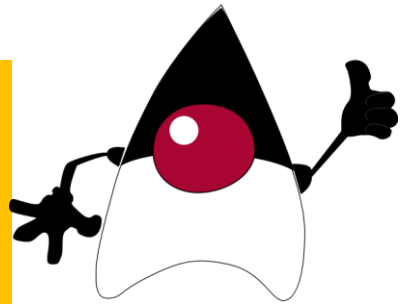
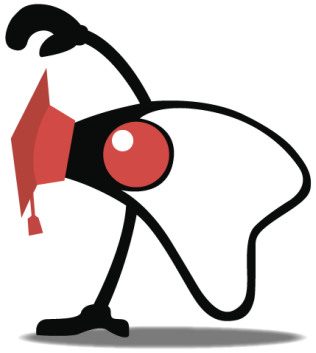
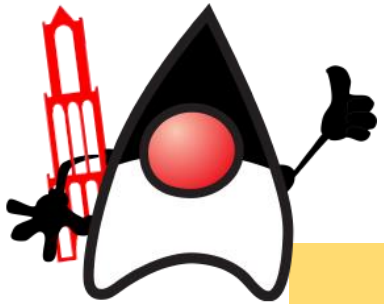
Curso FullStack Python

Codo a Codo 4.0







Python

Parte 1

Python

Es un lenguaje de programación de alto nivel cuya máxima es la **legibilidad del código**. Sus principales características son:

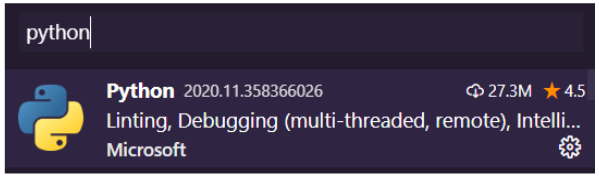
- **Multiparadigma**: Soporta la programación imperativa, programación orientada a objetos y funcional.
- **Multiplataforma**: Se puede encontrar un intérprete de Python para los principales sistemas operativos: Windows, Linux y Mac OS. Además, se puede reutilizar el mismo código en cada una de las plataformas.
- **Dinámicamente tipado**: El tipo de las variables se decide en tiempo de ejecución.
- **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.
- **Interpretado**: El código no se compila a lenguaje máquina, sino que ejecuta las instrucciones a medida que las va leyendo.

IDLE Python

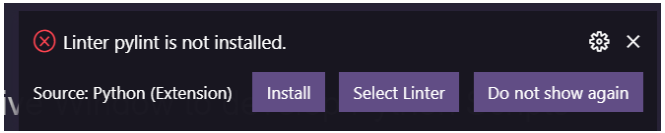
Sitio Web de descarga: https://www.python.org/downloads/
(de la versión 2 a 3 cambia mucho) Conviene descargar desde la versión 3.9 en adelante.
Es un entorno muy simple pero muy limitado.

Instalación en VS Code

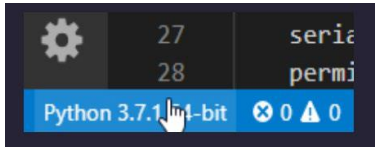
1. Descargar Python de https://www.python.org/downloads/ e instalar tildando la opción "*Agregar Python al PATH*". Al finalizar tocar en "*Disable path length limit*".
2. Descargar la extensión para VS Code:



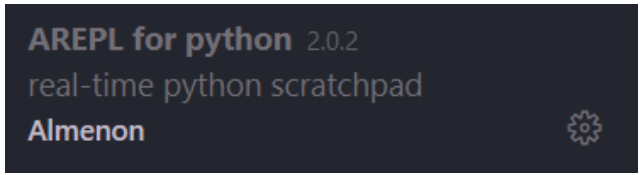
3. Al crear un archivo .py es probable que muestre el siguiente mensaje, clic en instalar.



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



*También conviene descargar **AREPL for Python**, que evalúa automáticamente el código Python en tiempo real a medida que escribe.*



Comenzando con el código...

Los archivos de Python tienen una extensión .py.

Para hacer correr un programa desde VSC hacemos clic en el ícono de Play:

hola_mundo.py

PY

```
ejemplo-clase-19 > hola_mundo.py
1   print("Hola mundo")
```

```
Hola mundo
```

Veremos el resultado en la terminal

La terminal es el intérprete, que interpreta, ejecuta y me muestra la salida. Con ***clear*** podemos limpiarla.

Tipos de datos

PY

```
#tipos de datos
cadena= "Hola Mundo en Python" #Strings
entero= 3 #int
flotante= 12.3 #float
logico= True # false (con primer letra mayúscula) boolean
```

hola_mundo.py

Expresiones y sentencias

Una expresión es una unidad de código que devuelve un valor y está formada por una combinación de operandos (variables y literales) y operadores. Ejemplos:

Expresiones

```py
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
```

expresiones.py

Sentencias

Una sentencia o declaración es **una instrucción que define una acción**. puede estar formada por una o varias expresiones, aunque no siempre es así. 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:

```py
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, recomienda una longitud de línea máxima de 72 caracteres).

Para dividir una sentencia en varias líneas se utiliza el carácter \. Por ejemplo:

```py
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:

```py
a = [1, 2, 7,
3, 8, 4,
9]
```

sentencias.py

Bloques de código (Indentación)

El código puede agruparse en bloques. 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**, que 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. La guía de estilo de Python recomienda usar los espacios en lugar de las tabulaciones para realizar el sangrado, se suelen utilizar 4 espacios.

