Operaciones de python

int(input()) (para dar valor numerico sin decimales)

float(input()) (para dar valor numerico con decimal)

input() (para escribir)

if … (para dar opciones/condiciones y poner una respuesta o consecuancia(solo numeros))

else (para continuar con if)

Definimos una lista con tres elementos:

lista=[10, 20, 30]

Imprimimos la cantidad de elementos que tiene la lista, en nuestro caso lo definimos con 3:

print(len(lista)) # imprime un 3

Agregamos una nuevo elemento al final de la lista llamando al método append:

lista.append(100)

Si llamamos nuevamente a la función len y le pasamos el nombre de nuestra lista ahora retorna un 4:

print(len(lista)) # imprime un 4

Round para redondear la cantidad de dijitos

raw\_input() solo escribir

Importante

los operadores:

relacionales (>, <, >=, <= , ==, !=)

matemáticos (+, -, \*, /, //, \*\*, %)

pero nos están faltando otros operadores imprescindibles:

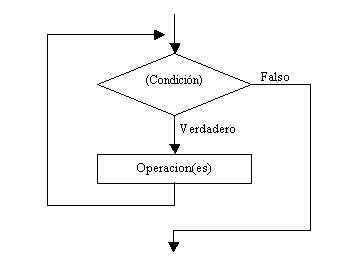
lógicos (and y or) (Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples)

Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:  
- La sentencia o las sentencias que se repiten.  
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las instrucciones.

## Estructura repetitiva while.

Representación gráfica de la estructura while:



x=1

while x<=num:

print(x)

x=x+1

### Problema 3:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

x=0

suma = 0

cant = int(input("Ingrese la cantidad de notas evaluadas: "))

while x<cant:

nota=float(input("Ingrese la nota "))

suma=suma+nota

x=x+1

promedio=suma/cant

print("La suma da")

print(suma)

print("y el promedio da")

print(promedio)

En general, la estructura repetitiva for se usa en aquellas situaciones en las cuales queremos que una variable vaya tomando un valor de una lista definida de valores.

Tenemos primero la palabra clave for y seguidamente el nombre de la variable que almacenará en cada vuelta del for el valor entero que retorna la función range.  
La función range retorna la primera vez el valor 0 y se almacena en x, luego el 1 y así sucesivamente hasta que retorna el valor que le pasamos a range menos uno (es decir en nuestro ejemplo al final retorna un 100)

Existen dos formas de definir comentarios en Python:

* Comentarios de una sola línea, se emplea el caracter #:

#definimos tres contadores

conta1=0

conta2=0

conta3=0

Todo lo que disponemos después del caracter # no se ejecuta

Comentarios de varias líneas:

"""Definimos tres contadores

que se muestran si son distintos a cero"""

conta1=0

conta2=0

conta3=0

Se deben utilizar tres comillas simples o dobles seguidas al principio y al final del comentario.

#definición de una variable entera

cantidad=20

#definición de una variable flotante

altura=1.92

Como vemos el intérprete de Python diferencia una variable flotante de una variable entera por la presencia del caracter punto.

A estos dos tipos de datos fundamentales (int y float) se suma un tipo de dato muy utilizado que son las cadenas de caracteres.

Una cadena de caracteres está compuesta por uno o más caracteres. También podemos iniciar una cadena de caracteres por asignación o ingresarla por teclado.

Inicialización de una cadena por asignación:

#definición e inicio de una cadena de caracteres

dia="lunes"

Igual resultado obtenemos si utilizamos la comilla simple:

#definición e inicio de una cadena de caracteres

dia='lunes'

Para la carga por teclado de una cadena de caracteres utilizamos la función input que retorna una cadena de caracteres:

raw\_input()

Importante

Para analizar cada caracter del string ingresado disponemos una estructura while utilizando un contador llamado x que comienza con el valor cero y se repetirá tantas veces como caracteres tenga la cadena (mediante la función len obtenemos la cantidad de caracteres):

while x<len(mail):

