U. T. 2 Elementos de un programa informático.



Introducción

La elección de un lenguaje de programación para aprender los entresijos de la programación es una opción muy importante y que no se debe tomar a la ligera. Tradicionalmente se usaban lenguajes muy estructurados para el aprendizaje como era Pascal o Basic, pero con el tiempo se impuso java como lenguaje de aprendizaje. Si bien, java está ampliamente extendido, no es un lenguaje sencillo ni fácil de aprender, la curva de aprendizaje al principio es muy elevada, lo que hace difícil la aproximación a la programación a través de este entorno. No quiero decir ni mucho menos que no sea posible, sino que no es el lenguaje para aprender para neófitos.

Por el contrario, Python incorpora muchas características que hacen que su curva de aprendizaje sea más plana y por tanto más fácil de dominar por parte de los nuevos programadores. De hecho, una gran mayoría de centros de estudios y universidades están utilizando ya Python como el lenguaje de aprendizaje para la programación.

Después de más de veinte años programando, y pasando por muchos lenguajes diferentes: C, C++, Lisp, Cobol, Pascal, Delphi, Basic, C#, Java, PHP, Python y Perl entre otros, me parece que Python reúne todas las características para que un alumno aprenda programación de manera sencilla, e impone restricciones que evitarán vicios difícilmente subsanables posteriormente, añadiendo una gran cantidad de utilidades y herramientas.

Un último dato para afianzar más que es el lenguaje actual que mejor se adapta a la enseñanza, en el año 2020 ha pasado a ser el segundo lenguaje más utilizado en proyecto, por delante de Java, continuando con su ascenso meteórico desde su creación.

Historia de Python

Guido Van Rossum es el creador y responsable de que Python exista. Se trata de un informático de origen holandés que fue el encargado de diseñar Python y de pensar y definir todas las vías posibles de evolución de este popular lenguaje de programación.

En las navidades de 1989 Van Rossum decidió empezar un nuevo proyecto como pasatiempo personal. Pensó en darle continuidad a ABC, un lenguaje de programación que se desarrolló en el mismo centro en el que estaba trabajando. ABC fue desarrollado a principios de los ochenta como alternativa a BASIC. Se trata de un lenguaje pensado para principiantes por su facilidad de aprendizaje y uso. Su código era compacto pero legible. Sin embargo, el proyecto no llegó mucho más lejos por las limitaciones del hardware de la época, así que Van Rossum decidió darle una segunda vida a su idea y partiendo de la base que tenía, empezó a trabajar en Python.

> ¿Por qué se llama Python? El nombre de este lenguaje de programación es en honor a los Monty Python, el famoso grupo de cómicos británicos.

Versiones

La versión 1.0, que se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008. Esta primera versión de Python ya incluía clases con herencias, manejo de excepciones, funciones y una de sus características fundamentales: funcionamiento modular. Esto permitía que fuese un lenguaje mucho más limpio y accesible para la gente con pocos conocimientos de programación, una característica que se mantiene hasta el día de hoy. Hasta el año 2018, el desarrollo de este popular lenguaje de programación estaba dirigido personalmente por Van Rossum, pero decidió apartarse y, desde 2019, son cinco las personas que deciden cómo evoluciona y se desarrolla Python. Un consejo que se renueva de forma anual.

Versión 1.0

Python es un lenguaje de programación que Van Rossum empezó a desarrollar mientras trabajaba en CWI. Fue este centro de investigación quien liberó, en 1995 la versión 1.2 de Python. A partir de este momento,

ya desvinculado de CWI, Van Rossum hizo aún más accesible el código y para el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo de BeOpen Python Labs.

Python 1.6.1 es exactamente lo mismo que 1.6, con algunos bugs arreglados y una nueva licencia compatible con GPL. La versión 1.6 de Python tuvo algunos problemas con su tipo de licencia hasta que la Free Software Foundation (FSF) consiguió cambiar Python a una licencia de Software Libre, que lo haría compatible con GPL.

Versión 2.0

En octubre del año 2000 se publica la segunda versión de Python. Una nueva versión en la que se incluyó la generación de listas, una de las características más importantes del lenguaje.

