Serie
Te acompaño a programar

Volumen I

Codifica en Python

Joe Llerena Izquierdo

Universidad Politécnica Salesiana

Codifica en Python

Serie

Te acompaño a programar

Volumen I

Codifica en Python

Joe Llerena Izquierdo

ABYA YALA | UNIVERSIDAD POLITÉCNICA SALESIANA

2020

CODIFICA EN PYTHON

1ra edición: Universidad Politécnica Salesiana
Av. Turuhuayco 3-69 y Calle Vieja
Cuenca-Ecuador
Casilla: 2074
P.B.X. (+593 7) 2050000
Fax: (+593 7) 4 088958
e-mail: rpublicas@ups.edu.ec
www.ups.edu.ec

CARRERA DE COMPUTACIÓN

Diagramación: Latex de autor

ISBN
Obra completa: 978-9978-10-428-6

ISBN
Volumen 1: 978-9978-10-429-3

Tiraje: 300 ejemplares

Impreso en Quito-Ecuador, agosto 2020

Publicación arbitrada de la Universidad Politécnica Salesiana

Codifica en Python

La serie: *Te acompaño a programar*

Joe Llerena Izquierdo

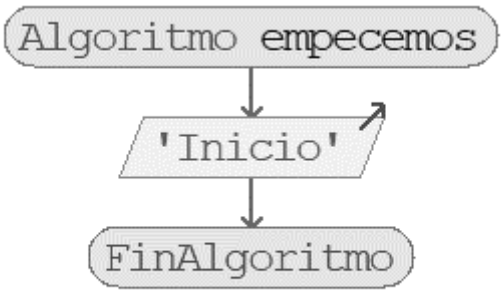
Contenido del libro

Tabla de Figuras

Tabla de Ejemplos

Tabla de Algoritmos

Tabla de Códigos



¡EMPECEMOS!

Presentación

El presente libro es una recopilación de problemas y ejercicios convertidos en actividades significativas de aprendizaje en el lenguaje de programación Python para estudiantes de los primeros años de estudio universitario. Una breve introducción a la iniciación en algoritmos y programas. Deseo que la lectura de este trabajo pueda ayudarte a conocer el mundo de la programación iniciando en este lenguaje llamado Python. El documento se compone de varios módulos cada uno de ellos con ejercicios explicativos. Los códigos de los programas y sus algoritmos los puedes ubicar en: https://github.com/joellerena/I_accompany_you_to_code disponible para todos.

Trataré de ir complementando los ejercicios con conceptos y explicaciones de forma clara y direccionando hacia las referencias oficiales del lenguaje. Adicionalmente existen ejercicios muy buenos en la web donde jóvenes, adultos, profesionales e interesados en el tema pueden también aportar con una mejor explicación. Es cierto que es una gran cantidad de información que se deba aprender, pero cada día es un paso en la forma de alcanzar los conocimientos que requerimos.

Ánimo, en esta lectura te acompaño.

Requerimientos

Para esta obra utilizaremos los diagramas de flujo del software de PSeInt ubicado en la siguiente dirección (en http://pseint.sourceforge.net/). Con él podrás representar el algoritmo desde su análisis y diseño; y, con el lenguaje de programación en Python 3.7 para su codificación y desarrollo como plataforma. El entorno de trabajo es Spyder 3.7.3 a septiembre del 2019 (*The Scientific Python Development Environment*) de Anaconda para transcribir el algoritmo a un programa de computador.

Python de https://www.python.org/

Anaconda de https://www.anaconda.com/

Spyder de https://anaconda.org/anaconda/spyder

Las versiones de cada uno de estos *softwares* no influyen en la realización de los algoritmos o programas que encontrarás en este documento. Recomendamos actualizar a las últimas versiones. ¡Puedes lograrlo!

Agradecimientos

Este trabajo es fruto de las jornadas de clases junto a los estudiantes de las carreras de Ingeniería en la universidad y desde mi iniciativa docente por hacer de cada clase un mejor espacio que pueda interesar y motivar a aprender. Ellos han logrado animarme con sus preguntas y sus dudas para encontrar la mejor explicación, un camino para entender la programación, a ustedes queridos jóvenes: por ustedes estudio, por ustedes trabajo en esto y a ustedes se los agradezco.

Las imágenes fueron creadas por el autor, excepto algunas diseñadas con Freepik en www.freepik.com.

El Autor

A Raquel, quien me enseña cada día de la vida y sus significados.
A nuestros hijos, con los que cada día soñamos en un mundo mejor.

A nuestros padres y maestros, que confiaron con nosotros.

MÓDULO 1

Módulo 1. Ideas básicas.

¿Qué es un algoritmo?

Un algoritmo es la secuencia de pasos, ordenados que alcanzan un fin determinado, una tarea o un resultado [1][2].

¿Qué es un programa?

Un conjunto de instrucciones que se escribe en un lenguaje de programación para que el computador puede entender o realizar una actividad [3].

¿Qué es un diagrama de flujo?

Es una forma de representar los algoritmos.

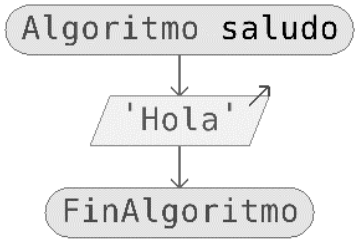
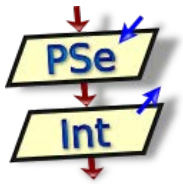


Figura 1. Representación en diagrama de flujo de un algoritmo

En este libro utilizaremos los gráficos de PSeInt que sirven para representar de forma gráfica un algoritmo [4], son diagramas de flujo que permiten entender el problema o ejercicio a realizar.



PSeInt puedes encontrarlo en
http://pseint.sourceforge.net/

Antes de iniciar el desarrollo de un programa, es prioritario entender el problema a resolver mediante un detallado análisis, luego representarlo con un diagrama de flujo y finalmente escribirlo en un lenguaje adecuado.

Figura 2. Etapas simplificadas en el desarrollo de una solución por computador

Mediante el uso de ejemplos didácticos, iremos especificando estas tres fases tratando de que con la práctica de ejercicios similares de forma recurrente puedas ir asimilando la lógica programática. Para el análisis, diseño y codificación tenemos estrategias aplicables, dependiendo del grado de dificultad en cada una de ellas.

Planteamiento de un problema

Para resolver un problema con el uso del computador, debemos entender bien el mismo en su contexto, analizando las posibles entradas y salidas que va a requerir el programa.

Ejemplo 1. Conversión de grados de temperatura

Realice un algoritmo que realice la conversión de grados Fahrenheit a Celsius. Utilice la fórmula adecuada.

Conversión de	a	Fórmula
Fahrenheit	Celsius	°C = (°F − 32) / 1.8
Celsius	Fahrenheit	°F = °C · 1.8 + 32
Fahrenheit	kelvin	K = (°F + 459.67) / 1.8
kelvin	Fahrenheit	°F = K · 1.8 − 459.67
Fahrenheit	Rankine	°Ra = °F + 459.67
Rankine	Fahrenheit	°F = °Ra − 459.67
Fahrenheit	Réaumur	°Ré = (°F − 32) / 2.25
Réaumur	Fahrenheit	°F = °Ré · 2.25 + 32

Figura 3. Fórmulas de conversión de temperatura

Solución: (Análisis)

El problema requiere de una fórmula específica [5], observe la tabla y utilice $°C = (°F - 32)/1.8$. Se ingresan valores enteros como entrada y se obtienen valores enteros o reales de salida. Visite https://es.noticias.yahoo.com/tiempo/ y presione la letra F o C visible.
Valor de entrada = F
Valor de salida = C
Proceso = fórmula identificada

Algoritmo (diseño)

Algoritmo 1: Conversión de temperatura

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 29 19:35:15 2019
"""
F = input("Ingrese un valor ")
C = round((int(F)-32)/1.8)
print("C: ",C)
```

```
Ingrese un valor 87
C:  31
```

Código 1: Conversión de temperatura

A simple vista, utiliza un código de combinaciones de funciones preestablecidas, más adelante se analizarán estas funciones. Para el ingreso de valores por teclado, se utiliza input ( ) y para

transformar, el valor ingresado a número entero requeriremos de int ( ). Compare el código actual con el siguiente.

Como desafío del ejercicio, se puede realizar un programa para las conversiones entre temperaturas.

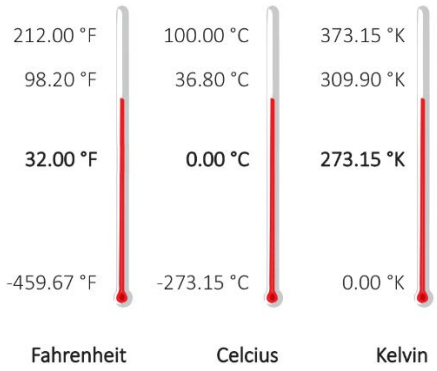


Figura 4. Comparación entre temperaturas

Ejemplo 2. Horas trabajadas de un empleado

Calcular el sueldo mensual de un trabajador conociendo la cantidad de horas trabajadas y el pago por hora.

Solución: (Análisis)

El problema requiere de una fórmula específica, $sueldo = horas\ trabajadas * costohora$
Se ingresan valores como entrada y se obtienen valores enteros o reales de salida.
Valor de entrada 1 al programa = cantidad de horas trabajadas = CHT
Valor de entrada 2 al programa = pago por hora = PPH
Valor de salida del programa = sueldo = CHT * PPH
Proceso = fórmula identificada

Algoritmo (diseño)

Algoritmo 2: Sueldo de un trabajador

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 29 19:53:47 2019
"""
CHT = input("Ingrese horas de trabajo: ")
print("Costo hora: ", end=" ")
PPH = input()
sueldo = int(CHT) * int(PPH)
print("Sueldo es: ",sueldo)
```

```
Ingrese horas de trabajo: 10
Costo hora:
3
Sueldo es:  30
```

Código 2: Sueldo de un trabajador

Las variables que se utilizan en estos ejemplos, por ahora son enteros y reales. Según su necesidad se utilizan funciones como int( ) para obtener el número de la cadena de caracteres y números que se ingresan por teclado.

Es decir, en este lenguaje, como en muchos, el uso de las variables requiere que se escriban de la misma forma durante todo el programa (usando el mismo nombre), conservando las mismas mayúsculas y minúsculas usadas. Los nombres deben utilizarse con una convención de tal forma que lo entiendan quiénes leen tu programa.

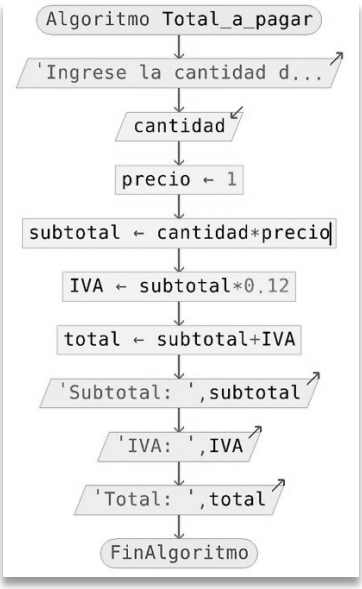
Ejemplo 3. Cobro del valor de IVA de un producto comprado

Calcular el valor a cancelar de un producto que cuesta $ 1 dólar, el programa debe mostrar cómo se presenta en una factura, subtotal (cantidad por precio), IVA (del subtotal) y total a pagar (la suma del subtotal + el IVA). Use de IVA el 12%.

Solución: (Análisis)

El problema requiere de fórmulas específicas, una de ellas es $IVA = subtotal * 0.12$
Se ingresa al programa la cantidad de un producto comprado, por ejemplo 1 litro de leche, que cuesta $ 1 dólar, el subtotal sería cantidad por lo que cuesta.

Algoritmo (diseño)



Algoritmo 3: Valor a pagar de un producto con IVA

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 29 19:53:47 2019
"""
cantidad = int(input("Ingrese la cantidad del producto: "))
precio = 1
subtotal = cantidad * precio
IVA = subtotal * 0.12
total = subtotal +  IVA
print("Subtotal: ",subtotal)
print("IVA: ",IVA)
print("total: ",total)
```

```
Ingrese la cantidad del producto: 1
Subtotal:  1
IVA:  0.12
total:  1.12
```

Código 3: Valor a pagar de un producto con IVA

MÓDULO 2

Módulo 2. Primeros pasos.

El uso de mensajes en pantalla

En esta sección trabajaremos con el método o función print( ) que permite la visualización de los caracteres o mensajes en pantalla. Su sintaxis es:

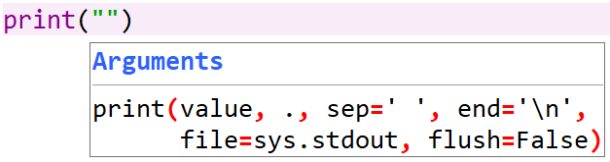


Figura 5. Función o método print ( ), imagen en Spyder

Se aprecian los argumentos o valores que se pueden "pasar" a la función print.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Aug 29 16:29:56 2019
4"""
5print("Hola mundo")
6print('Hola viejo mundo')
7print("Hola ",end='')
8print("mundo")
9
10print("Hola \"¿eres tú?\"")
```

```
Hola mundo
Hola viejo mundo
Hola mundo
Hola "¿eres tú?"
```

Explicación Código en Python

Línea 5: Uso de *print* con argumento "Hola mundo", uso de comillas dobles
Línea 6: Uso de *print* con argumento 'Hola mundo', uso de comillas simples
Línea 7: Uso de *end*=" para no agregar un salto de línea, une la línea en la que se encuentra con la siguiente (línea 8).
Línea 10: Uso comillas internas con la antibarra

Código 4: Función para mostrar caracteres en pantalla

A continuación se muestra otras combinaciones con la función print ( ), usando sep y end.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Aug 29 16:29:56 2019
4"""
5print("Hola","mundo",sep="-")
6print("Hola","mundo",sep=" ")
7print('Hola viejo mundo')
8print("Hola","viejo","mundo",sep='-')
9print("Hola","viejo","mundo",sep='\t')
10print("Hola","viejo","mundo",sep="")
11print("Hola","viejo","mundo",end="--")
```

```
Hola-mundo
Hola mundo
Hola viejo mundo
Hola-viejo-mundo
Hola    viejo    mundo
Holaviejomundo
Hola viejo mundo--
```

Explicación Código en Python

Línea 5: Uso de *print* con argumento "Hola", "mundo", y uso un separador guion medio.
Línea 6: Separador espacio en blanco
Línea 7: Uso normal de *print*
Línea 8: Uso del separador con guion medio.
Línea 9: Uso del separador tabulador
Línea 10: Uso del separador sin espacio
Línea 11: Uso del argumento *end* con doble guion medio al final de las expresiones, el uso de comas que separan las cadenas de caracteres, incluye un espacio entre ellas

Código 5: Diversos argumentos de print

Para el uso de los argumentos *file* y *flush,* print requiere de más argumentos. Específicamente *file* es utilizado para llevar un argumento hacia un archivo (usando *write*, más adelante lo trabajaremos como nos indica en el sitio web[1] de Python [6].

En el siguiente código el orden de presentación de los mensajes, por defecto tienen relación por el uso de un búfer, que se mantiene lleno o vacío dependiendo del parámetro en *flush*, por defecto es el valor *False* (no vaciar). Para el argumento de *file,* es muy adecuado para la escritura en archivos con *stdout* y mensajes de error con *stderr,* al no existir otro argumento, por defecto se muestra en pantalla dependiendo del almacenamiento (como una pila).

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Aug 29 16:29:56 2019
4 """
5 import sys
6 print("Mensaje para","error", file=sys.stderr, flush=True)
7 print("Salida desde archivo", file=sys.stdout, flush=True)
8 print()
9 print("Mensaje para","error", file=sys.stderr, flush=False)
10 print("Salida desde archivo", file=sys.stdout, flush=False)
```

```
Mensaje para error
Salida desde archivo

Salida desde archivo
Mensaje para error
```

Explicación Código en Python

Línea 5: Importación de *sys*
Línea 6: Mensaje almacenado en búfer por *file* pero con *flush=True* se presenta y vacía.
Línea 7: Mensaje almacenado en búfer por *file* al no encontrar archivo alguno, se presenta y vacía por *flush=True*.
Línea 9: Mensaje almacenado en búfer pero a estar *False*, no se vacía hasta que finalice el programa.
Línea 10: Mensaje se almacena en búfer y se presenta

Código 6: Función print con file y flush

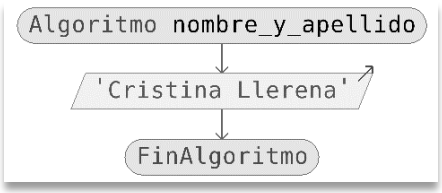
Ejemplo 4. Presentación de texto en pantalla

Muestre en pantalla su nombre y apellido.

Solución: (Análisis)

El problema requiere mostrar su nombre y apellido. Utilizaremos la función print.

Algoritmo (diseño)



Algoritmo 4: Presentar nombre y apellido en pantalla

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Aug 29 16:29:56 2019
4 """
5 print('Cristina Llerena')
6 print("Cristina Llerena")
7 print("\tCristina\tLlerena")
8 print("\tCristina\tLlerena",end="")
9 print("\nCristina","Llerena",sep="",end="")
10 print("\nCristina","Llerena",end="")
```

```
Cristina Llerena
Cristina Llerena
        Cristina        Llerena
        Cristina        Llerena
CristinaLlerena
Cristina Llerena
```

Código 7: Presentar texto en pantalla en varias formas

[1] Revise información oficial en https://docs.python.org/3/library/functions.html#print

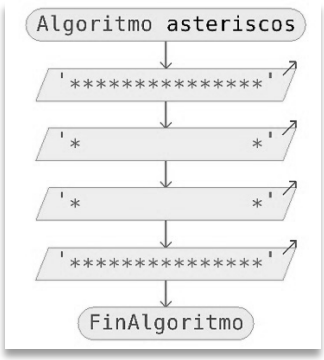
Ejemplo 5. Presentación de caracteres en pantalla

Muestre en pantalla un rectángulo de asteriscos.

Solución: (Análisis)

El problema requiere mostrar un rectángulo hecho con asterisco *. Utilizaremos varias líneas con la función print( ).

Algoritmo (diseño)



Algoritmo 5: Rectángulo de asteriscos

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  3 12:51:14 2019
"""
print("***************")
print("*             *")
print("*             *")
print("***************")
```

```
***************
*             *
*             *
***************
```

Código 8: Presentar un rectángulo hecho con asteriscos

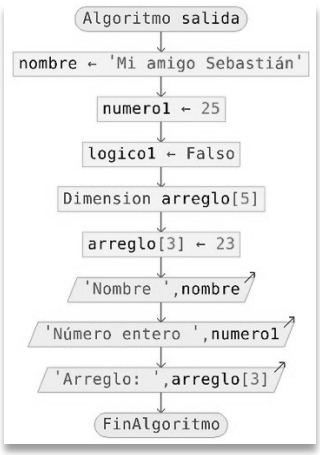
Ejemplo 6. Presentación de valores en pantalla

Muestre en pantalla un conjunto de tipos de datos: string, entero, lógico y un arreglo.

Solución: (Análisis)

El problema requiere mostrar un conjunto de datos que se pueden utilizar.

Algoritmo (diseño)



Algoritmo 6: Salida de valores en pantalla

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  3 13:01:54 2019
"""
nombre = "Mi amigo Sebastián"
numero1 = 25
logico1 = False
lista = [21,22,23,24,25]
print (nombre)
print (numero1)
print (logico1)
print (lista[2])
```

```
Mi amigo Sebastián
25
False
23
```

Código 9: Valores en pantalla

En el siguiente módulo indicaremos el significado de cada tipo de valor observado en este código.

Dando formato a los valores en pantalla

El método o función print( ) que permite la visualización en pantalla de los diferentes tipos de datos que Python nos ofrece, puede trabajar en conjunto con otros métodos o funciones para dar un formato adecuado en la presentación de los resultados, pondremos ejemplos comunes que se pueden requerir en algún momento de nuestra programación.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep  3 13:01:54 2019
4"""
5palabras = "Mi amiga María Cristina"
6numero = 19
7todo = "Primero: '{0}' y segundo: '{1}'".format(palabras,numero)
8print(todo)
9todo = "Primero: '{1}' y segundo: '{0}'".format(palabras,numero)
10print(todo)
```

```
Primero: 'Mi amiga María Cristina' y segundo: '19'
Primero: '19' y segundo: 'Mi amiga María Cristina'
```

Código 10: Función print con format( )

Explicación Código en Python

Línea 5: Una cadena de caracteres almacenada en la variable llamada *palabras.*

Línea 6: Un número entero almacenado en la variable llamada *numero* (las variables no llevan tilde).

Línea 7: El uso de *format ( ),* permite referenciar los índices según un orden requerido.

Línea 9: El uso de *format ( ),* cambiando el orden de los argumentos mediante referencias.

Trabajar con texto o números requerirá de un formato, para ello el uso de format ( ) es útil.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep  3 13:01:54 2019
4"""
5palabras = "Mi amiga María Cristina"
6numero = 19
7todo = "{:>28}".format("Mi amigo Sebastián")
8print(todo)
9print( "{:>28}".format(palabras) )
10print( "{:28}".format(palabras), end="")
11print("sigue")
12print( "{:^28}".format(palabras) )
13print( "{:.8}".format(palabras) )
14palabras = "Mi amiga María Cristina"
15otra_forma = f"Hola {palabras}"
16print(otra_forma)
```

```
          Mi amigo Sebastián
     Mi amiga María Cristina
Mi amiga María Cristina     sigue
  Mi amiga María Cristina
Mi amiga
Hola Mi amiga María Cristina
```

Código 11: Función format ( )

Explicación Código en Python

Línea 5: Cadena de texto

Línea 6: Número entero.

Línea 7: :>28 todo el texto en 28 espacios, alineación a la derecha.

Línea 9: Similar a la línea 7.

Línea 10: Utiliza 28 caracteres, luego de ellos muestra el siguiente texto.

Línea 12: :^28 centra el texto en 30 caracteres.

Línea 13: :.8 presenta 8 caracteres del contenido de *palabras*.

Línea 15: Uso de forma acortada para la función *format( )*.

MÓDULO 3

Módulo 3. Tipos de datos.

El uso de los distintos tipos de variables

En esta sección trabajaremos los tipos de datos que Python nos ofrece, datos simples y compuestos[2][7][8][9][10], de ellos: enteros, reales, cadenas de caracteres, listas, conjuntos diccionarios para este comienzo.

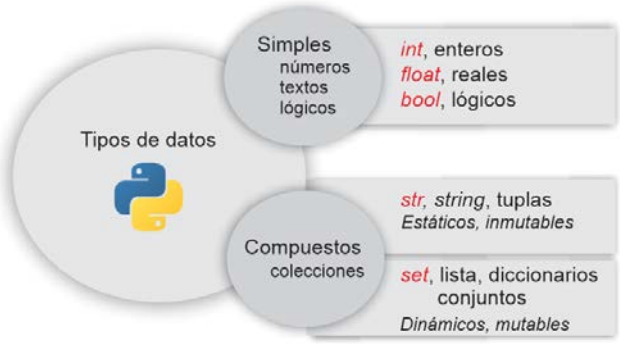


Figura 6. Tipos de datos en Python

Ejemplo 7. Presentación de formato de tipos de datos

Muestre en pantalla un conjunto de tipos de datos con el uso de la función format.

Solución: (Análisis)

El problema requiere mostrar un conjunto de datos almacenados en variables, se utilizará format( ) un método que referencia a lugares dentro de una cadena de caracteres, este método utiliza las llaves para la ubicación dentro de la cadena.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep  3 16:22:57 2019
4"""
5t = "palabra," #string
6n = 23 #entero
7print("Una palabra,",t,"ahora un número",n)
8m = "Una palabra, {} ahora un número '{}'".format(t,n)
9print(m) #uso de función de formato a referencia
10print(3 * t + t) #triplicar string y concatenar
```

```
Una palabra, palabra, ahora un número 23
Una palabra, palabra, ahora un número '23'
palabra,palabra,palabra,palabra,
```

Explicación Código en Python

Línea 5: Tipo de dato string.
Línea 6: Tipo de dato entero.
Línea 7: Uso de *format*( ) con referencia al contenido en m (tipo string).
Línea 8: Uso de *format*( ) con referencia al contenido en m (tipo string).
Línea 9: Tipo de dato string.
Línea 10: Propiedades de duplicidad y concatenación de cadenas de caracteres.

Código 12: Uso de referencias con format( )

Las variables son espacios de almacenamiento en memoria de un tipo de dato, con un tamaño específico. Llevan un nombre que identifique el contenido almacenado. En los siguientes

[2] Más información de variados ejercicios puedes encontrar en https://tutorial.recursospython.com/colecciones/

ejercicios se agregan nombres que relacionen lo que la variable almacena, de esta forma recordaremos dentro del programa aquellos valores que estamos realizando algún tratamiento.

Nombre	Tipo	Tamaño	Valor
m	str	1	Una palabra, palabra, ahora un número '23'
n	int	1	23
t	str	1	palabra,

Figura 7. Características de los tipos de datos

En la figura anterior, notamos el nombre de las variables utilizadas, el tipo de dato, su tamaño en bytes y contenido o valor almacenado.

Ejemplo 8. Trabajando con números enteros y reales

Muestre en pantalla un conjunto de números enteros o reales con sus operaciones comunes.

Solución: (Análisis)

El problema requiere mostrar operaciones de evaluación con números, por ejemplo, división con números enteros obtendremos números reales. Conocer los operadores como división entera o residuo serán útiles más adelante. Otras operaciones pueden ser desarrolladas con métodos o funciones existentes como la potencia de un números u operaciones a nivel bits.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  3 19:17:38 2019
"""
x = 7 / 5
y = 17 / 3 #decimales
z = 17 // 3 #sin redondeo
w = 17 % 3 #resto
v = 7 ** 2 #cuadrado
u = 7^2 #xor a nivel de bits 0111 xor 0010 = 0101 es 5
print(x)
print(y)
print(z)
print (w)
print (u)
```

```
1.4
5.666666666666667
5
2
5
```

Explicación Código en Python

Línea 5: División.
Línea 6: División.
Línea 7: División entera.
Línea 8: El residuo al dividir.
Línea 9: Potencia de un número.
Línea 10: Evaluación de números a nivel de bits.

Código 13: Uso de operadores y números enteros

Ejemplo 9. Definiendo el tipo de dato

Definir un número de tipo entero y uno de tipo real, luego asigne un valor coherente y finalmente presente su contenido en pantalla.

Solución: (Análisis)

El problema a simple vista requiere declarar dos tipos de datos uno entero y otro real, asigne el nombre que desee, siempre que no sea una palabra reservada. Luego asigne un número entero y otro real a cada variable creada. Presente en pantalla como en los anteriores programas.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep  4 22:09:08 2019
4"""
5numero = int()
6numero = 4
7sueldo = float()
8sueldo = 2600
9print(numero," ",sueldo)
```

```
4    2600
```

Código 14: Uso de la declaración del tipo de dato

Explicación Código en Python

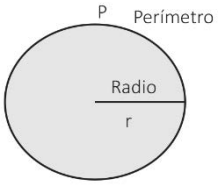
Línea 5: Declaración *int*.
Línea 6: Asignación.
Línea 7: Declaración *float*.
Línea 8: Asignación.
Línea 9: Presentación.

Ejemplo 10. Utilizando tipos de datos para la resolución de un ejercicio

Escriba un programa que calcule al perímetro y área de un círculo dado su radio.

Solución: (Análisis)

El problema nos pide el uso de fórmulas para encontrar el perímetro y área de una figura geométrica, para ello hay que recordar las fórmulas requeridas [11].

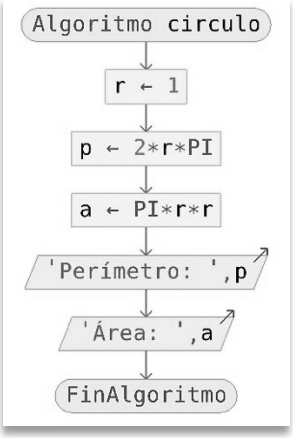


Para el perímetro tenemos: $p = 2\pi r$

Para el área tenemos: $a = \pi r^2$

Asignamos un valor a la variable radio, r, con el valor de 1. Con esto el resultado a obtenerse debe ser el valor de Pi.

Algoritmo (diseño)



Algoritmo 7: Área y perímetro de un círculo

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Sep  5 16:18:39 2019
4"""
5import math
6r = 1
7p = 2 * r * math.pi
8a = math.pi * r * r
9print("Perímetro: ", p)
10print("Área: ", a)
```

```
Perímetro:  6.283185307179586
Área:  3.141592653589793
```

Código 15: Área y perímetro de un círculo

Más adelante trataremos la librería math, que permite el uso de funciones trigonométricas.

Entendiendo las operaciones con los tipos de datos

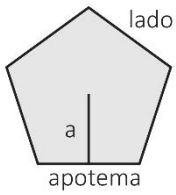
A medida que vayamos entendiendo el problema, el uso de los tipos de datos dependerá del contenido de las variables o valores a usar, por ejemplo, si requerimos usar una edad o trabajar con temperaturas, estos valores serán número enteros, pero si requerimos valores de sueldos, monedas, velocidad, áreas, distancias que usen punto decimal, el tipo de dato será real. En ejemplos más adelante usaremos los tipos de datos de texto y lógicos.

Ejemplo 11. Utilizando tipos de datos para geometría

Escriba un programa que calcule el área y perímetro de un pentágono regular.

Solución: (Análisis)

El problema nos pide el uso de fórmulas para encontrar el perímetro y área de una figura geométrica regular, para ello hay que recordar las fórmulas requeridas [12].

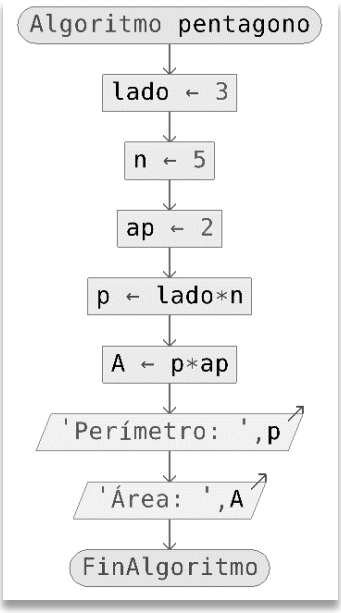


n = 5 por el número de lados

Para el perímetro tenemos: $p = lado\ x\ n$

Para el área tenemos: $A = p\ x\ a$ donde a es la apotema

Algoritmo (diseño)



Algoritmo 8: Perímetro y área de un pentágono

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Sun Sep  8 19:34:09 2019
4"""
5lado = 3
6n = 5
7ap = 2
8p = lado * n
9A = p * ap
10print("Perímetro: ",p)
11print("Área: ",A)
```

```
Perímetro:  15
Área:  30
```

Código 16: Perímetro y área de un pentágono

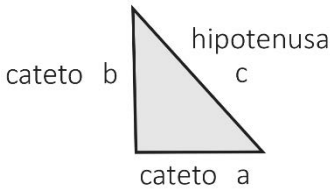
Cabe recordar que un programa es la secuencia de sentencias en un orden, en el programa anterior se puede deducir que las líneas de programación siguen una secuencia lógica para poder cumplir con las fórmulas utilizadas.

