

Python. Contenidos Generales:

Introducción. Entorno de trabajo. Hola mundo. Tipo de datos. Variables. Tipos de operadores. print. input.

Estructuras control. Condicionales. Repetitivas.

Tipo de datos compuestos. Listas. Cadenas de caracteres.

Funciones. Concepto. Parámetros y Argumentos. Valores de retorno. Parámetros mutables e inmutables. Llamada a función. Docstring.

POO. Paradigmas. Clases y Objetos.

Mensajes y Métodos. Colaboración entre clases. Variables de clase. Método especial `__str__`.

Encapsulamiento. Getters y Setters en Python.

Herencia. Polimorfismo. Herencia Simple, Herencia Múltiple, Clases Abstractas, Diagrama de Clases. Composición/Agregación.

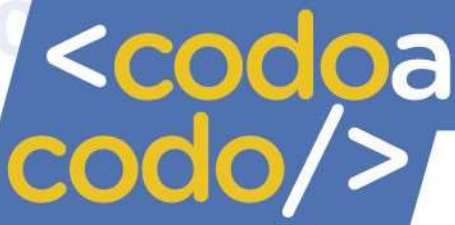
Manejo de excepciones. Módulos y packages. Librerías.

Introducción a Python

Características del lenguaje

Python es un lenguaje de programación de alto nivel cuya máxima es la legibilidad del código. Las principales características de Python son las siguientes:

Es multiparadigma, ya que soporta la programación imperativa, programación orientada a objetos y funcional.



Es multiplataforma: Se puede encontrar un intérprete de Python para los principales sistemas operativos: Windows, Linux y Mac OS. Además, se puede reutilizar el mismo código en cada una de las plataformas.

Es dinámicamente tipado: Es decir, el tipo de las variables se decide en tiempo de ejecución.

Es fuertemente tipado: No se puede usar una variable en un contexto fuera de su tipo. Si se quisiera, habría que hacer una conversión de tipos.

Es interpretado: El código no se compila a lenguaje máquina.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Instalación de Python







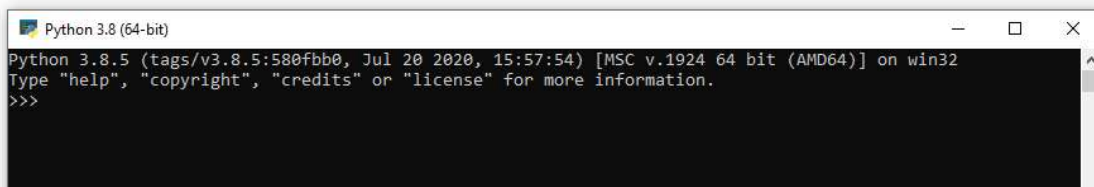
Agencia de
Aprendizaje
a lo largo
de la vida

Link de descarga: <https://www.python.org/downloads/>

Video recomendado “Cómo instalar Python y usar la herramienta IDLE para empezar a programar”:
https://youtu.be/F9eM_VoKGJQ