Dentro del ciclo while verificamos cada caracter mediante un if y contamos la cantidad de caracterers "@":

if mail[x]=="@":

cantidad=cantidad+1

Cuando sale del ciclo while procedemos a verificar si el contador tiene almacenado el valor 1 y mostramos el mensaje respectivo:

if cantidad==1:

print("Contiene solo un caracter @ el mail ingresado")

else:

print("Incorrecto")

Los string en Python son inmutables, esto quiere decir que una vez que los inicializamos no podemos modificar su contenido:

nombre="juan"

nombre[0]="m" #esto no se puede

### Métodos propios de las cadenas de caracteres.

Los string tienen una serie de métodos (funciones aplicables solo a los string) que nos facilitan la creación de nuestros programas.

Los primeros tres métodos que veremos se llaman: lower, upper y capitalize.

* upper() : devuelve una cadena de caracteres convertida todos sus caracteres a mayúsculas.
* lower() : devuelve una cadena de caracteres convertida todos sus caracteres a minúsculas.
* capitalize() : devuelve una cadena de caracteres convertida a mayúscula solo su primer caracter y todos los demás a minúsculas.

Si queremos conocer la longitud de un string en Python disponemos de una función llamada ***len*** que retorna la cantidad de caracteres que contiene

Para crear una lista por asignación debemos indicar sus elementos encerrados entre corchetes y separados por coma.

lista1=[10, 5, 3] # lista de enteros

lista2=[1.78, 2.66, 1.55, 89,4] # lista de valores float

lista3=["lunes", "martes", "miercoles"] # lista de string

lista4=["juan", 45, 1.92] # lista con elementos de distinto tipo

Primero definimos una lista por asignación con 5 elementos:

lista=[10,7,3,7,2]

Definimos un acumulador para sumar los elementos de la lista y un contador para indicar que posición de la lista accedemos:

suma=0

x=0

Mediante un ciclo while recorremos y sumamos cada elementos de la lista:

while x<len(lista):

suma=suma+lista[x]

x=x+1

Cuando llamamos a la función print pasando como dato una lista luego se muestra en pantalla todos los elementos de la lista entre corches y separados por coma tal cual como lo definimos:

print("Los elementos de la lista son")

print(lista)

Finalmente mostramos el acumulador:

print("La suma de todos sus elementos es")

print(suma)

Es una actividad muy común la búsqueda del mayor y menor elemento de una lista.  
Es necesario que la lista tenga valores del mismo tipo por ejemplo enteros. Pueden ser de tipo cadenas de caracteres y se busque cual es mayor o menor alfabéticamente, pero no podemos buscar el mayor o menor si la lista tiene enteros y cadenas de caracteres al mismo tiempo.

Primero procedemos a cargar por teclado los 5 valores en la lista:

for x in range(5):

valor=int(input("Ingrese valor:"))

lista.append(valor)

Para identificar el mayor de una lista primero tomamos como referencia el primer elemento, considerando a este en principio como el mayor de la lista:

mayor=lista[0]

Seguidamente disponemos un for que se repita 4 veces esto debido a que no debemos controlar el primer elemento de la lista (recordar que la primer vuelta del for x toma el valor 1, luego el 2 y así sucesivamente hasta el 4):

for x in range(1,5):

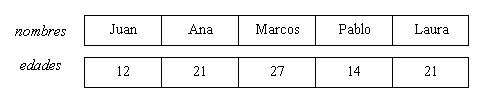
Dentro del for mediante una estructura condicional verificamos si el elemento de la posición x de la lista es mayor al que hemos considerado hasta este momento como mayor:

if lista[x]>mayor:

mayor=lista[x]

Como vemos en las dos líneas anteriores si el if se verifica verdadero actualizamos la variable mayor con el elemento de la lista que estamos analizando.

Podemos decir que dos listas son paralelas cuando hay una relación entre las componentes de igual subíndice (misma posición) de una lista y otra.



