Introducción al análisis de datos con Python

Profesor: Carlos Jiménez
cajimenezo@unal.edu.co

Contenido

1. Introducción (2h)
    a) Conceptos básicos
    b) Instalación de programas
    c) Conjuntos de datos
    d) Bucles y condicionales
    e) Definiciones
2. Estadística (4h)
    a) Media, mediana y promedio
    b) Datos de una variable, variables aleatorias
    c) Datos de dos variables y correlaciones
3. Pandas y visualización de datos (8h)
    a) Graficas con matplotlib
    b) Cargar y editar datos con pandas
    c) Seaborn y visualización avanzada de datos
4. Seguimiento a proyectos (10h)
    a) Proyecto básico con datos irreales
    b) Proyecto real con datos propios

Instalamos python y un editor

Página oficial

https://www.python.org/downloads/

Guia de instalación
https://www.digitalocean.com/community/tutorials/install-python-windows-10

Spyder

https://www.spyder-ide.org/

Autoevaluación

1. Escriba un código simple que imprima en pantalla la suma de todos los números desde el 1 hasta el 100
2. Escriba un código que pregunta tu edad y responda con estas dos opciones, “eres mayor de edad” o “eres menor de edad”
3. Escriba un código que genere un lista con los números impares que hay entre el 0 y el 80, excepto los números 15 y 39.

```
import numpy as np
# 1. La suma de los números desde el 1 hasta el 100
s = 0
for i in range(101):
    s = s + i
print("La suma de los números desde el 1 hasta el 100 es: ", s)

# 2. Pregunta por tu edad
print("¿Cuál es tu edad?")
x = int(input())
if x<18:
    print("Eres menor de edad")
else:
    print("Eres mayor de edad")

# 3. Lista de números impares
L = []
for i in range(40):
    L.append(2*i + 1)
L.remove(15)
L.remove(39)
print(L)
```

Conceptos básicos de programación

Lenguajes de programación científica

- Python
- R
- SQL
- Scala
- Julia
- Javascript
- Java
- C/C++
- MATLAB
- Wolfram

Python es un lenguaje de programación orientado a objetos de código abierto, que agrupa datos y funciones para lograr flexibilidad. En la ciencia de datos, Python suele utilizarse para el procesamiento de datos, la implementación de algoritmos de análisis de datos y el entrenamiento de algoritmos de aprendizaje automático y aprendizaje profundo. Python admite múltiples estructuras de datos y utiliza una sintaxis en inglés sencillo, lo que lo convierte en un gran lenguaje para los programadores principiantes.

Tutorial
https://www.w3schools.com/python/default.asp

Variable: Nombre simbólico que apunta a un objeto específico

Operadores aritméticos: Suma (+), resta (-), multiplicación (*), división (/).

Tipos de datos incorporados: Numéricos (enteros, flotantes, complejos), Secuencias (cadenas, listas, tuplas), Booleanos (Verdadero, Falso), Diccionarios y Conjuntos.

Expresiones booleanas: Expresiones en las que el resultado es True o False.

Condicional: Evalúa una expresión booleana y realiza algún proceso dependiendo del resultado. Se maneja mediante sentencias if/else.

Bucle: Ejecución repetida de bloques de código. Pueden ser bucles for o while.

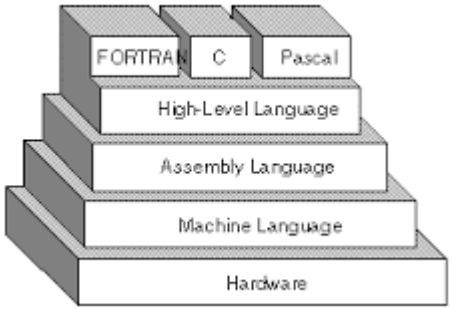
Funciones: Bloque de código organizado y reutilizable. Se crean con la palabra clave def.

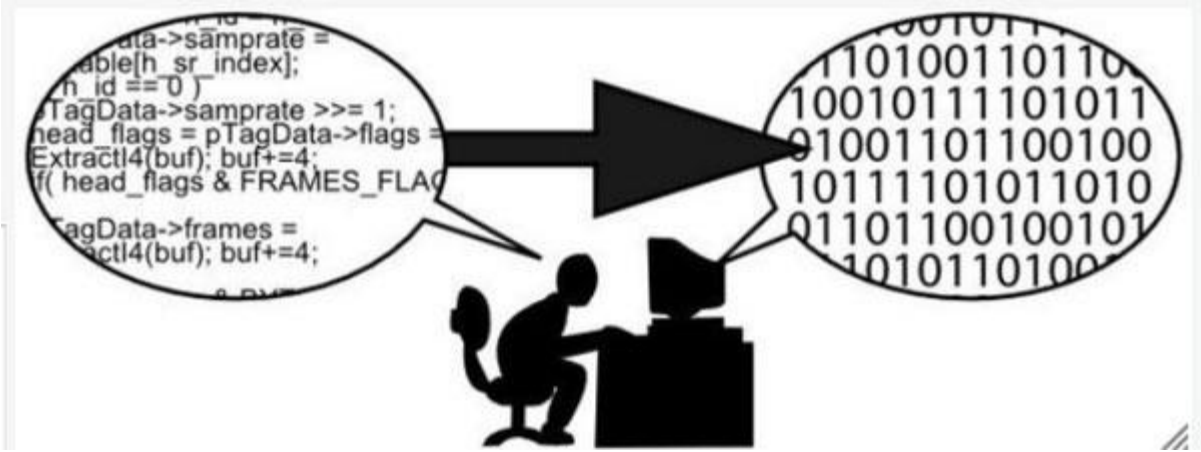
Argumentos: Objetos que se pasan a una función. Por ejemplo: sum([1, 2, 4])

Intérprete: Abre un terminal o línea de comandos y escribe «python <nombre del archivo>» o
Abre un shell de Python: Abre un terminal y escribe python o python3 dependiendo de tu sistema.

Editor: Ejecuta varias líneas de código (código fuente) ejemplo: IDLE, Spyder, Jupiter.

Lenguaje de alto nivel: Es una serie de órdenes con estructura coherente escrito por un programador y fácilmente entendible por otro humano (Código fuente)
Lenguaje de bajo nivel: Conjunto de órdenes dictadas al computador escrito en términos más básicos y que dependen más del hardware (Código binario, lenguaje máquina, lenguaje de ensamblador). Este es fácilmente entendible por un computador





En python utilizamos para escribir el código fuente un editor, en lugar del interprete. Utilizaremos principalmente Spyder o IDLE.

Pseudocódigo: Planteamiento de una idea para resolver un problema por pasos, es poco riguroso y entendible fácilmente por otro humano. Debe ser lo más simple posible

Algoritmo es un conjunto de instrucciones ordenadas, finitas y delimitadas que se crean con el fin de describir de forma sistemática la ejecución de una tarea, Es riguroso.

Diagrama de flujo: Planteamiento gráfico de la solución de un problema por medio de pasos y bloques enlazados de manera coherente, es fácil de entender y de implementar por un humano

Código: Planteamiento de un problema por pasos coherentes implementado en un lenguaje de programación específico, fácilmente entendible por un computador

Convención en un diagrama de flujo DF

Caja de Terminales - Inicio / Fin	
Entrada / Salida	
Proceso / Instrucción	
Decisión	
Conector / Flecha	

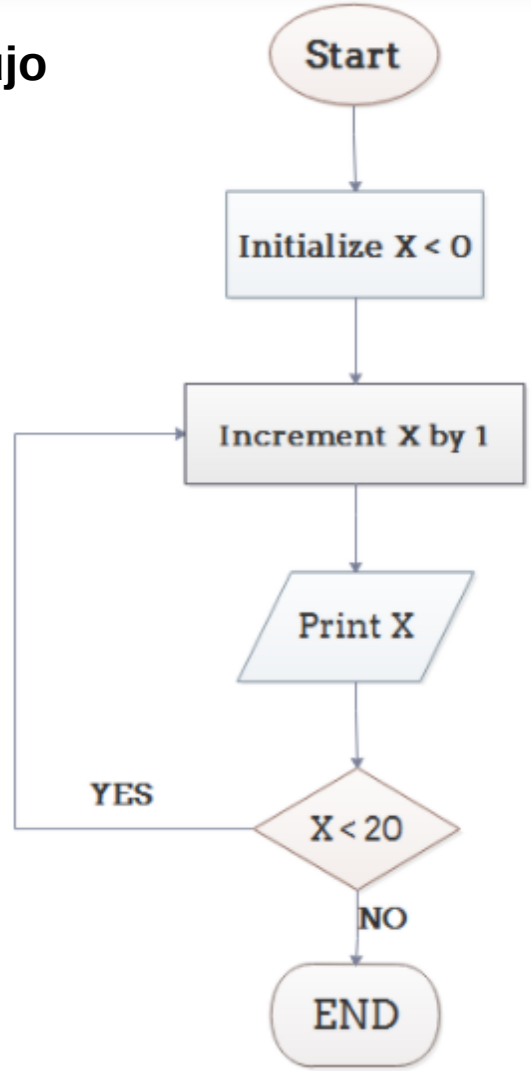
Ejemplo:
Imprimir los números del 1 al 20

Pseudocódigo: Imprima los números desde el 1 hasta el 20

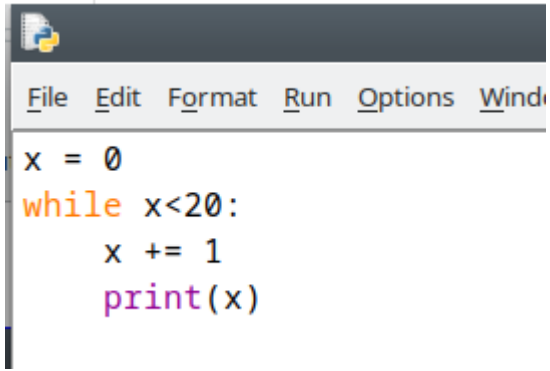
Algoritmo:

Paso 1: Inicializar X como 0,
Paso 2: Incrementa X en 1,
Paso 3: Imprimir X,
Paso 4: Si X es inferior a 20, vuelva al paso 2.

Diagrama de flujo



Código en Python



Tipos de datos o tipos primitivos

Python tiene cuatro tipos primitivos: enteros, flotantes, booleanos y cadenas o Strings.

- Int: Números enteros
- Float: Números decimales
- Booleanos: { 0 , 1 } o { True , False }
- Strings: Caracteres

```
#Enteros -> int

year = 2021
angulo = -45

#Flotantes -> float

temperatura = -5.55
edad = 26.0

#Booleanos -> "0,1"

esta_frio = True
es_bajo = False

#Strings -> caracteres

profesor = "Carlos Jiménez"
```

Data-types

Existen muchas maneras de agrupar variables

- Sets
- Lists
- Tuples
- Dictionary
- Array

Conjuntos de datos (set)

- Conjuntos (set): Es un colectivo desordenado de elementos, únicos e inmutables, aunque se pueden agregar más términos es no indexado

```
MySet = {1, 2, 3.14}
print(MySet)

MySet.add("pi")
print(MySet)

MySet.add(2)
print(MySet)

MySet[0]
print(MySet)
```

```
{1, 2, 3.14}
{1, 2, 3.14, 'pi'}
{1, 2, 3.14, 'pi'}
Traceback (most recent call last):


TypeError: 'set' object is not subscriptable
```

Conjuntos de datos (Lists)

Las listas (list): Me permiten guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos. Es un tipo indexado y mutable.

```
# Lists

MyList = [1, "Hola", 3.14]
print(MyList)

MyList.append("pi")
print(MyList)

print(MyList[1])

MyList[2] = MyList[3]
print(MyList)

List2 = [1, 2]
List3 = [3, 4]

print(List2 + List3 + MyList)
```

```
In [15]: runfile('C:/Users/User/Desktop/d
[1, 'Hola', 3.14]
[1, 'Hola', 3.14, 'pi']
Hola
[1, 'Hola', 'pi', 'pi']
[1, 2, 3, 4, 1, 'Hola', 'pi', 'pi']
```

Conjuntos de datos (Tuplas)

- Las tuplas (tuple): Sirven para lo mismo que las listas, pero en este caso es un tipo inmutable pero indexado. (No los usaremos mucho)

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Conjuntos de datos (Diccionarios)

- Diccionarios: Son ordenados, mutables y no permiten duplicarse, soportan datos acoplados por medio de una llave (key).

```
dic = {"llave1": "valor1" , "llave2":"valor2", "llave3": 3}
print(dic)

print(dic["llave2"])

dic["llave_nueva"] = "Valor_nuevo"
print(dic)

del dic["llave3"]
print(dic)
```

```
{'llave1': 'valor1', 'llave2': 'valor2', 'llave3': 3}
valor2
{'llave1': 'valor1', 'llave2': 'valor2', 'llave3': 3,
'llave_nueva': 'Valor_nuevo'}
{'llave1': 'valor1', 'llave2': 'valor2', 'llave_nueva':
'Valor_nuevo'}
```

Un arreglo es un conjunto de datos todos del mismo tipo indexados por uno o más índices, requiere la librería numpy. Muy útiles para nuestros propósitos, pues se permiten operar matemáticamente muy fácil

```
c = np.array([[2, 3, 4],
              [5, 7, -8]])
d = np.array([[1, -1],
              [5, 7],
              [0, 4]])
e = np.zeros((2, 3))
f = np.ones((2,3))
print(d)
print(d.shape)
print(f + c)
print(d[0,1])
```

```
In [28]: runfile('C:/Users/User/
[[ 1 -1]
 [ 5  7]
 [ 0  4]]
(3, 2)
[[ 3.  4.  5.]
 [ 6.  8. -7.]]
-1
```

```
e = np.zeros((2, 3))
f = np.ones((2,3))
print(e)

for i in range(2):
    for j in range(3):
        e[i,j] = (i + 1)* (j + 1)

print(e)
```

```
In [38]: runfile('C:/
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 2. 3.]
 [2. 4. 6.]]
```

Conjuntos de datos (rango)

Los rangos (range): Es un tipo de secuencias que nos permite crear secuencias de números. Es un tipo inmutable y se suele utilizar para realizar bucles.

```
# Enumera desde el cero hasta el 5
print(*range(6))

# A una lista vacía ingresa números desde el 4 hasta el 30 en pasos de a 3
lista2 = []
for i in range(4,30,3):
    lista2.append(i)
print(lista2)
```

```
0 1 2 3 4 5
[4, 7, 10, 13, 16, 19, 22, 25, 28]
>>>
```

```python
m = np.array(range(5, 40, 7))
n = np.linspace(5, 40, 6)
print("Desde 5 hasta 40 en pasos de 7", m)
print("Desde 5 hasta 40 y hay 6 valores", n)
print("Tenga cuidado con la diferencia entre una lista de listas y un arreglo")
g = [[1, 2, 3], [4, 5, 6]]
h = np.array([[1, 2, 3], [4, 5, 6]])

print(g)
print(h)
```

```
In [45]: runfile('C:/Users/User/Desktop/datos.py', wdir='C:/Users/User/I
Desde 5 hasta 40 en pasos de 7 [ 5 12 19 26 33]
Desde 5 hasta 40 y hay 6 valores [ 5. 12. 19. 26. 33. 40.]
Tenga cuidado con la diferencia entre una lista de listas y un arreglo
[[1, 2, 3], [4, 5, 6]]
[[1 2 3]
 [4 5 6]]
```

El comando input() permite una comunicación inicial con el usuario que sirve para ingresar valores, debe especificarse el tipo de datos de ingreso, ya sea int, float o string.

```
print("ingrese el número: x= :")
x = int(input())

print(x + 1)
```

```
print("Ingrese su nombre")
name = input()

print("Ingrese su edad")
age = input()
print(name, "tiene", age, "años")
```

```
Ingrese su nombre
carlos
Ingrese su edad
42
carlos tiene 42 años
```

Operadores lógicos

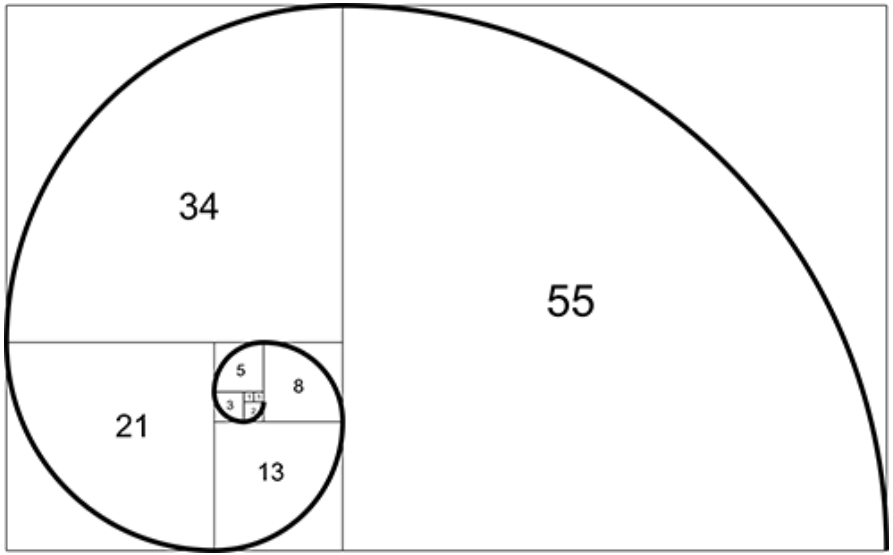
```python
import numpy as np

print("Operador igual")
for i in range(10):
    for j in range(10):
        if i*j == 24:
            print(i, j, i*j)
print("Operador distinto, negación, not")
for i in range(2,4):
    for j in range(2,4):
        if i*j != 6:
            print(i, j, i*j)
print("Operador menor o igual")
for i in range(3, 8):
    for j in range(1, 4):
        if i*j <= 5:
            print(i, j, i*j)
print("Operador and, multiplicación")
for i in range(8):
    for j in range(8):
        if i*j == 24 and i - j == 2:
            print(i, j, i*j)
print("Operador or, suma")
for i in range(1, 8):
    for j in range(1, 8):
        if i/j == 2 or i*j == 2:
            print(i, j, i*j)
```

```
In [14]: runfile('/home/carlos/Desktop/untitled0.py'
Operador igual
3 8 24
4 6 24
6 4 24
8 3 24
Operador distinto, negación, not
2 2 4
3 3 9
Operador menor o igual
3 1 3
4 1 4
5 1 5
Operador and, multiplicación
6 4 24
Operador or, suma
1 2 2
2 1 2
4 2 8
6 3 18
```

Tarea

El número áureo resulta de la división del último número (calculado, pues esta es infinita) de la serie fibonacci sobre el penúltimo número. Haga un programa que calcule la sucesión Fibonacci y el número áureo.



Funciones, condicionales y bucles

Funciones

Una función te permite definir un bloque de código reutilizable que se puede ejecutar muchas veces dentro de tu programa. Las funciones te permiten crear soluciones más modulares para problemas complejos.