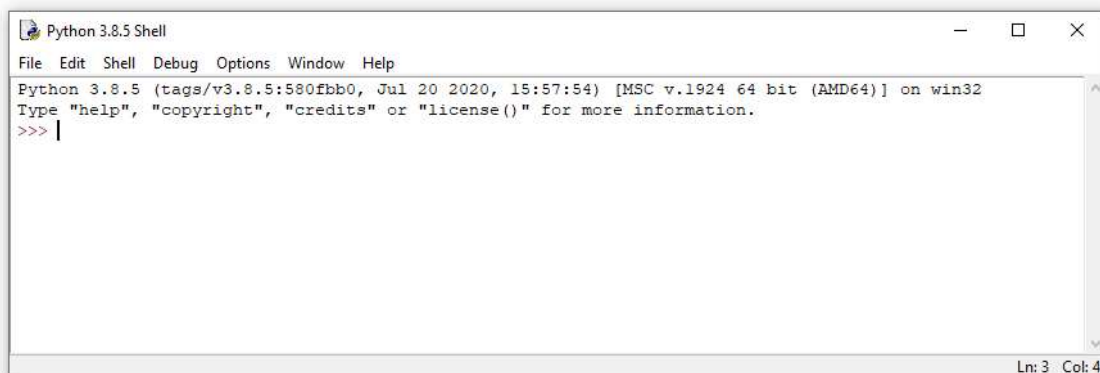
Python en Microsoft Windows

Nombre	Fecha de modificación	Tipo	Tamaño
 IDLE (Python 3.8 64-bit)	6/7/2020 11:47	Acceso directo	3 KB
 Python 3.8 (64-bit)	6/7/2020 11:46	Acceso directo	2 KB
 Python 3.8 Manuals (64-bit)	6/7/2020 11:47	Acceso directo	1 KB
 Python 3.8 Module Docs (64-bit)	6/7/2020 11:47	Acceso directo	3 KB



```
Python 3.8 (64-bit)
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Integrated Development Environment (IDLE)



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```




Python en MacOS

debe



MacTMOS

Python ya viene instalado en las versiones de MacOS, para abrirlo se acceder al Terminal (Aplicaciones -> Utilidades -> Terminal) y escribir **Python**. Ello abrirá la línea de comandos de Python.

Python en GNU/Linux

Una
con la

LinuxTM



vez descargada e instalada la versión que corresponda distribución de Linux, por lo general el directorio de instalación se encontrará en

/usr/

local/bin/Python3.8

En la Terminal de su distribución deberá tipear **Python3.8** para iniciar la interfaz de línea de comandos.

Entornos de desarrollo

<https://www.jetbrains.com/es-es/pycharm/download/>



PyCharm

Agencia de
Aprendizaje
a lo largo
de la vida

<https://www.eclipse.org/downloads/>



<http://www.aptana.com/>



aptana

<https://code.visualstudio.com/download>

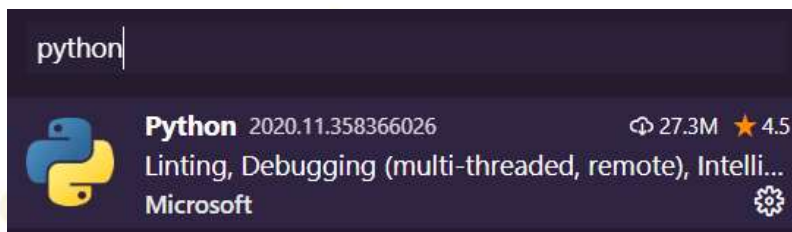


Visual Studio Code

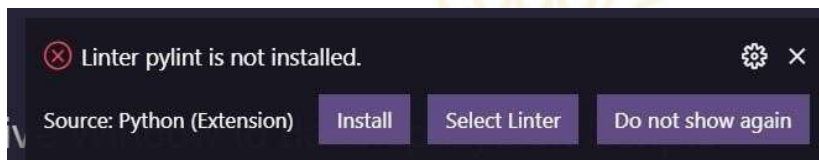
Pasos para instalar y utilizar Python en VS Code

1) Descargar Python de <https://www.python.org/downloads/> e instalar tildando la opción “Agregar Python al PATH”. Al finalizar tocar en “Disable path length limit”.

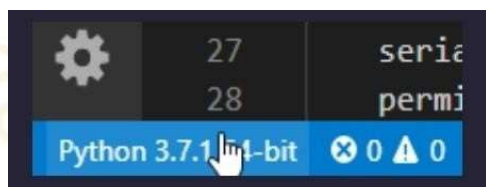
2) Descargar la extensión para VS Code.



3) Al crear un archivo .py es probable que muestre el siguiente mensaje, instalar.



4) Seleccionar el intérprete en la barra de status. (probar primero correr un programa, muchas veces no es necesario).



Hola Mundo!

Nuestro primer programa en Python

FUNCIÓN
PARÁMETROS
RESULTADO

```
>>> print("Hola, mundo!")  
Hola, mundo!  
>>> _
```

Estructura de un programa en Python

CODIGO REUTILIZADO

```
import weather  
import math
```

DEFINICIÓN DE UNA FUNCIÓN

```
def convert(TheStream):  
    global temp, celsius, Cstream  
    for temp in TheStream:  
        celsius = round((temp - 32) / 1.8)  
        Cstream.append(celsius)
```

PROGRAMA PRINCIPAL

