<codoa codo/>

Agencia de Aprendizaje a lo largo de la vida

FULL STACK PYTHON
Clase 25

PYTHON 1

BA

Fundamentos de Python

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 24

JOIN y Subconsultas

- JOIN: Inner, Left, Right.
- Funciones de agregación, GROUP BY, HAVING.
- Funciones escalares.
- Subconsultas.

Clase 25

Fundamentos de Python

- Introducción a Python.
- Entorno. Hola mundo.
- Salida por pantalla: print.
- Lectura por teclado: input.
- Tipo de datos: números enteros y flotantes, texto, booleanos.
- Tipos de operadores. Aritméticos y de asignación.
- Variables.

Clase 26

Controladores de flujo

- Estructuras control.
- Condicionales: sentencia if.
- Iterativas: sentencia while y for.
- Operadores lógicos y relacionales.

Python

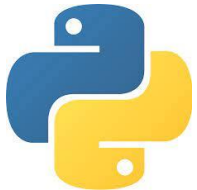
Python es un lenguaje de programación de alto nivel que se destaca, entre otras cosas, por la legibilidad del código. Sus principales características son:

- **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.
- **Dinámica y fuertemente tipado:** El tipo de las variables se decide en tiempo de ejecución. Pero no se puede usar una variable en un contexto fuera de su tipo sin efectuar una conversión.
- **Interpretado**: El código no se compila a lenguaje máquina, sino que se ejecutan las instrucciones a medida que se las lee.

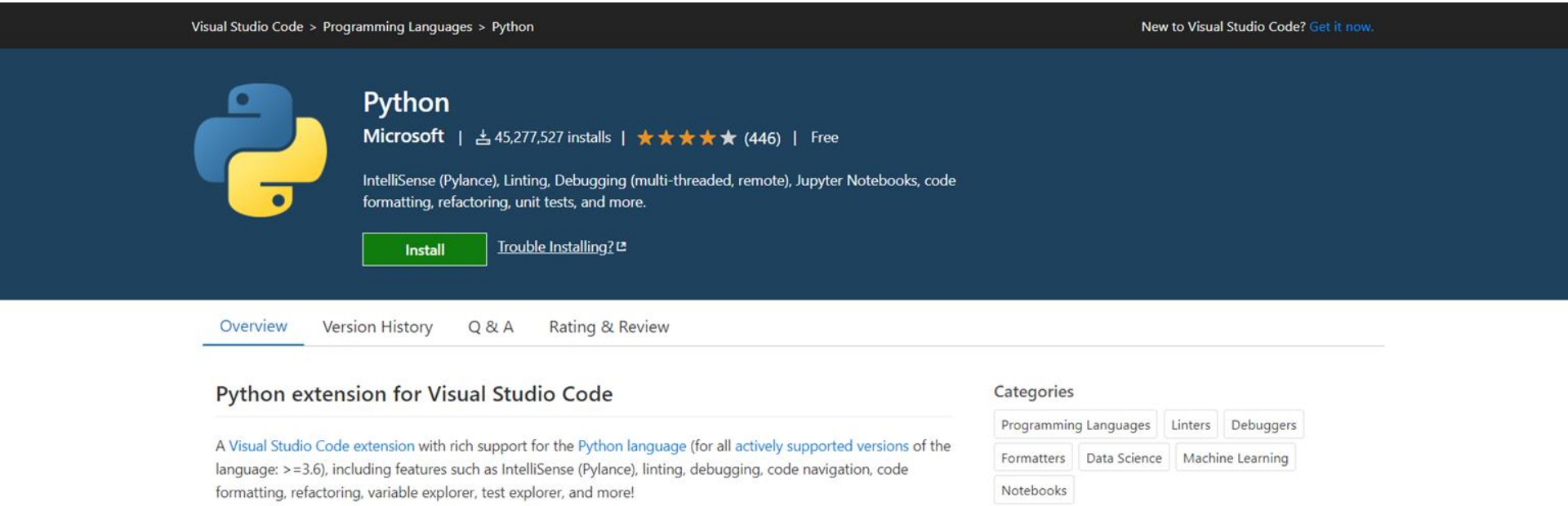
Instalación de Python y VSCode

Necesitamos instalar el **intérprete Python** desde su página oficial. Asegúrate de instalar la última versión (o como mínimo la versión 3.8.x). Recuerda instalar tildando la opción "*Agregar Python al PATH*". Al finalizar hacer click en "*Disable path length limit*".

