Sesión 1 - Introducción a Python

Contenidos:

---

1. ¡Hola Mundo! Básicos de Python.

Comentarios.

Para comenzar, aprenderemos a no escribir código en Python, es decir, realizar comentarios.

Un comentario de una línea tiene esta forma:

```
# Esto es un comentario
x = 5  # Lo podemos colocar tras una línea de código.
```

También podemos realizar comentarios utilizando comillas simples o dobles. Es la misma sintaxis que usaremos para las cadenas de texto.

```
'Podemos realizar comentarios utilizando texto normal'
"Sin embargo, de esta forma no podemos utilizarlo después de una variable"
a = 4 '(X) Esto esta prohibido!!'
```

Si queremos realizar comentarios de bloque, tenemos que utilizar la sintaxis de texto multilínea. Se realiza utilizando triples comillas simples o dobles.

```
''' Esto es
un comentario
en muchas
líneas'''
```

Función print().

Una forma recurrente de obtener información de un programa es mediante el uso de la función `print`. La función `print` imprime por pantalla cualquier cosa que se le pase como argumento. Por ejemplo:

```
print(5)  # -> 5
print('¡Hola mundo!')  # -> ¡Hola mundo!
print(1, 2, 3)  # -> 1 2 3
```

Podemos modificar un poco el formato de salida:

```
# Cambiar el separador:
print(1, 2, 3, sep=', ')  # -> 1, 2, 3

# Cambiar el carácter de finalización (por defecto salto de línea):
print('Hola a todos', end='!!!')  # -> Hola a todos!!!
```

Función input().

De forma análoga podemos obtener información con la función `input`. Esta acepta como argumento opcional los caracteres de espera de entrada (*promt*) y queda a la espera de la entrada de usuario.

Programa:

```
print('Introduzca su edad:')
edad = input('-->')
print('Su edad es de', edad, 'años.')
```

Consola:

```
Introduzca su edad:
-->
```

Consola tras introducir edad (número + enter):

```
Introduzca su edad:
--> 20
Su edad es de 20 años.
```

La salida de input es siempre texto. Más adelante veremos como cambiar esto.

2. Variables.

Una **variable** es un elemento que hace referencia a un objeto. (Ver diapositivas `01_Introducción_a_Python`).

Desde un punto de vista simplificado y pragmático, las variables son contenedores donde guardar datos. Según el dato que contengan las variables serán de un tipo u otro. Se verá más adelante.

Para crear una variable en Python solo debemos de asignarle algún valor.

```
a = 50  # Variable a de tipo entero que vale 50
texto = 'Esta variable guarda un texto'
```

Existen limitaciones a la hora de nombrar variables:

- Solo caracteres alfanuméricos y '_'.
- No pueden empezar por un número.
- No pueden contener espacios.
- No pueden llamarse igual que las palabras reservadas: `False`, `None`, `True` `and`, `as`, `assert`, `async`, `await`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`.

Hay una serie de recomendaciones a la hora de nombrar variables:

- Aunque se aceptan caracteres no ascii, es mejor no utilizarlos.

  ```
  españa = 4  # La ñ no es ascii.
  漢字 = 17  # Se acepta símbolos en caulquier lenguaje
  𓅓 = 'osiris'  # Sin comentarios
  ```

- Las variables que guardan datos preferiblemente en minúscula separadas por `'_'`.

  ```
  nombre_largo_de_variable = 60
  ```

- Que prime la claridad y legibilidad:

  ```

  ```

```
x = 24  # No da mucha información.
distancia_cm = 24  # Mejor
```

En Python no existen las constantes como tal, pero podemos indicar que una varible no debería ser modificada poniendo el nombre completo con mayúsculas.

```
VERSION = 4.0  # Variable de solo lectura (aunque se pueda sobrescribir).
```

En sucesivas sesiones veremos más consejos para nombrar otros elementos.

3. Operadores.

Los operadores son elementos del lenguaje que interacturan con las variables. Sirven para realizar transformaciones sobre los objetos a los que referencian.

Por ejemplo, si queremos asignar a una variable `x` el valor correspondiente al valor de la variable entera `a` más 5 unidades utilizaremos los operadores suma (+) y asignación (=).

```
x = a + 5  # Suma y asignación.
```

El resultado de la suma del objeto que referencia `a` con el objeto entero de valor `5` se asigna a la variable `x`.

A continuación se muestra una relación de operadores disponibles en Python:

- **Aritméticos**. Realizan operaciones aritméticas.

```
4 + 5   # -> 9 // Suma (+)
6 - 3   # -> 3 // Resta (-)
4 * 5   # -> 20 // Multiplicación (*)
2 ** 3  # -> 8 // Potencia (**)
5 / 2   # -> 2.5 // División en coma flotante (/)
5 // 2  # -> 2  // División entera (//)
5 % 2   # -> 1 // Resto de división (%)
```

- **Operadores a nivel de bit**. Se verán en detalle más adelante. Son `&`, `|`, `^`, `>>` y `<<`.
- **Asignación**. Asigna objetos a variables.

```
x = 4        # a la variable x se le ha asigando el objeto entero 4
x, y = 2, 3  # Asignaciones en paralelo (x = 2, y = 3)
```

- **Asignación combinada**. Combina cualquier operador aritmético o a novel de bit con asignación.

```
x = 4
x *= 2  # -> 8 // Equivale a x = x * 2
```

- **Operadores de Comparación**. Comparan valores. Devuelven `True` o `False`.

```
x, y = 4, 2
x == 4    # -> True // Igualdad (==)
y != 2    # -> False // Desigualdad (!=)
x > 3     # -> True // Mayor que (>)
y >= 2    # -> True // Mayor o igual (>=)
x < 4     # -> False // Menor que (<)
x <= 4    # -> True  // Menor o igual (<=)
```

- **Booleanos**. Operan con `True` o `False` y realizan operaciones lógicas.

```
x, y = True, False
not x       # -> False // Negación (not)
x and True  # -> True  // Conjunción lógica (and)
x or y      # -> True  // Disyunción lógica (or)
(not x and y) or (not y and x)  # No existe xor
```

- **Identidad**. Muestran relaciones de identidad entre dos objetos. Son `is` e `in`. Se verán en detalle en futuras sesiones.

Cada operador se comportará de forma distinta dependiendo del tipo de las variables involucradas en la operación. No es lo mismo el operador suma (`+`) para números (los sumará) que para texto (junta dos textos).

Practica

Realiza las siguientes operaciones:

- 4 veces 6 elevado a 4.
- La división entera de 1223 entre 67.
- El resto de dividir 65964 entre 324.
- Comprueba que se cumple la propiedad de la división entera con 37 entre 7.
- Comprueba que 43 es menor que 53 y a su vez mayor que 24.
- Asigna el valor 2 a una variable y duplica, triplica y decuplica su valor.

4. Tipos de datos.

Dentro del ordenador la información se guarda en bruto en codificación binaria con unos y ceros. Necesitamos de un discriminador que aporte información de cómo interpretar esa información. Ese discriminador es lo que llamamos **tipo**.

En Python llamamos tipo a la **clase** del objeto. Esto no es así en otros lenguajes. Sin embargo en Python todo son objetos y por tanto están definidos mediante "plantillas" que indican qué datos guardan los objetos y qué podemos hacer con ellos. Podemos obtener información del tipo de dato de una variable con la función `type()`.

```
x = 5
print(type(x))  # -> int
```

Existen una serie de tipos base que vienen incluidos en Python que nos permiten trabajar con los datos más comunes. Podemos dividirlos entre tipos **mutables** (cuyo valor contenido en los objetos puede variar) e **inmutables** (es fijo y no puede cambiar).

Los tipos inmutables son los que podríamos relacionar con **literales**, es decir, datos que podemos representar directamente como números o texto.

Tipos inmutables.

En Python tenemos los siguientes tipos inmutables:

- `int`: representa números enteros de tamaño arbitrario.
- `float`: representa números en coma flotante de precisión arbitraria.
- `complex`: representa números complejos.
- `str`: representa texto. No existen los caracteres en solitario.
- `bytes`: reprenta un bloque de bytes en bruto.

Cada uno de estos tipos está asociado con uno o más **literales**:

```
a = 5  # Número sin decimales -> int
b = 1.2  # Número con decimales -> float
c = 1e3  # Númeroen notación científica -> float
d = 1+3j  # Número complejo -> complex
e = 'Hola mundo!'  # Texto -> str
f = b'\x01\x02\x03\x04'  # Cadena de bytes en hexadecimal -> bytes
```

Conversión de tipos.

Podemos realizar conversión entre tipos utilizando el identificador para cada clase y pasando los datos que queramos convertir como argumento.

```
texto_numero = '25'
numero = int(texto_numero)
print(numero, type(numero))  # -> 25, int

temperatura = 15
temperatura_texto = str(temperatura)
print(temperatura_texto, type(temperatura_texto))  # -> 15, str
```

```
codigo = '0xf3'
valor_codigo = int(codigo, 16)  # Indicamos la base
print(codigo, valor_codigo)  # -> 0xf3 243

binario = '0b0010'
valor_binario = int(binario, 0)  # Con 0 le pedimos que lo deduzca
print(binario, valor_binario)  # -> 0b0010 2
```

No todos los tipos se pueden convertir unos a otros.

```
numero_texto = '25'
numero = int(numero_texto)
print(numero, type(numero))  # -> 25, int
```

5. Operaciones con strings.

Sobre los strings.

Los **strings** son cadenas de carácteres donde guardamos información textual. Son muy útiles por razones obvias: la información está en un formato que entendemos las personas. Los strings son tipos de datos inmutables, es decir, no puedes cambiar el valor de una cadena una vez asignado.

Los **strings** se pueden definir con comillas simples `'` o dobles `"` y en una o varias líneas:

```
texto_simples = 'Hola Mundo'
texto_dobles = "Hola mundo"
texto_multi_simple = '''Hola
esto es
texto en
varias lineas'''
texto_multi_doble = """Esto también.
Y puedo usar comillas simples sin 'miedo' a que pase nada."""
texto_unicode = '🐍🐍🐍'  # Vale cualquier texto incluido emojis.
```

Nos referimos al tipo texto como cadenas (*strings*) porque están definidas como una sucesión de carácteres encadenados a los que podemos acceder individualmente.

```
texto = 'Hola Mundo'
print(texto[0]) # imprime 'H'
print(texto[-1]) # imprime la última letra  'o'
print(type(texto[3])) # -> str // No existe el tipo char
texto[5] = 'y' # error, no puedes reasignar
```

> Importante: los índices en Python empiezan en 0.

También podemos partir y obtener trozos de cadena usando el símbolo de dos puntos `:`. A esto se le llama *slicing* y su sintaxis es:

```
texto[inicio:(final):(salto)]
```

Veámoslo en ejemplos.

```
A = 'Hola Mundo!'
print(A[5:])  # imprime Mundo!
print(A[:4])  # imprime Hola
print(A[2:7])  # imprime "la Mu"
print(A[::2])  # imprime "Hl ud!"
print('patatas'[2:6])  # imprime "tata", también vale con literales
print(A[::-1])  # imprime "!odnuM aloH", da la vuelta a la cadena.
```

Operadores de strings.

Los operadores disponibles para strings son los siguientes:

- Suma (`+`): concatena dos cadenas. E.g. `"Hola" + " Mundo!"` → `"Hola Mundo!"`
- Producto (`*`) : junto a un entero repite la cadena. E.g. `"ja" * 4` → `"jajajaja"`
- Dentro de (`in`): determina si hay una subcadena dentro de otra. E.g. `"melaza" in "patatas y melaza"` → `True`

También cabe mencionar como un operador a la función `len()`, que nos devuelve la longitud de la cadena. E.g. `len('hola')` → `4`

Practica

Intenta adivinar la salida de las siguientes expresiones:

```
A = 'Python!'
print(len(A))
print(A[-7])
print(A[:-1])
print(A[-2:])
print(A[7::-2])
print('p' in A)
```

Métodos de strings

Hemos mencionado que las clases tienen funciones que pueden llamarse para interaccionar con los datos que contienen los objetos. Llamamos a estas funciones **métodos**. Para acceder a los métodos hay que usar el operador de acceso punto (`.`). Por ejemplo el siguiente método devuelve una copia del texto en minúsculas

```
print("HOLA".lower())  # imprime "hola"
```

En el caso de los strings los métodos siempre nos van a devolver **copias de la cadena modificada** y nunca modificará los datos de la original (¡recuerda que `str` es un tipo de dato inmutable!).

Tenemos una lista de todos los métodos en la referencia oficial de Python. Ver

Algunos métodos útiles son:

- `capitalize()`: Pone la primera letra en mayúscula.
- `lower()`: Letras a minúsculas
- `upper()`: Letras a mayúsculas
- `find(substring)`: Devuelve la primera posición de la subcadena si la encuentra. -1 si no.
- `count(substring)`: Cuenta el número de veces que aparece una subcadena.
- `replace(old, new, (n))`: Devuelve una copia del string remplazando todas las subcadenas `old` por la cadena `new`. Si se le indica `n` lo solo hara n veces.
- `startswith(substring)`: Verdadero si coincide el primer substring.
- `endswith(substring)`: Verdadero si coincide el último substring
- `isalpha()`: Verdadero si solo hay letras en la cadena
- `isnumeric()`: Verdadero si solo hay numeros en la cadena

Hay muchos más, así como modificaciones y variaciones de los aquí presentados.

Formato a strings.

A veces es necesario introducir en una cadena algún dato con el que estemos trabajando con el formato que queremos. Para ello hay varios métodos, los más usados son:

- Usando el método `.format(...)`: con este modo podemos sustituir en una cadena posiciones y nombres por variables y valores.

  Programa:

  ```
  a = 'Hola a {}'.format("todos")
  b = 'Mis hermanos pequeños son {0} y {1}. La más pequeña es {1} con
  {2} años.'.format('Miguel', 'Marta', 14)
  c = 'Si dividimos el dinero entre los 3 tocamos a {:.2f}
  €.'.format(10000/3)
  d = 'La x vale {x}.'.format(x=5)
  print(a, b, c, d, sep='\n')
  ```

  Consola:

  ```
  Hola a todos
  Mis hermanos pequeños son Miguel y Marta. La más pequeña es Marta con
  14 años.
  Si dividimos el dinero entre los 3 tocamos a 3333.33€.
  La x vale 5.
  ```

- Usando **fstrings**: anteponiendo una `f` a la cadena podemos formatear in-situ. Simplemente hay que indicar el contenido entre las llaves.

  Programa:

  ```
  x, y, z = 1, 2, 3
  a = f'x + y = {x+y}'
  b = f'y / z = {y/z:.3f}'
  c = f'z vale {z:04d}' # al menos 4 números con ceros delante.
  print(a, b, c, sep='\n')
  ```

  Consola:

  ```
  x + y = 3
  y / z = 0.667
  z vale 0003
  ```

Ambos métodos usan la sintaxis para el formato de `printf` de C (utilizado también en Java). Aquí una referencia para C++.

Ejercicios para clase.

E1: Formulario.

Rellena y muestra el siguiente formulario a través de la terminal utilizando input y print:

Nombre:
Apellidos:
NIF:
Edad:
Aficiones:

E2: Extraer Información.

Recibimos cada pocos minutos mensaje desde una estación meteorológica con este formato:

> "temp:00X-hum:X-lluvia:[si|no]"

Es decir, la temperatura en kelvin con precisión de un grado, la humedad en porcentaje relativo (10-99) y 'si' o 'no' cuando hay lluvia.

Un ejemplo de mensaje puede ser: "temp:290-hum:54-lluvia:si"

Extrae la información y muestra por pantalla la misma información con mejor formato y la temperatura en grados centígrados.

Apéndice:

Glosario

- **Sintaxis**: conjunto de normas que definen la correcta secuenciación de elementos en un lenguaje de programación.
- **Expresión**: combinación de valores, variables, operadores y funciones que da por resultado un valor.