```
Cstream = []  
Fstream = weather.get_forecasts('Blacksburg, VA')  
print(Fstream)  
convert(Fstream)  
print(Cstream)
```

Expresiones y sentencias en Python

Expresión

Una expresión es una unidad de código que devuelve un valor y está formada por una combinación de operandos (variables y literales) y operadores. Los siguientes son ejemplos de expresiones (cada línea es una expresión diferente):

5 + 2 # Suma del número 5 y el número 2

`a < 10 # Compara si el valor de la variable a es menor que 10`
`b is None # Compara si la identidad de la variable b es None`
`3 * (200 - c) # Resta a 200 el valor de c y lo multiplica por 3`

Sentencia

Por su parte, una sentencia o declaración es una instrucción que define una acción. Una sentencia puede estar formada por una o varias expresiones, aunque no siempre es así. En definitiva, las sentencias son las instrucciones que componen nuestro programa y determinan su comportamiento.

Ejemplos de sentencias son la asignación `=` o las instrucciones `if`, `if ... else ...`, `for` o `while` entre otras.

Una sentencia está delimitada por el carácter Enter (`\n`).

Sentencias de más de una línea

Normalmente, las sentencias ocupan una sola línea. Por ejemplo: `a = 2 + 3 # Asigna a la variable <a> el resultado de 2 + 3`

Sin embargo, aquellas sentencias que son muy largas pueden ocupar más de una línea (la guía de estilo PEP 8 <https://www.python.org/dev/peps/pep-0008/>, recomienda una longitud de línea máxima de 72 caracteres).

Para dividir una sentencia en varias líneas se utiliza el carácter \. Por ejemplo: `a = 2 + 3 + 5 + \`

```
7 + 9 + 4 + \
```

```
6
```

Además de la separación explícita (la que se realiza con el carácter \), en Python la continuación de línea es implícita siempre y cuando la expresión vaya dentro de los caracteres (), [] y {}.

Por ejemplo, podemos inicializar una lista del siguiente modo: `a = [1, 2, 7, 3, 8, 4,`

```
9]
```

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Bloques de código (Indentación)

Lo último que veremos sobre sentencias en esta introducción a Python es cómo se pueden agrupar en bloques de código.

Un bloque de código es un grupo de sentencias relacionadas bien delimitadas. A diferencia de otros lenguajes como JAVA o C, en los que se usan los caracteres {} para definir un bloque de código, en Python se usa la indentación o sangrado.

El sangrado o indentación consiste en mover un bloque de texto hacia la derecha insertando espacios o tabuladores al principio de la línea, dejando un margen a la izquierda.

Un bloque comienza con un nuevo sangrado y acaba con la primera línea cuyo sangrado sea menor. De nuevo, la guía de estilo de Python recomienda usar los espacios en lugar de las tabulaciones para realizar el sangrado. Yo suelo utilizar 4 espacios.

Veamos todo esto con un ejemplo:

```
1 def suma_numeros(numeros): # Bloque 1
2     suma = 0                # Bloque 2
3     for n in numeros:       # Bloque 2
4         suma += n           # Bloque 3
5         print(suma)         # Bloque 3
6     return suma             # Bloque 2
```

decía

Como te
en la sección
anterior, no

hace falta todavía que entiendas lo que hace el ejemplo. Simplemente debes comprender que en la línea 1 se define la función suma_numeros. El cuerpo de esta función está definido por el grupo de sentencias que pertenecen al bloque 2 y 3. A su vez, la sentencia for define las acciones a realizar dentro de la misma en el conjunto de sentencias que pertenecen al bloque 3.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Convenciones de nombres y palabras reservadas

Convenciones de nombres en Python

A la hora de nombrar una variable, una función, un módulo, una clase, etc. en Python, siempre se siguen las siguientes reglas y recomendaciones:

Un identificador puede ser cualquier combinación de letras (mayúsculas y minúsculas), números y el carácter guión bajo (_).

Un identificador no puede comenzar por un número.

A excepción de los nombres de clases, es una convención que todos los identificadores se escriban en minúsculas, separando las palabras con el guión bajo. Ejemplos: contador, suma_enteros.

Es una convención que los nombres de clases sigan la notación Camel Case, es decir, todas las letras en minúscula a excepción del primer carácter de cada palabra, que se escribe en mayúscula. Ejemplos: Coche, VehiculoMotorizado.

No se pueden usar como identificadores las palabras reservadas.

Como recomendación, usa identificadores que sean expresivos. Por ejemplo, contador es mejor que simplemente c.

Python diferencia entre mayúsculas y minúsculas, de manera que variable_1 y Variable_1 son dos identificadores totalmente diferentes.

Palabras reservadas de Python

Python tiene una serie de palabras clave reservadas, por tanto, no pueden usarse como nombres de variables, funciones, etc.

Estas palabras clave se utilizan para definir la sintaxis y estructura del lenguaje Python. La lista de palabras reservadas es la siguiente:

and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, yield, while y with

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Constantes en Python

Terminamos esta introducción a Python señalando que, a diferencia de otros lenguajes, en Python no existen las constantes.

Entendemos como constante una variable que una vez asignado un valor, este no se puede modificar. Es decir, que a la variable no se le puede asignar ningún otro valor una vez asignado el primero.

Se puede simular este comportamiento, siempre desde el punto de vista del programador y atendiendo a convenciones propias, pero no podemos cambiar la naturaleza mutable de las variables.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Entrada/Salida en Python

Para el ingreso de un dato por teclado y mostrar un mensaje se utiliza la función **input**, esta función retorna todos los caracteres escritos por el operador del programa:

```
lado=input("Ingrese la medida del lado del cuadrado:")
```

La variable lado guarda todos los caracteres ingresados pero no en formato numérico, para esto debemos llamar a la función int:

Un formato simplificado para ingresar un valor entero por teclado y evitarnos escribir las dos líneas anteriores es:

```
lado=int(input("Ingrese la medida del lado del cuadrado:"))
```

Procedemos a efectuar el cálculo de la superficie luego de ingresar el dato por teclado y convertirlo a entero:

```
superficie=lado*lado
```

Para mostrar un mensaje por pantalla tenemos la función print que le pasamos como parámetro una cadena de caracteres a mostrar que debe estar entre simple o doble comillas:

```
print("La superficie del cuadrado es")
```

Para mostrar el contenido de la variable superficie no debemos encerrarla entre comillas cuando llamamos a la función print:

```
print(superficie)
```

Algunas consideraciones

Python es sensible a mayúsculas y minúsculas, no es lo mismo llamar a la función input con la sintaxis: Input.

Los nombres de **variables** también son sensibles a mayúsculas y minúsculas. Son dos variables distintas si en un lugar iniciamos a la variable "superficie" y luego hacemos referencia a "Superficie"

Los nombres de variable no pueden tener espacios en blanco, caracteres especiales y empezar con un número.

Todo el código debe escribirse en la misma columna, estará **incorrecto** si escribimos:

```
lado=input("Ingrese la medida del lado del cuadrado:")  
lado=int(lado) superficie=lado*lado
```

```
print("La superficie del cuadrado es")print(superficie)
```


Forma correcta:

```
print(superficie)
```

```
lado=input("Ingrese la medida del lado del cuadrado:")lado=int(lado) superficie=lado*lado
```

```
print("La superficie del cuadrado es")print(superficie)
```

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Definición de comentarios en el código fuente

Un programa en Python puede definir además del algoritmo propiamente dicho una serie de comentarios en el código fuente que sirvan para aclarar los objetivos de ciertas partes del programa.

Tengamos en cuenta que un programa puede requerir mantenimiento del mismo en el futuro. Cuando hay que implementar cambios es bueno encontrar en el programa comentarios sobre el objetivo de las distintas partes del algoritmo, sobre todo si es complejo. Existen dos formas de definir comentarios en Python:

Comentarios de una sola línea, se emplea el caracter #: #definimos tres contadores

```
conta1=0conta2=0conta3=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=0conta3=0
```

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

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Docstrings

Los docstrings son un tipo de comentarios especiales que se usan para documentar un módulo, función, clase o método. En realidad son la primera sentencia de cada uno de ellos y se encierran entre tres comillas simples o dobles.

Los docstrings son utilizados para generar la documentación de un programa. Además, suelen utilizarlos los entornos de desarrollo para mostrar la documentación al programador de forma fácil e intuitiva.

Veámoslo con un ejemplo:

```
def suma(a, b):
```

```
    """Esta función devuelve la suma de los parámetros a y b"""  
    return a + b
```

Más adelante desarrollaremos el concepto de funciones, clases y métodos en Python.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Variables

Hasta este momento hemos visto algunos ejemplos de cómo definir variables enteras y flotantes. Realizar su carga por asignación y por teclado.

Para iniciarlas por **asignación** utilizamos el operador = #definición de

una variable `enteracantidad=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 carácter punto.

Para realizar la carga por teclado utilizando la función `input` debemos llamar a la función `int` o `float` para convertir el dato devuelto por `input`:

```
cantidad= int(input("Ingresar la cantidad de personas:"))
```

```
altura= float(input("Ingresar la altura de la persona en metros ej:1.70:"))
```

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Tipos de datos en Python

En programación, el tipo de datos es un concepto importante. Las variables pueden almacenar datos de diferentes tipos y diferentes tipos pueden hacer cosas diferentes. Python tiene los siguientes tipos de datos integrados de forma predeterminada:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Obtener el tipo de datos

Puede obtener el tipo de datos de cualquier objeto utilizando la función **type()**: Ejemplo:

```
x = 5 print(type(x))
```

Definición del tipo de dato

En Python, el tipo de dato se establece cuando asigna un valor a una variable:

Ejemplo	Tipo de Dato
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Definición de un tipo de dato específico

Si desea especificar el tipo de dato, puede utilizar las siguientes funciones de conversión:

Ejemplo	Tipo de Dato
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool



x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Fuente:

https://www.w3schools.com/python/python_datatypes.asp

Operadores en Python

Los operadores se utilizan para realizar operaciones en variables y valores. En el siguiente ejemplo, usamos el operador + para sumar dos valores:

Ejemplo:

```
print(10 + 5)
```

https://www.w3schools.com/python/trypython.asp?filename=demo_oper

Algunos grupos de operadores en Python: Operadores

aritméticos

Operadores de asignación Operadores de
comparación Operadores lógicos
Operadores de pertenencia

Fuente:

https://www.w3schools.com/python/python_operators.asp

Operadores aritméticos

Operad or	Descripción
+	Suma dos operandos.
-	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un float).
%	Operador modulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Operadores de asignación

El operador de asignación se utiliza para asignar un valor a una variable. Como te he mencionado en otras secciones, este operador es el signo `=`.

Además del operador de asignación, existen otros operadores de asignación compuestos que realizan una operación básica sobre la variable a la que se le asigna el valor.

Por ejemplo, `x += 1` es lo mismo que `x = x + 1`. Los operadores compuestos realizan la operación que hay antes del signo igual, tomando como operandos la propia variable y el valor a la derecha del signo igual.

A continuación, aparece la lista de todos los operadores de asignación compuestos:

Operador	Ejemplo	Equivalencia
<code>+=</code>	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 2</code>	<code>x = x % 2</code>
<code>//=</code>	<code>x //= 2</code>	<code>x = x // 2</code>
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Operadores de comparación

Los operadores de comparación se utilizan, como su nombre indica, para comparar dos o más valores. El resultado de estos operadores siempre es True o False.

Operador	Descripción
>	Mayor que. True si el operando de la izquierda es estrictamente mayor que el de la derecha; False en caso contrario.
>=	Mayor o igual que. True si el operando de la izquierda es mayor o igual que el de la derecha; False en caso contrario.
<	Menor que. True si el operando de la izquierda es estrictamente menor que el de la derecha; False en caso contrario.
<=	Menor o igual que. True si el operando de la izquierda es menor o igual que el de la derecha; False en caso contrario.
==	Igual. True si el operando de la izquierda es igual que el de la derecha; False en caso contrario.
!=	Distinto. True si los operandos son distintos; False en caso contrario.

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Operadores lógicos

Operación	Resultado	Descripción
a or b	Si a se evalúa a falso, entonces devuelve b, si no devuelve a	Solo se evalúa el segundo operando si el primero es falso
a and b	Si a se evalúa a falso, entonces devuelve a, si no devuelve b	Solo se evalúa el segundo operando si el primero es verdadero
not a	Si a se evalúa a falso, entonces devuelve True, si no devuelve False	Tiene menos prioridad que otros operadores no booleanos

Ejemplos:

x = True y = False

x or y #resultado True

x and y #resultado False not x #resultado True

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (list, tuple, dict, set o str).

Todavía no hemos visto estos tipos, pero son operadores muy utilizados.

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.

A continuación vemos unos ejemplos que son muy intuitivos: lista = [1, 3, 2, 7, 9, 8, 6] 4 in lista

#False

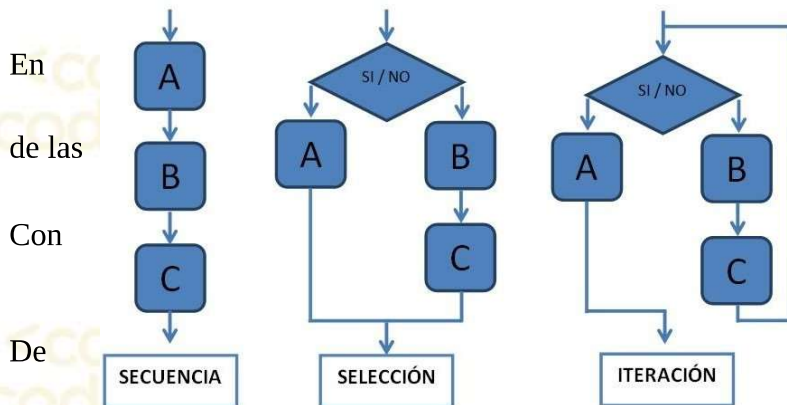
3 in lista #True 4 not in lista #True

Fuente:

<http://www.tutorialesprogramacionya.com>

<https://j2logo.com>

Estructuras de Control



programación, las estructuras de control permiten modificar el flujo de ejecución instrucciones de un programa.

las estructuras de control se puede:

acuerdo a si se cumple con una condición, ejecutar un grupo u otro de sentencias (if)

Ejecutar un grupo de sentencias mientras se cumpla una condición (while) Repetir un grupo de sentencias un número determinado de veces (for)

Todos los lenguajes de programación modernos tienen estructuras de control similares. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis (como se invoca la instrucción).

Cada lenguaje tiene una sintaxis propia para expresar la estructura.

Secuencial: las instrucciones se ejecutan una después de la otra, en el orden en que están escritas, es decir, **en secuencia**. Este proceso se conoce como ejecución secuencial.

Condicional (Selección o De Decisión): Las estructuras de control condicionales ejecutan un bloque de instrucciones u otro, o saltan a un subprograma o subrutina según se cumpla o no una condición.

Iterativa (Repetitiva): las estructuras de control iterativas o de repetición, inician o repiten un bloque de instrucciones si se cumple una condición o **mientras** se cumple una **condición**.

Estructura de Control Secuencial

La estructura secuencial se refiere a una serie de acciones que se ejecutan una seguida de la otra. Luego de la ejecución de una acción o instrucción, el control se transfiere a la siguiente.

Es importante recordar que la ejecución del programa es lineal y de arriba hacia abajo, podemos simbolizar esta estructura de control de la siguiente forma:

Lineal / Secuencial:



Ejemplo Problema Secuencial:

Ingresar dos números enteros e informar su suma.

Análisis y Algoritmo

Este ejercicio sigue siendo simple, pero ahora contamos con entrada de datos, un proceso y salida de información. Analizamos primero la entrada y salida de datos

Entrada de datos

Un número entero a guardar en la variable num1 Otro número entero a guardar en la variable num2

Salida de datos

La suma de los dos números a guardar en la variable suma En pseudocódigo:

INICIO

1. Solicitar Ingresar por teclado un número, guardarlo en num1.
2. Solicitar Ingresar por teclado un número, guardarlo en num2.
3. Realizar la suma de ambos números y guardarlo en suma.
4. Mostrar un mensaje por pantalla: "La suma es " suma. FIN

Código Fuente

