UT. Python I

Introducción

ÍNDICE DE CONTENIDO

UT. PYTHON

1.QUÉ ES PYTHON

Python es un lenguaje de programación **interpretado** cuya filosofía hace hincapié en la legibilidad de su código.

Es un lenguaje **interpretado, dinámico y multiplataforma.**

Lenguaje Interpretado

Un lenguaje de programación interpretado es aquel que no necesita ser compilado a código máquina antes de su ejecución. En cambio, el código es ejecutado directamente por un programa llamado ***intérprete***.

Algunas características clave de los lenguajes interpretados son:

- Ejecución Directa: El intérprete ejecuta el código línea por línea, lo que facilita la detección de errores en tiempo de ejecución.
- Flexibilidad: Los cambios pueden probarse rápidamente sin necesidad de una etapa de compilación.
- Ejemplos: Python, JavaScript y Ruby son lenguajes interpretados.

Lenguaje Dinámico

Un lenguaje dinámico tiene varias características que lo diferencian de los lenguajes estáticos, como:

- Tipado Dinámico: Las variables pueden cambiar de tipo en tiempo de ejecución. No es necesario declarar el tipo de variable antes de usarla.

  En Python, ej:

```
x = 5              # x es un entero
x = "Hola"         # x ahora es una cadena
```

- Meta-programación: Capacidad de examinar y modificar la estructura del programa en tiempo de ejecución.

Lenguaje Multiplataforma

Un lenguaje de programación multiplataforma es aquel que puede ejecutarse en diferentes sistemas operativos sin necesidad de realizar cambios significativos en el código. Algunas ventajas de los lenguajes multiplataforma incluyen:

- Portabilidad: El mismo código fuente puede ejecutarse en Windows, macOS, Linux, entre otros.
- Amplia Compatibilidad: Facilita el desarrollo y distribución de aplicaciones en diferentes entornos.
- Ejemplos: Además de Python, otros lenguajes multiplataforma son Java y C++.

PYTHON

Administrado por la **Python Software Foundation**, posee una licencia de **código abierto**, denominada Python Software Foundation License. Python se clasifica constantemente como uno de los lenguajes de programación más populares.

Python fué diseñado por **Guido van Rossum** (creador de Python) en **1991**. La última versión estable es la **Python 3.13.1**, lanzada el 3 de diciembre de 2024

En 1996 Guido van Rossum escribió:

> *"Hace seis años, en diciembre de 1989, estaba buscando un proyecto de programación como hobby que me mantuviera ocupado durante las semanas de Navidad. Mi oficina estaría cerrada y no tendría más que mi ordenador de casa a mano. Decidí escribir un intérprete para el nuevo lenguaje de scripting que había estado ideando recientemente: un descendiente de ABC que gustaría a los hackers de Unix/c. Elegí el nombre de Python para el proyecto, encontrándome en un estado de ánimo ligeramente irreverente (y siendo un gran fan de Monty Python)"*

Monty Python (a veces conocidos como Los Pythons) fue un grupo británico de seis humoristas que sintetizó en clave de humor la idiosincrasia británica de los años 1960 y 1970.

Video de los Monty Python: https://www.youtube.com/watch?v=rGbe5qy5274

Más información: https://es.wikipedia.org/wiki/Python

2.CARACTERÍSTICAS PRINCIPALES DE PYTHON

- **Lenguaje de propósito general**

  Eso significa que no está orientado a un fin concreto, como puede ser PHP, pensado sobre todo para hacer páginas de internet.

  Con Python podrás crear páginas sin tener un alto conocimiento (con Javascript como un poderoso aliado), pero también hacer scripts o software para el sistema operativo Windows.

- **Es multiparadigma**

  ¿Y qué significa eso? ¿Multiparadigma? Pues aunque su fuerte sea la programación orientada a objetos (es un lenguaje de alto nivel), existen otros paradigmas o estilos de programación para sus usuarios, como es la programación imperativa (con sentencias de bucle) o la programación funcional (con módulos y funciones).

  Así que si no sabes nada de objetos y sólo sabes escribir código mediante métodos, puedes usar Python perfectamente, cosa que en otros lenguajes hacer eso es imposible.

- **Python es un lenguaje interpretado**

  Cuando programamos en Python, no compilamos el código fuente a código máquina, sino que hay un intérprete que es el que ejecutará el programa basándose en el código directamente.

  Aunque esto puede ser una desventaja, los programas pueden ser más lentos.

- **Es multiplataforma**

  Al contrario que muchos lenguajes como Visual Basic, que principalmente solo puedes hacer cosas para Windows, con Python tienes la posibilidad de usarlo en muchos dispositivos y sistemas operativos, ya que se han creado intérpretes para Unix, Linux, Windows y sistemas Mac.

- **Es de tipado dinámico**

  Cuando declaramos una variable, no es necesario decirle de qué tipos son los datos (si es int, string, float, etc.). La variable se adapta a lo que escribimos cuando se ejecuta el programa.

  Antes esta característica siempre ha sido criticada en otros lenguajes, por la optimización de la memoria, errores a la hora de escribir código, etc. pero con Python el objetivo es que el lenguaje ayude a la creación de software, no tener que lidiar con peculiaridades propias del lenguaje.

  Igualmente, **Python es fuertemente tipado**, por ejemplo, no podrás sumar números y texto (una variable del tipo int con una de tipos cadenas) porque daría error.

- **Es orientado a objetos**

Ya hemos dicho que podemos aplicar otro estilo de programación, hacer software orientado a objetos conlleva una serie de ventajas estándar, sobre todo a la hora de reutilizar los componentes gracias a la herencia y sus funciones de polimorfismo.

- Otras propiedades de Python:
  - De **libre distribución.**
  - Gracias a su popularidad, existen una **gran cantidad de librerías y funciones** ya hechas, que podemos utilizar gracias a su extensa biblioteca.
  - Tiene soporte para **múltiple variedad de bases de datos**.
  - Tiene un gran soporte gracias a su **comunidad**. Por ejemplo, la última versión de Visual Studio te permite desarrollar en Python, o la comunidad de la página oficial de Python, dónde vemos todas las actividades que hacen en el mundo.

3. PROGRAMAS HECHOS CON PYTHON

Ahora vamos a nombrar algunos programas famosos que están hechos con Python, como por ejemplo:

- **Youtube**
- **Spotify**
- **Netflix**
- **Instagram**
- **Dropbox**
- **Reddit**
- **Odoo**: ERP
- **Calibre**: el mejor gestor de e-books para todos los usuarios.
- **GNU MailMan:** un programa para manejar listas de correo.
- **BitTorrent**: programa para compartir ficheros de tipo torrent estándar.

4. PYTHONISMO, PYTHONISTA Y PYTHONICO

Son términos que se utilizan entre los programadores de Python. Considerando una filosofía, una persona o un código.

El código pythonico podría resumirse "*meter la mayor cantidad de código en el menor espacio posible*"

Comunidad Pythonista : https://www.pythonista.io

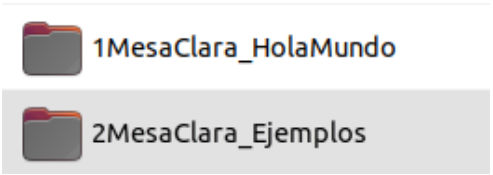
UT5.P0. InstalacionPythonUbuntu

Crear un pequeño tutorial donde se instale Python en Ubuntu en el IDE Visual Studio.
La versión de Python debe ser 3.10
Instala la extensión de Python en Visual Studio Code. Ve al Marketplace, busca "Python" e instálala.
En las capturas de pantalla debe aparecer tu apellido/nombre (puede ser en la terminal).

5. CREACIÓN DE PROYECTOS PYTHON EN ECLIPSE

En vuestro directorio personal de Ubuntu (/home/usuario) tendréis un directorio **eclipse-workspace** donde se irán guardando todos vuestros proyectos:

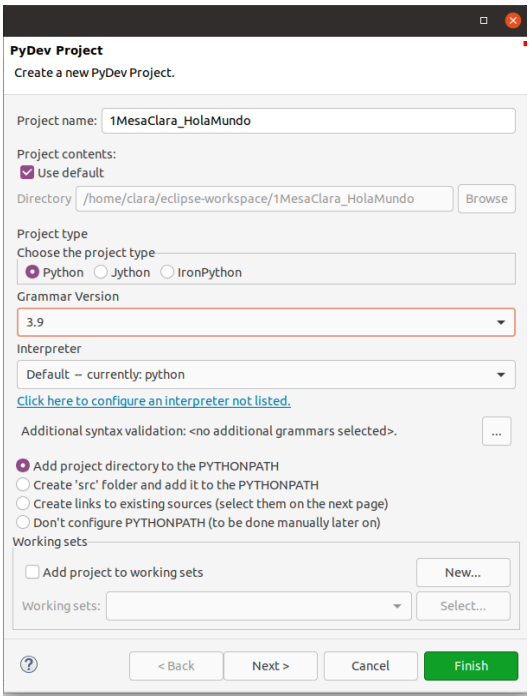
Dentro de este directorio estarán cada uno de los directorios de los proyectos que vayamos creando:



Creación de proyectos Pyhton en Eclipse:

1. File → New → PyDev Proyect (nombre ordenado: 1ApellidoNombre_NombredelEjercicio)

En el desplegable **Grammar Version:** elegimos **3.9**



Seleccionando nuestro proyecto:

2. File → New → Source Folder

Crearemos tantos directorios como sean necesarios:

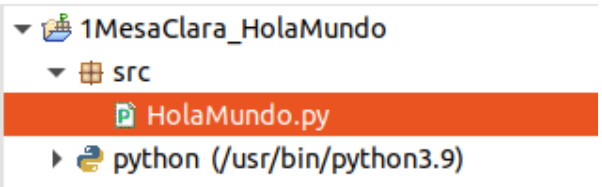
src → Donde guardaremos nuestros ficheros .py

data → Donde guardaremos nuestros .csv (por ejemplo)

Seleccionamos el directorio src:

3. File → New → PyDev Module

(no es necesario guardarlo como .py , automáticamente se guardarán con dicha extensión)



6. CREACIÓN DE PROYECTOS PYTHON EN VISUAL STUDIO CODE

1. Crea un directorio en tu sistema de archivos `(~/Python_Ejercicios)`**.** Aquí guardarás de forma **ORGANIZADA** todos los proyectos que realices.
2. Crea dentro de este directorio un subdirectorio llamado `1ApellidoNombre_Holamundo`
3. Crea un archivo `main.py` dentro de este directorio, y escribe el siguiente código de ejemplo:

```
print("¡Hola, mundo!")
```

4. Abre una terminal dentro de Visual Studio Code. (Ve al menú Terminal y selecciona Nueva terminal)

5. Ejecuta tu script Python: `python3 main.py`

```
clara@clara-Lenovo:~/Python_Ej/1.HolaMundo$ python3 main.py
¡Hola, mundo!
clara@clara-Lenovo:~/Python_Ej/1.HolaMundo$
```

Deberías ver el mensaje `¡Hola, mundo!` en la terminal.

> **UT5.P1. "Hola mundo "**
>
> Como es típico crea "Hola mundo SOY JUAN CARLOS FORNIELES(cada uno el suyo :)) " en Python.
> El proyecto se llamará: 1ApellidoNombre_Holamundo

7. INSTRUCCIONES Y FUNCIONES

Un programa Python está formado por **instrucciones** (a veces también son llamadas **sentencias**). Cada instrucción se escribe normalmente en una línea. Por ejemplo, la siguiente instrucción sirve para imprimir el mensaje "Hola, mundo!" en la pantalla.

```
print("Hola mundo")
```

Funciones

`print`, es una **función predefinida** (*built-in function*).

Una función puede ser "llamada" (también se dice "invocada") desde un programa escribiendo su nombre y a continuación unos paréntesis de apertura y cierre. En ocasiones, las funciones reciben **parámetros**, que se escriben en la llamada dentro de los paréntesis, separados por comas si hay más de uno.

Hay muchas funciones predefinidas, vienen incluidas "por defecto" en Python, por ejemplo la función **help** nos proporciona ayuda sobre cualquier otra función.

```
help(print)    # Nos muestra información de ayuda sobre la función predefinida print
```

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
# Para insertar un comentario
```

Los programadores pueden definir sus propias funciones, como se muestra a continuación:

```
def saluda(nombre):
    '''
    Esta función imprime un saludo personalizado con el nombre indicado mediante el parámetro "nombre".
    '''
    print("Hola, " + nombre)
```

La función que acabamos de definir se llama `saluda` y recibe un único parámetro. Después de la primera línea (llamada cabecera o prototipo de la función) vienen una o varias instrucciones (llamadas cuerpo de la función), que serán ejecutadas cuando alguien llame a la función.

Las instrucciones que conforman el cuerpo de la función aparecen indentadas, es decir, tienen un tabulador delante.

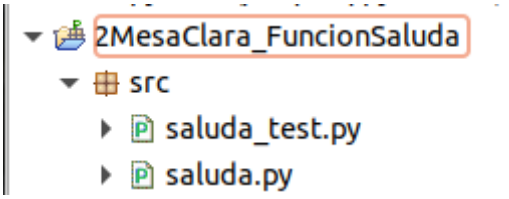
Las primeras líneas del cuerpo de la función, que comienzan y acaban con tres comillas simples, se llaman comentario de documentación de la función, y, aunque no son obligatorias, sí es recomendable incluirlas. Cuando alguien llame a la función predefinida help para pedir ayuda sobre nuestra función, se mostrará precisamente el texto aquí incluido.

Al definir una función conseguimos reutilizar la funcionalidad implementada las veces que queramos, simplemente llamando a la función en las partes del programa en que necesitemos dicha funcionalidad.

Estructura del código en distintos módulos

Dentro de nuestro proyecto y del directorio src crearemos dos ficheros:

.py	Contiene todas las funciones
_test.py	Se importan los archivos y se hace la llamada a la función principal



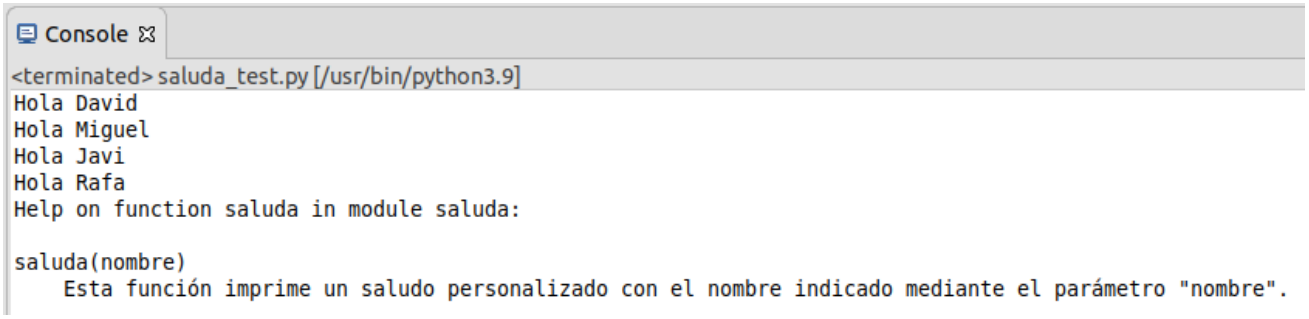
saluda.py

```
6  def saluda(nombre):
7      '''
8      Esta función imprime un saludo personalizado con el nombre indicado mediante el parámetro "nombre".
9      '''
10     print("Hola " + nombre )
```

saluda_test.py

```
6  from saluda import *
7  def main():
8      saluda("David")
9      saluda("Miguel")
10     saluda("Javi")
11     saluda("Rafa")
12     help(saluda)
13 if __name__=='__main__':
14     main()
```

Resultado al ejecutar:



Como puedes observar al invocar a la función help(saluda) aparece el texto de ayuda incorporado entre las comillas simples ‘’.

UT.P2. Saluda

Crea un proyecto 2ApellidoNombre_Saluda tal como se ha explicado en los apuntes.

7. EXPRESIONES Y TIPOS

Operadores aritméticos

Operador		Ejemplo	Resultado
+	suma	2+3	5
-	resta	5-2	3
*	producto	5*-2	-10
/	cociente	5/2	2.5
//	cociente división entera	5//2	2
%	resto división entera	5%2	1
**	potencia	5**2	25

Orden de prioridad de evaluación

1. **Funciones predefinidas**
2. **Potencias**
3. **Productos y cocientes**
4. **Sumas y restas**

Se puede saltar el orden de evaluación utilizando paréntesis ( ).

Ejemplo: (2+3)**2

Ejemplo de operaciones

```
 6 print(5+2)
 7 print(2-5)
 8 print(5-2)
 9 print(5*2)
10 print(5*-2)
11 print(5/2)
12 print(5//2)
13 print(5%2)
14 print(5**2)
15 print("*****************************")
16 print(3+5)
17 print(3+5*8)
18 print((3+5)*8)
19 print((3+5)*8/14)
20 print("*****************************")
21 print("El resultado de 3+5:  " , 3+5)
22 print("El resultado de 3+5*8:  " , 3+5*8)
23 print("El resultado de (3+5)*8:  " , (3+5)*8)
24 print("El resultado de (3+5)*8/14:  " , (3+5)*8/14)
```

Console

<terminated> tipos_datos.py [/usr/bin/python3.9]

```
7
-3
3
10
-10
2.5
2
1
25
*****************************
8
43
64
4.571428571428571
*****************************
El resultado de 3+5:   8
El resultado de 3+5*8:   43
El resultado de (3+5)*8:   64
El resultado de (3+5)*8/14:   4.571428571428571
```

Tipos de Datos Primitivos Simples

- **Números (numbers)**: Secuencia de dígitos (pueden incluir el - para negativos y el . para decimales) que representan números.
  Ejemplo. 0, -1, 3.1415.
- **Cadenas (strings)**: Secuencia de caracteres alfanuméricos que representan texto. Se escriben entre comillas simples o dobles.
  Ejemplo. 'Hola', "Adiós".
- **Booleanos (boolean)**: Contiene únicamente dos elementos True y False que representan los valores lógicos verdadero y falso respectivamente.

Tipos de datos primitivos compuestos (contenedores)

- **Listas (lists):** Colecciones de objetos que representan secuencias ordenadas de objetos de distintos tipos. Se representan con corchetes y los elementos se separan por comas.
  Ejemplo. [1, "dos", [3, 4], True]
- **Tuplas (tuples)**. Colecciones de objetos que representan secuencias ordenadas de objetos de distintos tipos. A diferencia de las listas son inmutables, es decir, que **no cambian durante la ejecución**. Se representan mediante paréntesis y los elementos se separan por comas.
  Ejemplo. (1, 'dos', 3)
- **Diccionarios (dictionaries):** Colecciones de objetos con una clave asociada. Se representan con llaves, los pares separados por comas y cada par contiene una clave y un objeto asociado separados por dos puntos.
  Ejemplo. {'pi':3.1416, 'e':2.718}

Clase de un dato (type())

La clase a la que pertenece un dato se obtiene con el comando `type()`

```
26 print(type(1))
27 print(type("Hola"))
28 print(type([1, "dos", [3, 4], True]))
29 print(type({'pi':3.1416, 'e':2.718}))
30 print(type((1,'dos',3)))
```

Resultado:

```
<class 'int'>
<class 'str'>
<class 'list'>
<class 'dict'>
<class 'tuple'>
```

Números (clases int y float)

Secuencia de dígitos (pueden incluir el - para negativos y el . para decimales) que representan números. Pueden ser enteros (int) o reales (float).

```
32 print(type(1))
33 print(type(-2))
34 print(type(2.3))
```

Resultado:

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

Operadores lógicos con números

Devuelven un valor lógico o booleano (true o false).

Operadores lógicos			
==	Igual que	!=	distinto de
>	Mayor que	>=	Mayor o igual que
<	Menor que	<=	Menor o igual que

```
38 print("Operadores lógicos")
39 print(3==3)
40 print(3.1<=3)
41 print(-3!=3)
```

Resultado:

```
Operadores lógicos
True
False
True
```

Cadenas (clase str)

Secuencia de caracteres alfanuméricos que representan texto. Se escriben entre comillas sencillas ' o dobles “.

- \n salto de línea
- \t tabulador

Resultado:

```
43 print("Ejemplo de clase str")
44 print(str('Hola chic@s'))
45 print(str("Python"))
46 print(str('Primera línea \nSegunda línea'))
```

```
****************************
Ejemplo de clase str
Hola chic@s
Python
Primera línea
Segunda línea
```

Acceso a los elementos de la cadena

Cada carácter tiene asociado un índice que permite acceder a él.

El índice del primer carácter de la cadena es 0.
El índice del último carácter de la cadena es -1.

También se pueden utilizar índices negativos para recorrer la cadena del final al principio.

Ejemplo:

c[i] devuelve el carácter de la cadena c con el índice i.

Cadena	p	y	t	h	o	n
Índice positivo	0	1	2	3	4	5
Índice negativo	-6	-5	-4	-3	-2	-1

```
47 print("****************************")
48 print('Python'[0])
49 print('Python'[-1])
50 print('Python'[6])
```

```
****************************
P
n
  File "/home/clara/eclipse-workspace/3MesaClara_TiposDatos/src/tipos_datos.py", line 50, in <module>
    print('Python'[6])
IndexError: string index out of range
```

Subcadenas

c[i:j:k] : Devuelve la subcadena de c desde el carácter con el índice i hasta el carácter anterior al índice j, tomando caracteres cada k.

```
51 print("*****************************")
52 print("Ejemplos de Subcadenas")
53 print('Python'[1:4])
54 print('Python'[1:1])
55 print('Python'[2:])
56 print('Python'[:])
57 print('Python'[:-2])
58 print('Python'[0:6:2])
```

Resultado:

```
*****************************
Ejemplos de Subcadenas
yth

thon
Python
Pyth
Pto
```

Operaciones con cadenas

- **c1 + c2 :** Devuelve la cadena resultado de concatenar las cadenas c1 y c2.
- **c * n** : Devuelve la cadena resultado de concatenar n copias de la cadena c.
- **c1 in c2 :** Devuelve True si c1 es una cadena contenida en c2 y False en caso contrario.
- **c1 not in c2** : Devuelve True si c1 es una cadena no contenida en c2 y False en caso contrario.

Utilizan el orden establecido en el código ASCII.

Funciones de cadenas

- **len(c)** : Devuelve el número de caracteres de la cadena c.
- **min(c)** : Devuelve el carácter menor de la cadena c.
- **max(c)** : Devuelve el carácter mayor de la cadena c.
- **c.upper()** : Devuelve la cadena con los mismos caracteres que la cadena c pero en mayúsculas.
- **c.lower()** : Devuelve la cadena con los mismos caracteres que la cadena c pero en minúsculas.
- **c.title()** : Devuelve la cadena con los mismos caracteres que la cadena c con el primer carácter en mayúsculas y el resto en minúsculas.
- **c.split(delimitador)** : Devuelve la lista formada por las subcadenas que resultan de partir la cadena c usando como delimitador la cadena delimitadora. Si no se especifica el

delimitador utiliza por defecto el espacio en blanco.

Cadenas formateadas (format())

- **c.format(valores):** Devuelve la cadena c tras sustituir los valores de la secuencia valores en los marcadores de posición de c. Los marcadores de posición se indican mediante llaves {} en la cadena c, y el reemplazo de los valores se puede realizar por posición, indicando en número de orden del valor dentro de las llaves, o por nombre, indicando el nombre del valor, siempre y cuando los valores se pasen con el formato nombre = valor.

Los marcadores de posición, a parte de indicar la posición de los valores de reemplazo, pueden indicar también el formato de estos. Para ello se utiliza la siguiente sintaxis:

- **{:n}** : Alinea el valor a la izquierda rellenando con espacios por la derecha hasta los n caracteres.
- **{:>n}** : Alinea el valor a la derecha rellenando con espacios por la izquierda hasta los n caracteres.
- **{:^n}** : Alinea el valor en el centro rellenando con espacios por la izquierda y por la derecha hasta los n caracteres.
- **{:nd}** : Formatea el valor como un número entero con n caracteres rellenando con espacios blancos por la izquierda.
- **{:n.mf}** : Formatea el valor como un número real con un tamaño de n caracteres (incluído el separador de decimales) y m cifras decimales, rellenando con espacios blancos por la izquierda.

Conversión de datos primitivos simples

Las siguientes funciones convierten un dato de un tipo en otro, siempre y cuando la conversión sea posible.

- **int()** convierte a entero.
  Ejemplo. int('12') → 12
  int(True) → 1
  int('c') → Error
- **float()** convierte a real.
  Ejemplo. float('3.14') → 3.14
  float(True) → 1.0
  float('III') → Error
- **str()** convierte a cadena.
  Ejemplo. str(3.14) → '3.14'
  str(True) → 'True'
- **bool()** convierte a lógico.
  Ejemplo. bool('0') → False
  bool('3.14') → True
  bool('') → False
  bool('Hola') → True

Variables

Una variable es un identificador ligado a algún valor.

A diferencia de otros lenguajes **no tienen asociado un tipo** y **no es necesario declararlas antes de usarlas (tipado dinámico).**

Para asignar un valor a una variable se utiliza el operador = y para borrar una variable se utiliza la instrucción **del**, aunque no es necesario borrarla.

Ejemplo:

```
x=3.14
nombre=”Maripepi”
y=3+2
x += 2          incremento, equivale a x=x+2
z -= 1          decremento, equivale a x=x-1
x=None          valor no definido
del x
```

Normas para la construcción de nombres de variables

Podemos usar los nombres que queramos para nuestras variables, siempre que cumplamos las siguientes reglas:

- Sólo podemos usar **letras, números y la barra baja** (_). No se pueden usar espacios.
- **El nombre puede comenzar por una letra o por la barra baja**.
- **No se pueden usar determinadas palabras clave** (keywords) que Python usa como instrucciones (por ejemplo, def o if) o como literales (por ejemplo, True). Aunque Python lo permite, tampoco es apropiado usar nombres de funciones predefinidas (por ejemplo, print).
- Los nombres tienen que ser **descriptivos** de lo que representa la variable (¡sin pasarse!).

Aquí tienes algunos ejemplos de nombres incorrectos de variables; observa los errores generados por Python.

```
4edad = 10
```

```
  File "<ipython-input-1-9b5cebdeb6a5>", line 1
    4edad = 10
         ^
SyntaxError: invalid syntax
```

```
if = 20
```

```
  File "<ipython-input-2-b8359eb0a5c6>", line 1
    if = 20
       ^
SyntaxError: invalid syntax
```

Palabras reservadas del sistema (Keywords)

Puedes consultar todas las palabras claves (keywords) existentes en Python de la siguiente forma:

```
import keyword

print(keyword.kwlist)
```

8. ENTRADA Y SALIDA POR TERMINAL

Entrada por terminal (input())

Para asignar a una variable un valor introducido por el usuario en la consola se utiliza la instrucción:

> **input(mensaje)** : Muestra la cadena mensaje por la terminal y devuelve una cadena con la entrada del usuario.

El valor devuelto siempre es una cadena, incluso si el usuario introduce un dato numérico.

Ejemplo:
edad = input (‘ ¿Cuál es tu edad? ‘)

Salida por terminal (print())

Para mostrar un dato por la terminal se utiliza la instrucción

> **print**(dato1, ..., sep=' ', end='\n', file=sys.stdout)

donde:

- **dato1, ...** son los datos a imprimir y pueden indicarse tantos como se quieran separados por comas.
- **sep** establece el separador entre los datos, que por defecto es un espacio en blanco ' '.
- **end** indica la cadena final de la impresión, que por defecto es un cambio de línea \n.
- **file** indica la dirección del flujo de salida, que por defecto es la salida estándar sys.stdout.

REFERENCIAS WEB

La mayor parte del contenido de estos documentos están basados en apuntes y ejemplos de:

https://aprendeconalf.es/docencia/python/manual/
https://www.cs.us.es/~jalonso/cursos/i1m/temas.php

Documentación Python: https://docs.python.org/3/