Primera sentencia en Python: `print`

In [6]:
```
print("Hola mundo")
```

```
Hola mundo
```

In [1]:
```
print("Bienvenidxs a DAW")
```

```
Bienvenidxs a DAW
```

In [2]:
```
print(25)
```

```
25
```

print es el nombre de una **función**. Las funciones son capaces de:

- Ejecutar una acción y causar un efecto
- Devolver algún valor
- Ambas cosas

Las funciones pueden:

- Estar integradas en Python
- Provenir de los complementos de Python
- Escritas por el programador

Las funciones pueden recibir argumentos, que se indican entre paréntesis. Aunque no reciban parámetros, deben indicarse igualmente los paréntesis. El argumento de print es "Hola mundo".

Cuando el intérprete llega a una función:

- Deja el código por un momento y salta dentro de la función que se desea invocar; por lo tanto, también toma los argumentos y los pasa a la función.
- La función ejecuta el código, provoca el efecto deseado (si lo hubiera), calcula el (los) resultado(s) deseado(s) y termina la tarea.
- Finalmente, Python regresa al código (al lugar inmediatamente después de la invocación) y reanuda su ejecución.

Bloque de ejercicios 1

1. Utiliza la función `print()` para imprimir la linea "¡Hola, Mundo!" en la pantalla.

In [7]:
```
print("Hola mundo")
```

```
Hola mundo
```

2. Una vez hecho esto, utiliza la función `print()` nuevamente, pero esta vez imprime tu nombre.

In [8]:
```
print("Marta")
```

```
Marta
```

3. Elimina las comillas dobles y ejecuta el código. Observa la reacción de Python. ¿Qué tipo de error se produce?

In [9]:
```
print(Hola mundo)
```

```
  Cell In[9], line 1
    print(Hola mundo)
          ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

Piensa que son dos variables y le falta la coma en medio para separar los argumentos.

In [10]:
```
print(hola)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 print(hola)

NameError: name 'hola' is not defined
```

Piensa que hola es una variable pero nos avisa de que no le hemos dicho qué contiene.

4. Luego, elimina los paréntesis, vuelve a poner las comillas dobles y vuelve a ejecutar el código. ¿Qué tipo de error se produce esta vez?

In [11]:
```
print"Hola mundo"
```

```
  Cell In[11], line 1
    print"Hola mundo"
    ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print
(...)?
```

5. Experimenta tanto como puedas. Cambia las comillas dobles a comillas simples.

In [12]:
```
print('Hola mundo')
```

```
Hola mundo
```

6. Utiliza múltiples funciones `print()` en la misma línea:

In [13]:
```
print("Hola")print("Mundo")
```

```
  Cell In[13], line 1
    print("Hola")print("Mundo")
                 ^
SyntaxError: invalid syntax
```

In [14]:
```
print("Hola"); print("Mundo")
```

```
Hola
Mundo
```

7. Utiliza múltiples funciones `print()` en líneas diferentes.

In [4]:
```
print("Uno")
print()
print("Dos")
```

```
Uno

Dos
```

Efectos

- Toma los argumentos
- los convierte (si es necesario) en un formato legible para el ser humano
- envía los resultados al dispositivo de salida.

In [15]:
```
print(2)
print("2")
```

```
2
2
```

In [16]:
```
print("Hola", 2, "DAW")
```

```
Hola 2 DAW
```

In [18]:
```
print("2+3")
print(2+3)
```

```
2+3
5
```

Argumentos

- Prácticamente todos los tipos de datos ofrecidos por Python.

Resultado devuelto

No devuelve resultados

Sintaxis básica de Python

- Es *caseSensitive*: diferencia mayúsculas de minúsculas

In [19]:
```
Print("Hola mundo")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 Print("Hola mundo")

NameError: name 'Print' is not defined
```

- Cada sentencia en una línea
- Las sentencias se ejecutan en orden
- No se pone ; al final de cada sentencia (excepto que dos sentencias se encuentren en la misma línea)

¿Qué pasa si ejecutamos una sentencia `print()` sin parámetros?

In [20]:
```
print("Adiós ríos, adiós fontes,")
print()
print("adiós regatos pequenos;")
```

```
Adiós ríos, adiós fontes,

adiós regatos pequenos;
```

Secuencias de escape

Secuencia	Significado
\n	Fin de línea
\"	"
\'	'
\\	\

In [21]:
```
print("Adiós vista dos meus ollos,\nnon sei cando nos veremos.")
```

```
Adiós vista dos meus ollos,
non sei cando nos veremos.
```

PREGUNTA ¿Cómo hacemos para representar las "? ¿Y las barras invertidas ?

In [22]:
```
print("Marta dijo "programad en Python"")
```

```
  Cell In[22], line 1
    print("Marta dijo "programad en Python"")
          ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

In [25]:
```
print('Marta dijo "programad en Python"')
print("Marta dijo 'programad en Python'")
```

```
Marta dijo "programad en Python"
Marta dijo 'programad en Python
```

In [27]:
```
print("Marta dijo \"programad en Python\"")
print("Marta dijo \'programad en Python\'")
```

```
Marta dijo "programad en Python"
Marta dijo 'programad en Python'
```

In [28]:
```
print("C:\Users")
```

```
  Cell In[28], line 1
    print("C:\Users")
                   ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in pos
ition 2-3: truncated \UXXXXXXXX escape
```

In [29]:
```
print("C:\\Users")
```

```
C:\Users
```

Función `print()` con varios argumentos

In [30]:
```
print("Uno", "Dos", "Tres")
```

```
Uno Dos Tres
```

Argumentos de palabra clave

- Los **argumentos posicionales** son aquellos cuyo significado viene dictado por su posición, por ejemplo, el segundo argumento se emite después del primero, el tercero se emite después del segundo, etc.
- Los **argumentos de palabra clave** son aquellos cuyo significado no está dictado por su ubicación, sino por una palabra especial (palabra clave) que se utiliza para identificarlos.

Los **argumentos de palabra clave** de print son:

- `end` Indica el carácter que se escribe después de haber escrito todos (no cada uno) los argumentos.
- `sep` Indica el carácter que se escribe entre cada par de argumentos.

In [31]:
```
print("Adiós vista dos meus ollos","non sei cando nos veremos.", sep="\n")
```

```
Adiós vista dos meus ollos
non sei cando nos veremos.
```

In [32]:
```
print("H", "O", "L", "A", sep="-")
```

```
H-O-L-A
```

In [33]:
```
print("Uno", "Dos", "Tres", sep="")
```

```
UnoDosTres
```

`""` es una cadena vacía.

In [34]:
```
print("Bienvenidxs a todxs", end="*")
print("a la clase de primero de DAW")
```

```
Bienvenidxs a todxs*a la clase de primero de DAW
```

Bloque de ejercicios 2

1. Escribe dos sentencias `print` para que la salida sea:

```
Fundamentos***Programación***en...Python
```

In [35]:
```
print("Fundamentos", "***", "Programación", "***", sep="", end="")
print("en", "...Python")
```

```
Fundamentos***Programación***en ...Python
```

Solución más optimizada:

In [37]:
```
print("Fundamentos", "Programación", "en", sep="***", end="...")
print("Python")
```

```
Fundamentos***Programación***en...Python
```

A partir de este código:

In [1]:
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
    *
   * *
  *   *
 *     *
***   ***
  *   *
  *   *
  *****
```

Modifícalo para:

2. Minimizar el número de invocaciones de la función print() insertando la secuencia \n en las cadenas.

In [4]:
```
print("    *", "   * *", "  *   *", " *     *",
      "***   ***", "  *   *", "  *   *", "  *****", sep ="\n")
```

```
    *
   * *
  *   *
 *     *
***   ***
  *   *
  *   *
  *****
```

Esta es una solución posible pero no es recomendable, puesto que no genera CÓDIGO LIMPIO, no se ve, de un vistazo, qué hace el código.

3. Duplicar la flecha, colocando ambas flechas lado a lado; nota: una cadena se puede multiplicar usando el siguiente truco: `"cadena" * 2` producirá `"cadenacadena"`

In [3]:
```
print("Hola" * 3)
```

```
HolaHolaHola
```

In [7]:
```
print("    *     " * 2)
print("   * *    " * 2)
print("  *   *   " * 2)
print(" *     *  " * 2)
print("***   *** " * 2)
print("  *   *   " * 2)
print("  *   *   " * 2)
print("  *****   " * 2)
```

```
    *         *
   * *       * *
  *   *     *   *
 *     *   *     *
***   *** ***   ***
  *   *     *   *
  *   *     *   *
  *****     *****
```

4. Elimina cualquiera de las comillas y observa detenidamente la respuesta de Python; presta atención a donde Python ve un error: ¿es el lugar en donde realmente existe el error? Haz lo mismo con algunos de los paréntesis.

In [8]:
```
print("    *")
print("   * *")
print("  *   *)
print(" *     *")
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
  Cell In[8], line 3
    print("  *   *)
          ^
SyntaxError: unterminated string literal (detected at line 3)
```

In [9]:
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print(***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
  Cell In[9], line 5
    print(***   ***")
                ^
SyntaxError: unterminated string literal (detected at line 5)
```

In [10]:
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *"
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
  Cell In[10], line 4
    print(" *     *"
         ^
SyntaxError: '(' was never closed
```

In [12]:
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print"***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
  Cell In[12], line 5
    print"***   ***")
                    ^
SyntaxError: unmatched ')'
```

5. Reemplaza algunas de las comillas por apóstrofes (comillas simples); observa lo que pasa detenidamente.

In [13]:
```
print("    *")
print("   * *")
print('  *   *")
print(" *     *")
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

```
  Cell In[13], line 3
    print('  *   *")
          ^
SyntaxError: unterminated string literal (detected at line 3)
```

In [14]:
```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print('***   ***')
print("  *   *")
print("  *   *")
print("  *****")
```

```
    *
   * *
  *   *
 *     *
***   ***
  *   *
  *   *
  *****
```