Ejemplo 12. Utilizando la librería math

Escriba un programa que calcule la hipotenusa de un triángulo rectángulo, dados los catetos.

Solución: (Análisis)

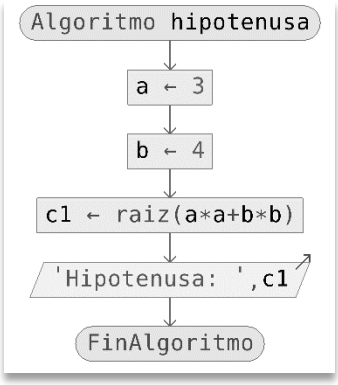
El problema nos pide el uso de la raíz cuadrada debido a que la hipotenusa es $c = \sqrt{a^2 + b^2}$, y para ello utilizaremos la librería matemática de Python en sus diferentes métodos o funciones que dispone. Utilizaremos las fórmulas conocidas [13].



a y b son catetos, y el valor hallar será c.

Python dispone del método o función sqrt para la raíz cuadrada y de hypot para encontrar la hipotenusa ingresando sólo los valores de los catetos y aplica el teorema de Pitágoras.

Algoritmo (diseño)



Algoritmo 9: Cálculo de la hipotenusa dado dos catetos

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  8 19:34:09 2019
"""
import math
a = 3
b = 4
c1 = math.sqrt(a*a + b*b)
c2 = math.hypot(a,b)
print("Hipotenusa: ",c1)
print("Hipotenusa: ",c2)
```

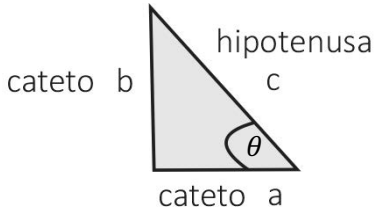
```
Hipotenusa:  5.0
Hipotenusa:  5.0
```

Código 17: Cálculo de la hipotenusa dado dos catetos

Escriba un programa que calcule la hipotenusa de un triángulo rectángulo, dados un cateto y un ángulo.

Solución: (Análisis)

El problema nos pide el uso de las funciones trigonométricas.



El ángulo θ y uno de los catetos son dados para el problema, se debe recordar las identidades trigonométricas [14].

$$Cos\ \theta = \frac{adyacente}{hipotenusa}$$

Python dispone de métodos o funciones trigonométricas que pueden ser de utilidad para problemas que lo requieran.

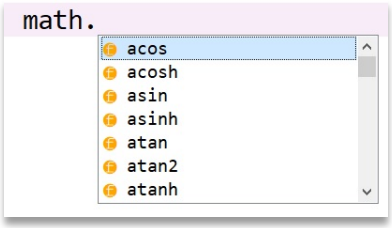


Figura 8. Librería matemática math

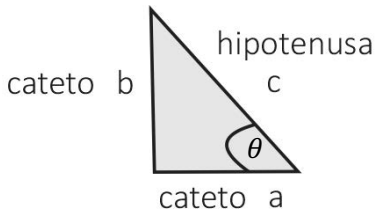
Para nuestro problema debemos utilizar el arcoseno o coseno inverso para obtener:

$$hipotenusa = \frac{adyacente}{\cos^{-1}\theta}$$

Si a = 3, b = 4, entonces c = 5

Para ello el ángulo θ es 53.13010235415598
lo vamos a demostrar.

Recordar que el cambio de grados a radianes viene dado por $\frac{\pi}{180}$ o al revés.



Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Sun Sep  8 19:34:09 2019
4"""
5import math
6a = 3 # adyacente a te
7b = 4 # opuesto a te
8c = 5 # hipotenusa
9angulo = 53.13010235415598
10h = a / math.cos(angulo * math.pi/180)
11print("Hipotenusa: ",h)
12identidad = 3 / 5
13te = math.acos(identidad)
14print("Ángulo: ",te * 180/math.pi)
```

Explicación Código en Python

Línea 5: Importación de los métodos o funciones de la librería *math*.
Línea 6: Cateto adyacente al ángulo.
Línea 7: Cateto opuesto al ángulo.
Línea 8: Hipotenusa a encontrar.
Línea 9: Ángulo ya conocido.
Línea 10: Hipotenusa aplicando la razón trigonométrica adyacente sobre el coseno del ángulo en radianes.
Línea 12: Identidad trigonométrica para el coseno del ángulo es igual a adyacente sobre hipotenusa, su resultado se le aplica el inverso del coseno o coseno inverso, el resultado al estar en radianes se multiplica por 180 sobre PI.

Código 18: Cálculo de la hipotenusa dado un ángulo y cateto

En los siguientes problemas, podemos utilizar los operadores aritméticos disponibles:

Descripción	Operador	Muestra	Resultado
Suma	+	n = 7 + 4	n = 11
Resta	–	n = 7 – 4	n = 3
Multiplicación	*	n = 7 * 4	n = 28
Potenciación	**	n = 7 ** 4	n = 2401
División	/	n = 7 / 4	n = 1.75
División entera	//	n = 7 // 4	n = 1
Resto o residuo	%	n = 7 % 4	n = 3
Negativo o inverso aditivo	–	n = – 7	Menos 7 o 7 negativo

Figura 9. Operadores aritméticos

MÓDULO 4

Módulo 4. Ingreso de datos.

Ingreso de valores desde teclado

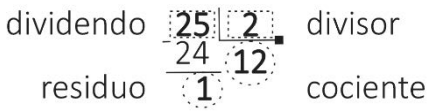
Hasta este momento se han realizado programas con valores inicializados, a partir de este momento realizaremos el ingreso de valores por teclado a menos que se indique lo contrario o para que la codificación no se extensa y podamos especificar sólo el código requerido para un mejor aprendizaje.

Ejemplo 13. Utilizando propiedades de las operaciones aritméticas

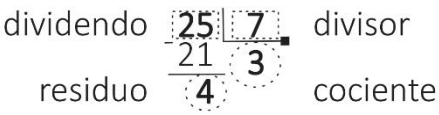
Escriba un programa que calcule el cociente y el residuo dados dos números enteros.

Solución: (Análisis)

El problema nos pide el ingreso de dos valores enteros por teclado y luego obtener el residuo.



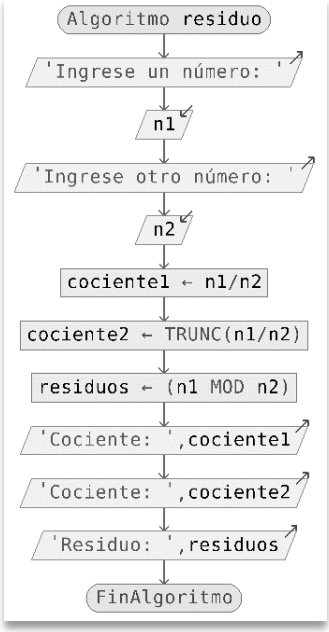
Un primer valor será el dividendo, el otro será el divisor, el resultado de dividirlos, será el cociente y un valor final llamado residuo es el que debemos mostrar.



Para cada uno de ellos, cociente y residuo, podemos obtenerlos mediante una división simple usando su operador o una función (dependiendo del lenguaje varía el nombre de la función).

Para este ejercicio debemos ingresar dos números y obtener el residuo, usamos MOD (de módulo o resto) o el operador % para obtenerlo.

Algoritmo (diseño)



Algoritmo 10: Cociente y residuo de dos números

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Mon Sep  9 12:05:19 2019
4"""
5n1 = int(input("Ingrese un número: "))
6n2 = int(input("Ingrese otro número: "))
7cociente1 = n1 / n2
8cociente2 = int(n1 / n2) # parte entera sin redondeo
9residuos = (n1 % n2)
10print("Cociente: ", cociente1)
11print("Cociente: ", cociente2)
12print("Residuo: ",residuos)
```

```
Ingrese un número: 25

Ingrese otro número: 7
Cociente:  3.5714285714285716
Cociente:  3
Residuo:  4
```

Código 19: Cociente y residuo de dos números

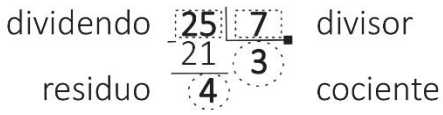
Cabe recordar, que se pueden realizar variaciones del mismo programa utilizando formas diferentes de aplicar las instrucciones con el uso de métodos o funciones existentes, incluso en

los módulos siguientes, el uso de funciones creadas o diseñadas por el propio programador puede encontrarse con un mundo de formas distintas de alcanzar matemáticamente el mismo resultado [15][16]. Como por ejemplo el uso de xor con Python[3].

Ejemplo 14. El uso de formateo en números decimales

Escriba un programa que calcule el cociente dados dos números enteros, muestre el resultado con varios dígitos decimales.

Solución: (Análisis)



El problema nos pide el ingreso de dos valores enteros por teclado y luego obtener el cociente con distintas cantidades de decimales, dos, tres, cuatro, etc.

Un primer valor será el dividendo, el otro será el divisor, luego se obtendrá el cociente de ellos mediante el uso del operador / y seguido usaremos un formateo para mostrar su resultado dependiendo de la cantidad de dígitos que obtengamos, finalmente usaremos tres nuevas funciones de redondeo, redondeo hacia arriba y hacia abajo.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 14:14:09 2019
"""
import math
n1 = int(input("Ingrese un número: "))
n2 = int(input("Ingrese otro número: "))
cociente1 = n1 / n2
cociente2 = int(n1 / n2) # parte entera sin redondeo
cociente3 = math.ceil(n1/n2)# redondeo hacia arriba
cociente4 = math.floor(n1/n2)# redondeo hacia abajo
cociente5 = round(n1 / n2)
print("Cociente1:    ", cociente1)
print("Cociente2 SR: ", cociente2)
print("Cociente3 CRArriba:", cociente3)
print("Cociente4 CRAbajo: ", cociente4)
print("Cociente5 CR: ", cociente5)
print("{:23.6f}".format(cociente1))
print("{:10.5f}".format(cociente1))# espacios blancos
print("{:010.5f}".format(cociente1))# con ceros
print("{:21d}".format(cociente2))# espacios blancos
print("{:4d}".format(cociente2))
print("{:3d}".format(cociente2))
print("{:2d}".format(cociente2))
print("{:1d}".format(cociente2))
print("{:04d}".format(cociente2))# con ceros
print("{:03d}".format(cociente2))
print("{:02d}".format(cociente2))
print("{:01d}".format(cociente2))
```

```
Ingrese un número: 25

Ingrese otro número: 7
Cociente1:     3.5714285714285716
Cociente2 SR:  3
Cociente3 CRArriba: 4
Cociente4 CRAbajo:  3
Cociente5 CR:  4
               3.571429
   3.57143
0003.57143
                     3
   3
  3
 3
3
0003
003
03
3
```

Explicación Código en Python

Línea 8: Cociente1 es la división normal.
Línea 9: Cociente2 es la división sin redondeo.
Línea 10: Cociente3 con redondeo hacia arriba.
Línea 11: Cociente4 con redondeo hacia abajo.
Línea 12: Cociente5, división con redondeo.

Línea 18: Contabiliza 23 espacios para todo el número obtenido con 6 dígitos decimales.
Línea 19: Utiliza 10 espacios para todo el número obtenido incluyendo el punto decimal, y llena con espacios en blanco de ser necesario a la izquierda.
Línea 20: Utiliza 10 espacios para todo, llenando con ceros a la izquierda si faltase.
Línea 21: Utiliza 21 espacios para todo el número, llena con espacios en blanco.
Línea 22: Utiliza 4 espacios para todo el número, llena con espacios en blanco a la izquierda.
Línea 23: Utiliza 3 espacios para todo el número, llena con espacios en blanco a la izquierda.

Línea 26: Utiliza 4 espacios para todo el número, llena con ceros a la izquierda.
Línea 27: Utiliza 3 espacios para todo el número, llena con espacios en blanco a la izquierda

Código 20: Formato a números decimales

[3] https://stackoverflow.com/questions/432842/how-do-you-get-the-logical-xor-of-two-variables-in-python

Ejemplo 15. Números Armstrong de tres cifras

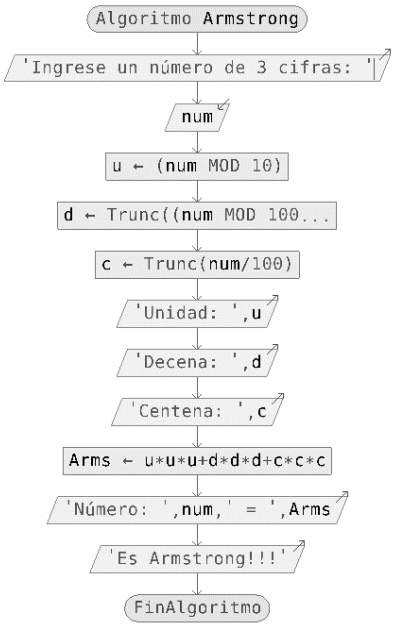
Escriba un programa que permita el ingreso de un número de tres dígitos y determine si es un número Armstrong [17][18]. Como el número que se ingresa posee 3 dígitos, la suma de cada uno de sus dígitos elevado a 3 debe ser igual al número.

Solución: (Análisis)

Figura 10. Análisis para determinar un número Armstrong de tres cifras

Ejemplo si ingreso el 153 es igual a $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.
El número 153 es un número Armstrong.

Algoritmo (diseño)



Algoritmo 11: Armstrong de tres cifras

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 16:27:20 2019
"""
num=int(input("Ingrese un número de tres cifras: "))
u=int(num % 10)
d=int(num % 100 /10)
c=int(num /100)
print("Unidad: ",u)
print("Decena: ",d)
print("Centena: ",c)
Arms = u*u*u + d*d*d + c*c*c
print("Número: ",num," = ",Arms)
print("Es Armstrong!!!")
```

```
Ingrese un número de tres cifras: 371
Unidad:  1
Decena:  7
Centena:  3
Número:  371  =  371
Es Armstrong!!!
```

Código 21: Armstrong de tres cifras

Cabe recordar que, en este ejemplo, el ingreso del valor de 371 (como otros pocos), ya conocemos que es Armstrong, más adelante realizaremos el código final donde se restringe a tres cifras el ingreso (validación) y el mismo programa identificará todos los Armstongs de tres cifras (uso de estructuras repetitivas).

Ejemplo 16. Números Armstrong de cuatro cifras

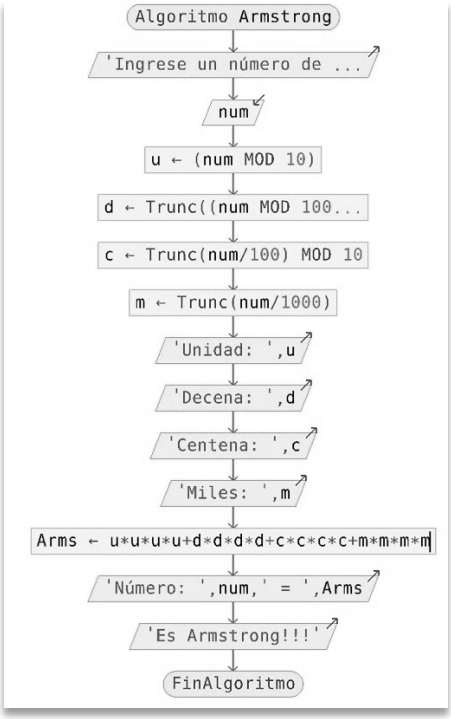
Escriba un programa que permita el ingreso de un número de cuatro dígitos y determine si es un número Armstrong [19][20]. Como el número que se ingresa posee 4 dígitos, la suma de cada uno de sus dígitos elevado a 4 debe ser igual al número.

Solución: (Análisis)

Figura 11. Análisis para determinar un número de Armstrong de cuatro cifras

Ejemplo: si ingreso el 8208 es igual a $8^4 + 2^4 + 0^4 + 8^4 = 4096 + 16 + 0 + 4096 = 8208$.
El número 8208 es un número Armstrong.

Algoritmo (diseño)



Algoritmo 12: Armstrong de cuatro cifras

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 11 10:21:21 2019
"""
num=int(input("Ingrese un número de cuatro cifras: "))
u=int(num % 10)
d=int(num % 100 /10)
c=int(num /100) % 10
m=int(num / 1000)
print("Unidad: ",u)
print("Decena: ",d)
print("Centena: ",c)
print("Miles: ",m)
Arms = u*u*u*u + d*d*d*d + c*c*c*c + m*m*m*m
print("Número: ",num," = ",Arms)
print("Es Armstrong!!!")
```

```
Ingrese un número de cuatro cifras: 8208
Unidad:  8
Decena:  0
Centena:  2
Miles:  8
Número:  8208  =  8208
Es Armstrong!!!
```

Código 22: Armstrong de cuatro cifras

Es posible que existan varias formas de colocar las fórmulas y de hallar el resultado de distintas maneras, y sobre todo a medida que se vaya conociendo más sentencias como operaciones en este lenguaje, el código irá reduciéndose debido a que se podrá resolver un mismo problema con creatividad.

MÓDULO 5

Módulo 5. Estructuras programáticas.

Estructuras secuenciales

Se reconoce a este tipo de estructura, cuando tenemos una secuencia de sentencias, que necesariamente deben de ir una a continuación de otra, por lógica y sentido común.

Ejemplo 17. Determine el valor resultante de las variables solicitadas

Escriba un programa que permita encontrar el valor final de las variables solicitadas.

1	2	3
a = 1	c = b	c = b
b = 2	a = c + b	presente a
c = a + b	b = a + 1	presente b
a = c	c = b + 4	presente c

Solución: (Análisis)

Si se observa el conjunto de columnas que van en orden, debemos ir realizando un seguimiento, línea a línea, de arriba abajo de cada una de las variables [21].

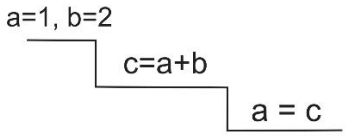
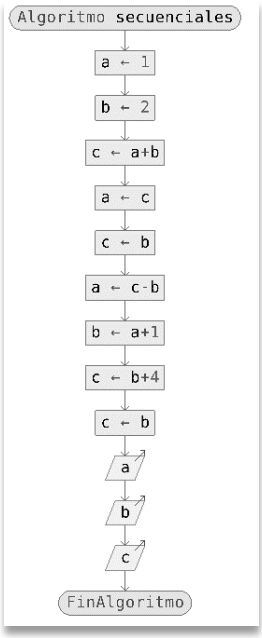


Figura 12. La solución tiene una secuencia lógica

Ejemplo, si la primera línea a es igual a 1 y la segunda b es igual a 2, entonces la tercera línea que es c, tiene el valor de a sumado b.

Algoritmo (diseño)



Algoritmo 13: Estructura secuencial

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 11 20:05:51 2019
"""
a = 1
b = 2
c = a + b
a = c
c = b
a = c - b
b = a + 1
c = b + 4
c = b
print("a: ",a)
print("b: ",b)
print("b: ",c)
```

```
a:  0
b:  1
b:  1
```

Nombre	Tipo	Tamaño	Valor
a	int	1	0
b	int	1	1
c	int	1	1

Código 23: Estructura secuencial y visualización del explorador de variables[4]

[4] Ver la ayuda de Spyder en https://docs.spyder-ide.org/

Estructuras selectivas

Estructuras selectivas simple

Cuando en un programa tenemos la elección de un camino entre dos existentes, se evalúa una condición, y de ella, su valor de verdadero o falso, dependerá la secuencia a seguir del programa.

Figura 13. Diagrama de flujo de la estructura selectiva simple

Estas estructuras, se rigen por una condición C, que utiliza operadores de relación, y su resultado, luego de evaluar es un valor booleano, verdadero o falso. Al "seleccionar" el camino a seguir (V o F) se ejecutan o realizan un bloque de sentencias, que pueden ser similares estructuras anidadas o un salto a subcódigos como funciones, métodos o archivos externos.

Ejemplo 18. Determine si una persona es mayor de edad

Escriba un programa que permita el ingreso de un número entero, que simbolice la edad de una persona, muestre a continuación sólo si es mayor de edad.

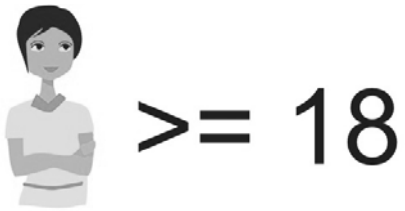


Figura 14. Una persona es mayor de edad, si es mayor e igual a 18 años

Solución: (Análisis)

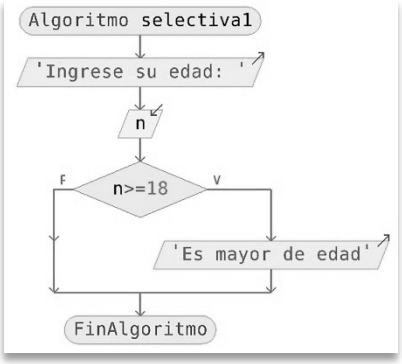
Si analiza el problema, tenemos que utilizar los operadores de relación.

Los operadores relacionales que encontramos en este lenguaje, así como en otros son:

Descripción	Operador	Muestra – condición	Resultado
¿son iguales?	==	c = 7 == 4	FALSE
¿no son iguales?	!=	c = 7 != 4	TRUE
¿es menor?	<	c = 7 < 4	FALSE
¿es mayor?	>	c = 7 > 4	TRUE
¿es menor e igual?	<=	c = 7 <= 4	FALSE
¿es mayor e igual?	>=	c = 7 >= 4	TRUE

Figura 15. Operadores de relación

Algoritmo (diseño)



Algoritmo 14: Estructura selectiva simple

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 11 20:35:05 2019
"""
n=int(input("Ingrese su edad: "))
if (n>=18):
    print("Es mayor de edad")
```

```
Ingrese su edad: 19
Es mayor de edad
```

Código 24: Estructura selectiva simple

Ejemplo 19. Determine la identidad y pertenencia

En Python encontraremos operadores que determinan la identidad de un objeto en una referencia, así como la pertenencia de un valor en una variable. En la siguiente figura, podemos utilizar los operadores de pertenencia e identidad disponibles:

Descripción	Operador	Muestra – condición	Resultado
Pertenencia – verificador Se aplica a los valores	**in**	palabra = "universo" "s" in palabra	TRUE
Identidad – referencia Se aplica a las variables	**is**	a = 2, b = 2, c = none a is b	TRUE
		a is not b	FALSE
		a is c	FALSE

Figura 16. Operadores de identidad y pertenencia

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 11 22:12:36 2019
"""
num1 = 3
num2 = 3.0
letra = "3"
num3 = 5
num4 = 3
conjunto = [1, 2, 3, 4]
palabra = "universidad"
lista1 = ["cristina","sebastián"]
tupla1 = ("cristina","sebastián")
#identidad
print("Identidad")
print("1: ",num1 is num2)
print("2: ",num1 is letra)
print("3: ",num1 is num3)
print("4: ",num1 is num4)
print("5: ",num1 is palabra)
print("6: ",lista1 is tupla1)
#pertenencia
print("Pertenencia")
print("7: ","univer" in palabra)
print("8: ",num1 in conjunto)
```

Código 25: Identidad y pertenencia

Explicación Código en Python

Línea 16: 1: El valor de 3 no es 3.0. (entero con real)
Línea 17: 2: El valor de 3 no es la letra "3".
Línea 18: 3: El valor de 3 no es 5
Línea 19: 4: El valor de 3 SÍ es 3
Línea 20: 5: El valor de 3 no es "universidad"
Línea 21: 6: La lista1 no es una tupla1

Línea 24: 7: "univer" está en palabra
Línea 25: 8: La variable num1 está en conjunto

```
Identidad
1:  False
2:  False
3:  False
4:  True
5:  False
6:  False
Pertenencia
7:  True
8:  True
```

Ejemplo 20. Determine si un número en positivo

Escriba un programa que permita el ingreso de un número entero, y el programa determine si es positivo.

Figura 17. La recta numérica divide a los números en negativos, positivos y el cero

Solución: (Análisis)

Si analiza el problema, el valor que debe ser ingresado, debe cumplir una condición, que n > 0, con esto sólo se verifica si el número n, es mayor que cero y con esto, la respuesta es: positivo.

En Python la identación o tabulación es importante observe la línea 7 del código.

Algoritmo (diseño)

Algoritmo 15: Estructura selectiva simple que valida un número mayor a cero

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Sep 12 08:49:45 2019
4"""
5n = int(input("Ingrese un número: "))
6if n > 0:
7    print(n," es positivo")
```

```
Ingrese un número: 56
56  es positivo
```

Código 26: Estructura selectiva simple que valida un número mayor a cero

Ejemplo 21. Determine si divisible para dos

Escriba un programa que permita el ingreso de un número entero, y luego determine si el número ingresado es divisible para dos.

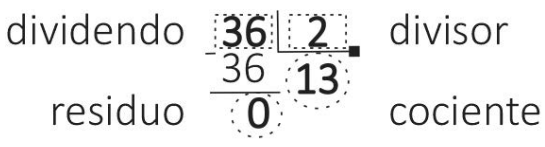
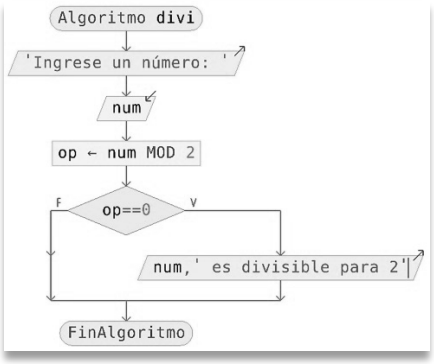


Figura 18. Elementos componentes de una división

Solución: (Análisis)

Si analiza el problema, sobre el valor ingresado debe realizarse una operación aritmética y sobre su resultado, cumplir una condición, que su residuo deba ser 0, con esto sólo se verifica el número n, siendo n divisible para dos. En Python los dos puntos indican el inicio de una secuencia de código de una estructura o bloque de sentencias.

Algoritmo (diseño)



Algoritmo 16: Estructura selectiva simple que valida una operación

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 10:21:42 2019
"""
num = int(input("Ingrese un número: "))
op = num % 2
if (op==0):
    print(num," es divisible para 2")
```

```
Ingrese un número: 36
36  es divisible para 2
```

Código 27: Estructura selectiva simple que valida una operación

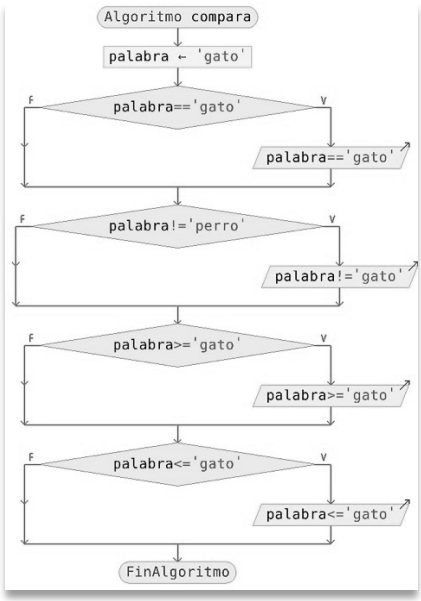
Ejemplo 22. Determine el valor de verdad entre dos cadenas de caracteres

Escriba un programa que permita comparar dos cadenas de caracteres usando if.

Solución: (Análisis)

Más adelante trataremos cadenas de caracteres y formas de evaluarlas. Por ahora usaremos la estructura if en el mejor de los casos. Sólo se da a escoger por verdadero.

Algoritmo (diseño)



Algoritmo 17: Estructura selectiva simple que valida una cadena

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 12:32:21 2019
"""
palabra = "gato"
if (palabra == "gato"):
    print(palabra == "gato")
if (palabra != "perro"):
    print(palabra != "gato")
if (palabra >= "gato"):
    print(palabra >= "gato")
if (palabra <= "gato"):
    print(palabra <= "gato")
```

```
True
False
True
True
```

Código 28: Estructura selectiva simple que valida una cadena

Hay que indicar que el segundo if del código anterior, el resultado de la condición es verdadero ya que *palabra* (que es gato), no es “perro”, entonces por verdadero ingresa y presenta la condición que deseamos inicialmente presentar *palabra* != “gato” que es falso porque sí lo es.

Estructuras selectivas doble

Cuando en una secuencia de instrucciones tenemos la elección de un camino entre dos existentes, la condición se evalúa, y de ella, su valor puede ser de verdadero o falso. Ahora trabajaremos con las dos respuestas.

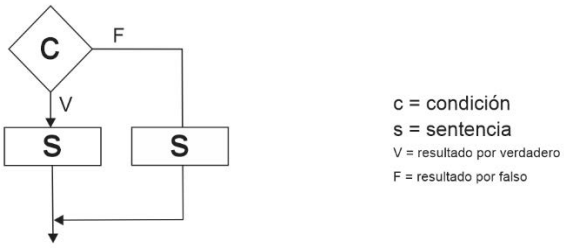


Figura 19. Diagrama de flujo de la estructura selectiva doble

Para los siguientes problemas tendremos la posibilidad de tener una u otra respuesta, evaluando a condición.

Ejemplo 23. Determine si un número es par o impar

Escriba un programa que permita el ingreso de un número y determine si es par o impar, asuma el cero como par.

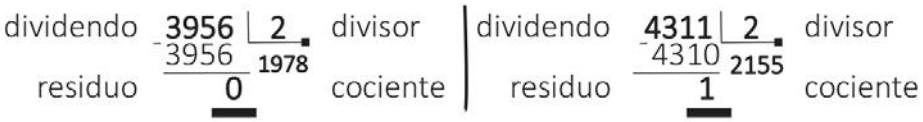
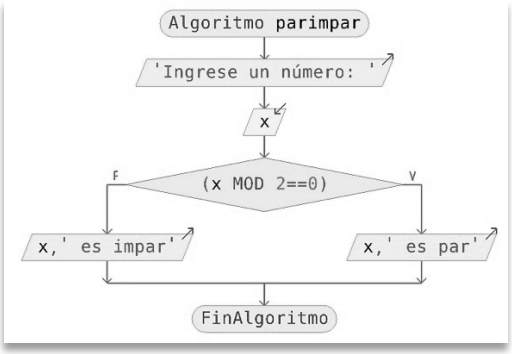


Figura 20. Elementos de la división de dos números

Solución: (Análisis)

Utilizaremos la estructura selectiva doble. Adicionalmente evaluaremos con el operador del módulo o resto, si obtenemos cero, por verdadero tendremos un número par, caso contrario un número impar. Con la aparición del else: cambia la identación al nivel del if, recuerde los dos puntos.

Algoritmo (diseño)



Algoritmo 18: Estructura selectiva doble

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 13:24:27 2019
"""
x = int(input("Ingrese un número: "))
if (x % 2 == 0):
    print(x," es par")
else:
    print(x," es impar")
```

```
Ingrese un número: 3956
3956  es par
Ingrese un número: 4311
4311  es impar
```

Código 29: Estructura selectiva doble

Estructuras selectivas anidada

En alguna ocasión es probable que tengamos más de una opción a escoger, con ello las bifurcaciones van teniendo aparición una detrás de otra a medida que las condiciones se cumplen o no.

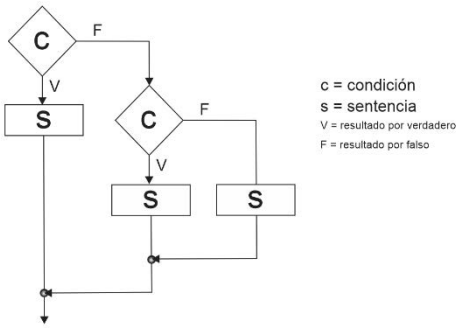


Figura 21. Diagrama de flujo de la estructura selectiva anidada

Ejemplo 24. Determine el estado de la persona según su edad

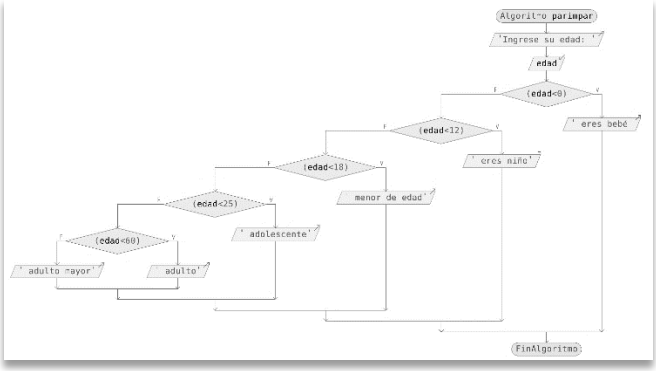
Escriba un programa que permita el ingreso de un número que represente la edad.

Figura 22. Edades de una persona, "Designed by Freepik"

Solución: (Análisis)

Utilizaremos edad < 0, bebé, edad < 12, niño, edad < 18, menor de edad, edad < 25, adolescente, edad < 60, adulto, caso contrario adulto mayor. Menor a cero no es admitido.

Algoritmo (diseño)



Algoritmo 19: Estructura selectiva anidada

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 14:07:44 2019
"""
edad=int(input("Ingrese su edad: "))
if(edad < 0):
    print("eres bebé")
elif (edad < 12):
    print("eres niño")
elif (edad < 18):
    print("menor de edad")
elif (edad < 25):
    print("adolescente")
else:
    if (edad < 60):
       print("adulto")
    else:
       print("adulto mayor")
```

```
Ingrese su edad: 78
adulto mayor
```

Código 30: Estructura selectiva anidada

Estructuras selectivas múltiple

En Python, las estructuras aprendidas como switch – case – default no existen, para ello hay formas creativas programáticas que usa estructuras de almacenamiento para responder a su estructura.

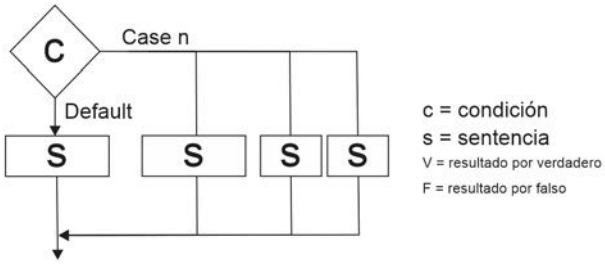


Figura 23. Diagrama de flujo de la estructura selectiva múltiple

Ejemplo 25. Determine el día de la semana dependiendo de un número ingresado

Escriba un programa que permita el ingreso de un número que represente un día de la semana. Su programa debe mostrar el nombre del día.

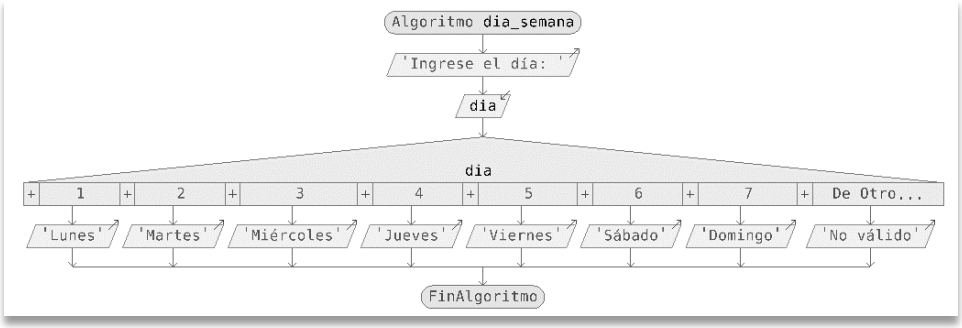


Figura 24. Edades de una persona, "Designed by Freepik"

Solución: (Análisis)

Utilizaremos una estructura llamada diccionario de almacenamiento para alojar los posibles nombres de los días a utilizar [22][23][24][25].

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 14:33:22 2019
"""
diccionario = {
        1: "lunes",
        2: "martes",
        3: "miércoles",
        4: "jueves",
        5: "viernes",
        6: "sábado",
        7: "domingo"
        }
dia=int(input("Ingrese el número del día: "))
print("Día: ", diccionario[dia])
```

```
Ingrese el número del día: 5
Día:  viernes
```

Código 31: Una variación para simular una estructura selectiva múltiple

Ejemplo 26. El mayor de tres números enteros con operadores relacionales

Escriba un programa que permita visualizar cuál es el mayor de tres números enteros ingresados por teclado.

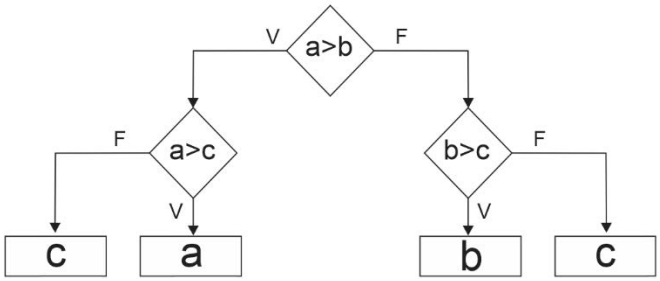
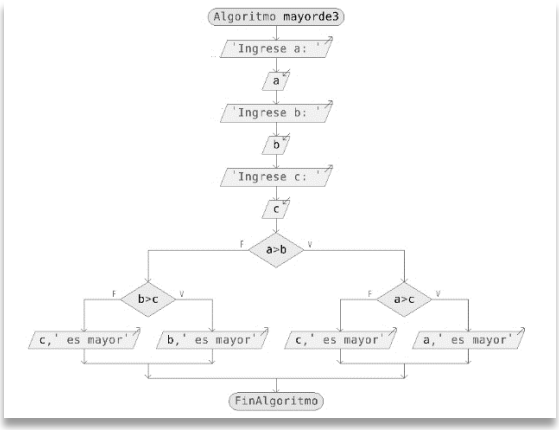


Figura 25. Mayor de tres números, operadores relacionales

Solución: (Análisis)

Dado el diagrama de flujo para su comprensión, hay que recordar que la validación entre números depende de las condiciones que agreguemos para ir filtrando cada posibilidad de comparación. A medida que se desee validar más números, más bifurcaciones.

Algoritmo (diseño)



Algoritmo 20: Estructura anidada que validad tres números

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 07:28:01 2019
"""
a = int(input("Ingrese el valor de a: "))
b = int(input("Ingrese el valor de b: "))
c = int(input("Ingrese el valor de c: "))
if(a>b):
    if(a>c):
        print(a," es mayor")
    else:
        print(c," es mayor")
else:
    if(b>c):
        print(b," es mayor")
    else:
        print(c," es mayor")
```

```
Ingrese el valor de a: 10

Ingrese el valor de b: 11

Ingrese el valor de c: 12
12  es mayor
```

Código 32: Estructura anidada que validad tres números

El uso de los operadores lógicos permite acortar las bifurcaciones y extender las expresiones.

Descripción	Operador	Muestra – condición	Resultado
Tabla de verdad de Y V and V es V, lo demás F	**Y and &&** **&&**	V and V V and F, F and V, F and F	TRUE FALSE
Tabla de verdad de O F or F es F, lo demás V	**O Or ||** **||**	F or F F or V, Vor F, V or V	FALSE TRUE
Tabla de negación	**No Not ~ !=**	Not(V)	FALSE

Figura 26. Operadores lógicos

Ejemplo 27. El mayor de tres números enteros con operadores lógicos

Escriba un programa que permita visualizar cuál es el mayor de tres números enteros ingresados por teclado, utilice operadores lógicos.

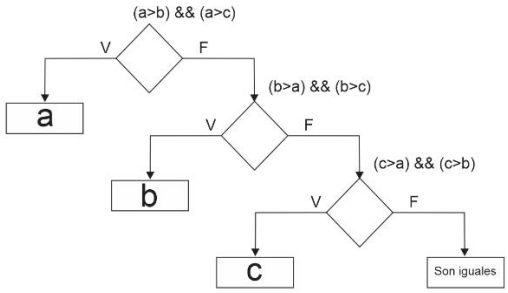
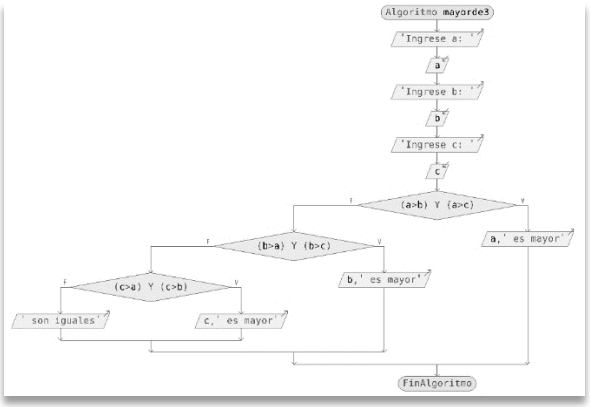


Figura 27. Mayor de tres números, operadores lógicos

Solución: (Análisis)

Dado el diagrama de flujo para su comprensión, hay que recordar que la validación entre números depende de las condiciones que agreguemos para ir filtrando cada posibilidad de comparación. A medida que se desee validar más números, más bifurcaciones.

Algoritmo (diseño)



Algoritmo 21: Estructura selectiva anidada con operadores lógicos

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 09:06:15 2019
"""
a = int(input("Ingrese el valor de a: "))
b = int(input("Ingrese el valor de b: "))
c = int(input("Ingrese el valor de c: "))
if((a>b) and (a>c)):
    print(a," es mayor")
else:
    if((b>a) and (b>c)):
        print(b," es mayor")
    else:
        if((c>a) and (c>b)):
            print(c," es mayor")
        else:
            print(" son iguales")
```

```
Ingrese el valor de a: 2

Ingrese el valor de b: 2

Ingrese el valor de c: 2
 son iguales
```

Código 33: Estructura selectiva anidada con operadores lógicos

Más adelante, seguiremos utilizando los distintos tipos de operadores, relacionales y lógicos que al combinarse con otros tipos de operadores podemos conseguir programas con códigos más cortos y estructuras de control con lógica más fácil de entender.

Compare los códigos de los programas anteriores con los siguientes, cada uno tiene algo nuevo que clarifica lo visto.

Estructuras repetitivas

Estructuras repetitivas mientras (while)

En Python, las estructuras repetitivas son dos, mientras y para, o while y for, como son conocidas. De esto, realizaremos ejercicios que nos permitan familiarizarnos con ellas.

La estructura mientras, o while, es una estructura restrictiva, ya que la condición se encuentra antes del bloque de sentencias.

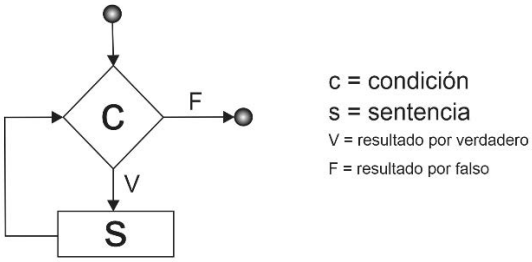


Figura 28. Estructura repetitiva mientras o while

Ejemplo 28. Muestre la secuencia de números del 1 al 10

Escriba un programa que permita visualizar los números del 1 al 10 de forma ascendente y luego descendente.

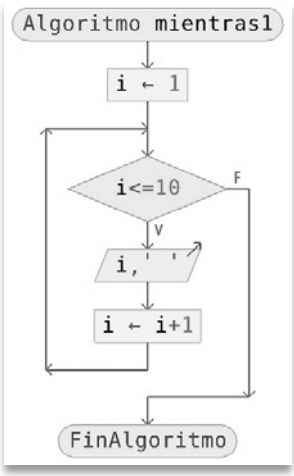
1 2 3 4 5 6 7 8 9 10 ascendente
10 9 8 7 6 5 4 3 2 1 descendente

Figura 29. Números ascendentes y descendentes

Solución: (Análisis)

Para el uso de la estructura repetitiva requiere de una variable de control, inicializada en un valor y luego incrementada o decrementada dependiendo de una cantidad, 1 generalmente.

Algoritmo (diseño)



Algoritmo 22: Estructura repetitiva while

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 23:08:55 2019
"""
i = 1
while i <= 10:
    print(i, end=" ")
    i += 1

print()

i = 10
while i >= 0:
    print(i, end=" ")
    i -= 1
```

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1 0
```

Código 34: Estructura repetitiva while

Ejemplo 29. Determine si un número es primo.

Escriba un programa que permita ingresar un número y determinar si es un número primo o no. Para ello debe recordar que, un número primo es aquel que es divisible para sí mismo y para la unidad.

1	2	3	5	7
11	13	17	19	
23	29			
31	37			
41	43	47		
53	59			
61	67			
71	73	79		
83	89			
97				

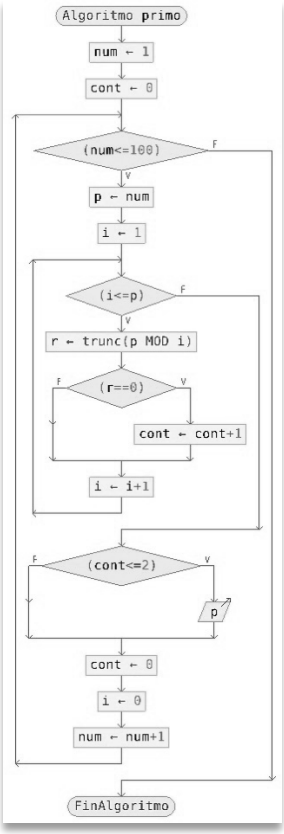
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Figura 30. Ascendencia y descendencia de números

Solución: (Análisis)

Puede realizar con distintas estructuras de repetición. Usaremos mientras. Adicionalmente se pueden anidar estructuras unas con otras. En este ejemplo tenemos un while dentro de otro debido a que uno avanza con los números del 1 al 100 y el otro permite realizar las divisiones del 1 hasta el mismo número, verificando que el residuo sea cero, si es cero es divisible. Recuerde divisible para uno para sí mismo.

Algoritmo (diseño)



Algoritmo 23: Estructura repetitiva anidada

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 10:14:10 2019
"""
num = 1
con = 0
while(num <= 100):
    if(num % 10 == 0):
        print("")
    p = num
    i = 1
    while(i <= p):
        r = p % i
        if (r == 0):
            con = con + 1
        i = i + 1
    if(con <= 2):
        print("\t",p,end=" ")
    con = 0
    i = 0
    num = num + 1

```

```
1       2       3       5       7
11      13      17      19
23      29
31      37
41      43      47
53      59
61      67
71      73      79
83      89
97
```

Código 35: Estructura repetitiva anidada

Estructuras repetitivas para (for)

La estructura repetitiva para, o for, es utilizada cuando conocemos la cantidad de iteraciones, repeticiones o ciclos que debe realizar el lazo o estructura.

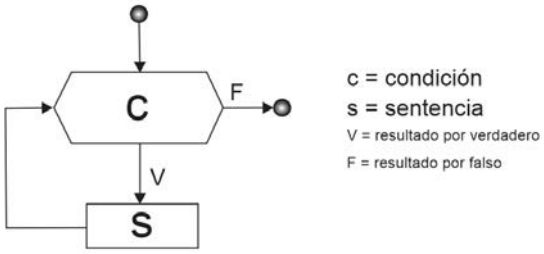


Figura 31. Estructura repetitiva para o for

Ejemplo 30. Muestre el rango de número del 501 al 550.

Escriba un programa que permita mostrar los números del 501 al 550 en bloques de 10.

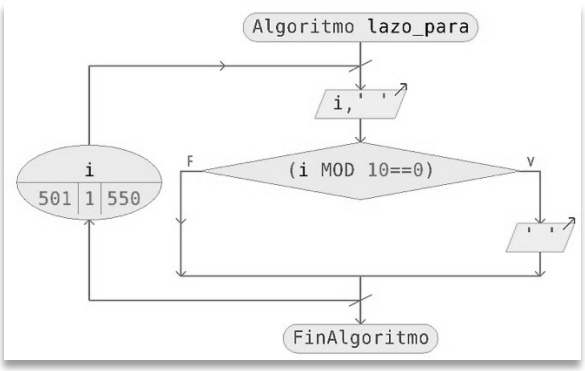
```
501 502 503 504 505 506 507 508 509 510
511 512 513 514 515 516 517 518 519 520
521 522 523 524 525 526 527 528 529 530
531 532 533 534 535 536 537 538 539 540
541 542 543 544 545 546 547 548 549 550
```

Figura 32. Bloques de resultados

Solución: (Análisis)

Cuando conocemos el número de repeticiones, es decir el inicio y fin de la secuencia de iteraciones, usaremos el lazo para o for.

Algoritmo (diseño)



Algoritmo 24: Estructura repetitiva para

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 13:21:28 2019
"""
for i in range(501, 551):
    print(i,end=" ")
    if(i % 10 == 0):
        print()
```

```
501 502 503 504 505 506 507 508 509 510
511 512 513 514 515 516 517 518 519 520
521 522 523 524 525 526 527 528 529 530
531 532 533 534 535 536 537 538 539 540
541 542 543 544 545 546 547 548 549 550
```

Código 36: Estructura repetitiva para

Ejemplo 31. Presente una tabla de multiplicar.

Escriba un programa que, dado un número entero, permita mostrar una tabla de multiplicar.

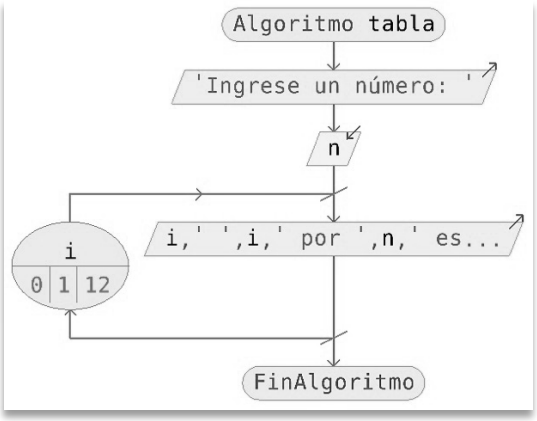
N	7	R
0	0 por 7	es 0
1	1 por 7	es 7
2	2 por 7	es 14
3	3 por 7	es 21
4	4 por 7	es 28
5	5 por 7	es 35

Figura 33. Tabla de multiplicar

Solución: (Análisis)

El ingreso por teclado de un valor entero, debe permitir generar sucesivas multiplicaciones, de 0 a 12, simulando una tabla de multiplicar.

Algoritmo (diseño)



Algoritmo 25: Estructura for, tabla de multiplicar

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 14:00:19 2019
"""
n = int(input("Ingrese un número: "))
for i in range (0, 12):
    print(i," ",i,"por",n,"es",(i*n) )
else:
    print("Finalizamos")
```

```
Ingrese un número: 7
0   0 por 7 es 0
1   1 por 7 es 7
2   2 por 7 es 14
3   3 por 7 es 21
4   4 por 7 es 28
5   5 por 7 es 35
6   6 por 7 es 42
7   7 por 7 es 49
8   8 por 7 es 56
9   9 por 7 es 63
10   10 por 7 es 70
11   11 por 7 es 77
Finalizamos
```

Código 37: Estructura for, tabla de multiplicar

El uso de series es muy común en los programas, se escribirá la secuencia o sucesión de Fibonacci [26], utilizando el lazo for.

Ejemplo 32. Presente la serie de Fibonacci.

Escriba un programa que, dado un número entero, permita mostrar los enésimos números de la sucesión de Fibonacci.

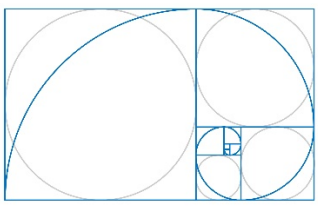
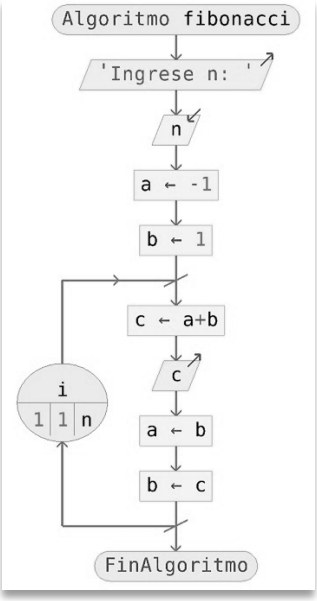


Figura 34. La espiral de Fibonacci, "Designed by Freepik"

Solución: (Análisis)

Se realiza el ingreso de un número entero por teclado, luego se presenta la sucesión.

Algoritmo (diseño)



Algoritmo 26: Sucesión de Fibonacci

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 14:40:44 2019
"""
n = int(input("Ingrese n: "))
a = -1
b = 1
for i in range(1, n+1):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
```

```
Ingrese n: 13
0 1 1 2 3 5 8 13 21 34 55 89 144
```

Código 38: Sucesión de Fibonacci

En la secuencia o sucesión de Fibonacci, se deben analizar sus diversas formas de solución y que se pueden encontrar en distintos lugares o libros. Una forma de entender la generación de esta sucesión es retrocediendo desde los dos primeros números 0 y 1, y encontrar una forma en retrospectiva de obtenerlos, la imagen siguiente se encuentra una explicación.

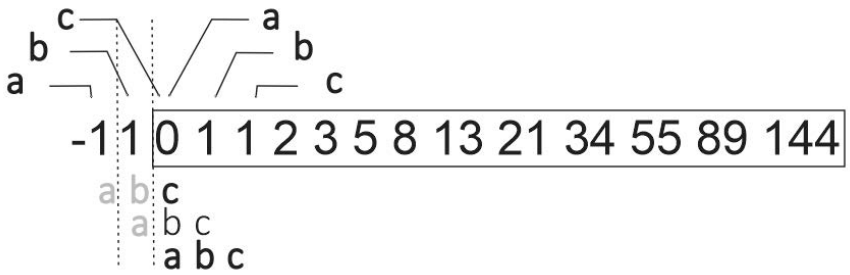


Figura 35. Generación de la sucesión de Fibonacci

Su análisis nos lleva a pensar que los primeros números requeridos para generar la sucesión son los valores -1 para *a* y 1 para *b*, siguiendo el código de que c = a + b.

Estructuras recursivas

Las estructuras recursivas permiten el uso de lo que conocemos como funciones o métodos que los utilizaremos más adelante. Por ahora representaremos funciones recursivas sencillas para entender su diseño.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Fri Sep 13 15:11:58 2019
4"""
5def fibonacci(n):
6    if (n==0):
7        return 0
8    if (n==1):
9        return 1
10    return(fibonacci(n-1)+fibonacci(n-2))
11
12n = int(input("Ingrese n: "))
13for i in range(0, n):
14    print(fibonacci(i), end=" ")
```

```
Ingrese n: 13
0 1 1 2 3 5 8 13 21 34 55 89 144
```

Explicación Código en Python

Línea 5: Uso de una función, def es la palabra reservada para su uso. Nombre: fibonacci, y tiene un argumento llamado n.
Línea 6: Para n igual 0,
Línea 7: … retorna 0.
Línea 8: Para n igual 1,
Línea 9: … retorna 1.
Línea 10: Se llama de forma recursiva a la misma función, con argumentos decrementados.

Línea 12: Se ingresa el valor de n
Línea 13: Se invoca varias veces a la función, ya que esta sólo retorna el enésimo término, con el for simulamos la serie.

Código 39: Fibonacci recursivo

Ejemplo 33. Presente el factorial de un número n.

Escriba un programa que determine el factorial de un número.

$$n! = 5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

Figura 36. Factorial del número 5

Solución: (Análisis)

Dado un número n, debemos utilizar un lazo for para recorrer la serie de números que intervienen en la acumulación de productos.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Fri Sep 13 15:55:57 2019
4"""
5n = int(input("Ingrese un número: "))
6acum = 1
7for i in range(0,n):
8    print((i+1),end=" ")
9    acum = acum * (i+1)
10print(" = ",acum)
```

Explicación Código en Python

Línea 5: Ingreso del valor de n.
Línea 6: Se utiliza una variable acumulador de productos que se **inicializa en 1**, debido a las multiplicaciones sucesivas, si **fuera suma sería 0**.
Línea 7: Lazo for que va de cero a n (n-1) cabe recordar que llega al valor n.
Línea 8: Un artificio para visualizar i sumado 1
Línea 9: El acumulador va guardando productos
Línea 10: Se muestra la acumulación de productos

Código 40: Factorial de un número n

Ejemplo 34. Presente el factorial de un número n, usando recursividad.

Escriba un programa que determine el factorial de un número, utilizando recursividad.

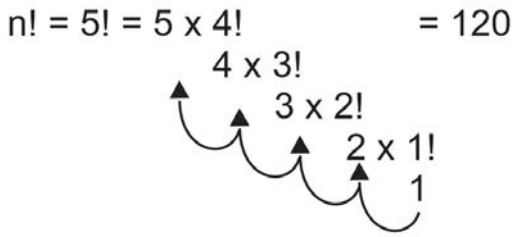


Figura 37. Factorial del número 5, con recursividad

Solución: (Análisis)

La recursividad es una forma elegante de desarrollar un problema iterativo. Dado un número n, debemos utilizar un método o función con multiplicaciones sucesivas invocadas hacia la misma función.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Fri Sep 13 16:22:01 2019
4"""
5def factorial(n):
6    if (n>0):
7        return (n * factorial(n-1))
8    else:
9        return 1
10
11n = int(input("Ingrese n: "))
12for i in range(1, n+1):
13    print(factorial(i), end=" ")
```

```
Ingrese n: 5
1 2 6 24 120
```

Explicación Código en Python

Línea 5: Uso de una función, def es la palabra reservada para su uso. Nombre: factorial, y tiene un argumento llamado n.
Línea 6: Si n es mayor a 0,
Línea 7: Retorna la multiplicación de n y se mantiene en memoria hasta la ejecución nuevamente de la función factorial con el argumento n - 1.
Línea 8: Si n no es mayor a cero, … es cero
Línea 9: … retorna 1.

Línea 11: Se ingresa el valor de n
Línea 12: Con el for efectuamos varias veces los factoriales, viendo el resultado el factorial de cada n.

Código 41: Factorial recursivo

Símil al ejercicio anterior, el uso de la recursividad no permite entender la autoinvocación de la misma función hasta una condición base, con ello las operaciones quedan en memoria, momentáneamente, hasta que se realicen todas las operaciones sucesivas.

Trabajar con rangos

En ciertos momentos, requeriremos valores límites, es decir valores que debemos tomar en cuenta de forma inclusiva. En matemáticas aprendimos a tomar estos valores identificando si es incluido o no en una operación. En la imagen vemos la recta numérica y la representación de un rango entre 10 y 90, mayores a 10 y menores a 90, en el centro el doble rayado, lo que debemos definir en un problema es que, si estos valores también se los evalúa o no, si se los toma en cuenta en el proceso o información final. En un módulo de desafíos se dejan problemas en los que debemos tener esta consideración en cuenta.

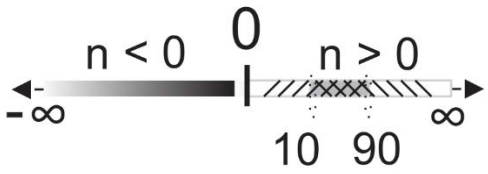


Figura 38. Rango de números entre 10 al 90

Como en matemáticas, usaremos un corchete o punto negro para representar la inclusión, en programación se usará el >= o el <= que representan la inclusión ya sea mayor igual o menor igual.

Ejemplo 35. Valide un número dentro de un rango.

Escriba un programa que, en un determinado rango, se valide un número. Utilice varias formas de escribir el mismo programa usando los operadores de relación y operadores lógicos.

Figura 39. Ejemplo de números límites en varios rangos

Solución: (Análisis)

Nos piden validar si un número es menor a cero (sin incluir), que esté entre 0 (inclusive) a 5 (inclusive) y si es mayor a 5 (sin incluir).