```
#Prog #Programa Suma: Suma dos números enteros ingresados por teclado  
  
nro1= nro1= int( input("Ingrese un número: "))  
nro2= nro2= int( input("Ingrese un número: "))  
suma suma = nro1 + nro2  
print print("La suma es: ", suma);
```

Todas las instrucciones se ejecutan una sola vez y finaliza el programa.

Estructuras de decisión o selección

Las estructuras de decisión permiten elegir entre ejecutar una acción u otra, dependiendo de que la condición sea VERDADERA o FALSA para el valor que tienen asignadas las variables que aparecen en la misma cuando se ejecuta esta instrucción.

La palabra clave asociada a esta estructura es **if**

La forma más simple de un condicional es un if (del inglés si) seguido de la condición a evaluar y dos puntos (:) al final de la línea. En las siguientes líneas e indentado, se escribe el código a ejecutar en caso de que dicha condición sea verdadera. Este bloque de instrucciones a ejecutar finaliza con la siguiente instrucción sin indentar.

if condición: sentencia 1

 sentencia 2

...

instrucción fuera del if

Ejemplo:

```
1 a= float(input("Ingrese un número: "))
2 if a < 10:
3     print("El numero ingresado es menor que 10")
```

NOTA: Siempre tiene que haber una instrucción a ejecutar cuando la condición resulta VERDADERA.

Otra forma de la sentencia alternativa if es la ejecución alternativa, en la que hay dos posibilidades y la condición determina cuál de las dos se ejecuta. La sintaxis en este caso es:

if condición: sentencia 1

sentencia 2 else:

sentencia3

Si se debe ejecutar alguna acción cuando la condición es falsa, esta debe estar precedida por la palabra **else:**, a la altura del if al que se asocia. Éste es opcional, o sea que no tiene que colocarse si en el caso de que la condición sea falsa no hay que ejecutar ninguna sentencia.

Ejemplo: ingresar un número distinto de cero y decir si es par o impar.