Utilizaremos un editor de texto. **Visual Studio Code** funciona perfectamente, y si aún no lo tienes, lo puedes descargar desde su página oficial.



Extensión para VS Code



Agencia de Aprendizaje
a lo largo de la vida

Escribiendo código Python

Los archivos de Python tienen una extensión **.py.**

Comentarios en línea, en bloque y docstrings

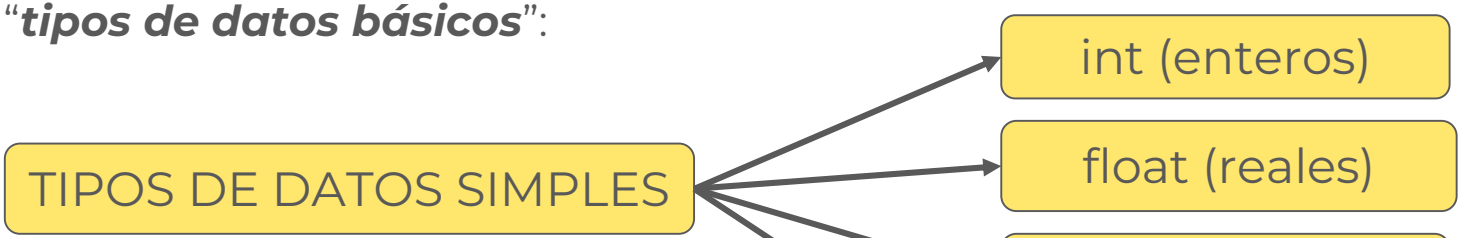
Los comentarios en el código sirven para aclarar los objetivos de ciertas partes del programa. Python permite comentarios **en línea** con **#**, o **en bloque** con triples comillas simples. También existen los **docstrings** (triples comillas dobles), utilizados para generar la documentación de un programa en forma automática con alguna herramienta externa.

Formas de realizar comentarios en el código Python

```
'''
Comentarios en bloque
Este es un ejemplo de los tipos de comentarios que posee Python.
'''
def suma(a, b): # Comentario en línea
    """Esta función devuelve la suma de los parámetros a y b"""
    return a + b
```

Tipos de datos en Python

Existen una gran variedad de datos en **Python**. Entre ellos algunos que se consideran "***tipos de datos básicos***":



Tipos de datos

```
cadena= "Hola Mundo en Python" #Strings
entero= 3 #int
flotante= 12.3 #float
logico= True # o False, boolean
```

Tipos de datos en Python

Algunos ejemplos de los valores que pueden tener los distintos tipos de datos:

- **Número Entero (int):** 3, 128, -453
- **Número Decimal (float):** 3.454 , - 12.4 , 3.14
- **Caracter (chr):** A, b, z, 6, %
- **Cadena de Texto (str):** Hola , Juan, A23, ¡Esto es otra cadena de texto!
- **Booleano (bool):** True, False

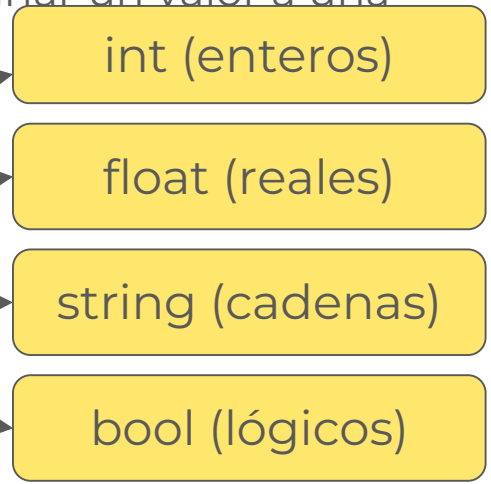
También disponemos de datos más complejos, a los que se denominan generalmente “colecciones” y que son muy importantes en Python, a los cuales nos referiremos en detalle más adelante.