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Sat Sep 14 19:13:12 2019
4"""
5num1 = 4
6if(num1>=0 and num1<=5):
7    print("Buen ingreso")
8else:
9    print("Mal ingreso")
10if(num1<0 or num1>5):
11    print("Mal ingreso")
12else:
13    print("Buen ingreso")
```

Explicación Código en Python

Línea 5: Asignamos el valor de 4 a la variable num1.
Línea 6: Si num1 >= 0 y num1 <=5,
Línea 7: Se muestra *Buen ingreso* y es correcto ya que el 4 está en la intersección.
Línea 8: Si no
Línea 9: Se muestra *Mal ingreso*.
Línea 10: Otra forma, Si num1 < 0 o num1 > 5
Línea 11: Se muestra *Buen ingreso*
Línea 12: Si no.
Línea 13: Se muestra *Mal ingreso*.

Código 42: Ejercicio con rangos

Del código anterior debemos analizar detenidamente varias formas de resolver el problema, y encontramos las siguientes. Observe detenidamente la figura y note los cambios posibles con sus operadores.

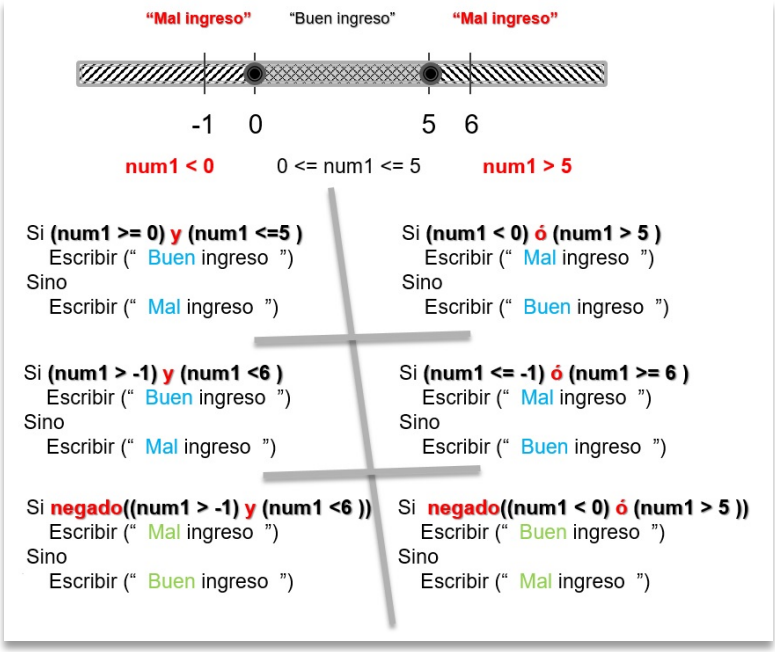


Figura 40. Posibles condiciones para un mismo problema, utilizando rangos

Entonces utilizando ahora todas las líneas de código, en 3 bloques tendremos:

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat Sep 14 19:13:12 2019
4 """
5 num1 = 4
6 ## bloque 1
7 if(num1>=0 and num1<=5):
8     print("Buen ingreso")
9 else:
10    print("Mal ingreso")
11 if(num1<0 or num1>5):
12    print("Mal ingreso")
13 else:
14    print("Buen ingreso")
15 ## bloque 2
16 if(num1>=-1 and num1<6):
17    print("Buen ingreso")
18 else:
19    print("Mal ingreso")
20 if(num1<=-1 or num1>=6):
21    print("Mal ingreso")
22 else:
23    print("Buen ingreso")
24    ## bloque 3
25 if not(num1>-1 and num1<6):
26    print("Mal ingreso")
27 else:
28    print("Buen ingreso")
29 if not(num1<0 or num1>5):
30    print("Buen ingreso")
31 else:
32    print("Mal ingreso")
```

Explicación Código en Python

Bloque 1
Línea 7-14: Es el ejemplo anterior

Bloque 2
Línea 16: Si num1 >= -1 y num1 <6.
Línea 17: Se muestra *Buen ingreso*
Línea 18: Si no
Línea 19: Se muestra *Mal ingreso*.
Línea 20: Otra forma, Si num1 <=-1 o num1 >=6
Línea 21: Se muestra *Mal ingreso*
Línea 22: Si no.
Línea 23: Se muestra *Buen ingreso*.

Similar al bloque 3, lo que se debe observar es cuándo se utiliza el apropiado mensaje, evaluando la condición. Y usando los operadores de relación adecuados > o >=, < o <= junto a los operadores lógicos, and, or o not podremos determinar la respuesta correcta.

Te motivamos a que encuentres otros código diferentes

Código 43: Ejercicio con rangos de varias formas

Hay que recordar que el uso del operador lógico AND (&&) es restrictivo, es decir, deben cumplirse todas las condiciones (intersección - disyunción) para que sea verdadero, en cambio que el operador OR (||) basta que se cumpla al menos una de ellas (conjunción) para que tome el valor de verdad.

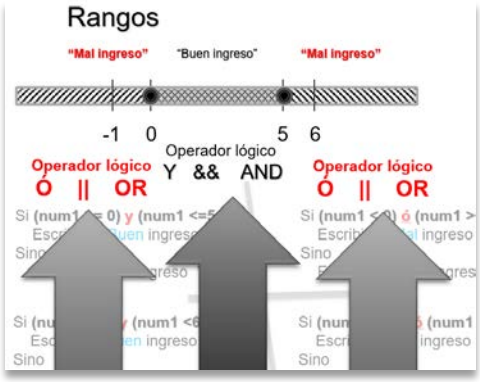


Figura 41. Utilidad de los operadores lógicos según los rangos

Trabajar con contadores, acumuladores e incrementadores/decrementadores

En programación tenemos términos que debemos asimilar, los contadores, acumuladores o incrementadores (o decrementadores) son variables que permiten contar, acumular o incrementar/decrementar valores o el caso de uno solo (sumar o restar 1).

Ejemplo 36. Programa que simula una libreta escolar.

Escriba un programa que permita el ingreso de tres notas de una asignatura, calcule el promedio y lo presenta en pantalla.

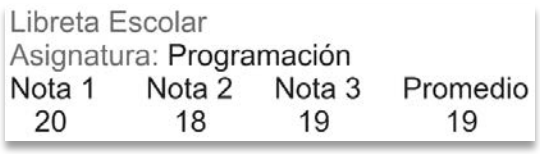


Figura 42. Formato de Libreta Escolar

Solución: (Análisis)

Para este programa, el uso del contador está explícito, siendo tres el número de notas, adicionalmente una variable está implícita, el acumulador de notas para luego obtener el promedio. Claramente usted debe ir pensando en el uso de las variables que requiera para lograr el resultado solicitado.

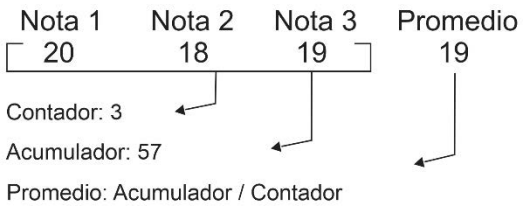


Figura 43. Descripción y valores de las variables del problema

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Mon Sep 16 15:15:26 2019
4"""
5print("LIBRETA ESCOLAR")
6print("Asignatura: Programación")
7acum = 0 #acumulador
8i = 0    #incrementador o contador
9promedio = 0.0
10while(i<3):
11    print("nota [",(i+1),"]", end=" ")
12    nota = int(input("Ingrese: "))
13    acum = acum + nota
14    i = i + 1
15promedio = acum / i
16print("Promedio: ",promedio)
```

```
LIBRETA ESCOLAR
Asignatura: Programación
nota [ 1 ]
Ingrese: 20
nota [ 2 ]
Ingrese: 18
nota [ 3 ]
Ingrese: 19
Promedio:  19.0
```

Explicación Código en Python

Línea 7: Uso de un acumulador (de notas).
Línea 8: Uso de un contador/incrementador en 1 .
Línea 9: Variable del promedio.
Línea 10: Uso de la estructura repetitiva y el contador
Línea 11: Artificio para visualizar en pantalla un valor más que el valor del incrementador.

Línea 13: Acumulador de notas.
Línea 14: Incrementador en 1.
Línea 15: Operación promedio

Nombre	Tipo	Tamaño	Valor
acum	int	1	57
i	int	1	3
nota	int	1	19
promedio	float	1	19.0

Código 44: Mientras controlado por contador

Hemos utilizado en el programa anterior, una estructura repetitiva conocida como while controlado por un contador. Pero existe otra manera de controlar al while conocida como while controlado por un centinela, el programa cambia desde la forma de entenderlo, comprenda bien el siguiente ejemplo

Ejemplo 37. Programa que simula el ingreso de n notas.

Escriba un programa que permita el ingreso de n notas de una asignatura, calcule el promedio y lo presenta en pantalla, adicionalmente para detener el ingreso deberá usar el número -99.

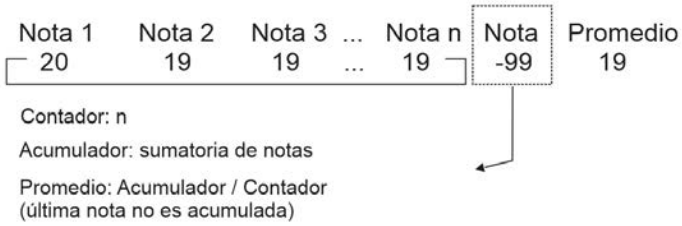


Figura 44. Formato de ingreso de n notas

Solución: (Análisis)

Para este programa, el uso del contador no está explícito, dependerá de la cantidad de notas que el usuario desee ingresar *n*, siendo el número -99 un valor de ingreso como una nota, pero sin ser tomado en cuenta. A este valor se lo conoce como valor centinela. Un valor señuelo que indica al programa la finalización de ingresos a través de la variable *nota*. Adicionalmente una variable está implícita, el acumulador de notas (que no acumula el último valor) para luego obtener el promedio. Para este problema, similar al anterior, utiliza nombres similares en sus

variables, es la forma de utilizar la estructura repetitiva lo que va a cambiar, observe la condición de este programa.

Código en Python (codificación) **Explicación Código en Python**

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 17 12:03:33 2019
4"""
5print("LIBRETA ESCOLAR")
6print("Asignatura: Programación")
7acum, i = (0, 0)
8nota, promedio = (0, 0.0)
9print("Ingrese las notas a continuación, -99 para finalizar")
10print("nota [",(i+1),"]", end=" ")
11nota = int(input("Ingrese: "))
12while(nota!=-99):
13    acum = acum + nota
14    i = i + 1
15    print("nota [",(i+1),"]", end=" ")
16    nota = int(input("Ingrese: "))
17if(i>0):
18    promedio = acum / i
19    print("Promedio: ",promedio)
20else:
21    print("Finalizó el programa")
```

```
LIBRETA ESCOLAR
Asignatura: Programación
Ingrese las notas a continuación, -99 para finalizar
nota [ 1 ]
Ingrese: 19
nota [ 2 ]
Ingrese: 18
nota [ 3 ]
Ingrese: 20
nota [ 4 ]
Ingrese: -99
Promedio:  19.0
```

Línea 7-8: Inicialización de dos valores simultáneamente.
Línea 9: Mensaje para finalizar.
Línea 10-11: Artificio para visualizar en pantalla un valor más que el valor del incrementador. Ingreso de una primera nota o el valor centinela.
Línea 12: Uso de la estructura repetitiva y el valor centinela. Note el uso del operador no es igual.
Línea 13-14: Acumulación de notas e incremento de i.
Línea 15-16: Similar a las líneas 10-11
Línea 17: Validar si *i* aumentó, prosigue con las operaciones caso contrario…
Línea 20-21: Finaliza el programa.

Nombre	Tipo	Tamaño	Valor
acum	int	1	57
i	int	1	3
nota	int	1	-99
promedio	float	1	19.0

Código 45: Mientras controlado por centinela

De los dos programas anteriores, puede observar la imagen del valor de las variables, especialmente la variable *nota*, para este ejercicio, el valor de *nota* quedó con un valor final de -99, que es el valor de la condición para finalizar la estructura repetitiva. La variable *i* mantiene los tres ingresos realizados (que es 3) y *acum* mantiene el valor acumulado (que es 57). Los ingresos para *nota* fueron 18, 19 y 20, y para el problema anterior se ingresó adicionalmente el valor de -99, pero no es considerado en *acum*.

Ejemplo 38. Sumatoria de los diez primeros números.

Escriba un programa que permita la sumatoria de los 10 primeros números. Utilice dos formas de realizar el problema mediante las estructuras repetitivas conocidas.

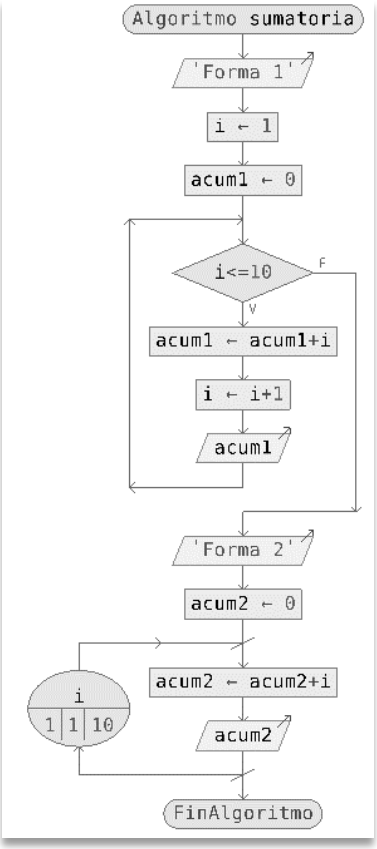
1 2 3 4 5 6 7 8 9 10
1 3 6 10 15 21 28 36 45 55

Figura 45. Ejemplo de la sumatoria de números

Solución: (Análisis)

Es el problema es un común trabajo para un acumulador, éste debe estar inicializado en 0, cuando es un acumulador de sumas o sumatoria se inicializa con 0, en el problema del factorial tratado anteriormente, el acumulador es de productos y se lo inicializaba con 1. Ahora presentaremos las acumulaciones a medida que recorren las repeticiones de número a número.

Algoritmo (diseño)



Algoritmo 27: Sumatoria de dos formas

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 17 13:43:44 2019
4"""
5print("Forma 1")
6i = 1
7acum1 = 0
8while(i<=10):
9    acum1 = acum1 + i
10    i = i + 1
11    print(acum1, end=" ")
12print(" ")
13print("Forma 2")
14acum2 = 0
15for i in range (1, 11, 1):
16    acum2 = acum2 + i
17    print(acum2, end=" ")
```

```
Forma 1
1 3 6 10 15 21 28 36 45 55
Forma 2
1 3 6 10 15 21 28 36 45 55
```

Código 46: Sucesión de Fibonacci

Note que la forma 1, se visualiza la variable de control *i,* inicializada en 1 (similar en forma 2), los acumuladores *acum1* y *acum2* se "enceran" o "inicializan en 0", y en las estructuras while como for, tienen sus valores límites de finalización, mientras que while la condición llega a menor e igual a 10 (considerando el 10), el lazo for dentro de la función range, se observa que debe llegar antes de 11, que es 10. Range se verá más adelante, pero podemos observar que los tres valores, representan el valor inicial, el valor final y el incremento. La línea 9 como la 16 son los incrementos en 1, como contadores. La línea 10 es el incremento de *i* para la forma 1. Finalmente el argumento end = " ", como se indicó en programas anteriores permite la visualización horizontal.

Ejemplo 39. La potencia de un número.

Escriba un programa que permita el ingreso de dos números enteros, uno representa la base y el otro el exponente de la operación matemática conocida como potenciación.

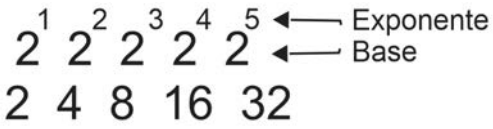
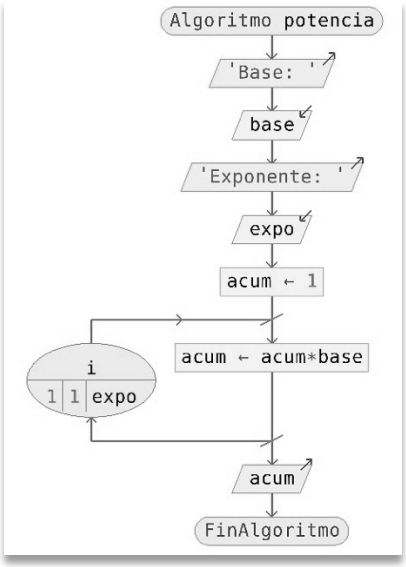


Figura 46. Ejemplo de potenciación

Solución: (Análisis)

En el problema dado, se ingresan dos números, el primero puede ser la base y el segundo puede ser el exponente. Con ello debe repetir la acumulación de exponente veces base.

Algoritmo (diseño)



Algoritmo 28: Potencia de un número

Código en Python (codificación)

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 17 15:16:45 2019
"""
print("Base: ", end=" ")
base = int(input())
print("Exponente: ", end=" ")
expo = int(input())
acum = 1
for i in range(1, (expo+1), 1):
    acum = acum * base
else:
    print(acum)
```

```
Base:
2
Exponente:
5
32
```

Código 47: Potencia de un número forma 1

Código en Python (codificación)

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 17 15:16:45 2019
"""
base = int(input("Base: "))
expo = int(input("Exponente: "))
acum = 1
for i in range(expo):
    acum = acum * base
else:
    print(acum)
```

```
Base: 2

Exponente: 5
32
```

Código 48: Potencia de un número forma 2

Código en Python (codificación)

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 17 15:16:45 2019
"""
import math as mat
base = int(input("Base: "))
expo = int(input("Exponente: "))
acum = mat.pow(base, expo)
print(acum)
```

```
Base: 2

Exponente: 5
32.0
```

Código 49: Potencia de un número forma 3

Utilizando todo lo que hemos aprendido, podemos notar que las tres formas de resolver este problema (pueden haber más) enfatizan el ingreso de los valores y asignación a las variables, además se entiende claramente el uso del acumulador de productos inicializado en uno o, el valor del acumulador inicialmente (acum = 1). En la forma 2, está implícito las veces que itera o se repite el lazo for, de 0 a 4 (ya que no llega a 5 en expo, y empieza siempre en cero) que son cinco veces. En la forma 3, finalmente el uso de la librería math renombrando a mat, como el objeto a usarse, utilizamos la función existente pow, que tiene de argumentos dos valores, la base y el exponente de la potencia de un número.

Ahora trataremos un problema de Cálculo en ingeniería, recordemos las fórmulas de Euler [27] que utilizan las funciones exponenciales y trigonométricas: $e^{iz} = \cos z + i \sin z$, para encontrar los valores de las potencias complejas:

$$e^z = e^{x+iy} = e^x(\cos y + i \sin y)$$

Figura 47. Fórmula de Euler

Siendo z la variable compleja, podemos demostrar estas expresiones, utilizando las series de Taylor [28][29]:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots + \infty$$

$$\sin x = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \cdots + \infty$$

$$\cos x = \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \cdots + \infty$$

Figura 48. Series de Taylor

De esto combinarlas para hallar el valor correspondiente.

A continuación encontraremos las expresiones de e^x, $\sin x$ y $\cos x$ para un enésimo término y siendo el valor de x igual a 1.

Ejemplo 40. Hallar e^x usando la serie de Taylor

Escriba un programa que permita obtener el resultado de la serie de Taylor, e^x.

Solución: (Análisis)

En el problema, se ingresan dos números, el primero puede ser el valor de x, y el valor n, que representará el enésimo término de la serie de Taylor a evaluar. Para estos problemas, usaremos el valor de x en 1, para comprender la lógica programática involucrada.

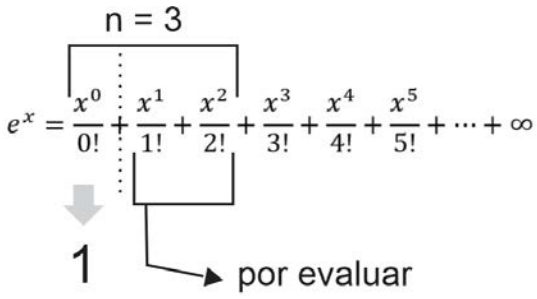


Figura 49. Tratamiento de solución a la serie de Taylor, e^x

Se debe evaluar el segundo y tercer término de la serie, debido a que el primer término su valor es 1, es decir podemos realizar la evaluación a partir del segundo término y al final sumaremos 1 al resultado obtenido.

Aplicaremos los códigos aprendidos de la potencia de un número y del factorial, utilizaremos un código simple de entender para que podamos explícitamente utilizar los valores como se muestran en la figura.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 17 16:56:20 2019
4"""
5x = int(input("Ingrese x: "))# x = 1
6n = int(input("Ingrese n: "))# n = 3
7acum = 0
8for i in range(1, (n), 1):
9    # potencia
10    pot = 1
11    for j in range(1, (i+1), 1):
12        pot = pot * x
13    #factorial
14    fac = 1
15    for j in range(1, (i+1), 1):
16        fac = fac * j
17    acum = acum + (pot/fac)
18print("e^x = ", (acum+1))
```

```
Ingrese x: 1

Ingrese n: 20
e^x =  2.7182818284590455
```

Explicación Código en Python

Línea 5: Ingreso de x. #usaremos 1
Línea 6: Ingreso de n. #usaremos 3, luego 20
Línea 7: Acumulador de sumas (términos de la serie).
Línea 8: Sumatoria de hasta n, 1 y 2, sin incluir el 3, recordar que el lazo for no llega a n.
Línea 10: Acumulador de productos. Por cada iteración i, se inicializar nuevamente a 1.
Línea 11 y 15: Por cada iteración i, se debe realizar la potencia en ese i y el factorial de i. Los lazos van de 1 a i (inclusive i+1)
Línea 17: Finalmente en cada iteración i, se realiza la debida división.
Línea 18: Se realiza el aumento de 1 como se dejó explicado, es el primer término elevado a la cero y factorial de cero, que es 1.

Código 50: Serie de Taylor e^x

Como observamos el código, si utilizamos una calculadora científica y utilizamos la función e^x, tendremos el mismo valor. En el programa utilizamos n con valor de 20.

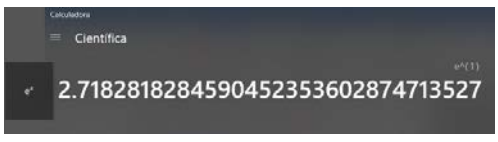


Figura 50. Uso de la calculadora para hallar e^x

Ejemplo 41. Hallar cos (x) usando la serie de Taylor

Escriba un programa que permita obtener el resultado de la serie de Taylor, cos(x).

Solución: (Análisis)

El problema es similar al anterior, pero ahora debemos considerar la alternancia de los signos y los números de las potencias y factorial son pares, se puede observar que cuando nos encontramos en la iteración i, de valor impar, los términos deben restar y cuando estamos en la iteración i, de valor par, los términos deben sumarse, al acumulador de sumas (términos).

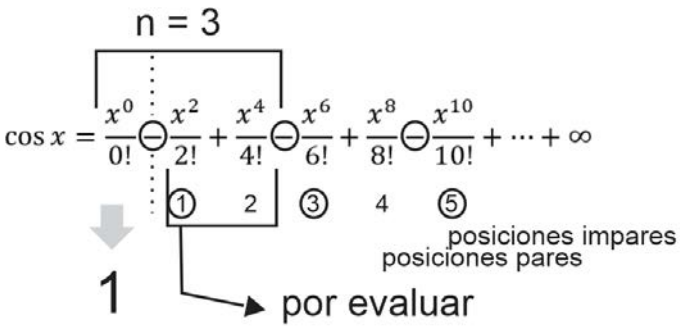


Figura 51. Tratamiento de solución a la serie de Taylor, cos(x)

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 17 16:56:20 2019
4"""
5x = int(input("Ingrese x: "))# x = 1
6n = int(input("Ingrese n: "))# n = 3
7acum = 0
8for i in range(1, (n), 1):
9    ii = i*2
10    # potencia
11    pot = 1
12    for j in range(1, (ii+1), 1):
13        pot = pot * x
14    #factorial
15    fac = 1
16    for j in range(1, (ii+1), 1):
17        fac = fac * j
18    if(i % 2 == 1):
19        acum = acum - (pot/fac)
20    else:
21        acum = acum + (pot/fac)
22print("e^x = ", (acum+1))
```

```
Ingrese x: 1

Ingrese n: 3
e^x =  0.5416666666666667
```

Explicación Código en Python

Línea 5: Ingreso de x. #usaremos 1
Línea 6: Ingreso de n. #usaremos 3
Línea 7: Acumulador de sumas (términos de la serie).
Línea 8: Sumatoria de hasta n, 1 y 2, sin incluir el 3, recordar que el lazo for no llega a n.
Línea 9: Para simular los números pares de la ecuación en cada iteración reemplazamos i = ii * 2.
Línea 11: Acumulador de productos. Por cada iteración i, se inicializar nuevamente a 1.
Línea 12 y 16: Por cada iteración i, se debe realizar la potencia en ese i y el factorial de i. Los lazos van de 1 a ii (inclusive ii+1)
Línea 18: Si la iteración i es impar se resta el término.
Línea 19-21: Finalmente en cada iteración impar, se realiza la debida división con signo negativo, caso contrario se suma.
Línea 22: Se realiza el aumento de 1 como se dejó explicado, es el primer término elevado a la cero y factorial de cero, que es 1.

Código 51: Serie de Taylor cos(x)

Como observamos el código, si utilizamos una calculadora científica y utilizamos y evaluamos término a término, tendremos el mismo valor. En el programa utilizamos n con valor de 3.

Figura 52. Uso de la calculadora para hallar cos(x)

Ejemplo 42. Hallar sen (x) usando la serie de Taylor

Escriba un programa que permita obtener el resultado de la serie de Taylor, sen(x).

Solución: (Análisis)

El problema es similar al anterior, pero ahora debemos considerar la alternancia de los signos y los números de las potencias y factorial son impares, se puede observar que cuando nos encontramos en la iteración i, de valor impar, los términos deben restar y cuando estamos en la iteración i, de valor par, los términos deben sumarse, al acumulador de sumas (términos).

Además, en el programa anterior el valor de i debió multiplicarse por dos para simular el inicio de los números pares, ahora son impares y empiezan en tres. Usaremos la siguiente expresión matemática, ii = i * 2 + 1.

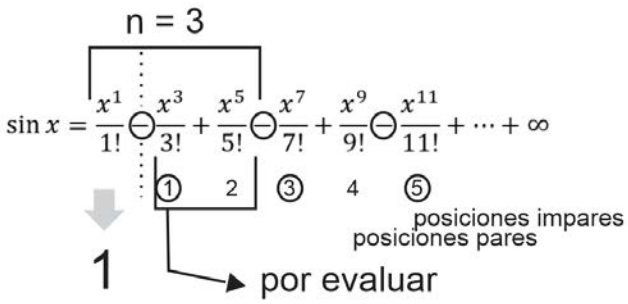


Figura 53. Tratamiento de solución a la serie de Taylor, sen(x)

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 17 16:56:20 2019
4"""
5x = int(input("Ingrese x: "))# x = 1
6n = int(input("Ingrese n: "))# n = 3
7acum = 0
8for i in range(1, (n), 1):
9    ii = i*2+1
10    # potencia
11    pot = 1
12    for j in range(1, (ii+1), 1):
13        pot = pot * x
14    #factorial
15    fac = 1
16    for j in range(1, (ii+1), 1):
17        fac = fac * j
18    if(i % 2 == 1):
19        acum = acum - (pot/fac)
20    else:
21        acum = acum + (pot/fac)
22print("e^x = ", (acum+1))
```

```
Ingrese x: 1

Ingrese n: 3
e^x =  0.8416666666666667
```

Explicación Código en Python

Línea 5: Ingreso de x. #usaremos 1
Línea 6: Ingreso de n. #usaremos 3
Línea 7: Acumulador de sumas (términos de la serie).
Línea 8: Sumatoria de hasta n, 1 y 2, sin incluir el 3, recordar que el lazo for no llega a n.
Línea 9: Para simular los números pares de la ecuación en cada iteración reemplazamos i = ii * 2+1.
Línea 11: Acumulador de productos. Por cada iteración i, se inicializar nuevamente a 1.
Línea 12 y 16: Por cada iteración i, se debe realizar la potencia en ese i y el factorial de i. Los lazos van de 1 a ii (inclusive ii+1)
Línea 18: Si la iteración i es impar se resta el término.
Línea 19-21: Finalmente en cada iteración impar, se realiza la debida división con signo negativo, caso contrario se suma.
Línea 22: Se realiza el aumento de 1 como se dejó explicado, es el primer término elevado a la cero y factorial de cero, que es 1.

Código 52: Serie de Taylor sen(x)

Como observamos el código, si utilizamos una calculadora científica y utilizamos y evaluamos término a término, tendremos el mismo valor. En el programa utilizamos n con valor de 3.

0.84166666666666633333333333333333

Figura 54. Uso de la calculadora para hallar sen(x)

La sucesión de Ulam o la conjetura de números de esta serie [30][31] se definen mediante las siguientes condiciones:

$$Número\ Ulam = \begin{cases} Si\ n\ es\ par, n = n/2 \\ Si\ n\ es\ impar, n = n * 3 + 1 \end{cases}$$

Ejemplo 43. Serie de Ulam

Escriba un programa que permita mostrar la serie de Ulam a partir de número entero ingresado por teclado.

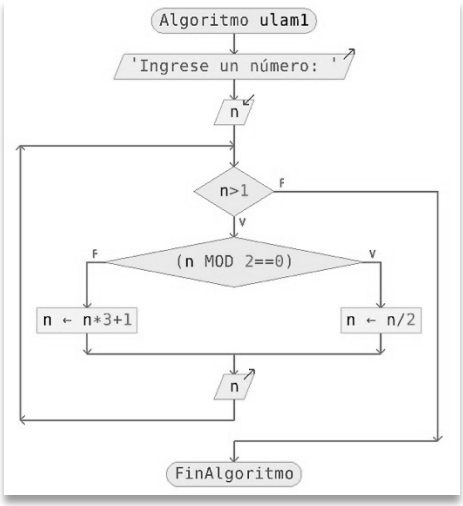
10 5 16 8 4 2 1

Figura 55. Serie de Ulam del número 10

Solución: (Análisis)

En esta ocasión, el problema indica que para la nueva generación de un número de la serie de Ulam debe aplicar un proceso diferente, las condiciones son de par e impar, luego se aplicar el proceso descrito. Hay que observar el valor de n, para las siguientes iteraciones/repeticiones en el bucle, se perderá hasta llegar al valor base o final que será 1 para esta serie.

Algoritmo (diseño)



Algoritmo 29: Serie de Ulam

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 12:17:48 2019
4"""
5n = int(input("Ingrese un número: "))
6while(n>1):
7    if(n % 2 == 0):
8        n = n // 2
9    else:
10       n = n * 3 + 1
11   print(n, end=" ")
```

```
Ingrese un número: 10
5 16 8 4 2 1
```

Código 53: Serie de Ulam

Observe que en la línea 8, se utiliza el operador de división entera, diferente a la de división, la diferencia una de otra es que una conlleva a generar números decimales al dividir enteros, la otra divide a los números enteros sin considerar decimales.