```
1 a = int(input("Ingrese un número: "))
2 if a % 2 == 0:
3     print(a, "es par")
4 else:
5     print(a, "no es par")
```

Si hay más de dos ramas en la salida de la condición se pueden resolver usando la sentencia **elif** como una abreviatura de **else if**.

Ejemplo:

Ingresar un dígito y mostrar su nombre en letras.

```
1 a = int(input("Ingrese un número: "))
2 if a == 1:
3     print("UNO")
4 elif a == 2:
5     print("DOS")
6 elif a == 3:
7     print("TRES")
8 elif a == 4:
9     print("CUATRO")
10 elif a == 5:
11     print("CINCO")
12 elif a == 6:
13     print("SEIS")
14 elif a == 7:
15     print("SIETE")
16 elif a == 8:
17     print("OCHO")
18 elif a == 9:
19     print("NUEVE")
```

Anidamiento de if

Los condicionales anidados se utilizan cuando una de las salidas de la primera condición tiene a su vez otras dos posibles salidas (verdadero o falso) y así hasta agotar todas las opciones posibles. La correcta indentación permite identificar a simple vista qué condición queda dentro de otra

Ejemplo

Ingresar un número y decir si es positivo, negativo o cero

```
1 numero = int(input("Ingrese un número: "))
2 if numero > 0:
3     print("el número es mayor que cero")
4 else:
5     if numero < 0:
6         print("el número es menor que cero")
7     else:
8         print("el número es cero")
```

Cada ELSE se corresponde con el IF más próximo que no haya sido emparejado, y deben tener la misma indentación.

Uso de operadores lógicos en la condición

Podemos reducir la cantidad de if anidados usando los operadores lógicos and (y) y or

(o) dentro de la misma condición. if

(condición1) and (condición2) :

if (condición1) or (condición2) :

Recordar:

Una condición con and es verdadera si todas las condiciones que conecta son verdaderas y que la condición or es verdadera si por lo menos una de las dos condiciones es verdadera. Por lo tanto, una condición and es falsa si alguna de las condiciones que conecta es falsa y una condición or es falsa si todas las condiciones son falsas.

Las siguientes tablas muestran todas las combinaciones posibles entre los valores de verdad de dos condiciones y el valor de verdad de la operación resultante

Condición1	and	Condición2
V	V	V
V	F	F
F	F	V
F	F	F

Condición1	or	Condición2
V	V	V
V	V	F
F	V	V
F	F	F

En ciertas ocasiones, también es útil usar el operador not, que devuelve el valor de verdad contrario al de la condición que afecta. Si la condición es verdadera, al anteponerle el operador not, el valor de verdad será falso; y por el contrario, si la condición es falsa, anteponiéndole el operador not obtendremos una condición verdadera

not Condición

F	V
V	F

El orden de precedencia para los operadores booleanos es el siguiente: not

and or

Este orden se puede alterar utilizando paréntesis. Ejemplo:

not condición1 and condición2 devuelve el valor de verdad del and entre la negación de la condición 1 y la condición 2

not (condición1 and condición2) devuelve la negación del and entre la condición1 y la condición2

Ejemplos:

Uso del and

En un aviso del diario piden ingenieros en sistemas con 5 años de experiencia a lo sumo, para ocupar un puesto laboral.

A la convocatoria se presenta:

Un licenciado en sistemas con 4 años de experiencia: NO LO TOMAN, pues la primera condición es falsa.

Un ingeniero en sistemas con 8 años de experiencia: NO LO TOMAN, pues la segunda condición es falsa.

Un analista programador con 8 años de experiencia: NO LO TOMAN, pues las 2 condiciones son falsas.

Un ingeniero en sistemas con 4 años de experiencia: LO TOMAN, pues las 2 condiciones son verdaderas.

Uso del or

Tengo invitados en casa y voy a comprar 1 kilo de helado. Sé que los únicos gustos que comen son chocolate o vainilla. Después de ir a varias heladerías encontré:

Hay chocolate pero no hay vainilla. LO COMPRO, pues la primer condición es verdadera.

Sólo hay vainilla, chocolate no. LO COMPRO, pues la segunda condición es verdadera.

Hay chocolate y vainilla. LO COMPRO, pues las dos condiciones son verdaderas.

Hay americana y dulce de leche. NO LO COMPRO, pues ninguna de las condiciones es verdadera.

Ejemplo

Realizar un programa que permita ingresar un número y decir si es de dos cifras o no. Una primera resolución podría ser:


```
1 '''
2 Realizar un programa que permita ingresar un número y decir si es de dos cifras o no.
3
4 '''
5 a = int(input("Ingrese un número: "))
6 if a >= 10:
7     if a <= 100:
8         print("El número ingresado es de dos cifras")
9     else:
10        print("El número ingresado no tiene dos cifras")
```

Pero si incluimos el uso de operadores lógicos puede quedar:

```
1 a = int(input("Ingrese un número: "))
2 if a >= 10 and a <= 100:
3     print("El numero ingresado es de dos cifras")
4 else:
5     print("El numero ingresado no tiene dos cifras")
```

También podemos

abreviar el and usando la notación utilizada en matemática.

```
1 a = int(input("Ingrese un número: "))
2 if 10 <= a <= 100:
3     print("El numero ingresado es de dos cifras")
4 else:
5     print("El numero ingresado no tiene dos cifras")
```

Otro

ejemplo: Ingresar dos

números enteros y decir si alguno es divisible por el otro.

```
1 a = int(input("Ingrese un número: "))
2 b = int(input("Ingrese otro número: "))
3 if a % b == 0 or b % a == 0:
4     print("Uno de los números es divisible por el otro")
5 else:
6     print("Ninguno de los números es divisor del otro")
```

Fuente:

https://www.w3schools.com/python/python_conditions.asp

<https://www.mclibre.org/consultar/python/lecciones/python-if-else.html>

Estructuras de Control Repetitivas

Una estructura de repetición le permite al programador repetir una acción, en tanto cierta condición se mantenga verdadera.

While

Permite ejecutar un bloque de sentencias mientras la condición a la que precede es verdadera. El ciclo finaliza cuando la condición del while es falsa

while condición: sentencia 1

sentencia 2

siguiente sentencia fuera del while

Si la condición es verdadera “entra” al while y se ejecutan las sentencias que están debajo de esta instrucción indentadas.

Si la condición es falsa no entra al while y ejecuta la próxima instrucción fuera del while (la primera que no tiene indentación respecto de ese while).

Esta instrucción no permite saber de antemano cuantas veces se va a repetir el bloque de sentencias que abarca, ya que el corte se produce por el valor de verdad de la condición.

Ejemplo:

Leer números y mostrarlos hasta que se ingrese un número negativo

```
1 num1 = int(input("Ingrese un número: "))
2 while num1 > 0:
3     print(num1)
4     num1 = int(input("Ingrese un número: "))
```


Ingreso 1:

5, 14, 6, 8, -3 (el quinto número ingresado es menor que cero por lo tanto sale del ciclo) Ingreso 2:

7, -6 (el segundo número ingresado es menor que cero por lo tanto sale del ciclo)

Más información en: https://www.w3schools.com/python/python_while_loops.asp

For

Esta sentencia de repetición se utiliza cuando sabemos exactamente la cantidad de repeticiones que se deben hacer.

```
for i in range(inicio, fin, paso): sentencia1
```

```
    sentencia2
```

```
primer sentencia fuera del for
```

Donde ***i*** es la variable que controla el número de veces que se ejecuta el bloque de instrucciones que queda dentro del for, ***inicio*** es el valor inicial de ***i***, ***fin*** el valor hasta el que se va a ir incrementando ***i*** de a tantas unidades como este indicado en ***paso***. (Tener en cuenta que el ciclo se repite mientras ***i*** sea menor que ***fin***)

Algunos parámetros de los que están entre paréntesis pueden no escribirse; por ejemplo, si solo se escribe un número, este corresponde al valor de fin y asume que el valor inicial es cero y el paso es uno. Y si se escriben solo dos valores dentro del paréntesis, se asume que son el de inicio y fin y que el paso es uno.

Ejemplos:

Ingresar cinco números y mostrarlos.

```
1 for i in range(5):  
2     a = int(input("Ingrese un número: "))  
3     print(a)
```

Mostrar los números del 10 al 20.

```
1 for i in range(10, 21):  
2     print(i)
```

Imprime el valor de la variable i, que tiene un valor inicial

de 10 y un valor final de 20.

Mostrar los múltiplos de 5 comprendidos entre 12 y 80.

```
1 for i in range(15,81,5):  
2     print(i)
```

Imprime el valor de la variable i, que tiene un valor inicial de 15 y se va incrementando de a 5, hasta llegar a un valor menor que 81.

Imprimir los números del 9 al 1.

```
1 for i in range(9,0,-1):  
2     print(i)
```



<codoa
codo/>

En este caso, como el valor inicial es mayor al valor final, el paso debe ser necesariamente un número negativo.

Más información en: https://www.w3schools.com/python/python_for_loops.asp