En 2001, se crea la Python Software Foundation, la cual a partir de Python 2.1 es dueña de todo el código, documentación y especificaciones del lenguaje. A mayores de esta nueva característica, esta nueva versión de Python también incluyó un nuevo sistema gracias al cual los programadores eran capaces de hacer referencias cíclicas y, de esta manera, Python podía recolectar basura dentro del código.

Versión 3.0

La última gran actualización de la historia de Python se produjo en el año 2008 con el lanzamiento de la versión 3.0, que venía a solucionar los principales fallos en el diseño de este lenguaje de programación.

Aunque Python mantiene su filosofía en esta última versión, como lenguaje de programación ha ido acumulando formas nuevas y redundantes de programar un mismo elemento. De ahí la necesidad de nuevas versiones que eliminen estos constructores duplicados.

Python 3.0 rompe la compatibilidad hacia atrás del lenguaje, ya que el código de Python 2.x no necesariamente debe correr en Python 3.0 sin modificación alguna. La última actualización de la versión 3 de Python a la hora de escribir estos apuntes se trata de la **versión 3.12**.

Recomiendo usar como mínimo la versión 3.11 de Python

Características de Python

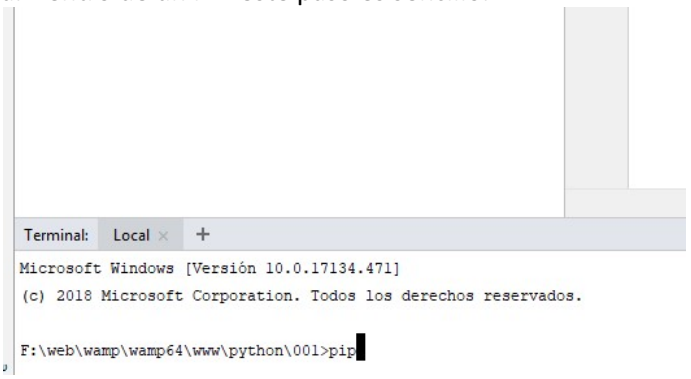
- Es un lenguaje fuertemente tipado, en concreto todos los elementos son objetos: los tipos, las funciones, etc.
- Es un lenguaje con tipado dinámico, lo que significa que el tipo de una variable se puede cambiar durante la ejecución. Esto implica que la definición de una variable no tendrá nunca un tipo, pero será necesaria su inicialización antes de usarla.
- El lenguaje incorpora un gestor de paquetes propio: **pip** que se utiliza para añadir funcionalidades al sistema. Se accederá desde una sesión de consola.

pip search cadena[1]	→ buscar paquetes
pip install paquete	→ instalar un paquete
pip install –U paquete	→ actualizar un paquete
pip uninstall paquete	→ desinstalar un paquete

```
C:\Users\arcipreste>pip search numpy
ERROR: XMLRPC request failed [code: -32500]
RuntimeError: PyPI no longer supports 'pip search' (or XML-RPC search). Please use https://pypi.org/search (via a browse
r) instead. See https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-methods for more information.
```

[1] Esta función está desactivada por la sobrecarga que conlleva en los servidores, https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-methods, en su defecto usar el interfaz web directamente y traspasar a pip install: https://pypi.org/search/

Hay que tener en cuenta que se debe ejecutar este comando dentro del entorno de ejecución si no queremos que se realice en todo el sistema. Dentro de un IDE este paso es sencillo.



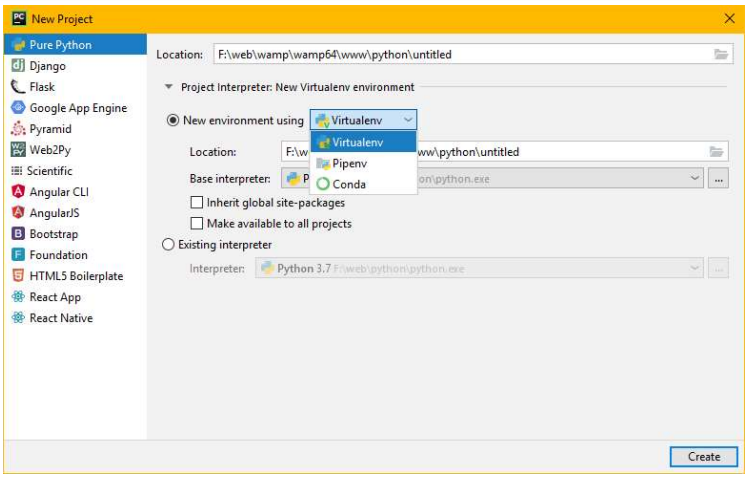
- Es un lenguaje interpretado, pero que se compila a una representación intermedia: byte code. Es un concepto similar a Java.
- El lenguaje está dirigido a todos los sistemas. Web, desktop, Programación de sistemas, Big data, programación científica.
- Es un lenguaje que se ejecuta en diversos sistemas operativos: Unix, Windows, etc.
- Se caracteriza por tener una gran librería. Es imprescindible estudiar la documentación.
- El lenguaje incorpora una gestión automática de recursos y de memoria, con un recolector de la misma.
- Es un lenguaje libre con varias implementaciones, la más conocida es la basada en C.
- El lenguaje impone pocas restricciones al programador.
- El lenguaje incorpora las facilidades de docstring para documentación del código generado.
- Por último, se asegura la compatibilidad a nivel de código dentro de la misma rama. Es decir, todo código de la versión 3.1 se ejecutará en cualquier 3.X, pero no bajo 4.X.
- La curva de aprendizaje para los no conocedores de la programación es muy suave, por el contrario, para aquellos con conocimientos arraigados puede costar cambiar la metodología.

Un vistazo rápido al interior de Python

El lenguaje está compuesto por los siguientes elementos: Gramática y sintaxis, implementaciones, librería estándar, librerías de terceros y Frameworks. Todo programa cuando se ejecuta pasa por las siguientes fases:

- Carga de la máquina virtual (Python.exe).
- Compilación en Byte Code. Se compila cada módulo y se guarda en caché dicha compilación (.pyc) no recompilándose nada a menos que sea necesario (La caché se guarda dentro del directorio: **__pycache__**).
- Ejecución del programa.
    - Se puede crear un entorno virtual de ejecución para cada proyecto, de tal manera que no interfieran unos en otros.

Es recomendable que cada proyecto tenga su propio entorno de ejecución. A La hora de crear el proyecto se debe inicializar este entorno y después instalar las librerías necesarias solo en el proyecto, no en el sistema.

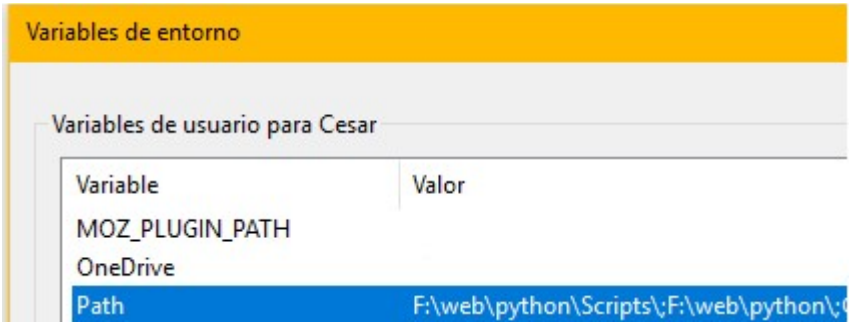


En Python existen utilidades para crear estos entornos virtuales: venv, pyvenv o virtualenv. En caso de usar un IDE será en el momento de la creación del proyecto cuando se determine si se usa el entorno general o se crea uno virtual. Veremos más adelante el uso y creación de estos entornos virtuales.