Supongamos que nos pidan generar todos los números Ulam del 15 al 20 e indicar quién genera la mayor lista de números.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Sep 18 12:27:45 2019
4 """
5 mayor = -999
6 for i in range(15, 21, 1):
7     n = i
8     cont = 0
9     print(n, end=": ")
10    print(n, end=" ")
11    while(n>1):
12        if(n % 2 == 0):
13            n = n // 2
14        else:
15            n = n * 3 + 1
16        cont = cont + 1
17        print(n, end=" ")
18    if(cont >= mayor):
19        mayor = cont
20        num = i
21    print()
22 print(num," genera la lista más larga con", mayor, "números")
```

```
15: 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
16: 16 8 4 2 1
17: 17 52 26 13 40 20 10 5 16 8 4 2 1
18: 18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
19: 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
20: 20 10 5 16 8 4 2 1
19  genera la lista más larga con 20 números
```

Código 54: Listados de números Ulam

Explicación Código en Python

Línea 5: Se utiliza una variable llamada *mayor* con valor de -999, una estrategia de colocar un valor muy pero muy bajo para que asuma el nuevo mayor el siguiente número positivo.
Línea 6: Se establecen los límites, inicial 15, final 20 (se muestra 21 ya que el lazo *for* en *range* no llega al 21, sino antes de 21), en pasos de 1.
Línea 7: Debido a que hemos aprendido que el valor de n se perderá y no deseamos que i se modifique, asignamos a una variable reutilizable n para asuma el valor de i en cada nueva iteración.
Línea 8: Utilizamos una variable contador llamada cont.
Línea 9-10: Utilizamos *end*= " " para una mejor presentación horizontal.
Línea 11-15: Reutilizamos el código creado en el ejercicio anterior de la serie de Ulam.
Línea 16: Por cada nueva generación de un número Ulam éste es contado a la iteración i.
Línea 18: Verificamos que el contador de la iteración i es mayor que valor mayor guardado.
Línea 19-20: Si es verdadero, guardamos el nuevo mayor generado por contador y guardamos la iteración i quien generó la lista más extensa.

Una forma de multiplicar dos números es conocida como el método de multiplicación rusa [32].

El procedimiento es representado en la siguiente imagen, supongamos multiplicar 11 por 12:

n1	n2	n1 es impar	Sumar n2
11	12	sí	12
5	24	sí	24
2	48	no	
1	96	sí	96
11 x 12 = 132			132

Figura 56. Esquema del método ruso de multiplicación

Ejemplo 44. Método de multiplicación rusa

Escriba un programa que permita multiplicar dos números mediante el método ruso de la multiplicación.

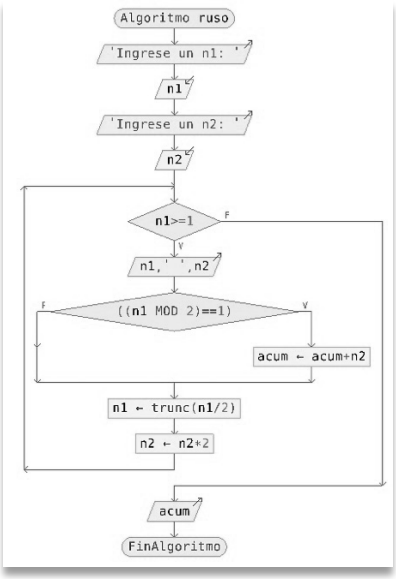
Solución: (Análisis)

Si ingresamos los números 11 como primer número y 12 como segundo número, debemos entender que el método realiza divisiones por dos sobre el primero y multiplicaciones por dos sobre el segundo, luego en el transcurso de cada proceso verifica si el primer número y sus respectivos cocientes, son impares (desde el primer número ingresado), si es verdadera esta condición, acumula en un sumador los respectivos segundos números o productos generados (del primer segundo número o segundo número ingresado)

Mientras que al primer número se lo divide (división entera) para dos, al segundo número se lo multiplica por dos, hasta que el primer número tienda a 1, en cada proceso sí el primer número (cocientes) es impar se sumará su correspondiente segundo número (productos).

Finalmente, la acumulación de los productos debe sumar al mismo valor que si hubiéramos multiplicados los dos primeros números. Para comprobar el resultado, invierta los números ingresados, es decir, primero el 12 y luego el 11, como se puede observar el 12 no es impar, se verificará en los siguientes cocientes.

Algoritmo (diseño)



Algoritmo 30: Método ruso de multiplicación

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 18 14:18:18 2019
"""
acum = 0
n1 = int(input("Ingrese primer número: "))
nn1 = n1
n2 = int(input("Ingrese segundo número: "))
nn2 = n2
while(n1 >= 1):
    print(n1," ",n2)
    if( n1 % 2 == 1):
        acum = acum + n2
    n1 = n1 // 2
    n2 = n2 * 2
print("\n",nn1," por ",nn2,
      " es ", (nn1*nn2)," = ", acum)
```

```
Ingrese primer número: 11

Ingrese segundo número: 12
11   12
5   24
2   48
1   96

 11  por  12  es  132  =  132
```

Código 55: Método ruso de multiplicación

Puede observar que la línea 11, donde se presentan los números, puede estar ubicada al final de la estructura *while*, intente realizar cambios de ubicación de la presentación de números para que comprenda lo importante de saber presentar los números o resultados en el momento o lugar indicado del código.

Asimismo, el uso del operador de doble división, que simboliza la división entera, es utilizado en este ejercicio. Finalmente la línea 16, el uso del *print*( ), permite un corte en la línea de código, dando continuidad a la línea anterior.

MÓDULO 6

Módulo 6. Arreglos y matrices.

Arreglos o vectores

Repasando lo anteriormente visto, podemos notar el uso de las variables, con sus distintos nombres y en su defecto, muchas de ellas.

Pensando en los arreglos o vectores, son elementos del mismo tipo de dato, uno a continuación de otro, nombrados con un mismo identificador pero ubicados con un índice, Adicionalmente, la estructura que nos permitirá realizar el recorrido

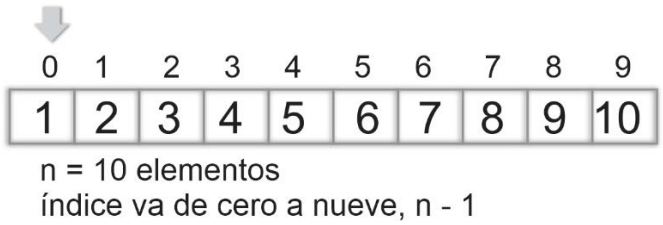


Figura 57. Características del arreglo o vector

Ejemplo 45. Llenado de un arreglo

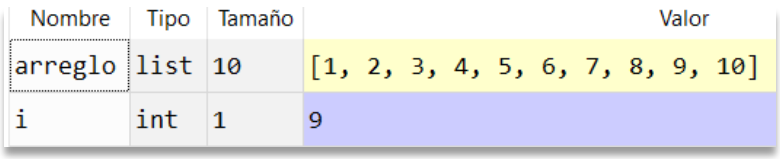
Escriba un programa que permita almacenar los diez primeros números en un arreglo.

Figura 58. Arreglo de números

Solución: (Análisis)

En python los vectores son llamados listas, una colección ordenada de cualquier tipo de dato. También se pueden identificar la tuplas, como un conjunto de valores agrupados secuencialmente. Los diccionarios, datos de un tipo que, asocian objetos en pares. Finalmente, los conjuntos, similar a las matemáticas, valores no repetidos en colecciones de forma desordenada.

Adicionalmente, podemos encontrar bibliotecas que se incorporan a Python para potenciar el uso del procesamiento de los vectores utilizando por ejemplo Numpy o Pandas que superan a Python List que por defecto viene en la instalación de Python. Realizaremos varios ejemplos de estos tipos de agrupaciones de almacenamiento de valores y de sus bibliotecas que no permiten desarrollar con mejores programas para su mayor comprensión.



Nombre	Tipo	Tamaño	Valor
arreglo	list	10	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
i	int	1	9

Figura 59. Un arreglo en Python es una lista de valores

El recorrido de la lista o el arreglo, toma en consideración la posición cero, con ello debemos recordar que la posición 10 de una lista de 10 elementos no existe, su primera referencia es cero y la última será 9. En los siguientes programas se muestran claramente lo indicado.

Usando listas

Se define una lista de diez elementos.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 14:27:39 2019
4"""
5arreglo = [1,2,3,4,5,6,7,8,9,10]
6print("Toda el arreglo: ")
7print(arreglo)
8print("Elementos del arreglo: ")
9for i in arreglo:
10    print(i, end=" ")
11print()
12print("Usando range..")
13print("Índices del arreglo: ")
14for i in range(len(arreglo)):
15    print(i, end=" ")
16print()
17print("contenido en arreglo[3]:",arreglo[3])
18print("contenido en arreglo[-3]:",arreglo[-3])
19print("Longitud del arreglo: ", end=" ")
20print(len(arreglo))
21print("Usando función list: ")
22print("list(range(10))")
23print(list(range(10)))
24print("list('hola')")
25print(list('hola'))
```

Explicación Código en Python

Línea 5: Lista de 10 elementos.
Línea 7: Se presenta la totalidad del arreglo.
Línea 9: Se presenta el arreglo elemento por elemento, de forma horizontal con la ayuda de *end*=“ ”.
Línea 14: Usando la función *range* el lazo *for* recorre desde el índice hasta la longitud total del arreglo (9 ya que no llega a 10, que es lo que guarda la función longitud).
Línea 17: Contenido de arreglo[3].
Línea 18: Contenido de arreglo[-3], presenta el valor desde la posición final a la inicial, recorrido inverso.
Línea 19-20: Muestra la longitud del arreglo.
Línea 23: Usando la función *list* y *range* se presenta un arreglo de 10 elementos desde 0 a 9.
Línea 22: Usando la función *list* sobre una palabra/*string* se presenta elemento a elemento (caracter por caracter).

```
Toda el arreglo:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Elementos del arreglo:
1 2 3 4 5 6 7 8 9 10
Usando range..
Índices del arreglo:
0 1 2 3 4 5 6 7 8 9
contenido en arreglo[3]: 4
contenido en arreglo[-3]: 8
Longitud del arreglo:  10
Usando función list:
list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list('hola')
['h', 'o', 'l', 'a']
```

Código 56: Almacenando valores en una lista o arreglo

Usando tuplas

Se definen dos tuplas, una de diez elementos y otra de tres elementos.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 16:56:15 2019
4"""
5tupla1 = (1,2,3,4,5,6,7,8,9,10)
6tupla2 = "192.168.1.1","IP","red"
7print("Toda la tupla 1: ")
8print(tupla1)
9print("Elementos de la tupla 2: ")
10for i in tupla2:
11    print(i, end=" ")
12print()
13print("Usando range..")
14print("Índices de la tupla 1: ")
15for i in range(len(tupla1)):
16    print(i, end=" ")
17print()
18print("contenido en tupla1[3]:",tupla1[3])
19print("contenido en tupla2[-3]:",tupla2[-3])
20print("Longitud de la tupla 2: ", end=" ")
21print(len(tupla2))
22algo1, algo2, algo3 = tupla2
23print(algo1," ",algo2," ",algo3)
```

Explicación Código en Python

Línea 5: tupla de 10 elementos.
Línea 6: tupla de 3 elementos.
Línea 8: Se presenta toda la tupla 1
Línea 10: Se presenta la tupla 2, elemento por elemento, de forma horizontal con la ayuda de *end*=“ ”.
Línea 14: Usando la función *range* el lazo *for* recorre desde el índice hasta la longitud total del arreglo (9 ya que no llega a 10, que es lo que guarda la función longitud).
Línea 22: Se asigna a tres variables, tipo cadena, el contenido de la tupla 2, desempaquetando la tupla.

```
Toda la tupla 1:
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Elementos de la tupla 2:
192.168.1.1 IP red
Usando range..
Índices de la tupla 1:
0 1 2 3 4 5 6 7 8 9
contenido en tupla1[3]: 4
contenido en tupla2[-3]: 192.168.1.1
Longitud de la tupla 2:  3
192.168.1.1   IP   red
```

Código 57: Almacenando valores en una tupla

Nombre	Tipo	Tamaño	Valor
algo1	str	1	192.168.1.1
algo2	str	1	IP
algo3	str	1	red
i	int	1	9
tupla1	tuple	10	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
tupla2	tuple	3	('192.168.1.1', 'IP', 'red')

Figura 60. Una Tupla de valores

El trabajo con tuplas, acorta el procedimiento de listar o recorrer el contenido almacenado y directamente pueden manipularse, compararse o asignarse de forma conjunta o desagregada en la misma cantidad de variables de la tupla.

Usando diccionarios

Se define un diccionario de cinco parejas de elementos.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Sep 18 17:40:35 2019
4 """
5 diccionario1 = {
6         "disco":"¿Cuál es tu Python?",
7         "cantante":"Anaconda",
8         "album":"Conda y Mini Conda",
9         "canciones":"10",
10        "productora":"Zoni"
11        }
12 indice = diccionario1.keys()
13 cantidad = diccionario1.items()
14 elementos = diccionario1.values()
15
16 for indice , elementos in cantidad:
17     print(indice + " : " + elementos)
```

Explicación Código en Python

Línea 5: diccionario de 5 parejas de elementos.
Línea 12: Característica de índices.
Línea 13: Característica de números de elementos.
Línea 14: Característica de elementos.
Se han usado .keys( ), .items( ) y .values( ) , funciones propias de los diccionarios, entre otras importantes.
Línea 16: Usando las llaves con sus elementos iterando sobre el número de elementos o cantidad.

```
disco : ¿Cuál es tu Python?
cantante : Anaconda
album : Conda y Mini Conda
canciones : 10
productora : Zoni
```

Código 58: Almacenando valores en un diccionario

Al trabajar con diccionarios, podemos comprender que es una estructura que registra un elemento asignándole un llave o índice determinado. Para este ejemplo, el diccionario tiene las propiedades de una estructura de datos.

Nombre	Tipo	Tamaño	Valor
diccionario1	dict	5	{'disco':'¿Cuál es tu Python?', 'cantante':'Anaconda', 'album':'Conda ...
elementos	str	1	Zoni
indice	str	1	productora

Figura 61. Un diccionario de valores

Usando conjuntos

Se define un conjunto de elementos, correspondientes a los días de la semana.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 18:32:43 2019
4"""
5conjunto1 = {
6        "Lunes","Martes",
7        "Miércoles","Jueves",
8        "Viernes","Sábado",
9        "Domingo",
10       }
11conjunto2 = {"Martes", "Jueves"}
12print(conjunto1)
13print(conjunto2)
14for i in conjunto1:
15    print(i, end=", ")
16print()
17print("Intersección: ",conjunto1 & conjunto2)
```

Explicación Código en Python

Línea 5: conjunto de 7 elementos.
Línea 11: conjunto de 2 elementos.
Línea 12: Presentación del conjunto1.
Línea 13: Presentación del conjunto2.
Línea 14: Recorrido iterable del conjunto.
Línea 15: Intersección de elementos del conjunto.

```
{'Jueves', 'Viernes', 'Miércoles', 'Lunes', 'Martes', 'Sábado', 'Domingo'}
{'Martes', 'Jueves'}
Jueves, Viernes, Miércoles, Lunes, Martes, Sábado, Domingo,
Intersección:  {'Martes', 'Jueves'}
```

Código 59: Almacenando valores en un conjunto o set

A diferencia de los diccionarios (que tienen llaves inmutables), las tuplas (que son inmutables)., un conjunto o set de elementos pueden ser modificados, es decir son mutables en el entorno.

Usando Numpy

Se define una lista de elementos enteros y, mediante la biblioteca de Numpy, con el objeto np, se utilizan los métodos o funciones disponibles para trabajar sobre la lista.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 18:56:50 2019
4"""
5import numpy as np
6lista = [4,6,5,10,1,9,8,3,7,2]
7maximo = np.amax(lista)
8minimo = np.amin(lista)
9lista1 = np.sort(lista)
10print("Lista: ",lista)
11print("Valor máx.",maximo)
12print("Valor min.",minimo)
13print("Ordena: ",lista1)
14print("Sumatoria: ",np.sum(lista))
```

Explicación Código en Python

Línea 5: se importa la biblioteca y se crea el objeto np.
Línea 6: se crea la lista de elementos.
Línea 7: se usa amax, función que halla el máximo valor de la lista.
Línea 8: se usa amin, función que halla el mínimo valor.
Línea 9: se ordena la lista con la función sort( ).

```
Lista:  [4, 6, 5, 10, 1, 9, 8, 3, 7, 2]
Valor máx. 10
Valor min. 1
Ordena:  [ 1  2  3  4  5  6  7  8  9 10]
Sumatoria:  55
```

Código 60: Uso de la librería numpy

Ahora, comenzamos a notar la importancia de los métodos y funciones que nos presentan las bibliotecas que podemos importar a nuestro proyecto o archivo. La forma de instalar estas librerías será según lo indica la página de Python (pip install pandas).

Usando Pandas

Se define una lista de elementos enteros y, mediante la biblioteca de Numpy, con el objeto *np*, se utilizan los métodos o funciones disponibles para trabajar sobre la lista.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 19:33:53 2019
4"""
5import pandas as pd
6lista = ["Do","Re","Mi","Fa","Sol","La","Si"]
7index = [1,2,3,4,5,6,7]
8serie = pd.Series(lista,index)
9print("Presenta: \n%s" % serie)
```

```
Presenta:
1     Do
2     Re
3     Mi
4     Fa
5    Sol
6     La
7     Si
dtype: object
```

Explicación Código en Python

Línea 5: se importa la biblioteca pandas.
Línea 6: se crea la lista de elementos.
Línea 7: se crea lista de índices.
Línea 8: se usa Series de pandas, función requiere de dos listas, una de elementos y otra de índices.
Línea 9: se presenta la información de las series de forma ordenada.

Más adelante utilizaremos estas librerías para la importación de archivos y las debidas graficas.

Código 61: Uso de la librería pandas

Las librerías que Python puede soportar complementan el uso eficiente de este lenguaje, con ello, las bondades de estos métodos que contienen las librerías, permiten un mejor tratamiento de la información que requiere en muchos casos de formas óptimas de obtener resultados.

Nombre	Tipo	Tamaño	Valor
index	list	7	[1, 2, 3, 4, 5, 6, 7]
lista	list	7	['Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si']
serie	Series	(7,)	Series object of pandas.core.series module

Figura 62. Tipos de datos en Pandas library

Ejemplo 46. Serie de Ulam con arreglos

Escriba un programa que permita almacenar la serie de Ulam un arreglo.

Figura 63. Serie de Ulam en un arreglo

Solución: (Análisis)

En python los vectores son llamados listas, como se ha trabajado hasta este momento, ahora debemos ir almacenando en una lista la serie de números de Ulam que se generan. Usaremos métodos o funciones que permitan almacenar, recorrer e insertar elementos de forma sencilla.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Sep 18 20:19:26 2019
4 """
5 lista = [0]
6 i = 0
7 lista[i] = int(input("Ingrese un número: "))
8 while(lista[i]>1):
9     if(lista[i] % 2 == 0):
10        lista.append(lista[i] // 2)
11    else:
12        lista.append(lista[i] * 3 + 1)
13    i = i + 1
14 print(lista, end=" ")
```

```
Ingrese un número: 10
[10, 5, 16, 8, 4, 2, 1]
```

Código 62: Serie de Ulam con arreglos

Explicación Código en Python

Línea 5: se crea una lista con un elemento.
Línea 6: *i* es la posición inicial de almacenamiento.
Línea 7: se almacena en el primer elemento.
Línea 8: se usa el código aplicado en el problema de Ulam sin arreglos, mientras el elemento en la posición *i* del arreglo sea mayor a 1, se aplican las operaciones dadas.
Línea 9: si es par, se adiciona a la lista, el elemento de la lista i dividido para dos.
Línea 12: si es impar, se adiciona a la lista, el elemento de la lista *i* multiplicado por tres y sumado uno.

Ejemplo 47. Método de multiplicación rusa usando arreglos

Escriba un programa que permita multiplicar dos números por el método ruso, utilice arreglos para almacenar la secuencia de números que se van generando.

n1	n2	n1 es impar	Sumar n2
11	12	sí	12
5	24	sí	24
2	48	no	
1	96	sí	96
11 x 12 = 132			132

Figura 64. Método de multiplicación rusa

Solución: (Análisis)

Observamos el problema presentado, requiere de dos arreglos, que llamaremos n1 y n2, aplicaremos el mismo procedimiento visto para la multiplicación rusa sin arreglos, manipulando los índices que se van creando a medida que requeriremos espacio de almacenamiento. Se usarán las variables de almacenamiento como se muestra en la figura, finalmente se compararán los valores almacenados en las primeras posiciones con el acumulador de productos obtenido.

Cabe recordar que para la lista n1 se dividen sus elementos para dos, la lista n2 sus elementos se multiplican por dos. Revise el código anteriormente elaborado para comprender el cambio al uso de arreglos en este mismo problema.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 18 21:15:01 2019
"""
n1 = [0]
n2 = [0]
i = 0
acum = 0
n1[i] = int(input("Ingrese primer número: "))
n2[i] = int(input("Ingrese segundo número: "))
while(n1[i]>=1):
    print(n1[i]," ",n2[i])
    if(n1[i] % 2 == 1):
        acum = acum + n2[i]
    n1.append(n1[i] // 2)
    n2.append(n2[i] * 2)
    i = i + 1
print("\n",n1[0],"por",n2[0],
      "es",(n1[0]*n2[0]),"=",acum)
```

```
Ingrese primer número: 11

Ingrese segundo número: 12
11   12
5   24
2   48
1   96

 11 por 12 es 132 = 132
```

Código 63: Método ruso usando arreglos

Explicación Código en Python

Línea 5: se crea una lista n1 con un elemento.
Línea 6: se crea una lista n2 con un elemento.
Línea 7: *i* es la posición inicial de almacenamiento.
Línea 8: acum es el acumulador de productos según condición.
Línea 9: se almacena en el primer elemento de n1.
Línea 10: se almacena en el primer elemento de n2.
Línea 11: el n1 en la iteración i es mayor e igual a 1.
Línea 12: se muestran ambas listas en la iteración i.
Línea 13: si el contenido del cociente es impar.
Línea 14: se acumula el correspondiente producto.
Línea 15: se añade a la lista la ubicación n1[i] // 2.
Línea 16: se añade a la lista la ubicación n2[i] * 2.
Línea 17: se incrementa i en 1.
Línea 18: se presenta la información de comprobación.

Nombre	Tipo	Tamaño	Valor
acum	int	1	132
i	int	1	4
n1	list	5	[11, 5, 2, 1, 0]
n2	list	5	[12, 24, 48, 96, 192]

Matrices

Si para un arreglo, o lista, tenemos entendido que son espacios de almacenamiento con un mismo nombre, pero ubicándolos en un determinado índice. Las matrices podemos decir que son un conjunto de arreglos. En álgebra lineal, aprendemos que los arreglos se pueden visualizar como filas en una matriz, y que estas tienen dimensiones asociadas a las filas (arreglos) por columnas (arreglos verticales).

En esta sección utilizaremos matrices para elaborar problemas adecuados al nivel de trabajo que estamos siguiendo.

Ejemplo 48. Suma de matrices

Escriba un programa que permita sumar dos matrices cuadradas de tres por tres, los valores son llenados de forma aleatoria.

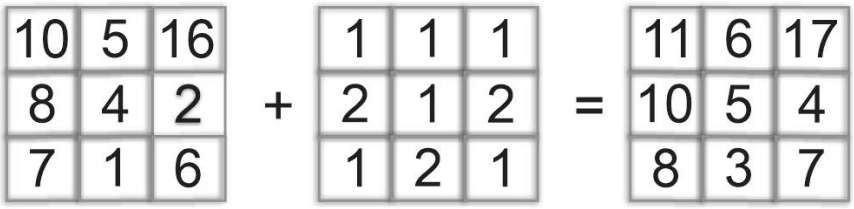


Figura 65. Suma de matrices con números aleatorios

Solución: (Análisis)

Utilizaremos las librerías aprendidas.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Wed Sep 18 21:36:28 2019
4"""
5import random
6import numpy as np
7# forma 1 con random
8matriz1 = [[0,0,0],[0,0,0],[0,0,0]]
9for i in range(len(matriz1)):
10    for j in range(len(matriz1[i])):
11        matriz1[i][j]=random.randrange(10)
12print()
13# forma 2 con numpy
14matriz2 = np.zeros((3,3))
15for i in range(len(matriz2)):
16    for j in range(len(matriz2[i])):
17        matriz2[i][j]=int(np.random.rand()*10)
18
19print("matriz1 \n",matriz1,"\n")
20print("matriz2 \n",matriz2,"\n")
21matriz3 = matriz1 + matriz2
22print("matriz3 (suma) \n",matriz3,"\n")
23
24matriz4 = [[0,0,0],[0,0,0],[0,0,0]]
25for i in range(len(matriz4)):
26    for j in range(len(matriz4[i])):
27        matriz4[i][j]= int(matriz1[i][j] + matriz2[i][j])
28print("matriz4 (suma) \n",matriz4,"\n")
```

Código 64: Suma de matrices

Explicación Código en Python

Línea 5: Importar librería random.
Línea 6: Importar librería numpy.
Línea 8: Matriz1 de 3x3.
Línea 9-11: Llenado de matriz con números aleatorios del 0 al 9 usando random.
Línea 14: Matriz2 de 3x3.
Línea 15-17: Llenado de matriz con números aleatorios del 0 al 9 usando numpy.
Línea 21: Suma de matrices.
Línea 25-27: Suma de matrices.

```
matriz1
 [[6, 3, 5], [8, 1, 6], [4, 3, 7]]

matriz2
 [[6. 7. 4.]
 [9. 0. 3.]
 [4. 6. 6.]]

matriz3 (suma)
 [[12. 10.  9.]
 [17.  1.  9.]
 [ 8.  9. 13.]]

matriz4 (suma)
 [[12, 10, 9], [17, 1, 9], [8, 9, 13]]
```

Podemos darnos cuenta a simple vista que al usar la librería numpy, el contenido de las matrices (línea 21) se suman de acuerdo a la ubicación de sus elementos, similar a (líneas 25-27).

Observe en cambio el código con las librerías estándares de Python.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Sep 19 10:15:32 2019
4 """
5 matriz1 = [[1,1,1],[1,1,1],[1,1,1]]
6 matriz2 = [[2,2,2],[2,2,2],[2,2,2]]
7 matriz3 = [[0,0,0],[0,0,0],[0,0,0]]
8 matriz3 = matriz1 + matriz2
9 print(matriz3)
```

Explicación Código en Python

Línea 5: Matriz cuadrada 1 llena de unos.
Línea 6: Matriz cuadrada 2 llena de dos.
Línea 7: Matriz cuadrada 3 encerada.
Línea 8: Concatenación de matrices que no es lo mismo del código anterior que sí sumaba elemento a elemento de cada matriz.
Línea 9: Contenido de Matriz 3.

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

Código 65: Unión o concatenación de matrices

Para realizar la multiplicación de matrices de matrices, debemos seguir un análisis desde lo visto en Álgebra Lineal, multiplicación escalar de las filas de la primera matriz por la columna de la segunda matriz, con ello encontraremos el elemento resultado posición a posición de la matriz final.

Ejemplo 49. Multiplicación de matrices

Escriba un programa que permita multiplicar dos matrices cuadradas de tres por tres, los valores son llenados de forma aleatoria.

1	2	3		1	2	3		6	12	18
1	2	3	x	1	2	3	=	6	12	18
1	2	3		1	2	3		6	12	18

Figura 66. Multiplicación de matrices con números aleatorios

Solución: (Análisis)

Utilizaremos las librerías aprendidas.

Código en Python (codificación) | **Explicación Código en Python**

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Sep 19 11:17:41 2019
4"""
5import numpy as np
6matriz1 = np.zeros((3,3))
7for i in range(len(matriz1)):
8    for j in range(len(matriz1[i])):
9        matriz1[i][j]=int(np.random.rand()*10)
10print("matriz1 \n",matriz1,"\n")
11matriz2 = np.zeros((3,3))
12for i in range(len(matriz2)):
13    for j in range(len(matriz2[i])):
14        matriz2[i][j]=int(np.random.rand()*10)
15print("matriz2 \n",matriz2,"\n")
16matriz3 = np.zeros((3,3))
17for i in range(len(matriz3)):
18    for j in range(len(matriz3[i])):
19        for k in range(len(matriz3[i])):
20            matriz3[i][j] += matriz1[i][k] * matriz2[k][j]
21print("matriz3 (multiplicación) \n",matriz3,"\n")
```

Línea 5-18: Similar al ejercicio anterior.
Línea 19: Resuelve el problema mediante las propiedades conocidas en álgebra, la sumatoria de multiplicaciones de cada fila por cada columna.

```
matriz1
 [[1. 7. 5.]
 [3. 0. 2.]
 [7. 7. 1.]]

matriz2
 [[6. 1. 1.]
 [7. 6. 6.]
 [2. 3. 2.]]

matriz3 (multiplicación)
 [[65. 58. 53.]
 [22.  9.  7.]
 [93. 52. 51.]]
```

Código 66: Multiplicación de matrices

Podemos utilizar el mismo ejemplo de la figura anterior y obtener el mismo resultado utilizando la fórmula de la línea 19.

Código en Python (codificación) | **Explicación Código en Python**

```
1# -*- coding: utf-8 -*-
2"""
3Created on Thu Sep 19 11:17:41 2019
4"""
5import numpy as np
6matriz1 = np.zeros((3,3))
7matriz1 = [[1,2,3],[1,2,3],[1,2,3]]
8print("matriz1 \n",matriz1,"\n")
9matriz2 = np.zeros((3,3))
10matriz2 = [[1,2,3],[1,2,3],[1,2,3]]
11print("matriz2 \n",matriz2,"\n")
12matriz3 = np.zeros((3,3))
13for i in range(len(matriz3)):
14    for j in range(len(matriz3[i])):
15        for k in range(len(matriz3[i])):
16            matriz3[i][j] += matriz1[i][k] * matriz2[k][j]
17print("matriz3 (multiplicación) \n",matriz3,"\n")
```

Línea 5-18: Similar al ejercicio anterior.
Línea 19: Resuelve el problema mediante las propiedades conocidas en álgebra, la sumatoria de multiplicaciones de cada fila por cada columna.

```
matriz1
 [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

matriz2
 [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

matriz3 (multiplicación)
 [[ 6. 12. 18.]
 [ 6. 12. 18.]
 [ 6. 12. 18.]]
```

Código 67: Multiplicación de matrices

Podemos entender que la multiplicación de matrices conlleva un procedimiento algebraico de solución. La multiplicación de los elementos de la fila de la primera matriz con los elementos de la columna de la segunda matriz, como se muestra en la figura.

$$\begin{pmatrix}1&2&3\\1&2&3\\1&2&3\end{pmatrix}\times\begin{pmatrix}1&2&3\\1&2&3\\1&2&3\end{pmatrix}=\begin{pmatrix}1\times1+2\times1+3\times1&1\times2+2\times2+3\times2&1\times3+2\times3+3\times3\\1\times1+2\times1+3\times1&1\times2+2\times2+3\times2&1\times3+2\times3+3\times3\\1\times1+2\times1+3\times1&1\times2+2\times2+3\times2&1\times3+2\times3+3\times3\end{pmatrix}$$
$$=\begin{pmatrix}6&12&18\\6&12&18\\6&12&18\end{pmatrix}$$

Figura 67. Multiplicación de matrices, disponible en https://matrixcalc.org/es/

Si dadas las matrices $A=\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\end{bmatrix}$ y $B=\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}b_{11}&b_{12}&c_{13}\\b_{21}&b_{22}&c_{23}\\b_{31}&b_{32}&c_{33}\end{bmatrix}$

entonces la ecuación resultante sería para la matriz C:

$$C = \begin{bmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{bmatrix} = \begin{bmatrix} a_{11}xb_{11} + a_{12}xb_{21} + a_{13}xb_{31} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

Es decir:

$c_{11} = a_{11}xb_{11} + a_{12}xb_{21} + a_{13}xb_{31}$, $c_{12} = a_{11}xb_{12} + a_{12}xb_{22} + a_{13}xb_{32}$, $c_{13} = a_{11}xb_{13} + a_{12}xb_{23} + a_{13}xb_{33}$
$c_{21} = a_{21}xb_{11} + a_{22}xb_{21} + a_{23}xb_{31}$, $c_{22} = a_{21}xb_{12} + a_{22}xb_{22} + a_{23}xb_{32}$, $c_{23} = a_{21}xb_{13} + a_{22}xb_{23} + a_{23}xb_{33}$
$c_{31} = a_{31}xb_{11} + a_{32}xb_{21} + a_{33}xb_{31}$, $c_{32} = a_{31}xb_{12} + a_{32}xb_{22} + a_{33}xb_{32}$, $c_{33} = a_{31}xb_{13} + a_{32}xb_{23} + a_{33}xb_{33}$

Obteniendo: $c_{ij} = a_{ik}xb_{kj} + a_{ik}xb_{kj} + a_{ik}xb_{kj}$, para k = 1, 2 y 3

$c_{ij} = a_{ik}xb_{kj} + a_{ik}xb_{kj} + a_{ik}xb_{kj}$, $c_{12} = a_{11}xb_{12} + a_{12}xb_{22} + a_{13}xb_{32}$, $c_{13} = a_{11}xb_{13} + a_{12}xb_{23} + a_{13}xb_{33}$
$c_{21} = a_{21}xb_{11} + a_{22}xb_{21} + a_{23}xb_{31}$, $c_{22} = a_{21}xb_{12} + a_{22}xb_{22} + a_{23}xb_{32}$, $c_{23} = a_{21}xb_{13} + a_{22}xb_{23} + a_{23}xb_{33}$
$c_{31} = a_{31}xb_{11} + a_{32}xb_{21} + a_{33}xb_{31}$, $c_{32} = a_{31}xb_{12} + a_{32}xb_{22} + a_{33}xb_{32}$, $c_{33} = a_{31}xb_{13} + a_{32}xb_{23} + a_{33}xb_{33}$

Es decir de forma resumida: $c_{ij} = c_{ij} + a_{ik}xb_{kj}$ para k=1, 2 y 3, de j = 1, 2 y 3, e i = 1, 2 y 3, tal como se encuentra en el código utilizado.

A continuación, trabajaremos con un conjunto de matrices y analizaremos las posiciones que nos indican llenar, como se muestra en la siguiente figura.

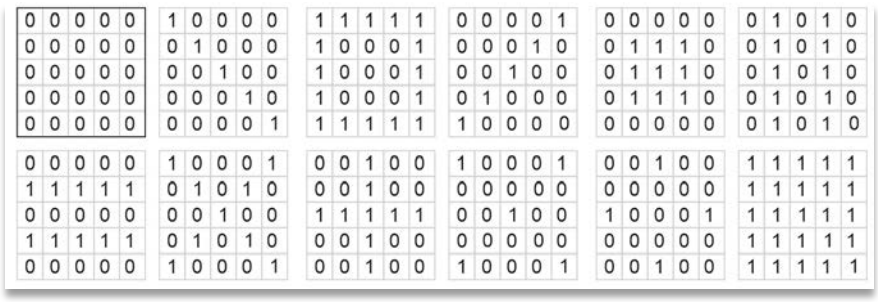


Figura 68. Varias matrices de 5x5

Ejemplo 50. Llenado de matriz de 5x5 versión 1

Escriba un programa que permita visualizar una matriz cuadrada de cinco por cinco, con los valores tal como se muestran en la figura a continuación.

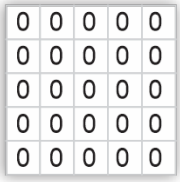


Figura 69. Llenado de matriz de 5x5 versión 1

Para estos ejercicios usaremos código con el uso de las bibliotecas, estándar de Python y Numpy.

Código en Python estándar

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz1 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz1:")
print(matriz1)

print("\nmatriz1:")
for i in range(len(matriz1)):
    for j in range(len(matriz1[i])):
        print(matriz1[i][j], end=" ")
    print()
```

```
matriz1:
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

```
matriz1:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Código 68: Matriz 1 estándar

Código en Python con Numpy

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz1 = np.zeros((5,5))
print("\nmatriz1:")
print(matriz1)

print("\nmatriz1:")
for i in range(len(matriz1)):
    for j in range(len(matriz1[i])):
        print(int(matriz1[i][j]), end=" ")
    print()
```

```
matriz1:
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]

matriz1:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Código 69: Matriz 1 con Numpy

Llenamos la diagonal principal de la matriz.

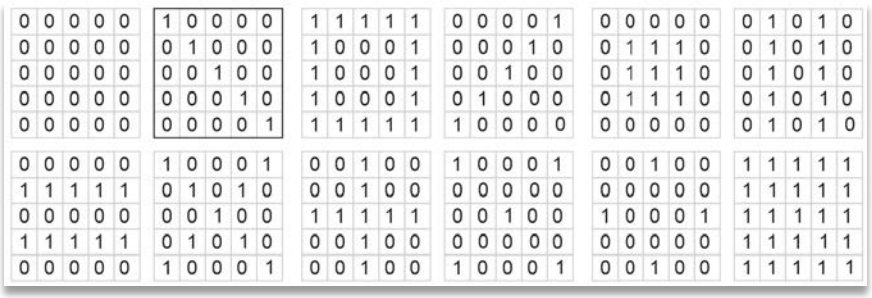


Figura 70. Varias matrices de 5x5

Ejemplo 51. Llenado de matriz de 5x5 versión 2

Escriba un programa que permita visualizar una matriz cuadrada de cinco por cinco, con los valores tal como se muestran en la figura a continuación.

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Figura 71. Llenado de matriz de 5x5 versión 2

Para estos ejercicios usaremos código con el uso de las bibliotecas, estándar de Python y Numpy.

Código en Python estándar

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz2 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz2:")
print(matriz2)

print("\nmatriz2:")
for i in range(len(matriz2)):
    for j in range(len(matriz2[i])):
        if(i == j):
            matriz2[i][j] = 1
        print(matriz2[i][j], end=" ")
    print()
```

```
matriz2:
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

Código 70: Matriz 2 estándar

Código en Python con Numpy

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz2 = np.zeros((5,5))
print("\nmatriz2:")
print(matriz2)

print("\nmatriz2:")
for i in range(len(matriz2)):
    for j in range(len(matriz2[i])):
        if(i == j):
            matriz2[i][j] = 1
        print(int(matriz2[i][j]), end=" ")
    print()
```

```
matriz2:
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

Código 71: Matriz 2 con Numpy

Llenamos la diagonal principal de la matriz.

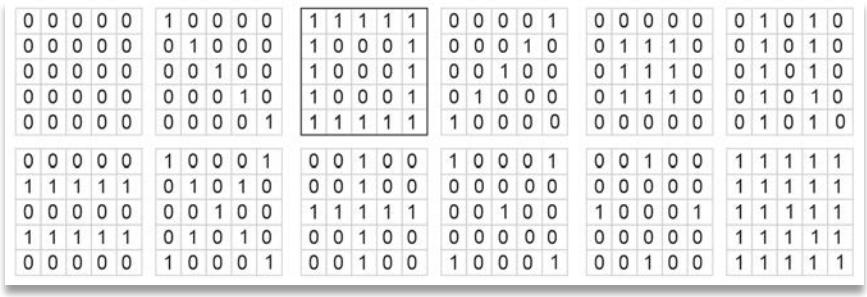


Figura 72. Varias matrices de 5x5

Ejemplo 52. Llenado de matriz de 5x5 versión 3

Escriba un programa que permita visualizar una matriz cuadrada de cinco por cinco, con los valores tal como se muestran en la figura a continuación.

1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1

Figura 73. Llenado de matriz de 5x5 versión 3

Para estos siguientes ejercicios usaremos código común.

Código en Python estándar | Numpy

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz3 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz3:")
print(matriz3)

print("\nmatriz3:")
for i in range(len(matriz3)):
    for j in range(len(matriz3[i])):
        if(i == 0):
            matriz3[i][j]=1
        if(i == 4):
            matriz3[i][j]=1
        if(j == 0):
            matriz3[i][j]=1
        if(j == len(matriz3)-1):
            matriz3[i][j]=1
        print(matriz3[i][j], end=" ")
    print()
```

```
matriz3:
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

Código 72: Matriz 3 estándar

Llenamos la diagonal secundaria de la matriz.

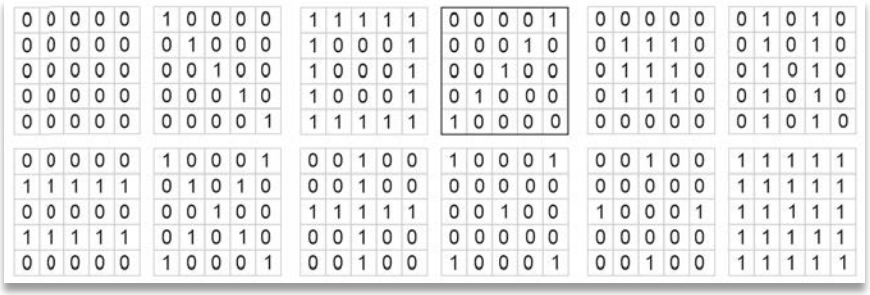


Figura 74. Varias matrices de 5x5

Ejemplo 53. Llenado de matriz de 5x5 versión 4

Escriba un programa que permita visualizar una matriz cuadrada de cinco por cinco, con los valores tal como se muestran en la figura a continuación.

0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0

Figura 75. Llenado de matriz de 5x5 versión 4

Para estos siguientes ejercicios usaremos código común.

Código en Python estándar | Numpy

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz4 = np.zeros((5,5))
print("\nmatriz4:")
print(matriz4)

print("\nmatriz4:")
for i in range(len(matriz4)):
    for j in range(len(matriz4[i])):
        if(j == 4-i):
            matriz4[i][j]=1;
        print(int(matriz4[i][j]), end=" ")
    print()
```

```
matriz4:
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
1 0 0 0 0
```

Código 73: Matriz 4 con Numpy

Presentamos a continuación las cuatro siguientes matrices y sus códigos.

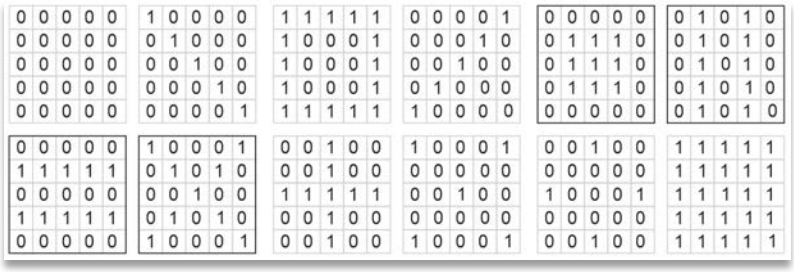


Figura 76. Varias matrices de 5x5

Código en Python estándar | Numpy

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz5 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz5:")
print(matriz5)

print("\nmatriz5:")
for i in range(len(matriz5)):
    for j in range(len(matriz5[i])):
        matriz5[i][j]=1
        if(i == 0):
            matriz5[i][j]=0
        if(i == 4):
            matriz5[i][j]=0
        if(j == 0):
            matriz5[i][j]=0
        if(j == len(matriz5)-1):
            matriz5[i][j]=0
        print(matriz5[i][j], end=" ")
    print()
```

```
matriz5:
0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0
```

Código 74: Matriz 5 estándar

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz6 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz6:")
print(matriz6)

print("\nmatriz6:")
for i in range(len(matriz6)):
    for j in range(len(matriz6[i])):
        if(j == 1):
            matriz6[i][j]=1
        if(j == 3):
            matriz6[i][j]=1
        print(matriz6[i][j], end=" ")
    print()
```

```
matriz6:
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
```

Código 75: Matriz 6 estándar

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz7 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz7:")
print(matriz7)

print("\nmatriz7:")
for i in range(len(matriz7)):
    for j in range(len(matriz7[i])):
        if(i == 1):
            matriz7[i][j]=1
        if(i == 3):
            matriz7[i][j]=1
        print(matriz7[i][j], end=" ")
    print()
```

```
matriz7:
0 0 0 0 0
1 1 1 1 1
0 0 0 0 0
1 1 1 1 1
0 0 0 0 0
```

Código 76: Matriz 7 estándar

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
matriz8 = [[0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0],[0,0,0,0,0],
           [0,0,0,0,0]]
print("\nmatriz8:")
print(matriz8)

print("\nmatriz8:")
for i in range(len(matriz8)):
    for j in range(len(matriz8[i])):
        if(i == j):
            matriz8[i][j]=1 # matriz 2
        if(j == 4-i):
            matriz8[i][j]=1 # matriz 4
        print(matriz8[i][j], end=" ")
    print()
```

```
matriz8:
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
```

Código 77: Matriz 8 estándar

Presentamos a continuación las cuatro últimas matrices y sus códigos.

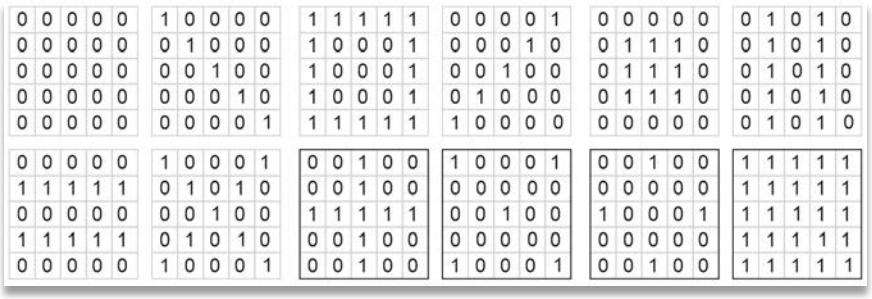


Figura 77. Varias matrices de 5x5

Código en Python estándar|Numpy

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz9 = np.zeros((5,5))
print("\nmatriz9:")
print(matriz9)

print("\nmatriz9:")
for i in range(len(matriz9)):
    for j in range(len(matriz9[i])):
        if(i == 2):
            matriz9[i][j]=1 # matriz 2
        if(j == 2):
            matriz9[i][j]=1 # matriz 4
        print(int(matriz9[i][j]), end=" ")
    print()
```

```
matriz9:
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
```

Código 78: Matriz 9 con Numpy

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz10 = np.zeros((5,5))
print("\nmatriz10:")
print(matriz10)

print("\nmatriz10:")
for i in range(len(matriz10)):
    for j in range(len(matriz10[i])):
        if((i==j)and(j%2==0)):
            matriz10[i][j]=1
        if((i==4-j)and(i%2==0)):
            matriz10[i][j]=1
        print(int(matriz10[i][j]), end=" ")
    print()
```

```
matriz10:
1 0 0 0 1
0 0 0 0 0
0 0 1 0 0
0 0 0 0 0
1 0 0 0 1
```

Código 79: Matriz 10 con Numpy

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz11 = np.zeros((5,5))
print("\nmatriz11:")
print(matriz11)

print("\nmatriz11:")
for i in range(len(matriz11)):
    for j in range(len(matriz11[i])):
        if((i==2)and(j%4==0)and(i!=j)):
            matriz11[i][j]=1
        if((i%4==0)and(j==2)and(i!=j)):
            matriz11[i][j]=1
        print(int(matriz11[i][j]), end=" ")
    print()
```

```
matriz11:
0 0 1 0 0
0 0 0 0 0
1 0 0 0 1
0 0 0 0 0
0 0 1 0 0
```

Código 80: Matriz 11 con Numpy

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 11:17:41 2019
"""
import numpy as np
matriz12 = np.zeros((5,5))
print("\nmatriz12:")
print(matriz12)

print("\nmatriz12:")
for i in range(len(matriz12)):
    for j in range(len(matriz12[i])):
        matriz12[i][j]=1;
        print(int(matriz12[i][j]), end=" ")
    print()
```

```
matriz12:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

Código 81: Matriz 12 con Numpy

MÓDULO 7

Módulo 7. Funciones o métodos.

Funciones que retornan valor

Cuando hablamos de funciones o métodos en Python, como en muchos lenguajes, debemos de pensar en dos grandes familias, aquellas que realizan alguna acción sobre variables o sobre otras líneas de código y aquellas de retornan o devuelven algún valor, como aprendimos en matemáticas y = f(x). Con esto podemos comprender que ahora vamos a codificar f.

Una función es también un código ejecutado/realizado que ha funcionado en algún momento y ahora queremos reutilizarlo, encapsularlo o empaquetarlo con un nombre para ser invocado y usado. A ello debemos entender quién es x y quién es y. Es decir, qué le paso a f, que es x, y qué recibe y, lo que devuelve f.

Ejemplo 54. Función o método con retorno

Escriba un programa que permita presentar la suma de dos números, utilice una función.

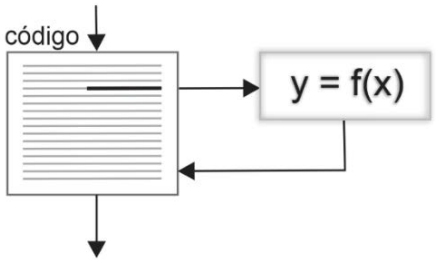


Figura 78. Esquema de una función

Solución: (Análisis)

Anteriormente se realizó la suma de dos números, se requiere dos variables de entrada, n1 y n2, luego una operación como suma = n1 + n2 y finalmente se muestra la variable de salida suma.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Editor de Spyder
4 """
5 def suma(num1, num2):
6     sumato = num1 + num2
7     return sumato
8
9 y = suma(5, 4)
10 print("La suma de 5 + 4 es: ",y)
```

```
La suma de 5 + 4 es:  9
```

Explicación Código en Python

Línea 5: El uso de la palabra reservada *def* permite definir una función o método, utilice los dos puntos para finalizar la línea de la definición del nombre de la función y parámetros a utilizar.
Línea 6: Las tabulaciones permiten distinguir en Python el bloque o conjunto de sentencias a ejecutar.
Línea 7: El uso de la palabra reservada *return*, permite devolver el valor de la operación realizada por la función.
Línea 9: La forma de invocar a estos métodos es a través de una variable del mismo tipo de dato que retorna o devuelve.

Código 82: Función suma de dos números

Podemos distinguir que depende ahora del tipo de dato que define a la función con la variable que retorna o devuelve el valor u objeto final. El tipo de dato de *sumato* es el mismo que *suma*.

Ejemplo 55. Función o método con retorno de un número real

Escriba un programa que permita presentar la división de dos números, utilice una función.

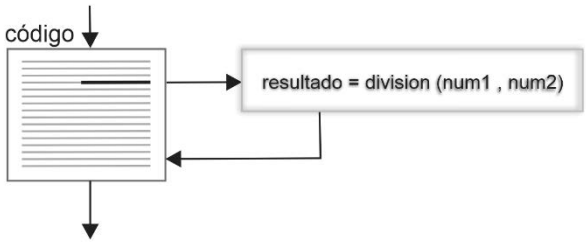


Figura 79. Función que devuelve una operación a resultado

Solución: (Análisis)

Anteriormente se realizó la suma de dos números con una función, ahora es la división, se debe observar que la operación requiere de un número real, es decir la variable que recibe el valor devuelto por la función debe ser del mismo tipo de dato de la variable retornada dentro de la función.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Mon Sep 23 05:38:40 2019
4"""
5def division(n1, n2 = 3):
6    divi = n1 / n2
7    return divi
8
9resultado = division(27)
10print("La división es: ", resultado)
```

```
La división es:  9.0
```

Explicación Código en Python

Línea 5: El uso de un valor ya establecido, permite automatizar la operación para ese número o dato.
Línea 6: La operación genera un número real.
Línea 7: El uso de datos enteros no necesariamente serán devueltos valores enteros, debe observarse la operación que se realiza (si se quería una división entera se usaría doble barra //).
Línea 9: La forma de invocar a estos métodos es a través de una variable del mismo tipo de dato que retorna o devuelve. Obsérvese que sólo es necesario n1

Código 83: Función división de dos números

Ejemplo 56. Función factorial de un número

Escriba un programa que permita presentar el factorial de un número, utilice una función.

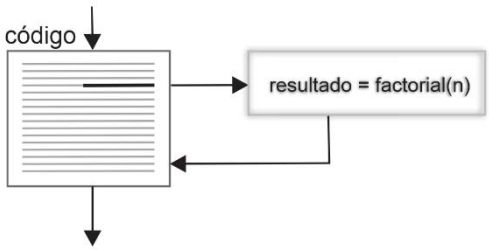


Figura 80. Función que devuelve una operación a resultado

Solución: (Análisis)

Utilizaremos el código anteriormente usado para el cálculo del factorial. Recordemos primero que es un acumulador de productos (*acum* = 1) y que se utiliza un lazo *for* para el recorrido. Matemáticamente, el factorial de 1 y 0 es siempre 1, no hay factorial para números negativos.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Mon Sep 23 06:01:28 2019
4"""
5def factorial(n):
6    acum = 1
7    for i in range(n):
8        v = i+1
9        acum = acum * v
10    return acum
11
12n = int(input("Ingrese n: "))
13print("Factorial de ",n)
14print("es ",factorial(n))
```

```
Ingrese n: 5
Factorial de  5
es  120
```

Explicación Código en Python

Línea 5: El uso del código ya anteriormente realizado, ahora nos facilita el trabajo debido a que sólo lo debemos reutilizar para la creación de nuestra función.
Línea 6: Variable acumulador de multiplicaciones o productos, asignado 1.
Línea 7: Estructura repetitiva, recordar que i empieza en 0, por ello, v se incrementa en 1.
Línea 9: Acumulador de productos.
Línea 10: Retorna acum.

Línea 14: Invocación o llamada del método y a la vez presentación en pantalla del resultado

Código 84: Función división de dos números

Recordemos el método de multiplicación rusa y ahora lo realizaremos con una función.

Ejemplo 57. Función para el método de multiplicación rusa

Escriba un programa que permita ingresar dos números y obtener el producto mediante el método ruso de la multiplicación, utilice una función.

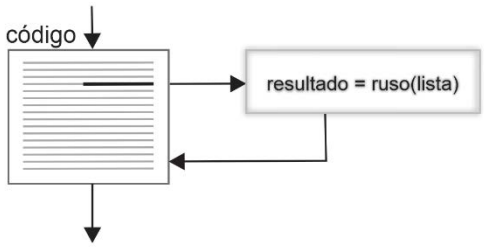


Figura 81. Función que recibe una lista de números

Solución: (Análisis)

Recordemos el código de la multiplicación rusa para usarlo. Mientras un primer número se dividía para dos, al segundo se multiplicaba por dos tantas veces hasta obtener el último cociente 1. Por cada cociente impar se sumaba su correspondiente producto obtenido. Luego la sumatoria de esos productos debe ser el mismo resultado, como si hubiera multiplicado los dos primeros números. Usaremos una lista para el paso de parámetros.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 23 07:54:58 2019
"""
def ruso(lista):
    i = 0
    acum = 0
    while(lista[i]>=1):
        a = lista[i]
        b = lista[i+1]
        if(a%2 == 1):
            acum = acum + b
        lista.append(a//2) #ubica al final
        lista.append(b*2)#ubica al final
        i = i + 2
    return acum

n1 = int(input("Ingrese primer número: "))
n2 = int(input("Ingrese segundo número: "))
resultado = ruso([n1, n2])
print("Respuesta es: ", resultado)
```

```
Ingrese primer número: 11

Ingrese segundo número: 12
Respuesta es:  132
```

Explicación Código en Python

Línea 5: El uso de la lista nos permite tener los valores en un espacio de memoria contiguo.
Línea 8: El recorrido por la lista (arreglo) será de acuerdo a la ubicación de los cocientes y productos.
Línea 11: Se utilizan variables temporales para el proceso.
Línea 13: Se ubica al final (lista) el siguiente cociente.
Línea 14: Se ubica al final (lista) el siguiente producto.
Línea 15: El valor de i avanza (en 2) de 0 a 2 para los cocientes y de 1 a 3 para los productos.

Recordamos que para cada código, hay varias formas más, de realizar el problema.

Código 85: Función ruso usando una lista

Cabe indicar que la lista, al tener varias propiedades (funciones que realizan operaciones como *append* que coloca un valor en la siguiente posición al final de la lista) pueden ser útiles al momento del almacenamiento de esta.

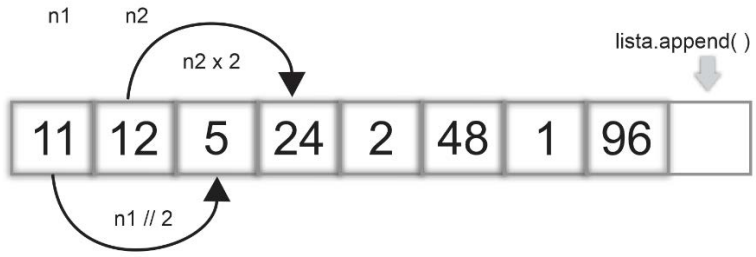


Figura 82. Uso de lista en una función

Con el código implementado, podemos observar que *acum*, guarda 12 + 24 + 96 = 132, que es lo mismo de haber multiplicado 11 x 12 = 132.

Funciones que NO retornan valor

Cuando decimos funciones o métodos, nos referimos a bloques de código que previamente han funcionado para una o unas operaciones específicas. Al retornar un valor determinado, este se lo devuelven a una variable o se visualiza directamente en el programa principal. Ahora nos referiremos los métodos que no retornan un valor. Estos métodos realizan alguna operación, dentro de todo el programa, pero su objetivo que realizar sentencias que afecten a otras líneas de código como parte de un todo.

Ejemplo 58. Función o método sin retorno

Escriba un programa que permita presentar un saludo.

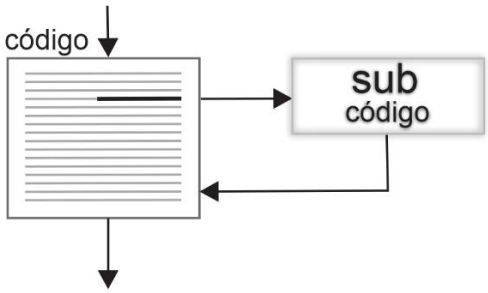


Figura 83. Función saludo

Solución: (Análisis)

Utilizaremos el método *print*( ) para presentación. Note que no existe la palabra reservada *return*.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Sep 23 12:25:00 2019
4 """
5 def saludo():
6     for i in range(5):
7         print(i, "Hola a todos")
8     else:
9         print("Finaliza la función")
10
11 saludo()
```

Explicación Código en Python

Línea 5: Definición de la función.
Línea 6: Lazo repetitivo de 0 a 5 veces.
Línea 7: Presentación del saludo.
Línea 8: Caso contrario.
Línea 9: Mensaje de finalización.

Línea 10: Llamada de la función.

```
0 Hola a todos
1 Hola a todos
2 Hola a todos
3 Hola a todos
4 Hola a todos
Finaliza la función
```

Código 86: Función saludo

Podemos utilizar este tipo de funciones para presentar series o conjeturas matemáticas.

Ejemplo 59. Función o método sin retorno que presenta la serie de Fibonacci

Escriba un programa que permita presentar la serie de Fibonacci. Utilice una función.

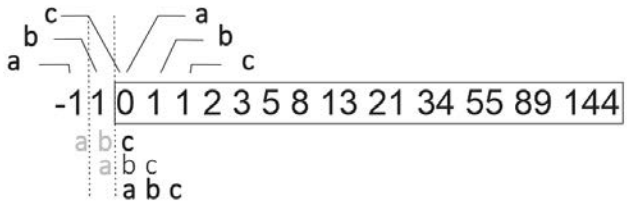


Figura 84. Función Fibonacci

Solución: (Análisis)

Ahora tenemos el mismo código de la serie de Fibonacci pero ahora dentro de un formato en función, debe observar que no existe la palabra reservada *return* y la forma de invocar al método / función es directamente desde la línea de código (sin asignación a nadie).

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Sep 23 06:01:28 2019
4 """
5 def fibonacci(n):
6     a = -1
7     b = 1
8     for i in range(n):
9         c = a + b
10        print(c,end=" ")
11        a = b
12        b = c
13
14 n = int(input("Ingrese enésimo término: "))
15 print("Serie de Fibonacci de ",n, end=" ")
16 print("es: ")
17 fibonacci(n)
```

Explicación Código en Python

Línea 5: Definición de la función.
Línea 6: Variable de inicialización.
Línea 7: Variable de inicialización.
Línea 8: Estructura repetitiva.
Línea 9-12: Suma e intercambio de valores, presentación de la suma del primero y segundo número.
Línea 14: Ingreso del valor de n.
Línea 17: Llamada de la función.

```
Ingrese enésimo término: 13
Serie de Fibonacci de  13 es:
0 1 1 2 3 5 8 13 21 34 55 89 144
```

Código 87: Función Fibonacci

Se puede ahora entender que, las funciones que no retornan valor alguno, como una función matemática trigonométrica, por ejemplo, devuelve un valor en base a una operación u operaciones. Mientras que las funciones que no retornan nada, realmente tienen un bloque de código que realiza acciones sobre variables, con la ayuda del *print* se puede visualizar la ejecución de los cambios que realiza la función.

MÓDULO 8

Módulo 8. Archivos.

Trabajar con archivos de texto

En Python hay varias formas de trabajar con archivos, veamos el trabajo con archivos de texto.

Ejemplo 60. Escritura en un archivo de texto

Escriba un programa que permita escribir texto en un archivo txt.

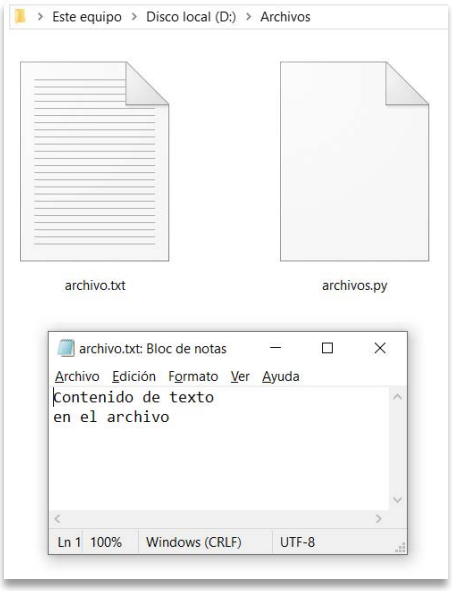


Figura 85. Archivo creado en un directorio

Solución: (Análisis)

Se va a crear un archivo en una carpeta o directorio vacío. A continuación se establecen las líneas de código de clase estándar File.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Sep 23 14:43:09 2019
4 """
5 #archivo creado
6 f = open("archivo.txt", "w")
7 #escribimos contenido
8 f.write("Contenido de texto \nen el archivo")
9 #cerramos el archivo
10 f.close()
```

Explicación Código en Python

Línea 6: open ( ). Permite crear un archivo y su tipo de acceso “w” modo escritura.
Línea 8: write ( ). Permite colocar texto dentro del archivo que ahora es el objeto f.
Línea 10: close ( ). Es el método o función.

Código 88: Archivo de texto

Se puede utilizar con la sentencia *with as* de Python. Ella permite la propia gestión del recurso de apertura y cierre, es decir, como bloque, pudiendo omitir *f.close*( ).

Solución: (Análisis)

Se va a crear un archivo llamado archivo1.txt en una carpeta o directorio vacío. Luego, se establecen las líneas de código de clase estándar File, para escribir en el archivo y cerrarlo.

Código en Python (codificación)

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 23 15:12:31 2019
4  """
5  #archivo creado
6  with open("archivo1.txt", "w") as f:
7      #escribimos contenido
8      f.write("Contenido de texto \nen el archivo")
9      #cerramos el archivo
10     f.close()
```

Explicación Código en Python

Línea 6: open ( ). Permite crear un archivo y su tipo de acceso “w” modo escritura, usando una estructura with as para f.
Línea 8: write ( ). Permite colocar texto dentro del archivo que ahora es el objeto f.
Línea 10: close ( ). Es el método o función.

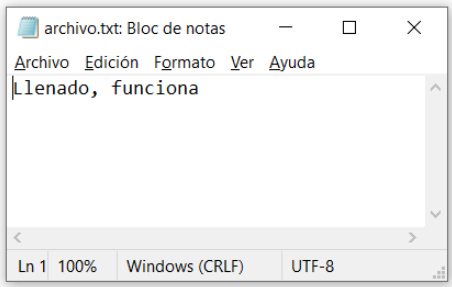
Código 89: Archivo de texto con with as

Solución: (Análisis)

Al observar el código, notamos que la sentencia *with as*, sentencias avanzadas de Python, permite una gestión de los recursos (archivos) sin recurrir al cerrado y mejora el codigo.

Código en Python (codificación)

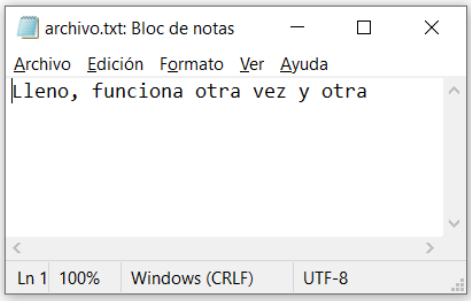
```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 23 14:43:09 2019
4  """
5  try:
6      f = open("archivo.txt", 'w')
7      f.write("Llenado, funciona")
8  except:
9      print ("error al abrir fichero")
10 finally:
11     if f:
12         f.close()
```



Código 90: Archivo de texto con try except finally

Código en Python (codificación)

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 23 14:43:09 2019
4  """
5  try:
6      with open("archivo.txt",'w') as f:
7          f.write("Lleno, funciona otra vez y otra")
8  except:
9      print ("no existe el archivo")
```



Código 91: Archivo de texto con with as

Manejo de archivos de datos

Con la ayuda de varias librerías podemos trata los datos de archivos existentes. Supongamos que tenemos un archivo en Excel con dos columnas, A y B, la primera, con doce números que representa a los meses y la segunda, con números reales, que representan rubros o pagos de un consumo.

	A	B
1	Meses	Rubros
2	1	100,00
3	2	80,00
4	3	200,00
5	4	45,00
6	5	210,00
7	6	30,00
8	7	50,00
9	8	100,00
10	9	110,00
11	10	105,00
12	11	96,00
13	12	80,00

Figura 86. Hoja de cálculo con datos en dos columnas

Y nos piden conocer, de forma gráfica, el mes con el menor rubro.

Ejemplo 61. Lectura de datos de un archivo de Excel con Pandas

Escriba un programa que permita leer un archivo de hoja de cálculo y graficar sus datos, estos datos representan valores de factura mensual de Energía Eléctrica. Muestre el punto o rubro más bajo de la gráfica.

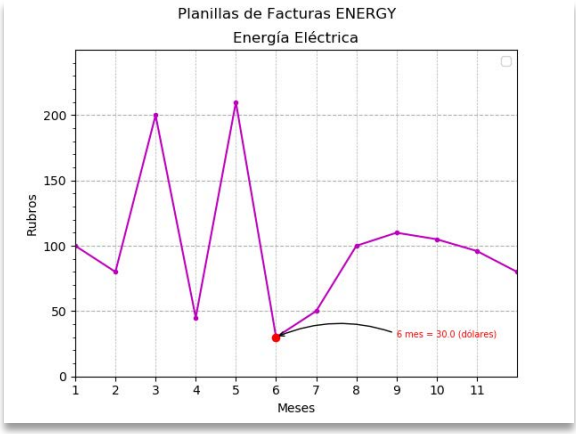


Figura 87. Gráfica generada con matplotlib

Solución: (Análisis)

Primero debemos configurar la terminal de IPython en la pestaña de Gráficas, utilizar el soporte automático en Salida: Automático | Tinker | Qt4….

En la gráfica que se muestra a continuación, se presenta la configuración establecida para las gráficas realizadas con los datos de los archivos de Excel usados.

Código en Python (codificación) p1-2

```python
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# defines el camino donde esta tu archivo de excel
filename = 'planilla.xlsx'
# lectura del archivo excel usando la función pd.read_excel
dataframe = pd.read_excel(filename)
mes = dataframe['Meses']
rubro = dataframe['Rubros']
fig, ax = plt.subplots()
a = 1
b = 12
c = 250
d = 0
# Major ticks every 20, minor ticks every 5
xmajor_ticks = np.arange(a, b, 1)
xminor_ticks = np.arange(a, b, 1)
ymajor_ticks = np.arange(d, c, 50)
yminor_ticks = np.arange(d, c, 10)
ax.set_xticks(xmajor_ticks)
ax.set_xticks(xminor_ticks, minor=True)
ax.set_yticks(ymajor_ticks)
ax.set_yticks(yminor_ticks, minor=True)
# And a corresponding grid
ax.grid(which='major', axis='both', linestyle='--')
ax.plot(mes, rubro, 'm.-') # violeta line with dots
```

Código en Python (codificación) p2-2

```python
#Encuentro el punto mínimo de la gráfica
py_rubro = rubro.min()
px_rubro = rubro.idxmin()+1
nota1 = str(px_rubro)+ ' mes = ' +str(py_rubro)+ ' (dólares)'
plt.plot(px_rubro, py_rubro, 'ro')
plt.text(px_rubro+3, py_rubro-0.26, nota1, color='r', fontsize=7)
print(py_rubro, px_rubro)
# Hago un señalización con flecha
pxz=px_rubro
pyz=py_rubro
pxz1=px_rubro+3
pyz1=py_rubro-0.26
flecha = plt.annotate(' ',xy=(pxz, pyz), xycoords='data',
                      xytext=(pxz1, pyz1), fontsize=9,
                      arrowprops=dict(arrowstyle="->",
                                      connectionstyle="arc3,rad=.2"))

# limito el área
plt.xlim(a, b)
plt.ylim(d, c)
plt.grid(True, 'major', 'x', ls='--', lw=.5, c='k', alpha=.3)
plt.legend(loc='best')
plt.xlabel('Meses')
plt.ylabel('Rubros')
plt.suptitle("Planillas de Facturas ENERGY ")
ax.set_title('Energía Eléctrica');
plt.show()
```

Código 92: Manejo de archivos para gráficas

El uso de Pandas [33], permite la importación de los datos del archivo *planilla.xlsx,* hacia un dataframe, y almacenar toda la columna “Meses” como “Rubros” en una lista. Con subplots de la librería Matplotlib [34] se permite la generación de la figura en una ventana mostrando, bajo código, la gráfica de los datos requerida. Un buen tutorial con sus respectivos códigos pueden encontrarse en Pyplot [35].

Explicación Código en Python

Parte 1-2

Línea 2-4: Importación de las librerías, pandas, numpy y matplotlib.pyplot con sus respectivos objetos de escritura estándar.

Línea 6: Objeto filename asignado el archivo de excel.

Línea 8: Objeto dataframe asignado por Pandas con el método read_excel( ).

Línea 9-10: Lista mes y rubro asignado por Pandas por medio del dataframe.

Línea 11-15: Función subplots con sus límites de gráfica, punto inicial en el eje x es a, punto final es b. Punto inicial en el eje y es d, punto final es c.

Línea 17-24: Rango de distancia mayor y menor en el eje x y en y, siendo el menor como prioritario.

Línea 26: Malla de la gráfica, visible para ambos ejes con estilo de línea.

Línea 27: Dibujo de la línea con color violeta código m.- con puntos en cada etiqueta de valor.

Parte 2-2

Línea 29-30: Identificación del rubro, valor mínimo en el eje x y en y.

Línea 31: Almacenamiento del contenido del mes y valor en una cadena para visualizar etiqueta.

Línea 32-34: Con plot( ) dibuja el punto color rojo (ro) en la posición px_rubro y py_rubro, con text( ) se coloca el texto dibujado en las posiciones desplazadas y de color rojo (‘r’), de tamaño 7.

Línea 36-43: Dibujo de la flecha indicadora en la gráfica.

Línea 46-53: Límite de la gráfica en la ventana con sus etiquetas de ejes y leyendas, color y estilo.

Línea 54: Presentación de la gráfica.

Ejemplo 62. Lectura de imagen con nibabel

Escriba un programa que permita leer un archivo de formato de imagen. Muestre en pantalla.

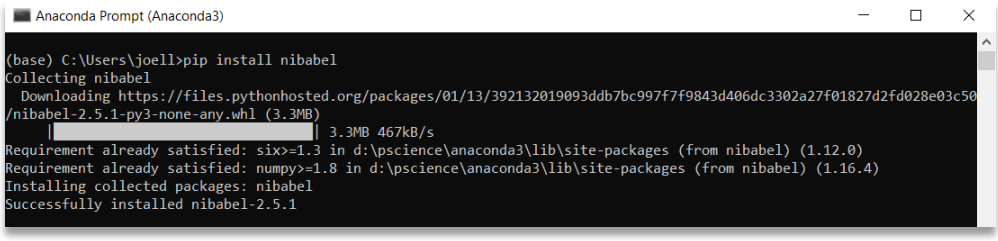


Figura 88. Instalación de librería en consola de Anaconda Prompt

Solución: (Análisis)

Para trabajos más complejos, como leer una imagen médica que tiene un formato estándar a nivel mundial, se deben incorporar al entorno de trabajo, librerías faltantes o requeridas [36]. Para ello, desde la consola de Anaconda (que estamos usando o desde el directorio base si fuera desde Python.org) se deben utilizar los comandos apropiados para descargar de los repositorios disponibles y confiables de Python.

Las librerías que requerimos adicionalmente son nibabel y nilearn [37]. En la imagen se muestran los comandos desde la consola de Anaconda:

- pip install nibabel
- pip install -U –user nilearn

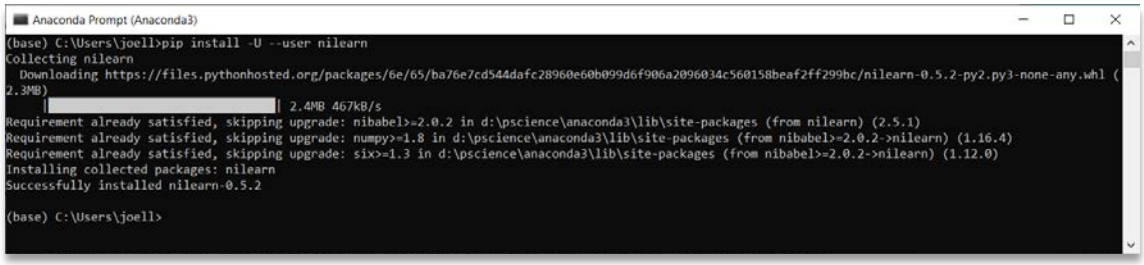


Figura 89. Instalación de librería nilearn

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Editor de Spyder
"""
import os
from nilearn import plotting
data_path = 'D:\P\Medica'
example_file = os.path.join(data_path, 'T1_brain_extractedBrainExtractionBrain.nii.gz
import nibabel as nib
img = nib.load(example_file)
plotting.plot_stat_map(img, threshold=3)
```

Código 93: Visualización del contenido de un archivo de imagen médica nifti

Explicación Código en Python

Línea 5: Importa la librería os para usarla en la línea 8.

Línea 6: Importa plotting para usarla en la línea 12.

Línea 7: El uso de una ruta en data_path que direcciona a la carpeta donde se encuentra el archivo de imagen médica **T1_brain_extractedBrainExtractionBrain.nii.gz** tipo nifti.

Línea 8: El uso de *asignar la ruta* y el nombre del archivo a un objeto.

Línea 10: El uso de nibabel para crear un objeto tipo nib com img de leer propiamente un archivo de tipo nifti dado por el objeto example_file.

Línea 11: Finalmente se carga a img por nibabel

Línea 12: Se visualiza el contenido del archivo nifti, que es un corte en tres dimensiones de una parte del cerebro. La imagen médica proviene de Alzheimer's Disease Neuroimaging Initiative (ADNI)

El archivo del código de Python junto al archivo imagen médica de la base ADNI [38], se colocan en una carpeta para su importación. Existen conjuntos de datasets disponibles que pueden descargarse desde la consola de Python o utilizando portales de distribución como ADNI.

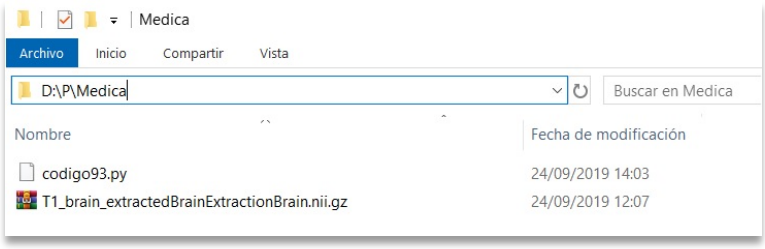


Figura 90. Archivo en formato nifti

Finalmente se muestra el contenido de una imagen en tres dimensiones con formato tipo nifti que utilizamos para este proyecto.

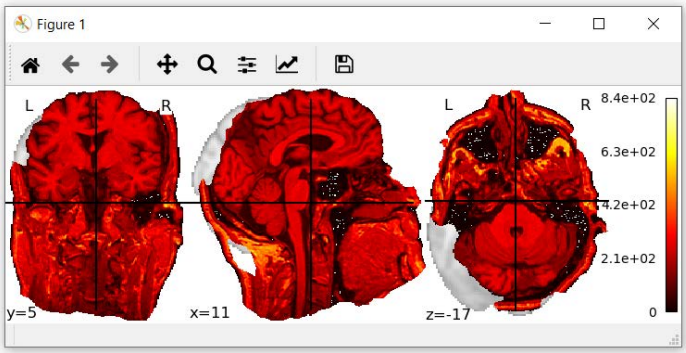


Figura 91. Contenido del archivo nifti

MÓDULO 9

Módulo 9. Procesos.

Manejo de hilos

En esta parte, trataremos de ejemplificar el uso de hilos en Python, para hacer llamadas a subprocesos de forma múltiple. Un ejemplo sencillo de entender es que existen procesos que quedan en espera mientras se ejecuta un programa, a ello Python permite ejecutar otro proceso mientras el primero espera.

Ejemplo 63. Muestre la ejecución de varios procesos

Escriba un programa que permita visualizar el inicio – espera – fin de varios procesos.

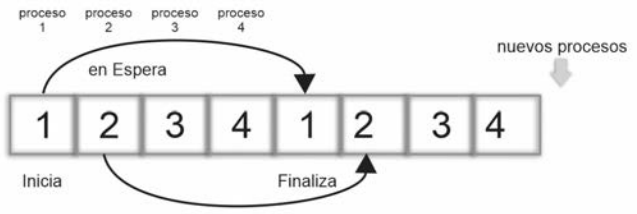


Figura 92. Gráfica de procesos

Solución: (Análisis)

Debemos comprender

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 24 14:47:46 2019
4"""
5import threading as th
6import time as tiempo
7import datetime as dt
8class Proceso(th.Thread):
9    def run(self):
10        ahora = dt.datetime.now()
11        print("{} ¡Inicia!".format(self.getName()))
12        print("en el tiempo: %s" % ahora)
13        tiempo.sleep(2)
14        luego = dt.datetime.now()
15        print("{} ¡Terminé!".format(self.getName()))
16        print("---> en el tiempo: %s" % luego)
17
18
19def main():
20    for i in range(4):
21        proceso = Proceso(name = "Thread-{}".format(i+1))
22        proceso.start()
23        tiempo.sleep(.5)
24
25if __name__ == '__main__':main()
```

Código 94: Tiempo para procesos

Explicación Código en Python

Línea 5-7: Importación de librerías threading, time y datetime.
Línea 8-16: Clase Proceso, que se define un método de ejecución en tiempo real, se inicia y finaliza un proceso, dejándolo en espera durante 2 segundos de cpu y manteniendo (contando) el tiempo de duración.
Línea 19: Se itera cuatro veces, para generar cuatro procesos, dándole un nombre, se inicia el proceso y se espera un tiempo de 0.5 segundos.

Detección de cámara de la portátil

En este espacio, se realiza la detección de la cámara del equipo y luego la generación de una captura mientras se ejecuta el programa. Se debe instalar las librerías externas de OpenCV [39] para Python. Para este trabajo ahora se debe elegir un archivo OpenCV del siguiente repositorio: https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv, de acuerdo a la versión de Python que se use y del sistema de archivo que se use.

Figura 93. Librerías de desarrolladores externos para Python

Identifique la versión de Python que tiene su equipo instalado.

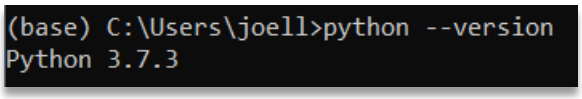


Figura 94. Uso del comando de versión de Python

Para el equipo donde se desarrolla este trabajo se usa: opencv_python-4.1.1+contrib-cp37-cp37m-win_amd64.whl para Python 3.7.3 en un ambiente Windows 10 de 64 bits.

Finalmente se ubica el archivo en algún directorio específico para su instalación.

Figura 95. Instalación de OpenCV

Finalmente se realiza una prueba de la instalación desde alguna carpeta alojando dos archivos, una imagen y el Python.

Código en Python (codificación)	**Explicación Código en Python**
1. import cv2 2. imagen = cv2.imread("imagen.jpg") 3. 4. cv2.imshow("prueba", imagen) 5. cv2.waitKey(0)	**Línea 1:** Importación de librería cv2. **Línea 2:** Objeto imagen que se le asigna la lectura o carga de imagen.jpg. **Línea 4:** Se muestra la imagen. **Línea 5:** De forma visible permanente.

Código 95: Prueba de la librería externa instalada

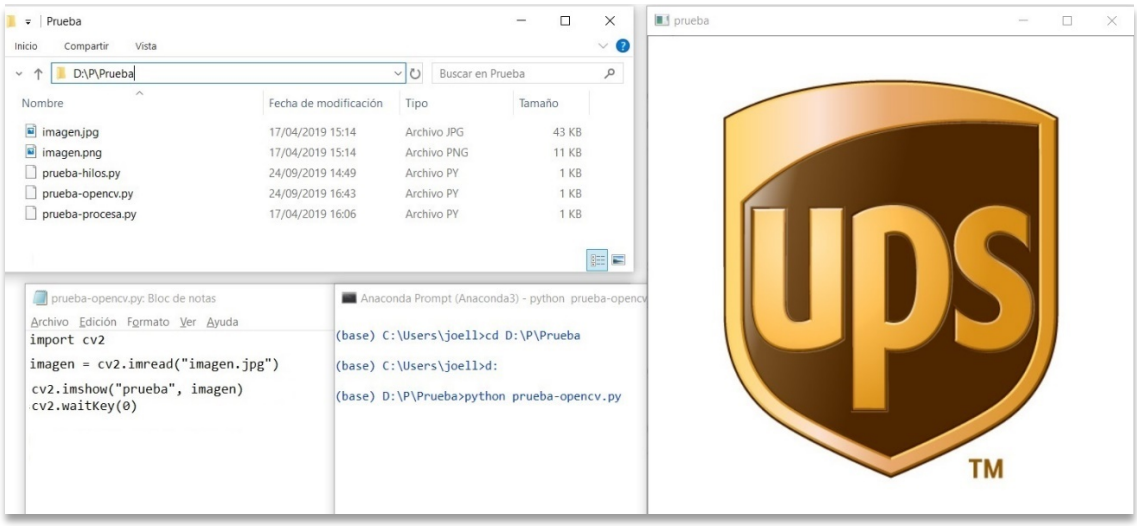


Figura 96. Prueba de la librería OpenCV

Ejemplo 64. Detección de cámara de portátil

Escriba un programa que permita reconocer la cámara del equipo portátil y visualizar la imagen del usuario.

Solución: (Análisis)

Debemos comprender que la utilidad de la librería OpenCV permite reconocer dispositivos de hardware para captura de imágenes. Con la ayuda de la librería Numpy podemos admitir valores con punto decimal para un procesamiento detallado complejo.

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Sep 24 17:04:51 2019
4 """
5 import cv2
6 import numpy as np
7 camera_port = 0
8 cam = cv2.VideoCapture(camera_port,cv2.CAP_DSHOW)
9 kernel = np.ones((5,5),np.uint8)
10 while (True):
11     ret,frame = cam.read()
12     rangomax = np.array([50,255,50])
13     rangomin = np.array([0,51,0])
14     print("Utilice Esc para salir")
15     mascara = cv2.inRange(frame, rangomin, rangomax)
16     opening = cv2.morphologyEx(mascara, cv2.MORPH_OPEN, kernel)
17     x,y,w,h = cv2.boundingRect(opening)
18     center = (np.float32(x+w/2),np.float32(y+h/2))
19     cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),4)
20     cv2.circle(frame,center,6,(0,0,100),-1)
21     cv2.imshow('camara' ,frame)
22     k = cv2.waitKey(1) & 0xFF
23     if k==27:
24         break
25 cam.release()
26 cv2.destroyAllWindows()
```

Explicación Código en Python

Línea 5-6: Importación de librerías cv2 y numpy.
Línea 8: Se asigna al objeto cam, la inicialización de la cámara de portable, con el puerto cero.
Línea 10-24: Se itera la captura de la imagen en un espacio determinado, con la tecla Esc se puede salir de la ejecución del programa.

Código 96: Captura de imagen continua desde la cámara de la portátil

El resultado de la ejecución del código anterior es:

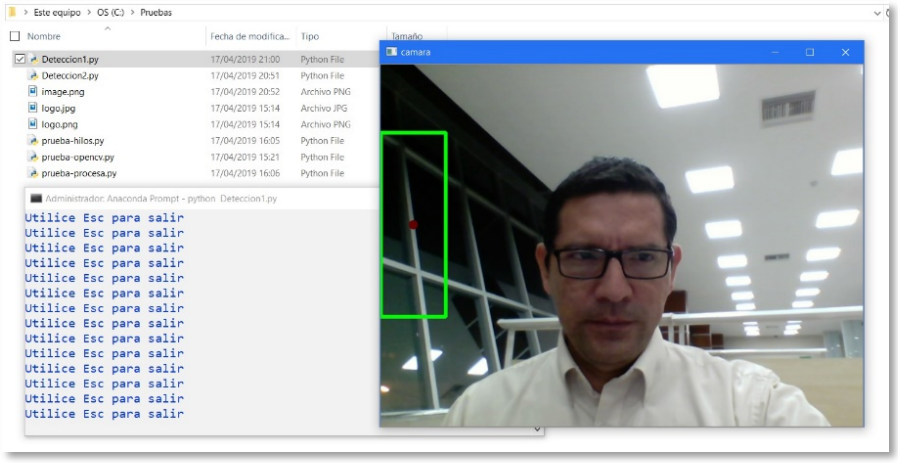


Figura 97. Imagen capturada

Ejemplo 65. Detección de cámara de portátil y generación de una imagen

Escriba un programa que permita reconocer la cámara del equipo portátil y capturar una instantánea de foto del usuario.

Solución: (Análisis)

Debemos comprender

Código en Python (codificación)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Sep 24 17:27:39 2019
4 """
5 import cv2
6 camera_port = 0
7 camera = cv2.VideoCapture(camera_port,cv2.CAP_DSHOW)
8 # Check if the webcam is opened correctly
9 if not camera.isOpened():
10     raise IOError("Cannot open webcam")
11
12 return_value, image = camera.read()
13 print("Te tomamos una foto, revisa la carpeta")
14 cv2.imwrite("image.png", image)
15
16 camera.release()
17 cv2.destroyAllWindows()
```

Explicación Código en Python

Línea 5-6: Importación de librerías cv2 y numpy.
Línea 8: Se asigna al objeto cam, la inicialización de la cámara de portable, con el puerto cero.
Línea 10-24: Se itera la captura de la imagen en un espacio determinado, con la tecla Esc se puede salir de la ejecución del programa.

Código 97: Imagen tomada desde la cámara de la portátil

El resultado de la ejecución del código anterior es:

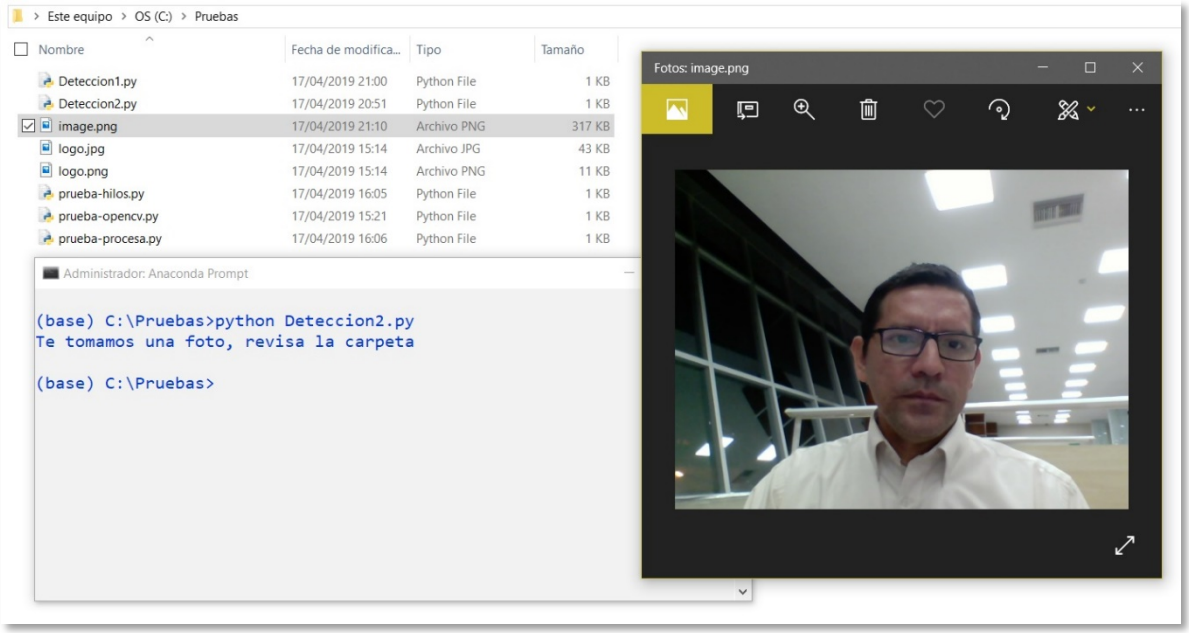


Figura 98. Imagen capturada por medio del código

MÓDULO 10

Módulo 10. Desafíos.

Entender los tipos de datos

Entender los tipos de datos, sus usos adecuados para realizar cálculos o almacenamiento de información requerirá del programador, un adecuado análisis para que todo funcione bien. En los lenguajes de programación se deja un amplio estudio al tipo de dato *cadena de caracteres,* o de tipo string.

Ejemplo 66. Visualización de los tipos de datos de las variables en Python

Escriba un programa que permita visualizar los diferentes tipos de datos en un programa.

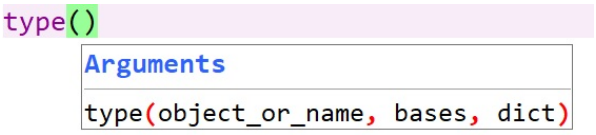


Figura 99. Uso de type

Solución: (Análisis)

Debemos recordar lo que al inicio de este trabajo se indicaba, los tipos de datos de Python pueden usarse dependiendo del buen análisis del problema y el manejo de ellos requiere la sentencia apropiada.

Código en Python (codificación)

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Sep 24 17:45:32 2019
4"""
5print(type(12.45))
6print(type(75))
7print(type("12.45"))
8print(type("Aplica Python"))
9print(type(["lunes", "martes"]))
10print(type(("lunes", "martes")))
11print(type({"lunes":"1", "martes":"2"}))
```

Salida del Código en Python

```
In [28]: runfile('D:/P/codigo98.py', wdir='D:/P')
<class 'float'>
<class 'int'>
<class 'str'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

Código 98: Visualización del tipo de dato en Python

Queda para una próxima edición, la elaboración de nuestras estructuras de datos que permitan nuevas formas de tratamiento o almacenamiento de ellos. De lo estudiado, cada vez más hay aportaciones a nivel mundial en la comunidad de desarrolladores del lenguaje.

Números aleatorios

La palabra *aleatorio* significa para algunos *al azar*, con ello un número aleatorio es un número no determinado [40]. Y se puede generar por medio de un método propio de la librería determinada de Python. Estos números permiten elaborar situaciones donde se requieren valores distintos y que permitan realizar simulaciones. También es muy común en la elaboración de juegos de entretenimiento.

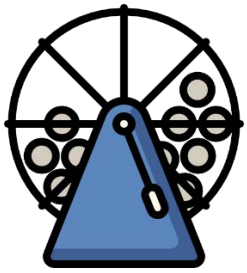


Figura 100. Imagen diseñada por Freepik de www.flaticon.com

Ejemplo 67. Genere una lista de números aleatorios

Escriba un programa que permita generar una lista de 10 números aleatorios y guardarlos en una lista.

81	32	3	17	7	19	4	96	3	14

Figura 101. Uso del número aleatorio

Solución: (Análisis)

Usaremos varias formas de generar números aleatorios, desde la librería estándar de Python así como nuevas librerías disponibles.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 24 18:06:08 2019
"""
import random
lista = []
for i in range(10):
    lista.append(int(random.random()*10))
print(lista)
```

Salida del Código en Python

```
In [1]: runfile('D:/P/codigo99.py', wdir='D:/P')
[9, 3, 9, 8, 6, 7, 9, 4, 6, 9]

In [2]: runfile('D:/P/codigo99.py', wdir='D:/P')
[1, 3, 3, 2, 9, 6, 3, 4, 4, 5]

In [3]: runfile('D:/P/codigo99.py', wdir='D:/P')
[0, 1, 7, 7, 1, 7, 7, 0, 5, 2]

In [4]: runfile('D:/P/codigo99.py', wdir='D:/P')
[5, 7, 2, 5, 9, 7, 1, 6, 3, 6]
```

Código 99: Números aleatorios en Python con random

Otra forma de utilizar la generación de número aleatorios es a través de métodos propios de las librerías. Cabe indicar que con random.random( )*10 se generan diez números aleatorios del 0 al 9, es decir el 10 no se genera aunque aparezca en la línea de código. Recuerde que de 0 a 9, hay diez números, que es lo que representa el 10 en la sentencia.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 24 19:17:09 2019
"""
import random
lista = []
for i in range(10):
    lista.append(int(random.randrange(1,10)))
print(lista)
```

Salida del Código en Python

```
In [1]: runfile('D:/P/codigo100.py', wdir='D:/P')
[2, 6, 8, 8, 8, 8, 4, 2, 5, 2]

In [2]: runfile('D:/P/codigo100.py', wdir='D:/P')
[6, 8, 9, 1, 6, 3, 9, 6, 8, 7]

In [3]: runfile('D:/P/codigo100.py', wdir='D:/P')
[3, 4, 6, 1, 4, 4, 6, 3, 3, 4]

In [4]: runfile('D:/P/codigo100.py', wdir='D:/P')
[3, 9, 8, 2, 5, 9, 1, 2, 4, 4]
```

Código 100: Números aleatorios en Python con randrange

En el código anterior, utilizamos random.randrange(1,10) que indica generar números aleatorios del 1 hasta el 10 sin incluirlo.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 24 19:24:55 2019
"""
from random import randint
lista = []
for i in range(10):
    lista.append(int(randint(0,9)))
print(lista)
```

Salida del Código en Python

```
In [1]: runfile('D:/P/codigo100.py', wdir='D:/P')
[2, 6, 8, 8, 8, 8, 4, 2, 5, 2]

In [2]: runfile('D:/P/codigo100.py', wdir='D:/P')
[6, 8, 9, 1, 6, 3, 9, 6, 8, 7]

In [3]: runfile('D:/P/codigo100.py', wdir='D:/P')
[3, 4, 6, 1, 4, 4, 6, 3, 3, 4]

In [4]: runfile('D:/P/codigo100.py', wdir='D:/P')
[3, 9, 8, 2, 5, 9, 1, 2, 4, 4]
```

Código 101: Números aleatorios en Python con randint

En el código que se muestra ahora, random.randint(0,9) indica que se debe generar números aleatorios del 0 inclusive al 9 incluido.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 24 19:40:22 2019
"""
import numpy as np
lista = []
for i in range(10):
    lista.append(np.random.randint(0,9))
print(lista)
```

Salida del Código en Python

```
In [1]: runfile('D:/P/codigo102.py', wdir='D:/P')
[0, 8, 0, 4, 2, 3, 0, 2, 4, 0]

In [2]: runfile('D:/P/codigo102.py', wdir='D:/P')
[2, 7, 4, 0, 2, 0, 3, 3, 7, 2]

In [3]: runfile('D:/P/codigo102.py', wdir='D:/P')
[3, 7, 3, 0, 1, 2, 4, 6, 6, 4]

In [4]: runfile('D:/P/codigo102.py', wdir='D:/P')
[8, 3, 3, 6, 6, 6, 6, 1, 8, 5]
```

Código 102: Números aleatorios en Python con Numpy

Utilizando la librería numpy, encontramos random.randint(0,9) genera números aleatorios del 0 inclusive al 9 sin incluir.

Ejemplo 68. Generación de dados por medio de números aleatorios

Escriba un programa que permita generar el lanzamiento de dos dados, diez veces, si la suma de los dos da 7, se muestra un mensaje de: "GANASTE", utilice la generación de números aleatorios.

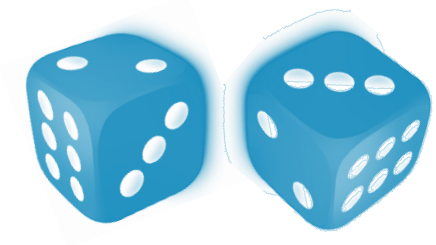


Figura 102. Números aleatorios en lanzamiento de dados

Solución: (Análisis)

De lo que hemos aprendido, podemos notar que se requiere de dos variables que almacenarán los dos números aleatorios correspondientes al dado 1 y al dado 2, si realizamos el lanzamiento de diez veces, usando una estructura repetitiva debemos utilizar una condición que permita sumar los dos valores aleatorios de los dados, si da siete, usaremos una variable acumuladora de las veces ganada. Adicionalmente, podemos visualizar el resumen de cuántas veces ganó el jugador.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 24 19:40:22 2019
"""
import random as ra
acum = 0
print("Lanzamientos:")
for i in range(1,11):
    dado1 = ra.randint(1,6)
    dado2 = ra.randint(1,6)
    if(dado1 + dado2 == 7):
        acum = acum + 1
        print((i),"dado 1:",dado1,"dado 2:",dado2,
              "Suma 7, ¡GANASTE!")
    else:
        print((i),"dado 1:",dado1,"dado 2:",dado2)
print("Ganaste ",acum," veces ")
```

Salida del Código en Python

```
Lanzamientos:
1 dado 1: 5 dado 2: 6
2 dado 1: 4 dado 2: 6
3 dado 1: 1 dado 2: 2
4 dado 1: 5 dado 2: 3
5 dado 1: 5 dado 2: 1
6 dado 1: 6 dado 2: 1 Suma 7, ¡GANASTE!
7 dado 1: 6 dado 2: 1 Suma 7, ¡GANASTE!
8 dado 1: 5 dado 2: 1
9 dado 1: 6 dado 2: 3
10 dado 1: 5 dado 2: 5
Ganaste  2  veces
```

Código 103: Dados con números aleatorios

De acuerdo al problema, el uso del lazo *for* con el rango de 1 a 11 (sin incluir, debido a que no llega al número final, sino hasta antes del 11), nos permite generar los lanzamientos. Con el uso de *randint*, definimos la generación de los números aleatorios del 1 al 6, debido a que el dado sólo tiene esos números, sin incluir el cero. Finalmente, el uso de la estructura selectiva doble nos permite valorar la condición por verdadero de la suma de los valores aleatorios de los dos dados. Con ello, incrementamos el acumulador para presentarlo al finalizar el lazo.

Ejemplo 69. Generación de veces ganadas

Escriba un programa que permita generar tantos lanzamientos de dos dados, cada uno de ellos de diez veces, si la suma de los dos da 7, se muestra un mensaje de: "GANASTE", utilice la generación de números aleatorios. Debe indicarnos finalmente en qué intento ganaríamos tres veces.

Figura 103. ¿Cuándo ganaré?

Solución: (Análisis)

Para este problema debemos generar tantos juegos o jugadas hasta lograr ganar, y en cada uno de esos juegos de lanzamientos indicar en qué intento de juego ganamos como está la condición, tres veces. El código anterior nos enseña que es el jugador (usuario) que, al hacer funcionar el programa, indica las veces que ganó o no. Ahora usaremos la potencia de la cpu para hacer que el equipo (PC) juegue tantas veces hasta que gane tres veces, puede ser tan rápida la respuesta de su equipo a medida que le pidamos que gane por ejemplo nueve veces de diez juegos, es decir el jugador debe usar el juego miles de veces hasta que exista la probabilidad de que gane nueve de diez veces y eso requiere de tiempo computacional, como lo vamos a ver ahora con tres juegos ganados y luego iremos aumentando el número de veces deseado.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 2):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 3):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Salida del Código en Python

```
Sacaste  3  veces 7 en juego  1
Sacaste  3  veces 7 en juego  2
Sacaste  3  veces 7 en juego  3
En el juego 4 lograste 3  veces ganar
```

Código 104: Uso de la potencia de la CPU v1.0

Como se observa el uso de un lazo repetitivo *while*, nos permite iterar tantas veces hasta obtener las tres veces ganadas.

Supongamos que deseamos ganar 5 veces el juego. ¿Cuántos intentos o veces jugar? para ganar. Debemos cambiar ahora el valor de la condición en el lazo *while*.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 4):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 3):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Salida del Código en Python

```
Sacaste  4  veces 7 en juego  2
Sacaste  3  veces 7 en juego  5
Sacaste  4  veces 7 en juego  9
Sacaste  3  veces 7 en juego  11
Sacaste  5  veces 7 en juego  12
En el juego 13 lograste 5  veces ganar
```

Código 105: Uso de la potencia de la CPU v2.0

Al querer conocer cuántos juegos deberé jugar para ganar la cantidad de veces deseada, notamos que el uso de la CPU demora más. Es decir, el tiempo para encontrar las veces que debe un jugador aumenta considerablemente. Ahora, queremos conocer cuántos intentos de juego debe jugar el usuario para ganar ocho veces.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 7):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 3):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Salida del Código en Python

```
Sacaste  3  veces 7 en juego  3
Sacaste  3  veces 7 en juego  8
Sacaste  4  veces 7 en juego  10
Sacaste  3  veces 7 en juego  20
Sacaste  3  veces 7 en juego  21
Sacaste  3  veces 7 en juego  22
Sacaste  3  veces 7 en juego  33
Sacaste  4  veces 7 en juego  35
En el juego 36 lograste 8  veces ganar
```

Código 106: Uso de la potencia de la CPU v3.0

Notamos, que el número de veces de juego va en aumento y el tiempo de CPU también, al ejecutar estos programas no piense que al no mostrar mensaje alguno implica que no está haciendo nada, sino mas bien está ejecutando tantas veces hasta ganar las veces que se solicita.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 8):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 3):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Código 107: Uso de la potencia de la CPU v4.0

Salida del Código en Python

```
Sacaste  4  veces 7 en juego  6
Sacaste  3  veces 7 en juego  30
Sacaste  3  veces 7 en juego  39
Sacaste  3  veces 7 en juego  47
Sacaste  3  veces 7 en juego  49
Sacaste  3  veces 7 en juego  62
Sacaste  4  veces 7 en juego  69
Sacaste  3  veces 7 en juego  77
Sacaste  4  veces 7 en juego  80
En el juego 81 lograste 9  veces ganar
```

Y ahora el mayor desafío, si deseamos que ganemos exactamente 8 veces siete en un juego, en nueve juegos. Observe el cambio.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 8):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 8):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Código 108: Uso de la potencia de la CPU v5.0

Salida del Código en Python

```
Sacaste  8  veces 7 en juego  47329
Sacaste  8  veces 7 en juego  47579
Sacaste  8  veces 7 en juego  81553
Sacaste  8  veces 7 en juego  103644
Sacaste  8  veces 7 en juego  139878
Sacaste  8  veces 7 en juego  179002
Sacaste  8  veces 7 en juego  221772
Sacaste  8  veces 7 en juego  404495
Sacaste  8  veces 7 en juego  422890
En el juego 422891 lograste 9  veces ganar
```

Finalmente, realice intente sacar 9 veces 7 en 9 juegos.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 21:33:58 2019
"""
import random as ra
acum = 0
juego = 0
contador = 0
while(contador <= 8):
    #print("Lanzamientos:")
    for i in range(1,11):
        dado1 = ra.randint(1,6)
        dado2 = ra.randint(1,6)
        if(dado1 + dado2 == 7):
            acum = acum + 1
    if(acum >= 9):
        contador = contador + 1
        print("Sacaste ",acum," veces 7 en juego ",juego)
    acum = 0
    juego = juego + 1
print("En el juego",juego,"lograste",contador," veces ganar")
```

Código 109: Uso de la potencia de la CPU v6.0

Salida del Código en Python

```
Sacaste  9  veces 7 en juego  1113857
```

El equipo queda casi paralizado para el siguiente resultado…

```
Sacaste  9  veces 7 en juego  1113857
Sacaste  9  veces 7 en juego  4121599
```

```
Sacaste  9  veces 7 en juego  1113857
Sacaste  9  veces 7 en juego  4121599
Sacaste  9  veces 7 en juego  5188060
Sacaste  9  veces 7 en juego  5746351
Sacaste  9  veces 7 en juego  6126153
Sacaste  9  veces 7 en juego  7934160
Sacaste  9  veces 7 en juego  9062891
Sacaste  10  veces 7 en juego  9318981
Sacaste  9  veces 7 en juego  9705959
En el juego 9705960 lograste 9  veces ganar
```

Ejemplo 70. Adivina el número

Escriba un programa que permita generar un número aleatorio y el usuario trate de adivinar el número generado.

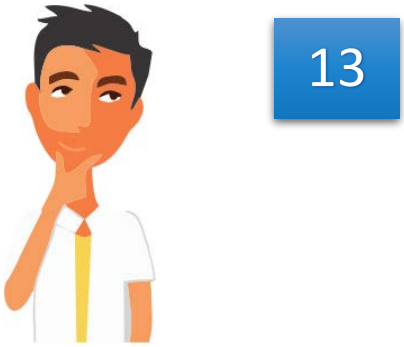


Figura 104. Adivine el número

Solución: (Análisis)

Si generamos un número aleatorio entre 1 al 10 es más fácil de adivinar por la pequeña probabilidad de pensar en el número, Cuando el rango aumenta a 100 o más, la probabilidad es mayor. Por eso colocaremos una condición que permita ayudar al usuario cuando el número adivinado en mayor, se mostrará un mensaje de que debe ingresar un número menor y lo contrario si ingresa un número menor.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import random as ra
adivina = ra.randint(1,100)
contar = 0
n = -99
while(n != adivina):
    n = int(input("Ingrese un número:"))
    if((n!=adivina)):
        if(n > adivina):
            print("Ingrese un número más bajo")
        else:
            print("Ingrese un número más alto")
    contar = contar + 1
print("GANASTE!! en el intento", contar)
```

Salida del Código en Python

```
Ingrese un número:45
Ingrese un número más bajo

Ingrese un número:25
Ingrese un número más bajo

Ingrese un número:15
Ingrese un número más bajo

Ingrese un número:5
Ingrese un número más alto

Ingrese un número:11
Ingrese un número más alto

Ingrese un número:14
Ingrese un número más bajo

Ingrese un número:12
Ingrese un número más alto

Ingrese un número:13
GANASTE!! en el intento 8
```

Código 110: Juego con aleatorios

Ejemplo 71. Matriz de números aleatorios

Escriba un programa que almacene números aleatorios en una matriz de cinco por cinco.

1	5	7	3	1
3	1	1	1	2
1	7	2	5	1
1	3	8	4	1
2	6	1	9	6

Figura 105. Matriz de aleatorios

Solución: (Análisis)

Por ahora utilizaremos la librería Numpy para el uso de la creación de una matriz de ceros del tamaño de 5 x 5. Luego Almacenaremos los números aleatorios mediante la función randint mediante dos lazos *for* , donde i recorre las filas y j, recorre las columnas.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import numpy as np
import random as ra
matriz=np.zeros((5,5))
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        matriz[i][j]=ra.randint(1,10)

print("matriz de aleatorios:\n",matriz)
```

Salida del Código en Python

```
matriz de aleatorios:
 [[3. 9. 4. 8. 4.]
 [2. 5. 1. 6. 5.]
 [9. 3. 6. 1. 5.]
 [1. 9. 2. 7. 5.]
 [7. 1. 4. 8. 3.]]
```

Código 111: Matriz de aleatorios

Finalmente mostramos toda la matriz gracias al formato que trae Numpy, como se muestra en la salida del código.

Ejemplo 72. Como una tabla de Bingo

Escriba un programa que permita generar una primera versión de una tabla de BINGO. Utilizando los números aleatorios.

B	I	N	G	O
1	5	7	46	1
3	16	31	1	2
15	7	2	5	61
1	30	8	60	75
2	6	45	9	6

Figura 106. Como una tabla de Bingo

Solución: (Análisis)

Este problema utiliza números aleatorios en cada una de las posiciones de la tabla. Lo que el libro no abarca, es la búsqueda de elementos repetidos. Por ahora realizaremos como si fuera una tabla de BINGO en una primera versión, es decir lograr que se generen los números aleatorios como si fuera una matriz. Adicionalmente las tablas de bingo tienen rangos de números que sólo pueden estar en una columna.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import numpy as np
import random as ra
matriz=np.zeros((5,5))
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        if(j==0):
            matriz[i][j]=int(ra.randint(1,15))
        if(j==1):
            matriz[i][j]=int(ra.randint(16,31))
        if(j==2):
            matriz[i][j]=int(ra.randint(31,45))
        if(j==3):
            matriz[i][j]=int(ra.randint(46,60))
        if(j==4):
            matriz[i][j]=int(ra.randint(61,75))

print("B\tI\tN\tG\tO\n")
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        print(int(matriz[i][j]),"\t",end=" ")
    print("\n")
```

Salida del Código en Python

```
B	I	N	G	O

1	19	37	47	65

2	30	31	46	67

15	26	43	53	62

15	21	31	60	70

10	31	35	54	67
```

Código 112: Como una tabla de Bingo

En esta primera versión notamos que j, tiene el control para generar un número en el rango específico cada vez que recorremos la columna j descrita en el código.

Ejemplo 73. Números de lotería

Escriba un programa que permita generar números aleatorios, que representen bolillas de números similar al número de la LOTERÍA.

Figura 107. Números de lotería

Solución: (Análisis)

Entendemos que detrás de todo está la generación de números aleatorios. El problema puede utilizar o no matrices como lo estamos haciendo, pero como se ha dicho anteriormente, todos los ejercicios pueden ser resueltos desde las distintas perspectivas de análisis. Con ello la variedad de posibilidades de resolver el problema son varias, trataremos de usar lo que hemos venido aprendiendo en este trabajo.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import numpy as np
import random as ra
matriz=np.zeros((4,4))
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
            matriz[i][j]=int(ra.randint(1,9))

print("Números de Lotería")
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        print(int(matriz[i][j]),end="")
    if(i==1):
        print("\n")
```

Salida del Código en Python

```
Números de Lotería
98431794

53535132
```

Código 113: Números de lotería

Es cierto que puede escribirse más ejemplos de códigos diferentes para este problema. Le animamos a encontrar, formas diferentes y más cortas para otra posible solución.

Ejemplo 74. Bolitas azules y verdes

Escriba un programa que permita generar un juego donde tenemos una funda de bolillas azules y verdes. En total son 20, 15 verdes y 5 azules. El juego permite realizar intentos de adivinar una bolita azul, que es lo que deseamos, sacar todas las azules y contar cuántas veces se realizó el juego hasta ganar.

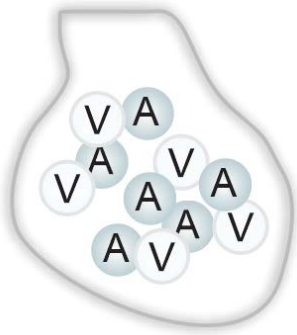


Figura 108. Bolitas azules y verdes

Solución: (Análisis)

Si la cantidad de bolitas es 20, sumando las azules y verdes, debemos pensar que por cada bolita azul o verde escogida (obtenida) se debe restar del total existente en la bolsita. Esta es una forma de encontrar la solución.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import random as ra
cantidad = 20
verde = 15
azul = 5
cverde = 0
cazul = 0
intento = 0
objetivo = 0
while(cazul!=azul):
    intento = intento + 1
    if((cverde != verde) or (cazul != azul)):
        bolita = int(ra.randint(1,2))
        if(bolita == 1):
            cazul = cazul + 1
        else:
            cverde = cverde + 1
    if(cverde == verde):
        bolita = 1;
    else:
        bolita = 2;
print("Cantidad de intentos: ",intento)
print("Bolitas azules sacadas: ",cazul)
print("Bolitas verdes sacadas: ",cverde)
print("De 5 bolitas azules faltantes: ",(azul - cazul))
print("De 15 bolitas verdes faltantes: ",(cantidad - azul - cverde))
```

Salida del Código en Python

```
Cantidad de intentos:  10
Bolitas azules sacadas:  5
Bolitas verdes sacadas:  5
De 5 bolitas azules faltantes:  0
De 15 bolitas verdes faltantes:  10
```

Código 114: Bolitas azules y verdes

Otra forma de realizar el problema y lo dejamos como desafío, es que, cómo cambiaría el programa si al escoger una bolita verde, como lo que deseamos es azul, la devolvemos a la misma fundita, es decir la probabilidad de escoger una azul la siguiente vez será menor ya que disminuyen las azules, pero se mantienen la verdes debido a que no la retiramos sino que la devolvemos.

Ejemplo 75. Piedra, papel y tijera

Escriba un programa que permita simular el juego de piedra, papel y tijera, entre el usuario y el pc, mediante números aleatorios.

Figura 109. Piedra, papel y tijera

Solución: (Análisis)

Por ahora trataremos de usar variables que representan al usuario y al pc, recordar que mediante un menú de opciones iremos solicitando al usuario el ingreso de un número que represente a la piedra, papel y tijera. Las indicaciones del juego son las mismas que conocemos, papel gana a piedra, piedra gana a tijera y tijera gana a papel, lo interesante es que usaremos aleatoriedad para similar un juego entre el usuario y el pc.

Código en Python (codificación)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 25 22:59:08 2019
"""
import random as ra
ganador = 3
gana_usuario = 0
gana_pc = 0
intento = 0
empate = 0
while((gana_usuario != ganador) and (gana_pc != ganador)):
    intento = intento + 1
    pc = ra.randint(1,3)
    print("\nEcoja: ")
    print("1.- Piedra")
    print("2.- Papel")
    print("3.- Tijera")
    usuario = int(input("Ingrese 1, 2 o 3: "))
    if((usuario == 1) and (pc == 3)):
        gana_usuario+=1
    else:
        if((usuario == 2) and (pc == 1)):
            gana_usuario+=1
        else:
            if((usuario == 3) and (pc == 2)):
                gana_usuario+=1
            else:
                if(usuario == pc):
                    empate+=1
                else:
                    gana_pc+=1
    print("Usuario: ",usuario,"PC: ",pc)
print("\nCantidad de intentos: ",intento)
print("Empates: ",empate)
print("Gana usuario: ",gana_usuario)
print("Gana pc: ",gana_pc)
if(gana_pc>gana_usuario):
    print("GANA PC!!!")
else:
    print("GANA USUARIO!!!")
```

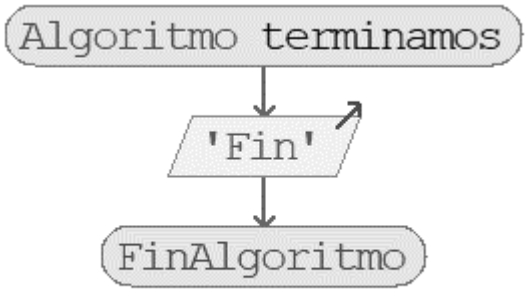
Salida del Código en Python

```
Ecoja:
1.- Piedra
2.- Papel
3.- Tijera

Ingrese 1, 2 o 3: 2
Usuario:  2 PC:  1

Ecoja:
1.- Piedra
2.- Papel
3.- Tijera

Ingrese 1, 2 o 3: 2
Usuario:  2 PC:  3

Cantidad de intentos:  5
Empates:  1
Gana usuario:  1
Gana pc:  3
GANA PC!!!
```

Código 115: Piedra, papel y tijera

Otra forma de trabajar el problema sería mediante un diccionario de datos donde los índices son 1, 2 y 3, mientras que las etiquetas serían los nombres de piedra, papel y tijera, realizar una búsqueda en el diccionario por índice y presentar su etiqueta debido a la comparación del valor buscado.

Y con esto hemos terminado el libro.



TERMINAMOS.
¡BUEN TRABAJO!

Bibliografía

[1] A. M. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.

[2] Y. N. Moschovakis, “What Is an Algorithm?,” in *Mathematics Unlimited — 2001 and Beyond*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 919–936.

[3] A. Turing, “Can a machine think,” *composingdigitalmedia.org*, 1956.

[4] P. Novara, “PSeInt,” 2003. [Online]. Available: http://pseint.sourceforge.net/. [Accessed: 11-Sep-2019].

[5] M. Zemansky and R. Dittman, “Calor y termodinámica,” 1979.

[6] The Python Software Foundation, “Python Software Foundation,” *Python 3.7.4 documentation*, 2019. [Online]. Available: https://docs.python.org/3/library/functions.html#print.

[7] Tutorial de Python, “Tutorial de Python.” [Online]. Available: https://tutorial.recursospython.com/. [Accessed: 11-Sep-2019].

[8] H. Costa, “Hektor Profe | Academia de programación,” 2018. [Online]. Available: https://www.hektorprofe.net/. [Accessed: 11-Sep-2019].

[9] The Python Software Foundation, “collections — Container datatypes — Python 3.7.4 documentation.” [Online]. Available: https://docs.python.org/3/library/collections.html. [Accessed: 11-Sep-2019].

[10] The Python Software Foundation, “8.3. collections — High-performance container datatypes — Python 2.7.16 documentation.” [Online]. Available: https://docs.python.org/2/library/collections.html. [Accessed: 11-Sep-2019].

[11] E. Swokowski, J. Abreu, and M. Oliveró, “Calculus with analytic geometry. Cálculo con geometría analítica,” 1989.

[12] E. Purcell, D. Varberg, and H. Castillo, “Cálculo con geometría analítica,” 1993.

[13] L. Leithold, “El cálculo: con geometría analítica,” 1992.

[14] W. Fleming and D. Varberg, “Álgebra y trigonometría con geometría analítica,” 1991.

[15] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, “What makes a good code example?: A study of programming Q&A in StackOverflow,” in *IEEE International Conference on Software Maintenance, ICSM*, 2012, pp. 25–34.

[16] Stack Overflow Team, “Stack Overflow - Where Developers Learn, Share, & Build Careers.” [Online]. Available: https://stackoverflow.com/. [Accessed: 11-Sep-2019].

[17] J. Madachy, “Madachy’s Mathematical Recreations,” 1979.

[18] G. H. (Godfrey H. Hardy and C. P. (Charles P. Snow, *A mathematician’s apology*. Cambridge University Press, 1992.

[19] E. Weisstein, “Narcissistic Number,” 2005.

[20] I. Taneja, “Flexible Power Narcissistic Numbers with Division,” 2016.

[21] MIT License, “Spyder: The Scientific Python Development Environment — Documentation — Spyder 3 documentation,” 2018. [Online]. Available: https://docs.spyder-ide.org/. [Accessed: 11-Sep-2019].

[22] JournalDev, “Python Tutorial - JournalDev.” [Online]. Available: https://www.journaldev.com/python-tutorial. [Accessed: 12-Sep-2019].

[23] W3Schools, “Python Tutorial.” [Online]. Available: https://www.w3schools.com/python/default.asp. [Accessed: 12-Sep-2019].

[24] Pythones, “Python básico desde CERO; GRATIS y en ESPAÑOL || Pythones.” [Online]. Available: https://pythones.net/. [Accessed: 12-Sep-2019].

[25] DataFlair, “Python Tutorial for Beginners - Learn Python Programming from Scratch - DataFlair.” [Online]. Available: https://data-flair.training/blogs/python-tutorials-home/. [Accessed: 12-Sep-2019].

[26] A. F. Horadam, “A Generalized Fibonacci Sequence,” *Am. Math. Mon.*, vol. 68, no. 5, pp. 455–459, May 1961.

[27] L. Euler, “De integratione aequationum differentialium altiorum graduum,” *Misc. Berolinensia*, Jan. 1743.

[28] E. Whittaker and G. Watson, “Forms of the remainder in Taylor’s series, § 5.41 in A Course in Modern Analysis,” 1990.

[29] H.-J. Weber and G. B. (George B. Arfken, *Essential mathematical methods for physicists*. Academic

Press, 2004.
[30] P. SELLERS, “Stanislaw M. Ulam’s Contributions to Theoretical Theory,” *dornsife.usc.edu*.
[31] S. Ulam, “Combinatorial Analysis in Infinite Sets and Some Physical Theories,” *SIAM Rev.*, vol. 6, no. 4, pp. 343–355, Oct. 1964.
[32] G. Tee, “Russian peasant multiplication and Egyptian division in Zeckendorf arithmetic,” 2002.
[33] NumFOCUS, “Python Data Analysis Library — pandas: Python Data Analysis Library.” [Online]. Available: https://pandas.pydata.org/. [Accessed: 24-Sep-2019].
[34] NumFOCUS, “Matplotlib: Python plotting — Matplotlib 3.1.1 documentation.” [Online]. Available: https://matplotlib.org/. [Accessed: 24-Sep-2019].
[35] NumFOCUS, “Pyplot tutorial — Matplotlib 3.1.1 documentation.” [Online]. Available: https://matplotlib.org/3.1.1/tutorials/introductory/pyplot.html. [Accessed: 24-Sep-2019].
[36] Brett Matthew, “Neuroimaging in Python — NiBabel 2.5.0 documentation.” [Online]. Available: https://nipy.org/nibabel/. [Accessed: 24-Sep-2019].
[37] NiPy ecosystem, “Nilearn: Machine learning for NeuroImaging in Python — Machine learning for NeuroImaging.” [Online]. Available: http://nilearn.github.io/introduction.html#installing-nilearn. [Accessed: 24-Sep-2019].
[38] A. J. Saykin *et al.*, “Alzheimer’s Disease Neuroimaging Initiative biomarkers as quantitative phenotypes: Genetics core aims, progress, and plans.,” *Alzheimers. Dement.*, vol. 6, no. 3, pp. 265–73, May 2010.
[39] Opencv Dev Team, “Welcome to opencv documentation! — OpenCV 2.4.13.7 documentation.” [Online]. Available: https://docs.opencv.org/2.4/index.html. [Accessed: 24-Sep-2019].
[40] J. Katz and Y. Lindell, “Introduction to modern cryptography,” 2014.

Carrera de Computación

Codifica en Python es el primero de una serie de libros que tiene como propósito enseñar Programación a los jóvenes de los primeros ciclos de esttudios universitarios o a quienes se inician en el mundo de la programación.

Más de 100 ejercicios con sus respectivas respuestas enfocados en el análisis, diseño y desarrollo de una solución mediante el uso de este lenguaje, son parte de este volumen. Los ejercicios desarrollados por el autor en su calidad de profesor en carreras de ingenierías, han sido escogidos por su importancia en el aprendizaje de la programación a partir de la experiencia con los estudiantes.

Al mismo tiempo, aspira a despertar la motivación por la aplicación de lógica algorítmica en problemas comunes que pueden ser resueltos de forma sencilla, abriendo las puertas para alternativas de solución; como fundamento de la ciencia de los datos, de la información, del conocimiento, de la inteligencia artificial, etc.

Una herramienta útil también para el profesor, tanto para la preparación como para el desarrollo de clases.

ABYA YALA | UNIVERSIDAD POLITÉCNICA SALESIANA

ECUADOR SALESIANOS DON BOSCO

ISBN: 978-9978-10-429-3