Operadores y expresiones

El **=** (igual) es muy importante en Python. Su función es diferente a la que habitualmente le damos en otros contextos, como la matemática. Se lo denomina "**operador de asignación**" y nos permite asignar un valor a una variable.

Variables

```
cantidad = 43
precio = 12.45
nombre = "Adrián"
encendido = True
```



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.

Expresiones

```
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
```

Una **sentencia** o **declaración** define una acción. Puede contener alguna(s) expresiones . Son las instrucciones que componen el código de un programa y determinan su comportamiento. Finalizan con un Enter.

Sentencias de más de una línea

Aquellas sentencias que son muy largas pueden ocupar más de una línea (se recomienda una longitud máxima de 72 caracteres). Para dividir una sentencia explícitamente en varias líneas se utiliza el carácter ****.

Ejemplo 1 (división explícita)

```
a = 2 + 3 + 5 + 7 + 9 + 4 + \
6 + 3 + 5 + 1
```

Ejemplo 2 (división implícita)

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

Además, en Python la continuación de línea es **implícita** siempre y cuando la expresión vaya dentro de los caracteres **()**, **[]** y **{}**.

Bloques de código (Indentación)

El código puede agruparse en bloques, que delimitan sentencias relacionadas. Python no usa los caracteres **{}** para definir un bloque, utiliza la **indentación o sangrado**, que consiste en mover el bloque de código hacia la derecha insertando **espacios** o **tabuladores** al principio de la línea, dejando un margen a su 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 espacios en lugar de tabulaciones.

Para realizar el sangrado, se suelen utilizar 4 espacios.

Bloques de código (Indentación)

Indentación

```
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
print()                     # Bloque 1
```

Si bien aún no sabemos exactamente qué hace el código anterior, se pueden ver los bloques de instrucciones indicados mediante los tabuladores sobre el margen izquierdo. Es posible incluir un bloque dentro de otro, para crear estructuras complejas.

Convenciones de nombres

Los nombres de variables, funciones, módulos y clases deben respetar las siguientes convenciones:

- Pueden ser cualquier combinación de letras (mayúsculas y minúsculas), dígitos y el carácter guión bajo (_), pero no puede comenzar por un dígito. Se escriben en minúsculas, separando las palabras con el guión bajo.
- No se pueden usar como identificadores las palabras reservadas.
- Se recomienda usar nombres que sean expresivos. Por ejemplo, contador es mejor que simplemente c.
- Solamente los nombres de clase pueden comenzar con mayúsculas, y siguen la notación CamelCase.
- Python es “case sensitive”, diferencia entre mayúsculas y minúsculas.

Convenciones de nombres

Python tiene una serie de palabras reservadas, que se utilizan para definir la sintaxis y estructura del lenguaje. No pueden usarse como identificadores.

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
```

En Python **no existen las constantes**. Sin embargo, se suelen utilizar variables, con su nombre en mayúsculas (para distinguirlas de las demás) y es responsabilidad del programador no cambiar su valor a lo largo del programa.

Convenciones de nombres

Algunos nombres de variables **válidos y recomendados**:

suma | total | importe_final | _saldo | area12

Algunos nombres de variables **válidos** pero **no recomendados**:

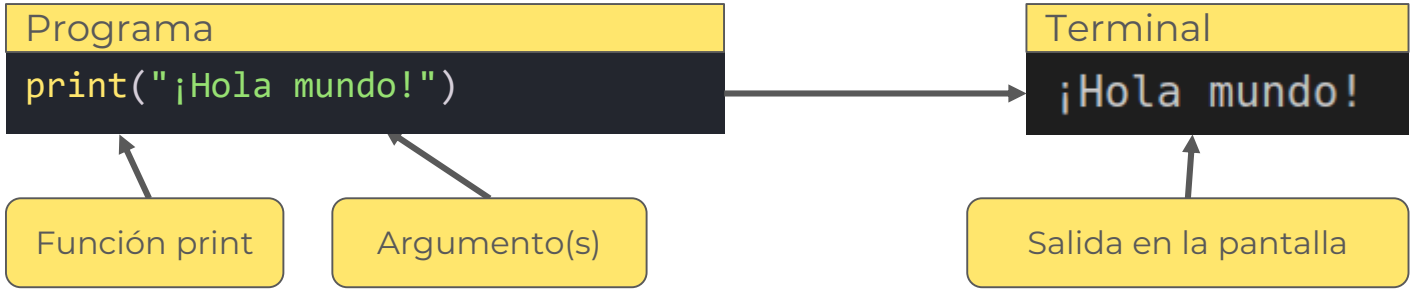
Suma | areacuadrado | ImporteFinal | w12e43rt41 | años

Algunos nombres de variables **no válidos** (Python reporta error):

mi saldo | 2pesos | for | 21%IVA | $a_pagar

Entrada / Salida: La función print()

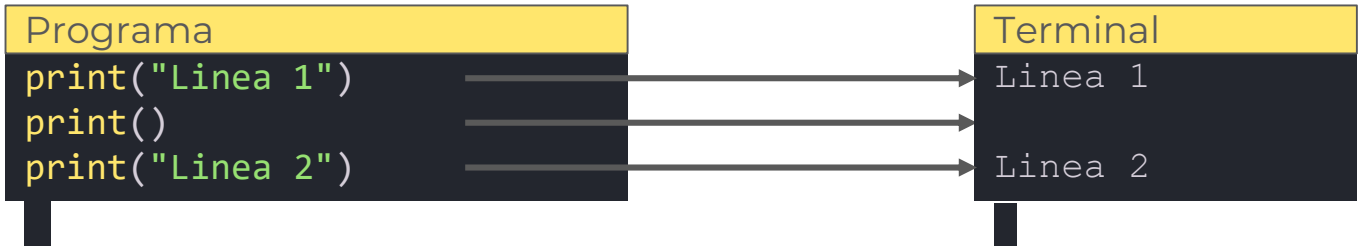
La función **print()** permite mostrar datos en la terminal. Recibe como parámetros variables y/o literales, separados por comas.



En Python las cadenas pueden delimitarse con comillas simples o dobles.

Entrada / Salida: La función print()

Luego de imprimir, **print()** realiza un salto de línea (pasa a la línea siguiente). Si se usa **print()** sin argumentos, sólo se muestra la línea en blanco:



Para evitar el salto de línea, podemos agregar el argumento `end=""` al final de la lista de argumentos:

Entrada / Salida: La función print()

Algunos ejemplos de **print()**:

Programa

```
nombre = "Adrián"
a = 40
b = 30
promedio = (a + b) / 2

print("Mi nombre es", nombre)

print("La suma de",a,"y",b,"es",a+b)

print("Promedio:" + str(promedio) )
```

Declaración de variables

Operaciones y asignación

Terminal

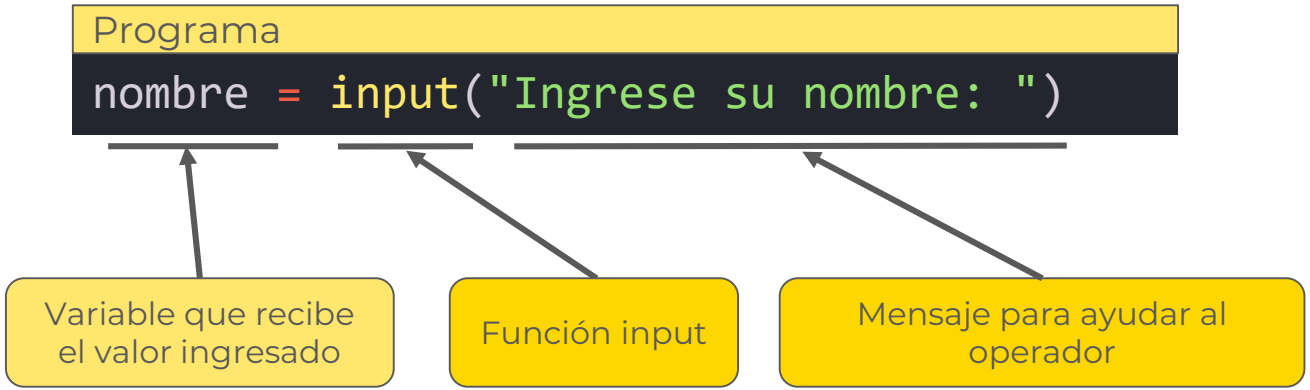
```
Mi nombre es Adrián

La suma de 40 y 30 es 70

Promedio:35.0
```

Entrada / Salida: La función input()

input() proporciona un mecanismo para que el usuario introduzca datos en nuestro programa. Muestra el cursor en la terminal, lee lo que se escribe, y cuando se presiona Enter, este contenido, en formato de cadena de caracteres, se puede asignar a una variable.



Entrada / Salida: La función input()

Dado que **input()** devuelve únicamente valores tipo *string*, es necesario realizar una conversión a algún formato numérico si se requiere operar matemáticamente con esos valores. Para ello, usamos las funciones **int()** y **float()**:

Programa

```
num1 = input("Ingrese un número: ")
numero = float(num1)
resultado = numero * 2
print(numero,"x 2 = ", resultado)
```

Terminal

```
Ingrese un número: 13.5
13.5 x 2 =  27.0
```

La cadena ingresada en el **input()** se convierte en un número de coma flotante, se almacena en "*numero*", se guarda en "*resultado*" su producto con 2, y se muestra en la terminal usando **print()**

Tipos de datos

Las variables pueden almacenar datos de diferentes tipos. En Python existen los siguientes tipos de datos:

Familia	Tipos
Texto	str
Numéricos	int, float, complex
Colecciones	list, tuple, range
Mapeos	dict
Conjuntos	set, frozenset
Booleanos	bool
Binarios	bytes, bytearray, memoryview

Se puede conocer el tipo de dato de cualquier objeto utilizando la función **type()**:

Programa

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

Terminal

```
<class 'int'>
```

Definición de tipo de dato

El tipo de dato de una variable se establece cuando se le asigna un valor:

Ejemplo	Tipo de Dato
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["higo", "pera", "uva"]	list
x = ("higo", "pera", "uva")	tuple
x = range(6)	range

Ejemplo	Tipo de Dato
x = {"name" : "John", "age" : 36}	dict
x = {"higo", "pera", "uva"}	set
x = frozenset({"higo", "pera", "uva"})	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 cero o más caracteres. Las cadenas pueden delimitarse con comillas simples o dobles.

Inicialización de una cadena por asignación:

```
dia="lunes" #definición e inicio de una cadena de caracteres
x=""        #x contiene una cadena de caracteres de longitud nula.
```

Una ventaja del hecho de poder delimitar cadenas con comillas simples o dobles es que si usamos comillas de una clase, las de otra clase puede utilizarse como parte de la cadena:

Uso de comillas simples y dobles

```
print("Mi perro 'Toby'") # Mi perro 'Toby'
print('Mi perro "Toby"') # Mi perro "Toby"
```

Cadenas de caracteres | Concatenación

Podemos unir (**concatenar**) dos cadenas utilizando el signo **+** (“más”):

Concatenación de cadenas:

```
var1 = 'Hola'
var2 = 'Python'
var3 = var1 + ' ' + var2
print(var3) # Imprime Hola Python
```

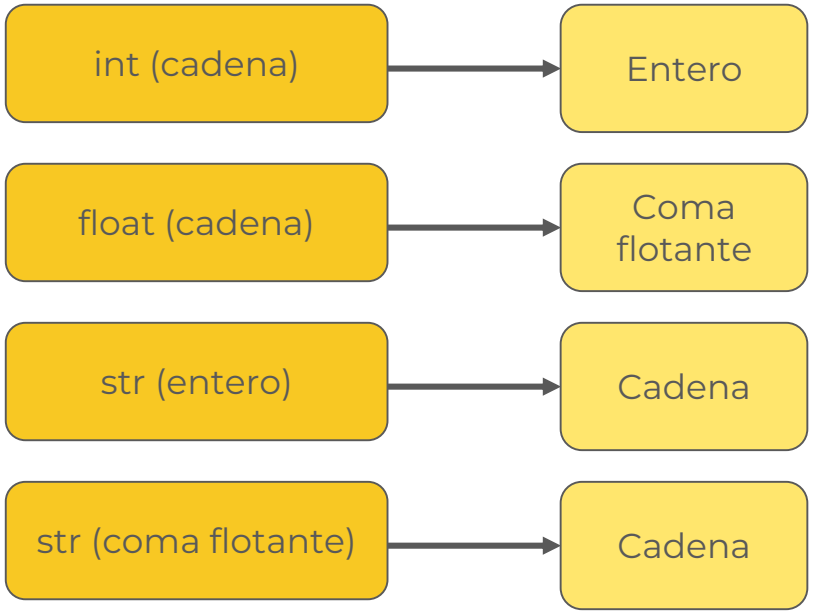
El mismo signo + se usa para sumar números o concatenar cadenas. Pero no podemos utilizarlo con datos mixtos, porque se obtiene un error:

Concatenación de cadenas:

```
var1 = 3 + 5     # 8
var2 = "3" + "5" # 35
var3 = 3 + "5"   # TypeError
```

Conversión de tipos de datos

En ocasiones es necesario aplicar conversiones de valores entre **tipos de datos** para manipular los valores de forma diferente. Por ejemplo, es posible que debamos concatenar valores numéricos con cadenas o representar posiciones decimales en números que se iniciaron como valores enteros. Python provee funciones que pueden hacer estas tareas:



Operador

Un operador es un carácter o conjunto de caracteres que actúa sobre una, dos o más variables y/o literales para llevar a cabo una operación con un resultado determinado.
Ejemplos de operadores comunes son los operadores aritméticos + (suma), - (resta) o * (producto), aunque en Python existen otros operadores.
Tipos de operadores:

- Operador de Asignación
- Operadores Aritméticos
- Operadores de pertenencia
- Operadores Relacionales (los abordaremos en la próxima presentación)
- Operadores Lógicos (los abordaremos en la próxima presentación)

Agencia de Aprendizaje a lo largo de la vida
Buenos Aires Ciudad Educación

Operadores aritméticos

Realizan operaciones aritméticas. Requieren uno o dos operandos (operadores unarios o binarios). Se aplican 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.
*	**Multiplicación:** 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 compuestos

Además del operador de asignación, existen los **operadores de asignación compuestos**. Realizan la operación indicada sobre la misma variable.

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

Por ejemplo, **x += 1** equivale a **x = x + 1**. Los operadores compuestos realizan la operación indicada antes del signo igual, tomando como operandos la propia variable y el valor a la derecha del signo igual. Y el resultado se guarda en la variable.

Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un caracter o cadena se encuentran dentro de otra.

OPERADOR	DESCRIPCION
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.

Ejemplos:

```
cadena = "Codo a Codo"
print("C" in cadena)     # True
print("n" in cadena)     # False
print("Codo" in cadena)  # True
print("A" not in cadena) # True
print("o" not in cadena) # False
```

Material extra

Artículos de interés

Material extra:

- Descarga Python de su sitio oficial.
- Guia de estilo PEP8, o cómo escribir buen código Python
- Apuntes de Majo
- Curso de Python, en Código Facilito

Videos:

- ¿Qué es Python? e Instalación de Python, en FAZT
- Curso de Python, en Píldoras informáticas

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios obligatorios.

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto