In [1]:
```
print("Hola mundo!")
```

```
Hola mundo!
```

😎 Elementos de un programa en Python

Un programa de Python es un archivo de texto (normalmente guardado con el juego de caracteres UTF-8) que contiene expresiones y sentencias del lenguaje Python. Esas expresiones y sentencias se consiguen combinando los elementos básicos del lenguaje. El lenguaje Python está formado por elementos (tokens) de diferentes tipos:

- **palabras reservadas (keywords)**
- **funciones integradas (built-in functions)**
- **literales**
- **operadores**
- **delimitadores**
- **identificadores**

Para que un programa se pueda ejecutar, el programa debe ser sintácticamente correcto, es decir, utilizar los elementos del lenguaje Python respetando su reglas.

Líneas y espacios

Un programa de Python está formado por líneas de texto, por ejemplo:

In [2]:
```
radio = 5
area = 3.14159242 * radio ** 2
print(area)
```

```
78.5398105
```

Se recomienda que cada línea contenga una única instrucción, aunque puede haber varias instrucciones en una línea, separadas por un punto y coma (;) Por motivos de legibilidad, se recomienda que las líneas no superen los 79 caracteres. Si una instrucción supera esa longitud, se puede dividir en varias líneas usando el caracter contrabarra (\)

In [3]:
```
radio = 5
area = 3.14159265358979323846 \
* radio ** 2
print(area)
```

```
78.53981633974483
```

Los elementos del lenguaje se separan por espacios en blanco (normalmente, uno), aunque en algunos casos no se escriben espacios:

- **Entre los nombres de las funciones y el paréntesis**
- **Antes de una coma (,)**
- **Entre los delimitadores y su contenido (paréntesis, llaves, corchetes o comillas)**

Excepto al principio de una línea, los espacios no son significativos, es decir, da lo mismo un espacio que varios.

Los espacios al principio de una línea (el sangrado) son significativos porque indican un nivel de agrupamiento.
El sangrado inicial es una de las características de Python que lo distingue de otros lenguajes, por eso una línea no puede contener espacios iniciales, a menos que forme parte de un bloque de instrucciones o de una instrucción dividida en varias líneas. Al ejecutar una instrucción con espacios iniciales, se mostrará un aviso de error de sintaxis.

Palabras reservadas (keywords)

Las palabras reservadas de Python son las que forman el núcleo del lenguaje Python y son las siguientes:

> **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** Estas palabras no pueden utilizarse para nombrar otros elementos (variables, funciones, etc.), aunque pueden aparecer en cadenas de texto.

Literales

Los literales son los datos simples que Python es capaz de manejar:

> números: valores lógicos, enteros, decimales y complejos, en notación decimal, octal o hexadecimal

> cadenas de texto

Operadores

Los operadores son los caracteres que definen operaciones matemáticas (lógicas y aritméticas). Son los siguientes:

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

Delimitadores

Los delimitadores son los caracteres que permiten delimitar, separar o representar expresiones. Son los siguientes: >

```
'       "       #       \
(       )       [       ]       {       }
,       :       .       ;       @       =       ->
+=      -=      *=      /=      //=     %=      @=
&=      |=      ^=      >>=     <<=     **=
```

Identificadores

Los identificadores son las palabras que se utilizan para nombrar elementos creados por el programador.

Esos elementos pueden ser **variables u objetos** que almacenan información, **funciones** que agrupan instrucciones, **clases** que combinan ambos, **módulos** que agrupan los elementos anteriores, etc.

Los identificadores están formados por letras (mayúsculas y minúsculas) números y el carácter guion bajo (_). El primer caracter del identificador debe ser una letra.

Funciones integradas (built-in functions)

Una función es un bloque de instrucciones agrupadas, que permiten reutilizar partes de un programa. Python incluye las siguientes funciones de forma predeterminada (es decir, estas funciones siempre están disponibles):

abs() - dict() - help() - min() - setattr() - all() - dir() - hex() - next() - slice() - any() - divmod() - id() - object() - sorted() - ascii() - enumerate() - input() - oct() - staticmethod() - bin() - eval() - int() - open() - str() - bool() - exec() - isinstance() - ord() - sum() - bytearray() - filter() - issubclass() - pow() - super() - bytes() - float() - iter() - print() - tuple() - callable() - format() - len() - property() - type() - chr() - frozenset() - list() - range() - vars() - classmethod() - getattr() - locals() - repr() - zip() - compile() - globals() - map() - reversed() -**import**() - complex() - hasattr() - max() - round() - delattr() - hash() - memoryview() - set()

Los nombres de las funciones integradas se pueden utilizar para nombrar variables, pero entonces las funciones ya no estarán disponibles en el programa. Si se eliminan las variables, las funciones vuelven a estar disponibles.

Funciones adicionales

- Un programa **puede definir nuevas funciones o redefinir las funciones integradas.**
- Los nombres de las funciones **no pueden coincidir con las palabras reservadas.**
- Un programa puede también **importar nuevas funciones que se encuentran definidas en otros archivos llamados módulos.** Python incluye una biblioteca de módulos (Biblioteca estándar) especializados en todo tipo de tareas. Además de la biblioteca estándar, existen miles de módulos escritos por diferentes programadores y accesibles en Internet. El principal repositorio de módulos es el Python Package Index (Índice de paquetes de Python), más conocido por PyPI.

Nombres de archivos

Se recomienda que los nombres de los archivos de programas:

- Estén escritos en minúsculas
- Utilicen guiones bajos para separar palabras o números Se recomienda también seguir un criterio común, por ejemplo, que el nombre del programa contenga una referencia a la práctica de ejercicios .

Comentarios adicionales

Los programas de Python **no necesitan realmente definir una función main()**. Es decir, los dos programas siguientes producirían el mismo resultado:

In [4]:
```
# Programa con función main():
def main():
    print("¡Hola, mundo!")

if __name__ == "__main__":
    main()
```

```
¡Hola, mundo!
```

In [5]:
```
# Programa sin función main()

print("¡Hola, mundo!")
```

```
¡Hola, mundo!
```

La diferencia entre ambos programas sólo se percibiría en caso de importarlos.

Salida por pantalla: la función print()

Salida por pantalla en el entorno interactivo

En un entorno interactivo para que Python nos muestre el valor de una variable basta con escribir su nombre.

In [6]:
```
a=2
a
```

Out[6]: 2

También se puede conocer el valor de varias variables a la vez escribiéndolas entre comas (el entorno interactivo las mostrará entre paréntesis), como muestra el siguiente ejemplo:

In [7]:
```
a = b = 2
c = 'Pepe'
a
```

Out[7]: 2

In [8]:
```
c,b
```

Out[8]: ('Pepe', 2)

La función print()

En los programas, para que Python nos muestre texto o variables hay que utilizar la función **print()**. La función print() permite mostrar texto en pantalla.

El texto a mostrar se escribe como argumento de la función:

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

```
Hola
```

Las cadenas se pueden delimitar tanto por comillas dobles **(")** como por comillas simples **(')**

In [10]:
```
print('Hola')
```

```
Hola
```

- La función **print()** admite varios argumentos seguidos.
- En el programa, los argumentos deben separarse por comas.
- Los argumentos se muestran en el mismo orden y en la misma línea, separados por espacios:

In [11]:
```
print("Hola", "Adiós")
```

```
Hola Adiós
```

Cuando se trata de dos cadenas seguidas, se puede no escribir comas entre ellas, pero las cadenas se escribirán seguidas, sin espacio en blanco entre ellas:

In [12]:
```
print("Hola" "Adiós")
```

```
HolaAdiós
```

Al final de cada print(), Python añade automáticamente un salto de línea:

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

```
Hola
Adiós
```

Para generar una línea en blanco, se puede escribir una orden print() sin argumentos

In [14]:
```
print("Hola")
print()
print("Adiós")
```

```
Hola

Adiós
```

Si no se quiere que Python agregue un salto de línea al final de un print(), se debe agregar al final el argumento end="":

In [15]:
```
print("Hola", end="")
print("Adiós")
```

```
HolaAdiós
```

En el ejemplo anterior, las dos cadenas se muestran pegadas. Si se quieren separar los argumentos en la salida, hay que incluir los espacios deseados (bien en la cadena, bien en el argumento end):

In [16]:
```
print("Hola. ", end="")
print("Adiós")
```

```
Hola. Adiós
```

In [17]:
```
print("Hola.", end=" ")
print("Adiós")
```

```
Hola. Adiós
```

Como las comillas indican el principio y el final de una cadena, si se escriben comillas dentro de comillas se produce un error de sintaxis.

In [18]:
```
print("Un persona le dice a otra: "¿Cómo estás?""
```

```
  Input In [18]
    print("Un persona le dice a otra: "¿Cómo estás?""
                                       ^
SyntaxError: invalid character '¿' (U+00BF)
```

Nota: Si nos fijamos en la forma como el editor colorea la instrucción, podemos darnos cuenta de que hay un error en ella.

Como las cadenas empiezan y acaban con cada comilla, el editor identifica dos cadenas y un texto en medio que no sabe lo que es. Para incluir comillas dentro de comillas, se puede escribir una contrabarra () antes de la comilla para que Python reconozca la comilla como carácter, no como delimitador de la cadena:

In [19]:
```
print("Una persona le dice a otra: \"¿Cómo estás?\"")
print('Y la otra le contesta: \'¡Muy bien!\'')
```

```
Una persona le dice a otra: "¿Cómo estás?"
Y la otra le contesta: '¡Muy bien!'
```

O escribir comillas distintas a las utilizadas como delimitador de la cadena:

In [20]:
```
print("Una persona le dice a otra: '¿Cómo estás?'")
print('Y la otra le contesta: "¡Muy bien!"')
```

```
Una persona le dice a otra: '¿Cómo estás?'
Y la otra le contesta: "¡Muy bien!"
```

La función print() permite incluir variables o expresiones como argumento, lo que nos permite combinar texto y variables:

In [21]:
```
nombre = "Pepe"
edad = 25
print("Me llamo", nombre, "y tengo", edad, "años.")
```

```
Me llamo Pepe y tengo 25 años.
```

In [22]:
```
semanas = 4
print("En", semanas , "semanas hay", 7 *
semanas, "días.")
```

```
En 4 semanas hay 28 días.
```

La función print() muestra los argumentos separados por espacios, lo que a veces no es conveniente. En el ejemplo siguiente el signo de exclamación se muestra separado de la palabra.

In [23]:
```
nombre = "Pepe"
print("¡Hola,", nombre, "!")
```

```
¡Hola, Pepe !
```

A partir de Python 3.6 se pueden utilizar las cadenas "f".

In [24]:
```
nombre = "Pepe"
print(f"¡Hola, {nombre}!")
```

```
¡Hola, Pepe!
```

Entrada por teclado: la función input()

La función **input()** permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla enter, como muestra el siguiente ejemplo:

In [25]:
```
print("¿Cómo se llama?")
nombre = input()
print(f"Me alegro de conocerle, {nombre}")
```

```
¿Cómo se llama?
Pepito
Me alegro de conocerle, Pepito
```

En el ejemplo anterior, el usuario escribe su respuesta en una línea distinta a la pregunta porque Python agrega un salto de línea al final de cada print().

Si se prefiere que el usuario escriba su respuesta a continuación de la pregunta, se podría utilizar el argumento opcional end en la función print(), que indica el carácter o caracteres a utilizar en vez del salto de línea. Para separar la respuesta de la pregunta se ha añadido un espacio al final de la pregunta.

In [26]:
```
print("¿Cómo se llama? ", end="")
nombre = input()
print(f"Me alegro de conocerle, {nombre}")
```

```
¿Cómo se llama? Pepito
Me alegro de conocerle, Pepito
```

Otra solución, más compacta, es aprovechar que a la función input() se le puede enviar un argumento que se escribe en la pantalla (sin agregar un salto de línea):

In [27]:
```
nombre = input("¿Cómo se llama? ")
print(f"Me alegro de conocerle, {nombre}")
```

```
¿Cómo se llama? Pepito
Me alegro de conocerle, Pepito
```

Conversión de tipos

De forma predeterminada, la función **input()** convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error:

In [29]:
```
cantidad = input("Dígame una cantidad en pesos: ")
print(f"{cantidad} pesos son {round(cantidad / 83.0, 2)} dólares")
```

```
Dígame una cantidad en pesos: 100

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [29], in <cell line: 2>()
      1 cantidad = input("Dígame una cantidad en pesos: ")
----> 2 print(f"{cantidad} pesos son {round(cantidad / 83.0, 2)} dólares")

TypeError: unsupported operand type(s) for /: 'str' and 'float'
```

Si se quiere que Python interprete la entrada como un número entero, se debe utilizar la función **int()** de la siguiente manera:

In [30]:
```
cantidad = int(input("Dígame una cantidad en pesos: "))
print(f"{cantidad} pesos son {round(cantidad / 183.0, 2)} dólares")
```

```
Dígame una cantidad en pesos: 100
100 pesos son 0.55 dólares
```

De la misma manera, para que Python interprete la entrada como un número decimal, se debe utilizar la función **float()** de la siguiente manera:

In [31]:
```
cantidad = float(input("Dígame una cantidad en dólares (hasta con 2 decimales): "))
print(f"{cantidad} dólares son {round(cantidad * 183.0)} pesos")
```

```
Dígame una cantidad en dólares (hasta con 2 decimales): 100.25
100.25 dólares son 18346 pesos
```

Pero si el usuario **NO** escribe un número, las funciones int() o float() producirán un error:

In [32]:
```
cantidad = float(input("Dígame una cantidad en dólares (hasta con 2 decimales): "))
print(f"{cantidad} dólares son {round(cantidad * 83.0)} pesos")
```

```
Dígame una cantidad en dólares (hasta con 2 decimales):

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [32], in <cell line: 1>()
----> 1 cantidad = float(input("Dígame una cantidad en dólares (hasta con 2 decimales): "))
      2 print(f"{cantidad} dólares son {round(cantidad * 83.0)} pesos")

ValueError: could not convert string to float: ''
```

De la misma manera, si el usuario escribe un número decimal, la función int() producirá un error:

In [33]:
```
edad = int(input("Dígame su edad: "))
print(f"Su edad son {edad} años")
```

```
Dígame su edad: 25.5

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [33], in <cell line: 1>()
----> 1 edad = int(input("Dígame su edad: "))
      2 print(f"Su edad son {edad} años")

ValueError: invalid literal for int() with base 10: '25.5'
```

Pero si el usuario escribe un número entero, la función float() no producirá un error, aunque el número se escribirá con parte decimal (.0):

In [34]:
```
peso = float(input("Dígame su peso en kg: "))
print(f"Su peso es {peso} kg")
```

```
Dígame su peso en kg: 68
Su peso es 68.0 kg
```

Variables como argumento de la función input()

La función input() sólo puede tener un argumento, pero las **cadenas "f"** permiten incorporar variables en el argumento de la función input()

In [35]:
```
nombre = input("Dígame su nombre: ")
apellido = input(f"Dígame su apellido, {nombre}: ")
print(f"Me alegro de conocerle, {nombre} {apellido}.")
```

```
Dígame su nombre: Pepito
Dígame su apellido, Pepito: Pepe
Me alegro de conocerle, Pepito Pepe.
```

In [36]:
```
numero1 = int(input("Dígame un número: "))
numero2 = int(input(f"Dígame un número mayor que {numero1}: "))
print(f"La diferencia entre ellos es {numero2 - numero1}.")
```

```
Dígame un número: 23
Dígame un número mayor que 23: 45
La diferencia entre ellos es 22.
```

In [ ]:
```

```