PY

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

*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.*

Convenciones de nombres

Estas reglas y recomendaciones se siguen a la hora de nombrar una variable, una función, un módulo, una clase, etc.:

- 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 ***guion 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

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. Estas son las palabras reservadas:

```
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, with
```

Constantes

En Python no existen las constantes. Recordemos que las constantes son variables 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.

Entrada/Salida

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:")
```
PY

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 es:

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

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

```
superficie=lado*lado
```
PY

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")
```
PY

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

```
print(superficie)
```
PY

entrada_salida.py

print.py

input.py

Comentarios en línea o en bloque

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, fundamentalmente si es complejo.

```py
# Comentario en línea
'''
Comentarios
en
bloque
'''
```

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.

```py
def suma(a, b):
    """Esta función devuelve la suma de los parámetros a y b"""
    return a + b
```

Variables

Una variable es una posición en memoria donde se puede almacenar un valor. Pueden ser de tipo entero (int), real (float) o booleano (bool)

Así como podemos asignar un valor a una variable **por teclado** también podemos hacerlo **por asignación** con el operador =:

```py
#definición de una variable entera
cantidad=20
#definición de una variable flotante
altura=1.92
```

El intérprete de Python diferencia una variable flotante de una variable entera por la presencia del caracter 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**:

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

Tipos de datos

Las variables pueden almacenar datos de diferentes tipos. 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

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

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

Resultado: <class 'int'>

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

Ejemplo	Tipo de Dato
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

Cadenas de caracteres

Una cadena de caracteres está compuesta por uno o más caracteres. Podemos iniciarla por asignación o ingresarla por teclado.

Inicialización de una cadena por asignación:

```py
dia="lunes" #definición e inicio de una cadena de caracteres
dia='lunes' #igual resultado obtenemos si utilizamos la comilla simple
```

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

```py
nombre=input(“Ingrese su nombre:")
```

En general una cadena de caracteres está formada por varios caracteres y podemos acceder en forma individual a cada caracter del string mediante un subíndice:

```py
nombre='juan'
print(nombre[0]) #se imprime una j
if nombre[0]=="j": #verificamos si el primer caracter del string es una j
    print(nombre)
    print("comienza con la letra j")
```

Los subíndices comienzan a numerarse a partir del cero.

Cadenas de caracteres

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

```py
nombre='juan'
print(len(nombre))
```

El programa anterior imprime un 4 ya que la cadena nombre almacena 'juan' que tiene cuatro caracteres.

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. Tres de estos métodos son: *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.

Concatenación:

```py
var1 = 'Hola'
var2 = 'Python'
var3 = var1 + ' ' + var2
```

cadenas.py

Ejemplos de tipos de datos, variables y operadores de asignación

PY

```python
#tipos de datos
cadena= "Hola Mundo en Python" #Strings
entero= 3 #int
flotante= 12.3 #float
logico= True # False (con primer letra mayúscula) boolean

# variables
# cadena_string: Esta nomenclatura se utiliza para funciones
# cadenaString: La nomenclatura camelCase se usa para variables
cadenaString= "123"
valorNumerico= 123

#Operador de asignación
variable= 12 #Entero
variable= 12.3 #Asignamos otro valor de otro tipo (tipado dinámico)
variable= "Cadena" #No es aconsejable reutilizar la misma variable con otro tipo
 de dato
# Estas variables son globales de por sí
```

Conversión de tipos de datos

Entre las más comunes encontramos:

- str -> int
- int -> str
- str -> float
- int -> float
- float -> int

conversion.py

Operadores

- De asignación (+=, -=, *=, /=, %=, //=, **=)
- Aritméticos (+, -, *, /, %, //, **)
- Relacionales o de comparación (>, >=, <, >=, ==, !=)
- Lógicos (or, and, not)
- De pertenencia (in, not in)

tipos_operadores.py

Operadores aritméticos

Los operadores aritméticos realizan operaciones matemáticas, como sumas o restas con operandos. Los operadores **unarios** realizan una acción con un solo operando. Los operadores **binarios** realizan acciones con dos operandos. En una expresión compleja (dos o más operandos), el orden de evaluación depende de las reglas de precedencia.

Operador	Descripción
+	**Suma:** Suma dos operandos.
-	**Resta:** Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	**Mutiplicación:** Producto de dos operandos.
/	**División:** Divide el operando de la izquierda por el de la derecha (el resultado siempre es un float).
%	**Operador módulo:** Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	**División entera:** 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.

Operadores de asignación

El operador de asignación se utiliza para asignar un valor a una variable, 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.

Operador	Ejemplo	Equivalencia
+=	x += 2	x = x + 2
-=	x -= 2	x = x – 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2

Operadores relacionales o de comparación

Se utilizan 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.

Operadores lógicos

Los operadores lógicos nos proporcionan un resultado a partir de que se cumpla o no una cierta condición, producen un resultado booleano, y sus operandos son también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a cierto o falso según su valor sea cero o distinto de cero).

Operador	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

PY

```
x = True
y = False
x or y #resultado True
x and y #resultado False
not x #resultado False
```

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).

Operador	Equivalencia
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.

PY

```
lista = [1, 3, 2, 7, 9, 8, 6]
4 in lista #False
3 in lista #True
4 not in lista #True
```

Ejemplos, cursos y guías de Python

- **Apunte de Majo – Python:** https://drive.google.com/file/d/12_1yUhaGeoH7wLGqrHiSx987FMdqM_Mv/view
- **Curso de Python – Código Facilito:** https://codigofacilito.com/cursos/Python
- **Curso de Python – Píldoras informáticas (lista de reproducción de YouTube):** https://youtube.com/playlist?list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS