



Lenguaje de Programación Web II

Lenguaje de Programación Python



Primero Pasos

- Escribir Código.
- Operaciones Básicas.
- Prueba del entorno.



```
DATA BREACHC6F6650358E1B419C86A548B18C69C11812A645033974F14894
0B1C05B535F...B39EE8CA16BE3B6009CYBER ATTACKCD7CE6FEB6A68D7B881F3
315EF0841AC4D50F7D8A2DA43DFDB9AF0832F08DE951647848C16...B1F4347
0D10E55A46...27F141E634999...F408007D99DAD6EB94SYSTEM PROTECTION5051
28...44A3SECURITY BREACH2C0C24ACE35D...BFD955FF3896C99DF73...
01...44A3SECURITY BREACH2C0C24ACE35D...BFD955FF3896C99DF73...
2E6A9AB6BA...51DE57AC1709...36...3AE764...067...F9E8
783907CA4F2...D4...A3A710D082C...05...25...3E...60...A49...F5C50
8AE5AF96685...D2...6356B4C8DD0C...0158...05...33...60...10C8845A8
36E9EF61F3...D7D81664D04B1DE...EF78343D...39...7A...40...3E...8A0
16F4D2B2D06F...24A36A662BD87...E2...60...00...7C2E9D5C732
AA954EC273C1606...60...A1228A52F0...A0...BE79C6E89B...67...10FE...62DA
FCEA319859213CA538...3ED2ABE310F42...01100B408B01...82D99AFD8
CB8D4BED935B35808...436D5400...CC62C0A8017929C60B...EED43A
3FE0000000SYSTEMAF023CA0000020405B401030801010402D4BED935B35894F
5E0000000450001FD28...SAFETY95AA1CA01...E556E834
```

Tu primer programa Python

- Ahora que tenemos Python en funcionamiento, podemos escribir nuestro primer programa en Python.
- Creemos un programa muy simple llamado **Hello World**. Un **"¡Hola, mundo!"** es un programa simple que se muestra **Hello, World!** en la pantalla. Dado que es un programa muy simple, a menudo se usa para presentar un nuevo lenguaje de programación a los que recién se inician.

Tu primer programa Python

- Escriba el siguiente código en cualquier editor de texto o un IDE y guárdelo como **hello_world.py**

```
print("Hello, world!")
```

- Luego, ejecute el archivo. Obtendrá el siguiente resultado.

```
¡Hola Mundo!
```



Identificadores y palabras clave de Python

Palabras clave de Python

- Las palabras clave son las palabras reservadas en Python.
- No podemos usar una palabra clave como nombre de variable, nombre de función o cualquier otro identificador. Se utilizan para definir la sintaxis y la estructura del lenguaje Python.
- En Python, las palabras clave se distinguen entre mayúsculas y minúsculas.
- Hay 33 palabras clave en Python 3.7. Este número puede variar ligeramente con el transcurso del tiempo.

Palabras clave de Python

- Todas las palabras clave Excepto **True**, **False** y **None** están en minúsculas y deben escribirse tal como son. La lista de palabras clave se muestra a continuación.

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



Identificadores de Python

- Un identificador es un nombre que se le da a entidades como clase, funciones, variables, etc. Ayuda a diferenciar una entidad de otra.

Reglas para escribir identificadores


- Los identificadores pueden ser una combinación de letras en minúscula (**de la a a la z**) o mayúsculas (**de la A a la Z**) o dígitos (**del 0 al 9**) o un guión bajo `_`. Nombres como *myClass*, *var_1* y *print_this_to_screen*, todos son ejemplos válidos.
- Un identificador no puede comenzar con un dígito. *1variable* no es válido, pero *variable1* es un nombre válido.
- Las palabras clave no se pueden utilizar como identificadores.

`global = 1`

- ¡No podemos usar símbolos especiales como `!`, `@`, `#`, `$`, `%` etc. en nuestro identificador. `a@ = 0`

Cosas para recordar

- Python es un lenguaje que distingue entre mayúsculas y minúsculas. Esto significa, *Variable* y *variable* no son lo mismo.
- Siempre dé a los identificadores un nombre que tenga sentido. Si bien `c = 10` es un nombre válido, la escritura `contador = 10` tendría más sentido y sería más fácil averiguar qué representa cuando mira su código.
- Se pueden separar varias palabras con un guión bajo, como `esta_es_una_variable_larga`.



Declaración, sangría y comentarios de Python

Declaración de varias líneas

- En Python, el final de una declaración está marcado por un carácter de nueva línea. Pero podemos hacer que una declaración se extienda por varias líneas con el carácter de continuación de línea (\). Por ejemplo:

```
a = 1 + 2 + 3 + \  
    4 + 5 + 6 + \  
    7 + 8 + 9
```

Declaración de varias líneas

- Esta es una continuación de línea explícita. En Python, la continuación de la línea está implícita entre paréntesis (), corchetes [] y llaves {}. Por ejemplo, podemos implementar la declaración de varias líneas anterior como:

```
a = (1 + 2 + 3 +  
     4 + 5 + 6 +  
     7 + 8 + 9)
```


Declaración de varias líneas

- Aquí, los paréntesis () hacen la continuación de línea implícitamente. Igual es el caso con [] y {}. Por ejemplo:

```
colors = ['red',  
          'blue',  
          'green']
```

- También podemos poner varias declaraciones en una sola línea usando punto y coma, de la siguiente manera:

```
a = 1; b = 2; c = 3
```

Sangría de Python

- La mayoría de los lenguajes de programación como C, C ++ y Java usan llaves {} para definir un bloque de código. Python, sin embargo, usa sangría.
- Un bloque de código (cuerpo de una función , bucle , etc.) comienza con sangría y termina con la primera línea sin sangría. La cantidad de sangría depende de usted, pero debe ser constante en todo ese bloque.
- Generalmente, se utilizan cuatro espacios en blanco para la sangría y se prefieren a las pestañas. Aquí hay un ejemplo.

```
for i in range(1,11):  
    print(i)  
    if i == 5:  
        break
```

Sangría de Python

- La aplicación de la sangría en Python hace que el código se vea limpio y ordenado. Esto da como resultado programas de Python que se ven similares y consistentes.
- La sangría se puede ignorar en la continuación de la línea, pero siempre es una buena idea sangrar. Hace que el código sea más legible. Por ejemplo:

```
if True:  
    print('Hello')  
    a = 5
```

y

```
if True: print('Hello'); a = 5
```

- ambos son válidos y hacen lo mismo, pero el estilo anterior es más claro.
- Se producirá una sangría incorrecta **IndentationError**.

Comentarios de Python

- Los comentarios son muy importantes al escribir un programa. Describen lo que está sucediendo dentro de un programa, de modo que una persona que mira el código fuente no tiene dificultades para averiguarlo.
- Es posible que olvide los detalles clave del programa que acaba de escribir en un mes. Por lo tanto, tomarse el tiempo para explicar estos conceptos en forma de comentarios siempre es provechoso.
- En Python, usamos el símbolo de almohadilla (#) para comenzar a escribir un comentario.

Comentarios de Python

- Se extiende hasta el carácter de nueva línea. Los comentarios son para que los programadores comprendan mejor un programa. El interprete de Python ignora los comentarios.

```
#This is a comment  
#print out Hello  
print('Hello')
```

Comentarios de varias líneas

- Podemos tener comentarios que se extiendan hasta varias líneas. Una forma es utilizar el símbolo de almohadilla (#) al principio de cada línea. Por ejemplo:

```
#This is a long comment  
#and it extends  
#to multiple lines
```


Comentarios de varias líneas

- Otra forma de hacer esto es usar comillas triples, ya sea ' ' ' o

“ “ “
.

- Estas comillas triples se utilizan generalmente para cadenas de varias líneas. Pero también se pueden utilizar como comentarios de varias líneas. A menos que no sean cadenas de documentos, no generan ningún código adicional.

```
"""This is also a  
perfect example of  
multi-line comments"""
```



▀ Variables, constantes y literales de Python

Variables de Python

- Una variable es una ubicación con nombre que se utiliza para almacenar datos en la memoria. Es útil pensar en las variables como un contenedor que contiene datos que se pueden cambiar más adelante en el programa. Por ejemplo,

```
number = 10
```

- Aquí, hemos creado una variable llamada **numero**. Le hemos asignado el valor **10** a la variable.

Variables de Python

- Puede pensar en las variables como una bolsa para almacenar libros y ese libro se puede reemplazar en cualquier momento.

```
number = 10  
number = 1.1
```

- Inicialmente, el valor de **number** estaba **10**. Más tarde, se cambió a **1.1**.

Asignar valores a variables en Python

- Como puede ver en el ejemplo anterior, puede usar el operador de asignación = para asignar un valor a una variable.
- Ejemplo 1: declarar y asignar valor a una variable

```
website = "apple.com"  
print(website)
```

- Salida

```
apple.com
```

- En el programa anterior, asignamos un valor **apple.com** a la variable **sitio web**. Luego, imprimimos el valor asignado a **sitio web** es decir **apple.com**

Asignar valores a variables en Python

- **Nota** : Python es un lenguaje de tipo inferido , por lo que no es necesario definir explícitamente el tipo de variable. Automáticamente sabe que **apple.com** es una cadena y declara el **sitio web** variable como una cadena.

Asignar valores a variables en Python

Ejemplo 2: cambiar el valor de una variable

```
website = "apple.com"  
print(website)  
  
# assigning a new variable to website  
website = "programiz.com"  
  
print(website)
```

Salida:

```
apple.com  
programiz.com
```

- En el programa anterior, hemos asignado apple.com al sitio web variable inicialmente. Luego, el valor se cambia a programiz.com.

Asignar valores a variables en Python

- Ejemplo 3: Asignar varios valores a varias variables

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)
```

Si queremos asignar el mismo valor a múltiples variables a la vez, podemos hacer esto como:

```
x = y = z = "same"

print (x)
print (y)
print (z)
```

El segundo programa asigna el mismo cadena a las tres variables x, y, z.

Constantes

- Una constante es un tipo de variable cuyo valor no se puede cambiar. Es útil pensar en las constantes como contenedores que contienen información que no se puede cambiar más adelante.
- Puede pensar en las constantes como una bolsa para almacenar algunos libros que no se pueden reemplazar una vez colocados dentro de la bolsa.

Asignar valor a constante en Python

- En Python, las constantes generalmente se declaran y asignan en un módulo. Aquí, el módulo es un nuevo archivo que contiene variables, funciones, etc., que se importa al archivo principal. Dentro del módulo, las constantes están escritas en mayúsculas y subrayados que separan las palabras.

Asignar valor a constante en Python

Ejemplo 3: Declaración y asignación de valor a una constante

- Crea un constant.py :

```
PI = 3.14  
GRAVITY = 9.8
```

- Crea un main.py :

```
import constant  
  
print(constant.PI)  
print(constant.GRAVITY)
```

- Salida

```
3,14  
9,8
```

Reglas y convenio de nomenclatura para variables y constantes

1. Los nombres de constantes y variables deben tener una combinación de letras en minúsculas (de la **a a la z**) o mayúsculas (de la **A a la Z**) o dígitos (**0 a 9**) o un guión bajo (**_**). Por ejemplo:

```
caso_serpiente  
MACRO_CASE  
el caso de Carmel  
CapWords
```

2. Crea un nombre que tenga sentido. Por ejemplo, **vocal** tiene más sentido que **v**.

Reglas y convenio de nomenclatura para variables y constantes

3. Si desea crear un nombre de variable que tenga dos palabras, use un guión bajo para separarlas. Por ejemplo:

```
mi nombre  
salario actual
```

4. Utilice letras mayúsculas posibles para declarar una constante. Por ejemplo:

```
Pi  
GRAMO  
MASA  
VELOCIDAD DE LA LUZ  
TEMPERATURA
```

5. Nunca use símbolos especiales como!, @, #, \$,%, Etc.
6. No empiece el nombre de una variable con un dígito.

Literales

- Literal es un dato sin procesar dado en una variable o constante. En Python, hay varios tipos de literales que son los siguientes:

Literales numéricos

- Los literales numéricos son inmutables (inmutables). Los literales numéricos pueden pertenecer a 3 tipos numéricos diferentes: **Integer**, **Float**, y **Complex**.

Literales numéricos

```
a = 0b1010 #Binary Literals  
b = 100 #Decimal Literal  
c = 0o310 #Octal Literal  
d = 0x12c #Hexadecimal Literal
```

```
#Float Literal  
float_1 = 10.5  
float_2 = 1.5e2
```

```
#Complex Literal  
x = 3.14j
```

```
print(a, b, c, d)  
print(float_1, float_2)  
print(x, x.imag, x.real)
```

```
10 100 200 300  
10,5 150,0  
3,14j 3,14 0,0
```

Literales numéricos

- En el programa anterior,
- Asignamos literales enteros a diferentes variables. Aquí, un es literal binario, segundo es un literal decimal, C es un literal octal y re es un literal hexadecimal.
- Cuando imprimimos las variables, todos los literales se convierten en valores decimales.
- 10,5 y 1.5e2 son literales de punto flotante. 1.5e2 se expresa con exponencial y es equivalente a $1,5 * 10^2$.
- Asignamos un literal complejo, es decir 3,14j en variable X. Luego usamos literal imaginario (x.imag) y literal real (x.real) para crear partes imaginarias y reales de números complejos.

Literales de cadena

- Un literal de cadena es una secuencia de caracteres entre comillas. Podemos usar comillas simples, dobles o triples para una cadena. Y un carácter literal es un carácter único rodeado de comillas simples o dobles.

Literales de cadena

```
strings = "This is Python"
char = "C"
multiline_str = """This is a multiline string with more than one line code."""
unicode = u"\u00dcnic\u00f6de"
raw_str = r"raw \n string"

print(strings)
print(char)
print(multiline_str)
print(unicode)
print(raw_str)
```

Esto es Python

C

Se trata de una cadena de varias líneas con más de un código de línea.

Ünicöde

cadena \ n sin procesar

Literales de cadena

En el programa anterior, `Esto es Python` es una cadena literal y `c` es un carácter literal.

El valor entre comillas triples `"""` asignado a la `multiline_str` es un literal de cadena de varias líneas.

La cuerda `u "\ u00dcnic \ u00f6de"` es un literal Unicode que admite caracteres distintos del inglés. En este caso, `\ u00dc` representa `Ü` y `\ u00f6` representa `ö`.

`r "cadena \ n sin procesar"` es un literal de cadena sin formato.



Literales booleanos

- Un literal booleano puede tener cualquiera de los dos valores: **True** o **False**.

Literales booleanos

- Ejemplo 8: ¿Cómo usar literales booleanos en Python?

```
x = (1 == True)
y = (1 == False)
a = True + 4
b = False + 10

print("x is", x)
print("y is", y)
print("a:", a)
print("b:", b)
```

- Salida

```
x es cierto
y es falso
a: 5
b: 10
```

Literales booleanos

En el programa anterior, usamos el literal booleano `True` y `False`. En Python, `True` representa el valor como `1` y `False` como `0`. El valor de `x` es `True` porque `1` es igual a `True`. Y, el valor de `y` es `False` porque `1` no es igual a `False`.

De manera similar, podemos usar `True` y `False` en expresiones numéricas como valor. El valor de `un` es `5` porque sumamos `True` que tiene un valor de `1` con `4`. Similar, `segundo` es `10` porque agregamos el `False` valor de tener `0` con `10`.

Literales especiales

- Python contiene un literal especial, es decir **None**. Lo usamos para especificar que el campo no ha sido creado.

Literales especiales

- Ejemplo 9: ¿Cómo usar literales especiales en Python?

```
drink = "Available"  
food = None  
  
def menu(x):  
    if x == drink:  
        print(drink)  
    else:  
        print(food)  
  
menu(drink)  
menu(food)
```

- Salida

```
Disponible  
Ninguna
```

Literales especiales

- En el programa anterior, definimos una **menú** función. En el interior **menu**, cuando establecemos el argumento como **drink** entonces, se muestra **Available**. Y, cuando el argumento es **food**, se muestra **None**.

Colecciones literales

- Hay cuatro colecciones literales diferentes: Literales de lista, literales de tupla, literales de Dict y literales de conjunto.

Colecciones literales

- Ejemplo 10: ¿Cómo usar colecciones de literales en Python?

```
fruits = ["apple", "mango", "orange"] #list
numbers = (1, 2, 3) #tuple
alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary
vowels = {'a', 'e', 'i', 'o', 'u'} #set

print(fruits)
print(numbers)
print(alphabets)
print(vowels)
```

- Salida

```
['manzana', 'mango', 'naranja']
(1, 2, 3)
{'a': 'manzana', 'b': 'pelota', 'c': 'gato'}
{'e', 'a', 'o', 'i', 'u'}
```

Colecciones literales

- En el programa anterior, creamos una lista de **frutas**, una tupla de **números**, un diccionario **dictar** tener valores con claves designadas para cada valor y un conjunto de **vocales**.