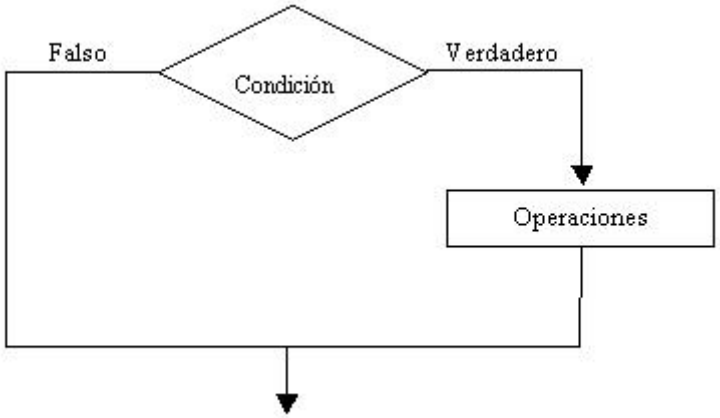
Estructuras condicionales

Fuente: https://www.tutorialesprogramacionya.com

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.
En nuestra vida diaria se nos presentan situaciones donde debemos decidir.
¿Elijo la carrera A o la carrera B?
¿Me pongo este pantalón?
Para ir al trabajo, ¿Elijo el camino A o el camino B?
Al cursar una carrera, ¿Elijo el turno mañana, tarde o noche?
Es común que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.
Representación gráfica de una estructura condicional a utilizar en un diagrama de flujo:

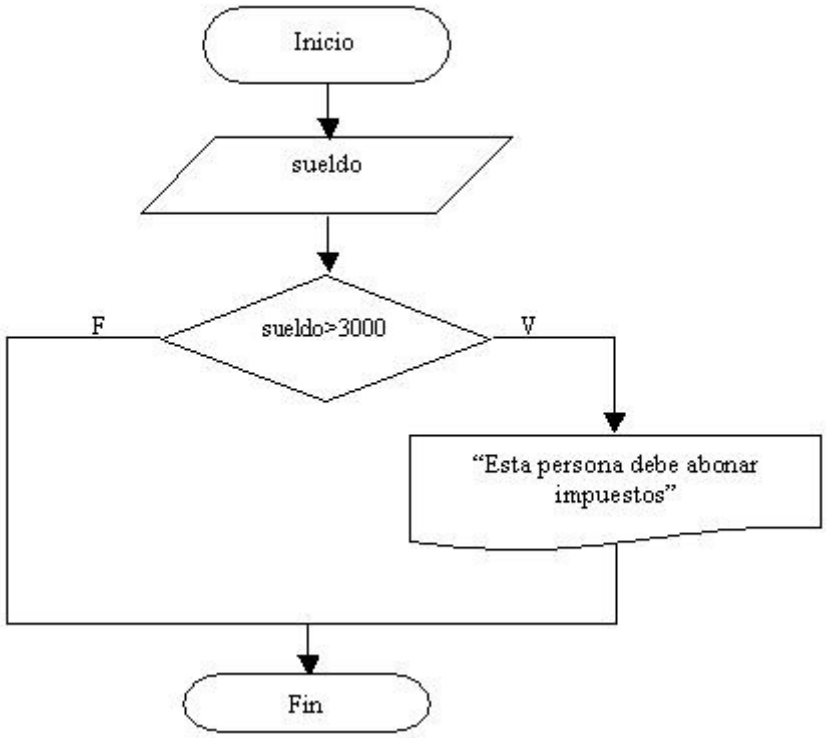


Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda donde no hay ninguna actividad.
Se trata de una estructura CONDICIONAL SIMPLE porque por el camino del verdadero hay actividades y por el camino del falso no hay actividades.
Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

Problema:

Ingresar el sueldo de una persona, si supera los 3000 dolares mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo:

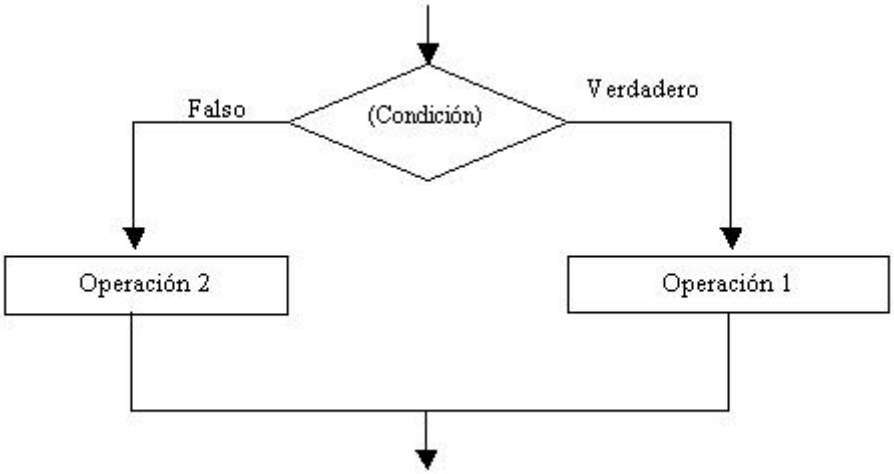


Podemos observar lo siguiente: Siempre se hace la carga del sueldo, pero si el sueldo que ingresamos supera 3000 dólares se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 3000 o menos no aparece nada por pantalla.

La palabra clave "if" indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición y finalizamos la línea con el caracter dos puntos.
La actividad dentro del if se indenta generalmente a 4 espacios.
Todo lo que se encuentre en la rama del verdadero del if se debe disponer a 4 espacios corrido a derecha.
La indentación es una característica obligatoria del lenguaje Python para codificación de las estructuras condicionales, de esta forma el intérprete de Python puede identificar donde finalizan las instrucciones contenidas en la rama verdadera del if.

Estructura condicional compuesta

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir, tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.
Representación gráfica:

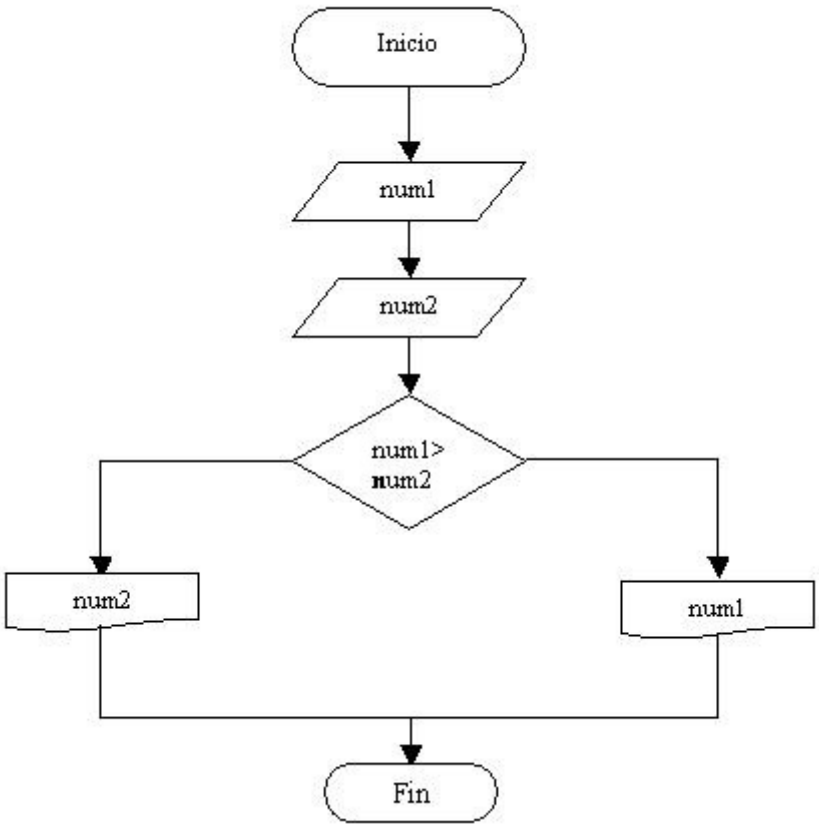


En una estructura condicional compuesta tenemos actividades tanto por la rama del verdadero como por la rama del falso.

Problema:

Realizar un programa que solicite ingresar dos números distintos y muestre por pantalla el mayor de ellos.

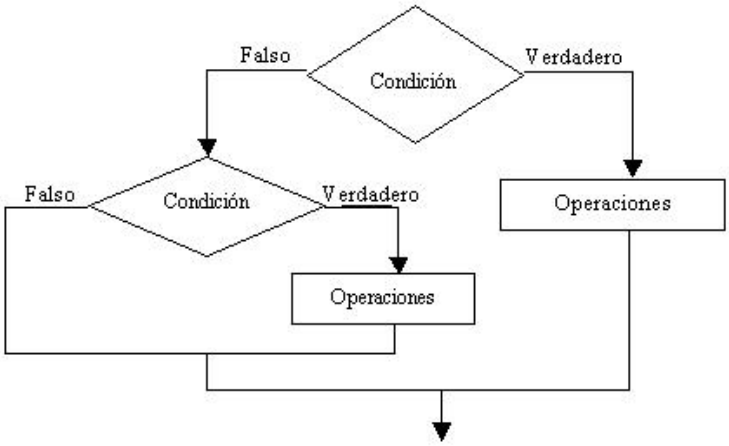
Diagrama de flujo:



Se hace la entrada de num1 y num2 por teclado. Para saber cuál variable tiene un valor mayor preguntamos si el contenido de num1 es mayor (>) que el contenido de num2, si la respuesta es verdadera vamos por la rama de la derecha e imprimimos num1, en caso que la condición sea falsa vamos por la rama de la izquierda (Falsa) e imprimimos num2.
Como podemos observar, nunca se imprimen num1 y num2 simultáneamente.
Estamos en presencia de una ESTRUCTURA CONDICIONAL COMPUESTA ya que tenemos actividades por la rama del verdadero y del falso.

Estructuras condicionales anidadas

Estamos en presencia de una estructura condicional anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.



El diagrama de flujo que se presenta contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primera estructura.
Es común que se presenten estructuras condicionales anidadas aún más complejas.

Problema:

Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:
Si el promedio es >=7 mostrar "Promocionado".
Si el promedio es >=4 y <7 mostrar "Regular".
Si el promedio es <4 mostrar "Reprobado".

Ejemplo con condicionales anidados y elif:

```
if prom>=7:
    print("Promocionado")
else:
    if prom>=4 and prom<7:
        print("Regular")
    else:
        print("Reprobado")
```

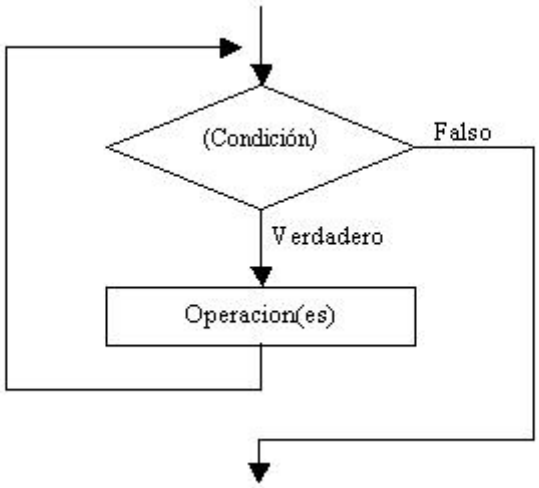
```
if prom>=7:
    print("Promocionado")
elif prom>=4 and prom<7:
    print("Regular")
else:
    print("Reprobado")
```

Estructura repetitivas

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



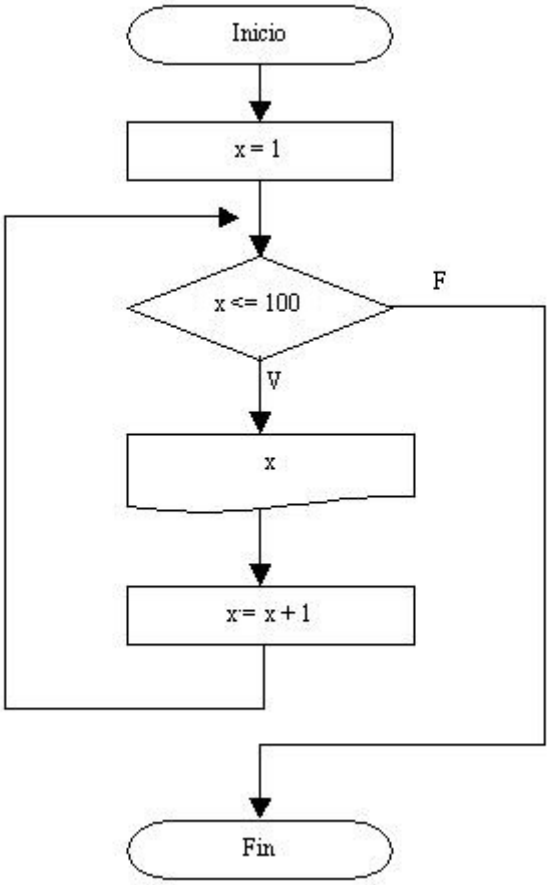
No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)
Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.
A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.
En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.
El bloque se repite MIENTRAS la condición sea Verdadera.
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación lógico, nunca finalizará el programa.

Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.
Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Iniciamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

Diagrama de flujo:

Si continuamos con el diagrama veríamos que es casi interminable.
Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en Python muy largo.
Ahora veamos la solución empleando una estructura repetitiva while:



Es muy importante analizar este diagrama:
La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición (x <= 100), se lee MIENTRAS la variable x sea menor o igual a 100.

Al ejecutarse la condición retorna VERDADERO porque el contenido de x (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene una salida y una operación.
Se imprime el contenido de x, y seguidamente se incrementa la variable x en uno.

La operación x=x+1 se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.
Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.

Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.
Lo más difícil es la definición de la condición de la estructura while y que bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición x >=100 (si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.
Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer un seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

x
1
2
3
4
.
.
100
101 Cuando x vale 101 la condición de la estructura repetitiva retorna falso, en este caso finaliza el diagrama.

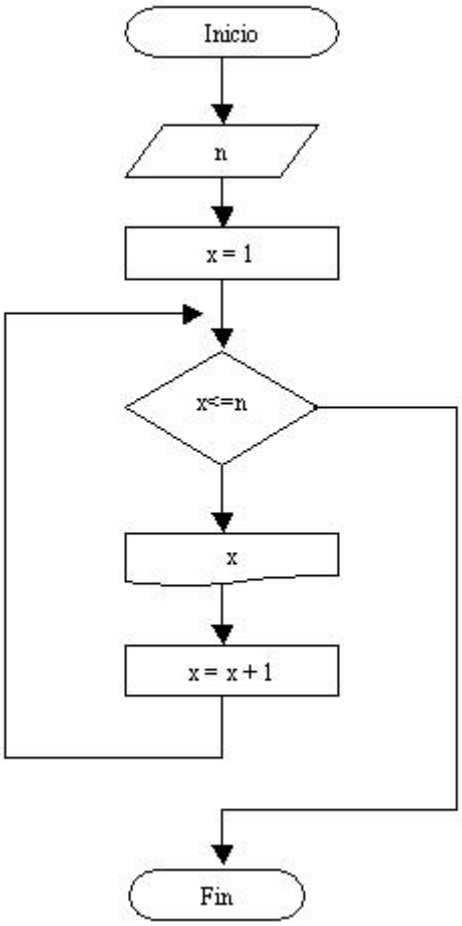
Importante: Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso la primera vez.
La variable x debe estar inicializada con algún valor antes que se ejecute la operación x=x+1 en caso de no estar inicializada aparece un error de compilación.

Recordemos que un problema no estará 100% solucionado si no codificamos el programa en nuestro caso en Python que muestre los resultados buscados.
Es importante notar que seguido de la palabra clave while disponemos la condición y finalmente los dos puntos. Todo el código contenido en la estructura repetitiva debe estar indentado (normalmente a cuatro espacios)

Problema 2:

Codificar un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.
Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.
Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en Python de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

Diagrama de flujo:



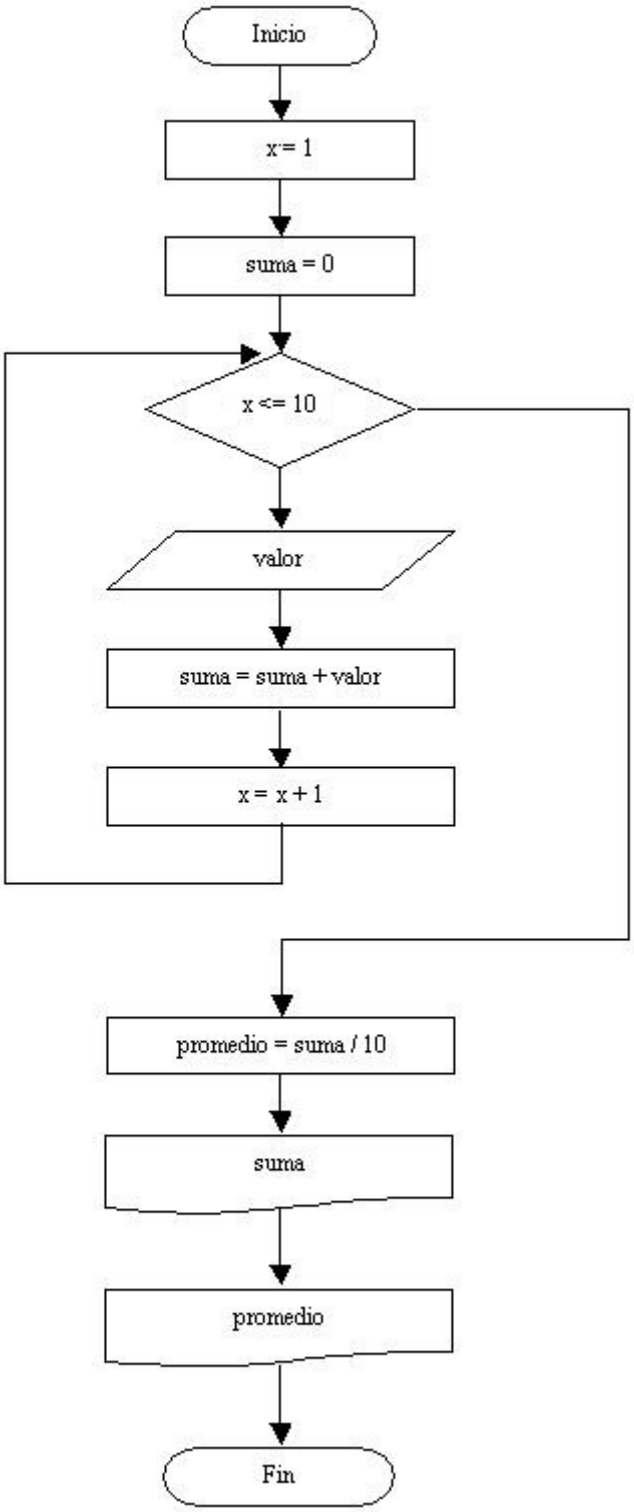
Podemos observar que se ingresa por teclado la variable n. El operador puede cargar cualquier valor.
Si el usuario carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es “Mientras x<=n ”, es decir “mientras x sea menor o igual a 10”; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

Los nombres de las variables n y x pueden ser palabras o letras (como en este caso)
La variable x recibe el nombre de CONTADOR. Un contador es un tipo especial de variable que se incrementa o disminuye con valores constantes durante la ejecución del programa.
El contador x nos indica en cada momento la cantidad de valores impresos en pantalla.

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.

Diagrama de flujo:



En este problema, a semejanza de los anteriores, tenemos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while.
También aparece el concepto de ACUMULADOR (un acumulador es un tipo especial de variable que se incrementa o disminuye con valores variables durante la ejecución del programa)
Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

El promedio se calcula al salir de la estructura repetitiva (es decir primero sumamos los 10 valores ingresados y luego los dividimos por 10)
Hay que tener en cuenta que cuando en la variable valor se carga el primer valor (en este ejemplo 5) al cargarse el segundo valor (16) el valor anterior 5 se pierde, por ello la necesidad de ir almacenando en la variable suma los valores ingresados.

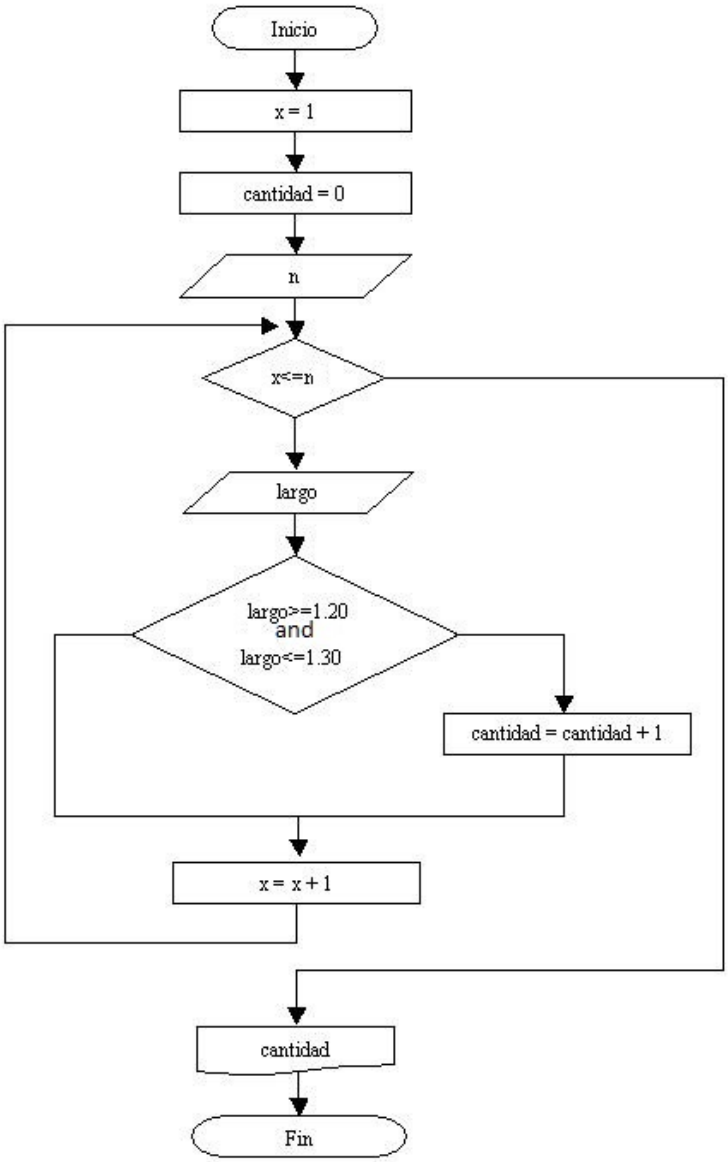
El resultado del promedio es un valor real es decir con coma. Si queremos que el resultado de la división solo retorne la parte entera del promedio debemos utilizar el operador //:
promedio=suma//10

El intérprete de Python sabe que el promedio se calcula al finalizar el while ya que se encuentra codificado en la columna 1. Las tres instrucciones contenidas en el while están indentadas.

Problema 4:

Una planta que fabrica perfiles de hierro posee un lote de **n** piezas.
Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1.20 y 1.30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

Diagrama de flujo:



Podemos observar que dentro de una estructura repetitiva puede haber estructuras condicionales (inclusive puede haber otras estructuras repetitivas que veremos más adelante)
En este problema hay que cargar inicialmente la cantidad de piezas a ingresar (n), seguidamente se cargan n valores de largos de piezas.
Cada vez que ingresamos un largo de pieza (largo) verificamos si es una medida correcta (debe estar entre 1.20 y 1.30 el largo para que sea correcta), en caso de ser correcta la CONTAMOS (incrementamos la variable cantidad en 1)
Al contador cantidad lo inicializamos en cero porque inicialmente no se ha cargado ningún largo de pieza.
Cuando salimos de la estructura repetitiva porque se han cargado n largos de piezas mostramos por pantalla el contador cantidad (que representa la cantidad de piezas aptas)
En este problema tenemos dos CONTADORES:

Estructura repetitiva for

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while, pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones que tenemos que recorrer una lista de datos.

Problema 1:

Realizar un programa que imprima en pantalla los números del 0 al 100. Este problema lo podemos resolver perfectamente con el ciclo while pero en esta situación lo resolveremos empleando el for.

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)
Tengamos en cuenta que este mismo problema resuelto con la estructura while debería ser:

Problema 2:

Realizar un programa que imprima en pantalla los números del 20 al 30.

La función range puede tener dos parámetros, el primero indica el valor inicial que tomará la variable x, cada vuelta del for la variable x toma el valor siguiente hasta llegar al valor indicado por el segundo parámetro de la función range menos uno.

Problema 3:

Imprimir todos los números impares que hay entre 1 y 100.

La función range puede tener también tres parámetros, el primero indica el valor inicial que tomará la variable x, el segundo parámetro el valor final (que no se incluye) y el tercer parámetro indica cuánto se incrementa cada vuelta x.
En nuestro ejemplo la primer vuelta del for x recibe el valor 1, la segunda vuelta toma el valor 3 y así sucesivamente hasta el valor 99.

Problema 4:

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. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for para repetir la carga de los diez valores por teclado.

Como vemos la variable f del for solo sirve para iterar(repetir) las diez veces el bloque contenido en el for.
El resultado hubiese sido el mismo si llamamos a la función range con los valores:
range(1,11)

Problema 5:

Escribir un programa que solicite por teclado 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

Nuevamente utilizamos el for ya que sabemos que el ciclo repetitivo debe repetirse 10 veces. Recordemos que si utilizamos el while debemos llevar un contador y recordar de incrementarlo en cada vuelta.

Problema 6:

Escribir un programa que lea 10 números enteros y luego muestre cuántos valores ingresados fueron múltiplos de 3 y cuántos de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y de 5 a la vez.

Operaciones con Strings, Listas, Tuplas y Diccionarios

Fuente: https://python-para-impacientes.blogspot.com

Operaciones con cadenas y listas

```
cadena1 = 'tengo una yama que Yama se llama'  # declara variable
lista1 = ['pera', 'manzana', 'naranja', 'uva']          # declara lista

longitud = len(cadena1)                                 # 32, devuelve longitud de la cadena
elem = len(lista1)                                      # 4, devuelve nº elementos de la lista

cuenta = cadena1.count('yama')                          # 1, cuenta apariciones de 'yama'
print(cadena1.find('yama'))                             # 10, devuelve posición de búsqueda

cadena2 = cadena1.join('***')                           # inserta cadena1 entre caracteres
lista1 = cadena1.split(' ')                             # divide cadena por separador → lista
tupla1 = cadena1.partition(' ')                         # divide cadena por separador → tupla

cadena2 = cadena1.replace('yama','cabra',1)             # busca/sustituye términos
numero = 3.14                                           # asigna número con decimales

cadena3 = str(numero)                    # convierte número a cadena

if cadena1.startswith("tengo"):                  # evalúa si comienza por “tengo” (Falta definir el
bloque)
if cadena1.endswith("llama"):            # evalúa si termina por “llama” (Falta definir el bloque)
if cadena1.find("llama") != -1:          # evalúa si contiene “llama” (Falta definir el bloque)

cadena4 = 'Python'                                      # asigna una cadena a una variable
print(cadena4[0:4])                                     # muestra desde la posición 0 a 4: "Pyth"
print(cadena4[1])                                       # muestra la posición 1: "y"
print(cadena4[:3] + '-' + cadena4[3:])                  # muestra "Pyt-hon"
```

```
print(cadena4[:-3])                    # muestra todo menos las tres últimas: "Pyt"

# declara variable con cadena alfanumérica
cadena5 = "  abc;123  "

# suprime caracteres en blanco por la derecha
print(cadena5.rstrip())                # "  abc;123"

# suprime caracteres en blanco por la izquierda
print(cadena5.lstrip())                # "abc;123  "

# suprime caracteres en blanco por derecha e izquierda
print(cadena5.strip())                 # "abc;123"

cadena6 = "Mar"                        # declara una variable
print(cadena6.upper())                 # convierte a mayúsculas: "MAR"
print(cadena6.lower())                 # convierte a minúsculas: "mar"
```

Operaciones con listas y tuplas

```
lista1 = ['uno', 2, True]              # declara una lista heterogénea
lista2 = [1, 2, 3, 4, 5]               # declara lista numérica (homogénea)
lista3 = ['nombre', ['ap1', 'ap2']]    # declara lista dentro de otra
lista4 = [54,45,44,22,0,2,99]          # declara una lista numérica

print(lista1)                          # ['uno', 2, True], muestra toda la lista
print(lista1[0])                       # uno, muestra el primer elemento de la lista
print(lista2[-1])                      # 5, muestra el último elemento de la lista
print(lista3[1][0])                    # ap1, primer elemento de la lista anidada
print(lista2[0:3:1])                   # [1,2,3], responde al patrón inicio:fin:paso
print(lista2[::-1])                    # devuelve la lista ordenada al revés

lista1[2] = False                      # cambia el valor de un elemento de la lista
lista2[-2] = lista2[-2] + 1            # 4+1 → 5 – cambia valor de elemento

lista2.pop(0)                          # borra elemento indicado o último si no indica
lista1.remove('uno')                   # borra el primer elemento que coincida
del lista2[1]                          # borra el segundo elemento (por índice)

lista2 = lista2 + [6]                  # añade elemento al final de la lista
lista2.append(7)                       # añade un elemento al final con append()
lista2.extend([8, 9])                  # extiende lista con otra por el final
lista1.insert(1, 'dos')                # inserta nuevo elemento en posición

del lista2[0:3]                        # borra los elementos desde:hasta
lista2[:] = []                         # borra todos los elementos de la lista

print(lista1.count(2))                 # cuenta el nº de veces que aparece 2
print(lista1.index("dos"))             # busca posición que ocupa elemento

lista4.sort()                          # ordena la lista
lista4.sort(reverse=True)              # ordena la lista en orden inverso
```

```
lista5 = sorted(lista4)                 # ordena lista destino

tupla1 = (1, 2, 3)                      # declara tupla (se usan paréntesis)...
tupla2 = 1, 2, 3                        # ...aunque pueden declararse sin paréntesis
tupla3 = (100,)                         # con un elemento hay terminar con “,”
tupla4 = tupla1, 4, 5, 6                # anida tuplas
tupla5 = ()                             # declara una tupla vacía
tupla6 = tuple([1, 2, 3, 4, 5])         # Convierte una lista en una tupla
tupla2[0:3]                             # (1, 2, 3), accede a los valores desde:hasta
```

Operaciones con diccionarios

```
dic1 = {'Lorca':'Escritor', 'Goya':'Pintor'}    # declara diccionario
print(dic1)                                     # {'Goya': 'Pintor', 'Lorca': 'Escritor'}

dic2 = dict((('mesa',5), ('silla',10)))         # declara a partir de tupla
dic3 = dict(ALM=5, CAD=10)                      # declara a partir de cadenas simples

print(dic1['Lorca'])                            # escritor, acceso a un valor por clave
print(dic1.get('Gala', 'no existe'))            # acceso a un valor por clave

if 'Lorca' in dic1: print('está')               # comprueba si existe una clave

print(dic1.items())                             # obtiene una lista de tuplas clave:valor
print(dic1.keys())                              # obtiene una lista de las claves
print(dic1.values())                            # obtiene una lista de los valores

dic1['Lorca'] = 'Poeta'                         # añade un nuevo par clave:valor
dic1['Amenabar'] = 'Cineasta'                   # añade un nuevo par clave:valor
dic1.update({'Carreras' : 'Tenor'})             # añadir con update()
del dic1['Amenabar']                            # borra un par clave:valor
print(dic1.pop('Amenabar', 'no está'))          # borra par clave:valor
```

Recorrer secuencias y diccionarios con for...in

```
artistas = {'Lorca':'Escritor', 'Goya':'Pintor'}        # diccionario
paises = ['Chile','España','Francia','Portugal']        # declara lista
capitales = ['Santiago','Madrid','París','Lisboa']      # declara lista
for c, v in artistas.items(): print(c,':',v)            # recorre diccionario
for i, c in enumerate(paises): print(i,':',c)           # recorre lista
for p, c in zip(paises, capitales): print(c,' ',p)      # recorre listas
for p in reversed(paises): print(p,)                    # recorre en orden inverso
for c in sorted(paises): print(c,)                      # recorre secuencia ordenada
```

Operadores para secuencias: in, not in

```
cadena = 'Python'                                       # asigna cadena a variable
lista = [1, 2, 3, 4, 5]                                 # declara lista
if 'y' in cadena: print('“y” está en “Python”')         # contiene
if 6 not in lista: print('6 no está en la lista')       # no contiene
```

```
if 'abcabc' == 'abc' * 2: print('Son iguales')                # son iguales
```