La definición de una función tiene las siguientes características (Python):

- La palabra clave def
- Un nombre de función
- Paréntesis ’()’, y dentro de los paréntesis los parámetros de entrada, aunque los parámetros de entrada sean opcionales.
- Dos puntos ’:’
- Algún bloque de código para ejecutar
- Una sentencia de retorno (opcional)

Cada programa tiene incorporado muchas funciones definidas por defecto, sin embargo se pueden construir muchas otras según la necesidad del código, las cuales pueden llamarse en otra parte.

```
def SayHello(name):
    return print("Hi", name)
SayHello("Carlos")

def Suma(x, y):
    return x + y

print("La suma de 2 + 3 es: ", Suma(2, 3))

def VolEsfera(r):
    return (4/3)*np.pi*r**3
print("El volumen de la esfera de radio ", 2, "es ", round(VolEsfera(2), 3))
```

```
In [52]: runfile('C:/Users/User/Desktop/untitled0
Hi Carlos
La suma de 2 + 3 es:  5
El volumen de la esfera de radio  2 es  33.51
```

Ejercicio:

Diseñe una función que solicite un valor al usuario y devuelva una respuesta

Este código solicita cuatro números para calcular la suma de dos fracciones

```
print("Ingrese los valores (Separados por espacio)")
print("a, b, c, d para calcular la suma a/b + c/d")
a, b, c, d = input().split()
a, b, c, d =[int(a), int(b), int(c), int(d)]
def num(a, b, c, d):
    return (a*d + b*c)

def den(b, d):
    return b*d
print("La suma de ", a,"/", b, "+", c, "/", d, "es: ", num(a, b, c, d), "/", den(b, d))
```

```
In [68]: runfile('C:/Users/User/Desktop/untitle
Desktop')
Ingrese los valores (Separados por espacio)
a, b, c, d para calcular la suma a/b + c/d
1 2 3 4
La suma de  1 / 2 + 3 / 4 es:  10 / 8

In [69]:
```

Podemos definir funciones pensando en el contexto de las matemáticas, para ello se pueden utilizar la función def, o la función lambda, así:

```
def f(x):
    return x**2
print(f(4))

def ff(x,y):
    return x**2 - y**2
print(ff(4,2))

#Funcion lambda

g = lambda x: np.sin(x)
print(g(np.pi/6))

h = lambda x, y: np.exp(-(x*2-y**2))
print(h(1, 2))
```

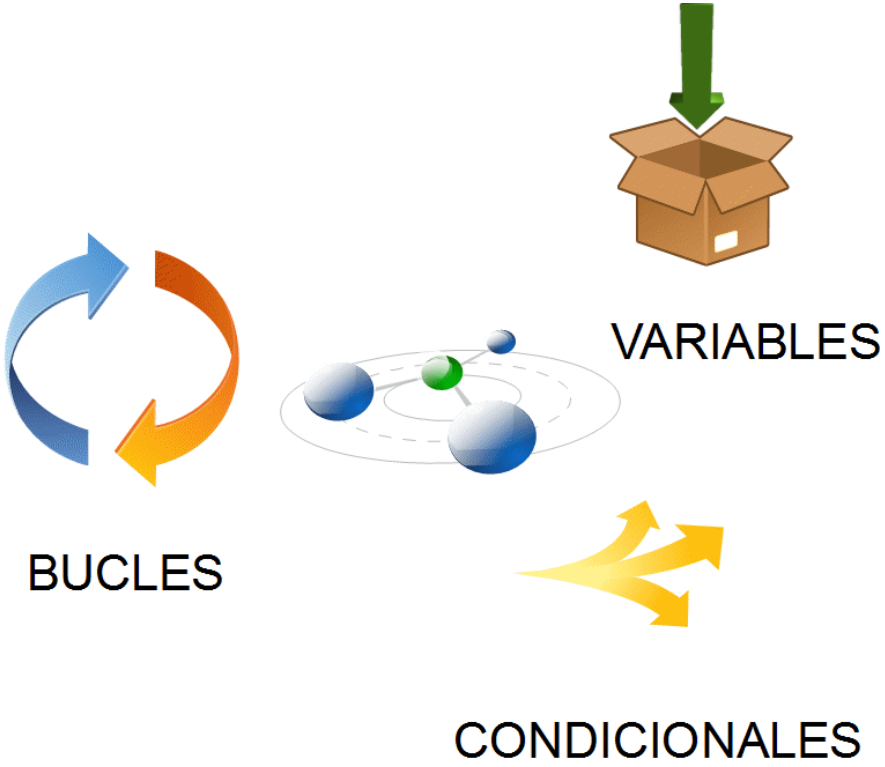
```
16
12
0.49999999999999994
7.38905609893065

In [81]:
```

Condicionales y bucles

Los condicionales y bucles alteran el flujo normal de un programa

En programación existen múltiples recursos que permiten evaluar una condición que da lugar a la ejecución de una orden, existen de muchos tipos, pero en python existe el condicional “if” y los bucles “while” y “for”



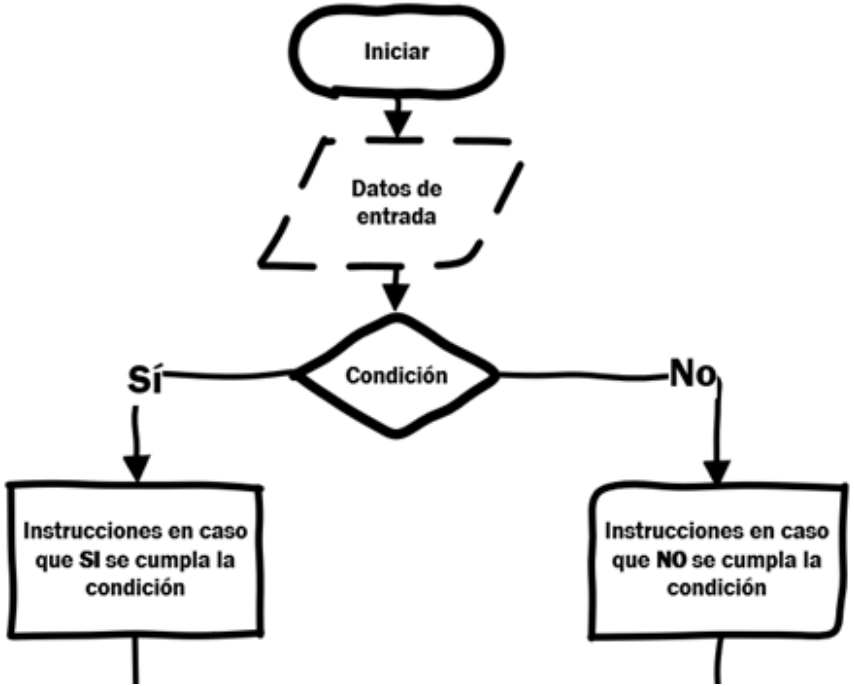
Condicionales y bucles If

Las sentencias condicionales nos permiten alterar el flujo de un algoritmo dependiendo si ciertas condiciones se cumplen o no.

```
if True:
  print('¡el bloque If se ejecutará!')
```

```
x = 5

if x > 4:
  print("¡La condición era verdadera!") #Esta sentencia se ejecuta
```



Condicionales y bucles If

Es posible agregar una condición adicional con el comando “else” e incluso agregar múltiples condiciones con el comando “elif”.

```
print("ingrese un número")
N = int(input())

if N>8:
    print("Este número es mayor que:", 8)
elif N == 8:
    print("Este número es igual a:", 8)
else:
    print("Este número es menor que:", 8)
```

If anidado: Es posible utilizar un condicional dentro de otro

```
print("Ingrese un número N")
N = float(input())

if N>=10:
    if N<=20:
        print("Este número está entre 10 y 20")
    else:
        print("Este número es mayor a 20")
else:
    print("Este número es menor a 10")
```

Condicionales y bucles (while)

Bucle while:

En este tipo de iteración, siempre y cuando la prueba se evalúe como true, la declaración o bloque de instrucciones se seguirán ejecutando. Por lo tanto, el flujo ejecutará todas las sentencias dentro del bloque de bucle y si el condicionante del mismo es false, se ejecutarán las siguientes sentencias después del “while”. Si el condicionante da siempre como resultado true, en ese caso, obtendremos lo que se denomina como un bucle infinito.

```
i = 0
while i<10:
    print(i**2)
    i = i + 1

a = [0,1]
j = 0
while j<10:
    a.append(a[j+1] + a[j])
    j = j + 1
print(a)
```

```
0
1
4
9
16
25
36
49
64
81
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

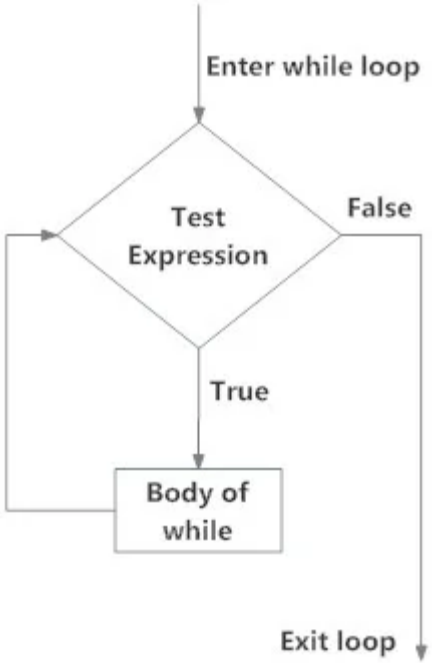


Fig: operation of while loop

Condicionales y bucles

Es posible mezclar bucles y condicionales anidados.
El comando break puede romper el flujo del programa, útil sii por error se escribe un bucle infinito, también se puede hacer manualmente tumbando el Kernel o oprimiendo ctrl + c

```
x = 1
while x<10:
    if x == 5:
        x += 1
    print(x)
    x +=1
```

```
x = 1
while (x <=
10):
   if(x == 5):
                x
= x+1

continue
   print(x)
   break
```

```
lenguajes = ["Python", "C", "C++", "Java"]
i = 0

while i < len(lenguajes):
   print(lenguajes[i])
   i += 1
```

Condicionales y bucles

El bucle “for” trabaja de forma similar

```
for i in <collection>:
  <loop body>
```

```
for i in range(1, 11):
    print(i)
```

```
lenguajes = ["Python", "C", "C++", "Java"]

for lenguaje in lenguajes:
    print(lenguaje)

nun = []
for i in range(2,124,13):
    nun.append(i)

print(nun)
```

```
Python
C
C++
Java
[2, 15, 28, 41, 54, 67, 80, 93, 106, 119]
```

“Los ciclos for son utilizados para ciclos en los cuales ya está definido el número de iteraciones, y los ciclos while sirven mejor para ciclos donde el número de iteraciones puede variar dependiendo de una condición”

Ejemplos y problemas

Ejemplo:
Imprimir de una lista dada los términos que tenga una característica en común.

```
names = ["Superman", "Thor", "Iron man", "Flash", "Mujer maravilla"]
namesWithm = []
for name in names:
    if "m" in name:
        namesWithm.append(name) # add this element to this array.
print(namesWithm)
```

```
['Superman', 'Iron man', 'Mujer maravilla']
```

Problema 1

Calcula el número Pi hasta 10 decimales de precisión.
Hay muchas maneras de hacerlo, por ejemplo utilizando el problema de Basilea

$$\pi = \sqrt{6\left(\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots\right)}$$

Problema 2

Diseñe un código que calcule el porcentaje de grasa corporal corporal (%GC) utilizando la fórmula de Deurenberg. Debe salir un mensaje según su (%GC)

%GC (% Grasa corporal) = 1,2 x (IMC) + 0,23 x (Nuestra edad) – 10,8 x (sexo) – 5,4

IMC= masa / estatura^2 en kg y m, respectivamente
Edad en años
Sexo: 1 para hombres y 0 para mujeres

- **Si su IMC es menos de 18.5**, se encuentra dentro del rango de peso insuficiente.
- **Si su IMC es entre 18.5 y 24.9**, se encuentra dentro del rango de peso normal o saludable.
- **Si su IMC es entre 25.0 y 29.9**, se encuentra dentro del rango de sobrepeso.
- **Si su IMC es 30.0 o superior**, se encuentra dentro del rango de obesidad.