Si tenemos dos listas que ya hemos inicializado con 5 elementos cada una. En una se almacenan los nombres de personas en la otra las edades de dichas personas.  
Decimos que la lista nombres es paralela a la lista edades si en la componente 0 de cada lista se almacena información relacionada a una persona (Juan - 12 años)  
Es decir hay una relación entre cada componente de las dos listas.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en las estructuras de datos.

Para imprimir los nombres de la personas mayores de edad procedemos a analizar dentro de un for y mediante un if cada una de las edades almacenadas en la lista "edades", en el caso que su valor sea mayor o igual a 18 mostramos el elemento de la lista nombres de la misma posición:

for x in range(5):

if edades[x]>=18:

print(nombres[x])

Otro algoritmo muy común que debe conocer y entender un programador es el ordenamiento de una lista de datos.

El ordenamiento de una lista se logra intercambiando las componentes de manera que:  
lista[0] <= lista[1] <= lista[2] etc.

El contenido de la componente lista[0] sea menor o igual al contenido de la componente lista[1] y así sucesivamente.  
Si se cumple lo dicho anteriormente decimos que la lista está ordenado de menor a mayor. Igualmente podemos ordenar una lista de mayor a menor.

Tengamos en cuenta que la estructura de datos lista en Python es mutable, eso significa que podemos modificar sus elementos por otros.

Se puede ordenar tanto listas con componentes de tipo int, float como cadena de caracteres. En este último caso el ordenamiento es alfabético.

Si hay 5 componentes hay que hacer 4 comparaciones, por eso el for se repite 4 veces.  
Generalizando: si la lista tiene N componentes hay que hacer N-1 comparaciones.

Cuando se tienen listas paralelas y se ordenan los elementos de una de ellas hay que tener la precaución de intercambiar los elementos de las listas paralelas.

Lo nuevo se presenta cuando ordenamos la lista de notas de mayor a menor. La condición dentro de los dos ciclos repetitivos depende de la lista notas, pero en el caso que se verifique verdadera intercambiamos tanto los elementos de la lista notas como el de la lista alumnos con el fin que continúen paralelas:

for k in range(4):

for x in range(4-k):

if notas[x]<notas[x+1]:

aux1=notas[x]

notas[x]=notas[x+1]

notas[x+1]=aux1

aux2=alumnos[x]

alumnos[x]=alumnos[x+1]

alumnos[x+1]=aux2

Imprimimos las dos listas:

for x in range(5):

print(alumnos[x],notas[x])

Para definir y crear por asignación una lista de listas tenemos:

lista=[[1,2,3], [4,5,6], [7,8,9], [10,11,12]]

Cuando pasamos a la función print el primer elemento de la lista:

print(lista[0])

Nos muestra la lista contenida en la primer componente de la lista principal:

[1, 2, 3]

Si queremos acceder al primer entero almacenado en la lista contenida en la primer componente de la lista principal:

print(lista[0][0])

Nos muestra:

1

Para acceder mediante un for a todos los elementos de la lista contenida en la primer componente de la lista principal debemos codificar:

for x in range(len(lista[0])):

print(lista[0][x])

Recordemos que la función len retorna la cantidad de elementos que contiene una lista. En este caso le pasamos como parámetro lista[0] que hace referencia a la primer componente de la lista principal.

El resultado de len(lista[0]) es un 3 que es la cantidad de elementos que tiene la lista contenida en la primer componente de la lista principal.

Cada ciclo del for accedemos a: lista[0][0] cuando x vale 0, lista[0][1] cuando x vale 1 y lista[0][2] cuando x vale 2.

Mediante este ciclo podemos acceder a cada elemento y procesarlo.

Por último con el ciclo anidado k podemos acceder a cada elemento de la lista principal y mediante el for interno acceder a cada elemento entero de las listas contenidas en la lista principal:

for k in range(len(lista)):

for x in range(len(lista[k])):

print(lista[k][x])

Otra característica fundamental de las listas en Python es que podemos eliminar cualquiera de sus componentes llamando al método pop e indicando la posición del elemento a borrar:

lista.pop(0)

Como es posible que se eliminen 0, 1, 2 etc. elementos de la lista utilizamos un ciclo while (no podemos usar un for, ya que el contador del mismo no indicará correctamente el subindice a analizar)

Llevamos un contador llamado "posicion" que nos indica que elemento de la lista estamos verificando en el if, en el caso que se debe borrar llamamos al método pop pasando el contador y no incrementamos en uno el contador "posicion" ya que los elementos de la derecha se desplazan una posición a izquierda.  
En el caso que no se debe borrar se incrementa en uno el contador "posicion" para analizar el siguiente elemento de la lista en la próxima vuelta del ciclo:

posicion=0

while posicion<len(lista):

if lista[posicion]==5:

lista.pop(posicion)

else:

posicion=posicion+1

### Acotación: otra forma de eliminar elementos de una lista

Para eliminar elementos de una lista también es empleada la función "del" pasando como parámetro la referencia de la componente a eliminar:

lista=[10, 20, 30, 40, 50]

print(lista)

del(lista[0])

del(lista[1])

del(lista[2])

print(lista) # 20 40

**Funciones**

La programación estructurada busca dividir o descomponer un problema complejo en pequeños problemas. La solución de cada uno de esos pequeños problemas nos trae la solución del problema complejo.

En Python el planteo de esas pequeñas soluciones al problema complejo se hace dividiendo el programa en funciones.

Una función es un conjunto de instrucciones en Python que resuelven un problema específico.

La forma de organizar nuestro programa cambia en forma radical.

El programa ahora no empieza a ejecutarse en la línea 1.

El programa principal comienza a ejecutarse luego del comentario "programa principal":

# programa principal

presentacion()

carga\_suma()

finalizacion()

La sintaxis para declarar una función es mediante la palabra clave def seguida por el nombre de la función (el nombre de la función no puede tener espacios en blanco, comenzar con un número y el único caracter especial permitido es el \_ )

Ahora veremos que una función puede tener parámetros para recibir datos. Los parámetros nos permiten comunicarle algo a la función y la hace más flexible.

Ahora para resolver este pequeño problema hemos planteado una función llamada mostrar\_mensaje que recibe como parámetro un string (cadena de caracteres) y lo muestra en pantalla.

Los parámetros van seguidos del nombre de la función encerrados entre paréntesis (y en el caso de tener más de un parámetro los mismos deben ir separados por coma):

def mostrar\_mensaje(mensaje):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(mensaje)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

Un parámetro podemos imaginarlo como una variable que solo se puede utilizar dentro de la función.

Los parámetros son la forma para que una función reciba datos para ser procesados. Ahora veremos otra característica de las funciones que es la de devolver un dato a quien invocó la función (recordemos que una función la podemos llamar desde el bloque principal de nuestro programa o desde otra función que desarrollemos)

Aparece una nueva palabra clave en Python para indicar el valor devuelto por la función: return

La función retornar\_superficie recibe un parámetro llamado lado, definimos una variable local llamada sup donde almacenamos el producto del parámetro lado por sí mismo.

Hay un cambio importante cuando llamamos o invocamos a una función que devuelve un dato:

superficie=retornar\_superficie(va)

Es decir el valor devuelto por la función retornar\_superficie se almacena en la variable superficie.

Es un error lógico llamar a la función retornar\_superficie y no asignar el valor a una variable:

retornar\_superficie(va)

La función retornar\_superficie devuelve un entero y se lo pasamos a la función print para que lo muestre.

Cuando una función encuentra la palabra return no sigue ejecutando el resto de la función sino que sale a la línea del programa desde donde llamamos a dicha función.

Para no complicarla:

def mayor(lista):

may=lista[0]

for x in range(1,len(lista)):

if lista[x]>may:

may=lista[x]

return may