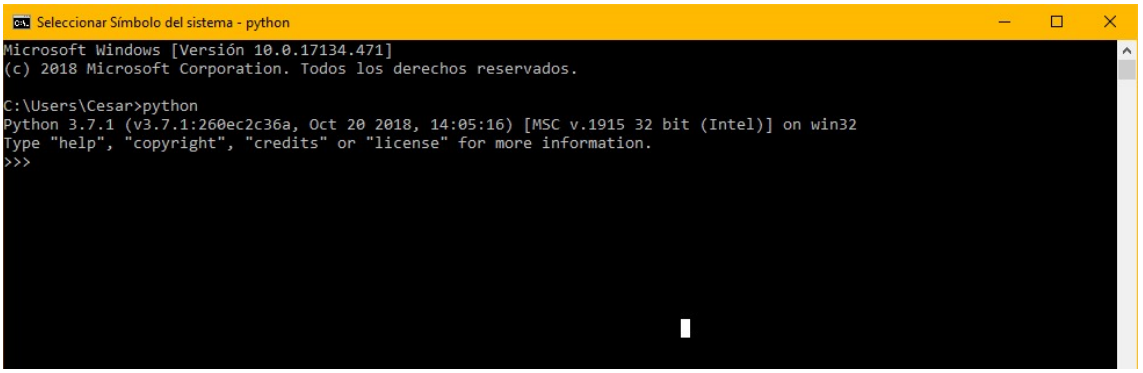
Instalación de Python

La instalación está dirigida sobre todo a Windows sabiendo que la mayoría del alumnado usa este tipo de sistemas, pero para otros sistemas operativos la instalación es igual o más sencilla.

Desde la página https://www.python.org/downloads/ nos descargamos el ejecutable correspondiente (generalmente x64) y lo instalamos. Una vez finalizado sería recomendable establecer las variables de entorno (PATH del sistema) de forma adecuada si no se hace en la instalación:



Con estas configuraciones, podremos acceder a una consola Python desde el símbolo de sistema o un terminal ejecutando simplemente **Python**.



En caso que no deseemos instalar todo desde cero, se puede utilizar la distribución Anaconda de Python, si bien está orientada a la programación científica se puede usar como plataforma de aprendizaje: https://www.anaconda.com/

> En un entorno Unix, la propia distribución incluirá los paquetes necesarios para la instalación. Así en sistemas Debian/Ubuntu será tan sencillo como **apt-get install python**

Entornos IDE

Para el desarrollo de un proyecto Python es altamente recomendable utilizar un IDE que nos proporcione todas las características, si bien no es imprescindible y es posible usar: atom, sublime, etc. como entornos de

desarrollo, lo más indicado es utilizar uno. Los IDEs más extendidos a día de hoy son: PyCharm y Microsoft Visual Studio. Este curso usaremos PyCharm.

> Si nos encontramos que PyCharm no muestra ni llaves, corchetes, etc. hay que añadir "actionSystem.force.alt.gr=true" al final del fichero **bin\idea.properties** en el directorio de instalación.

La instalación de PyCharm es bastante sencilla no siendo necesaria ninguna configuración si tenemos instalado antes Python en el sistema.

Elementos de un programa

Estructura General de un programa

Un programa de Python es un fichero 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.

Conceptos básicos a recordar

- Los ficheros bajo Python se codifican en formato UTF-8 de forma general, pero se puede cambiar añadiendo en la primera línea del fichero fuente la codificación deseada: **# -*- coding: cp-1252**
- Cada línea representará una instrucción, no hace falta un delimitador final (como en otros lenguajes) ya que la indentación es parte imprescindible de la sintaxis. Así para crear un bloque sintáctico simplemente se indentarán todas las líneas del bloque el mismo número de espacios, marcando la línea principal de bloque con (:) dos puntos. A la hora de indentar no se pueden mezclar tabuladores y espacios, se recomienda usar cuatro espacios para indentar.
- Python usa comentarios de fin de línea exclusivamente, no existe ningún otro tipo de comentario (#). Es posible simular comentarios multilíneas con la notación extendida de representación de cadenas en varias líneas: “““ ”””
- El lenguaje diferencia entre mayúsculas y minúsculas tanto en las palabras clave como en los nombres de variables y funciones.

```
print("Hola Mundo") # mi primer comentario

# Los comentarios son sólo de fin de línea

for cnt in [1, 2, 3, 4]:
    print(cnt)

"""
 Aunque de esta manera se puede simular
 un comentario de varias líneas sin serlo.
"""
```

Funciones introductorias

Para poder desarrollar los ejemplos y avanzar de forma adecuada en el conocimiento del lenguaje debemos introducir varias funciones que nos servirán de ayuda.

- **print**(,,,end=’’). Esta función imprimirá un conjunto de parámetros en una línea por la salida estándar del programa, generalmente la pantalla. El parámetro **end** indica el último carácter a añadir a la línea en la salida, por defecto \n.
- **input**(“texto”). Esta función se utiliza para recoger una cadena de texto del usuario. En caso que el valor a tratar sea numérico es imprescindible realizar la conversión. Utilizará por defecto la entrada estándar del programa.

- **str**(valor), **int**(valor) e **float**(valor) convierten respectivamente a cadena un objeto cualquiera, a entero cualquier valor no numérico y a decimal. En el último caso no convertirá si el valor contiene alguna letra.
- **dir**(objeto) devuelve la lista de los métodos y propiedades existentes en el objeto.
- **type**(objeto) devuelve el tipo del objeto.
- **pass**. No ejecución. Se utiliza para crear bloques vacíos ya que no es posible que se deje un bloque sin instrucciones.

Ejemplo: Ejecutar un programa

Vamos a crear nuestro primer programa para familiarizarnos con todos los procedimientos.

Ejemplo: Crear el siguiente programa.

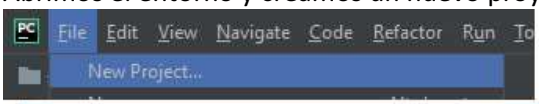
```
print("Prueba de entrada")

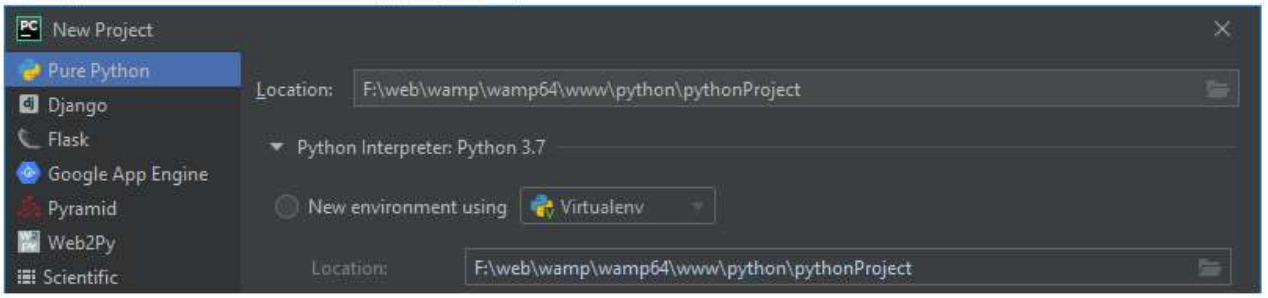
num1 = int(input("Primer Valor:"))
num2 = int(input("Segundo Valor:"))

print("La suma es:", str(num1 + num2))
```

code_0000.py

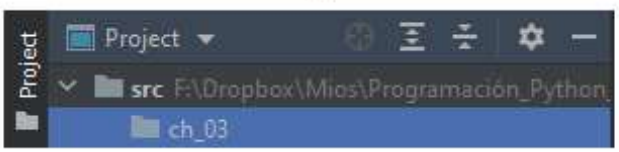
1. Abrimos el entorno y creamos un nuevo proyecto, estableciendo el directorio de trabajo.



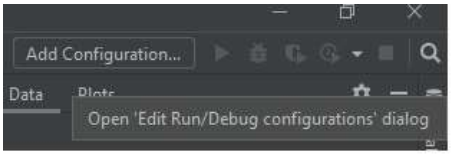


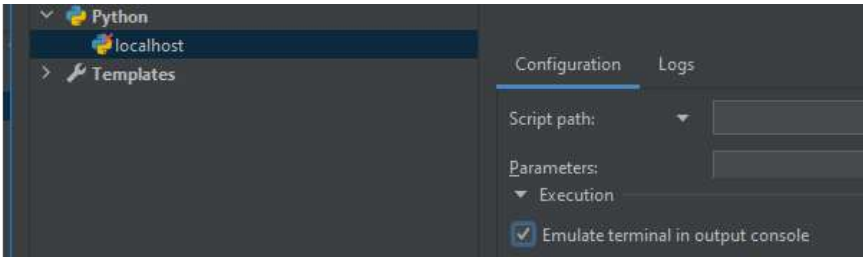
2. Creamos un nuevo directorio para contener los ficheros fuentes y tenerlos organizados: ch_03.



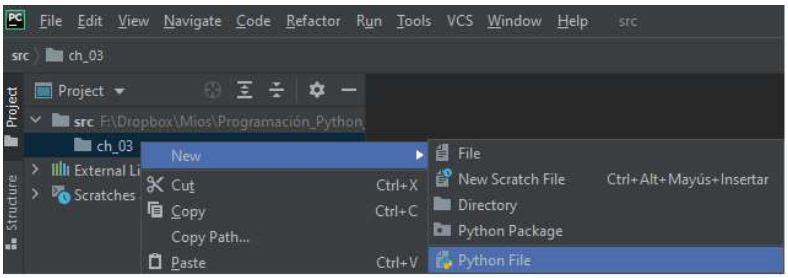


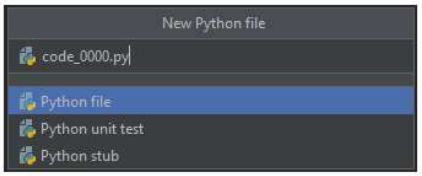
3. Añadimos una configuración de ejecución al entorno, para poder ejecutar ficheros. Se pulsará en la parte superior el botón más (+) y de la lista desplegable seleccionaremos Python.

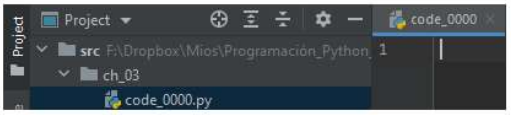




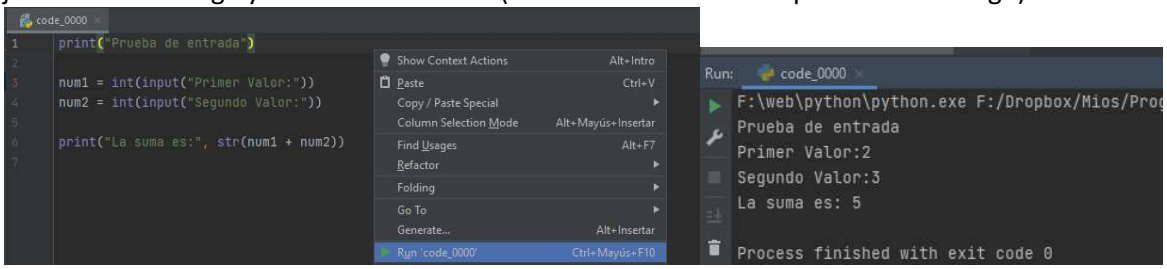
4. Creamos el fichero.







5. Introducimos el código fuente tal y como aparece.
6. Ejecutamos el código y vemos el resultado (botón derecho sobre la pestaña del código).



Hemos procedido a crear un fichero y todas las estructuras necesarias para la ejecución del código, pero no es necesario utilizar un IDE tal y como dijimos anteriormente. Se puede hacer la programación en cualquier editor de texto y ejecutarlo llamando a la orden **python nombre_archivo**, veámoslo.

1. Se abre el editor de texto preferido (write.exe, Notepad.exe, vi, joe, etc.) y se escribe el fichero fuente, almacenándolo en formato UTF-8. Esto último es obligatorio si no queremos que de un error de sintaxis.
2. Se abre una consola del sistema (cmd.exe en Windows, bash en Linux).
3. Se localiza el fichero fuente con la ruta completa.
4. Se ejecuta pasándolo como parámetro a la orden **python**.

```
Microsoft Windows [Versión 10.0.18363.1379]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Cesar>python F:\Dropbox\Mios\Programación_Python_DAW\libro\src\ch_03\code_0000.py
Prueba de entrada
Primer Valor:2
Segundo Valor:3
La suma es: 5

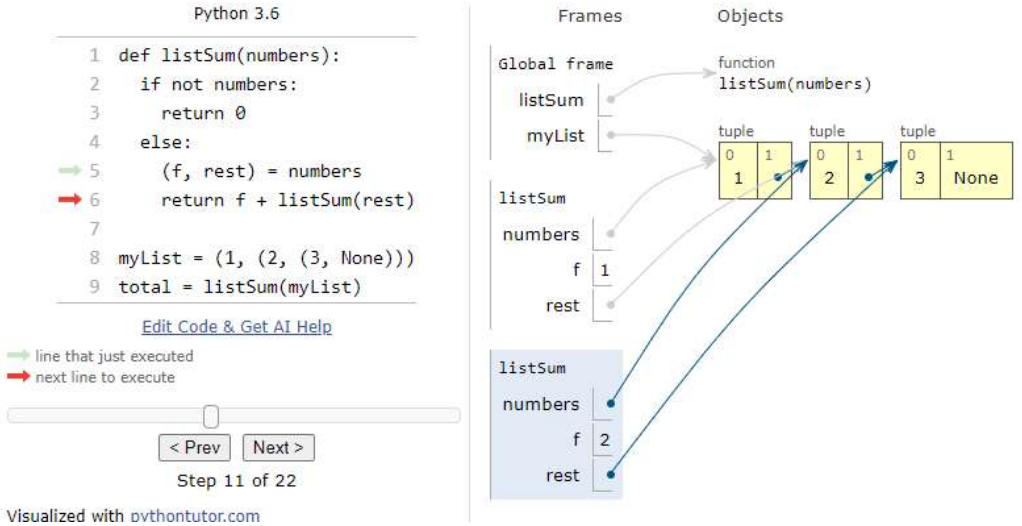
C:\Users\Cesar>_
```

Explicación del código

- **print("Prueba de entrada")**. Muestra por pantalla un texto.
- **num1 = int(input("Primer Valor:"))**. Crea una variable llamada num1 y le asigna el valor entero que el usuario introduzca por teclado, si no puede convertirlo dará error.
- **num2 = int(input("Segundo Valor:"))**. Similar pero llamada num2.
- **print("La suma es:", str(num1 + num2))**. Imprime un texto y el resultado de la operación de sumar las dos variables.

Ejemplo: Crear un programa con el típico: Hola mundo.

Herramienta de depuración: https://pythontutor.com/



Líneas y espacios en blanco

- Un programa Python se divide en un número de **líneas lógicas**. El final de una línea lógica está representado por el token NEWLINE (nueva línea, no hace falta ningún otro terminador como el punto y coma). Una línea lógica se construye a partir de una o más líneas físicas siguiendo las reglas de unión de líneas.
- Una **línea física** es una secuencia de caracteres terminada por una secuencia de final de línea. En los archivos fuente y las cadenas, se puede utilizar cualquiera de las secuencias de terminación de línea de la plataforma estándar: Unix utiliza ASCII LF, Windows utiliza la secuencia ASCII CR LF. Todas estas formas pueden ser utilizadas por igual, independientemente de la plataforma.
- **Dos o más líneas físicas pueden unirse en líneas lógicas** utilizando caracteres de barra invertida (\), de la siguiente manera: cuando una línea física termina en una barra invertida que no es parte de literal de cadena o de un comentario, se une con la siguiente formando una sola línea lógica

```
if 1900 < year < 2100 and 1 <= month <= 12 \
   and 1 <= day <= 31 and 0 <= hour < 24 \
   and 0 <= minute < 60 and 0 <= second < 60:
      return 1
```

Una línea que termina en una barra invertida no puede llevar un comentario. No se puede usar una barra invertida para continuar un comentario.

- Las expresiones entre paréntesis, entre corchetes o entre llaves pueden dividirse en más de una línea física sin usar barras invertidas. Por ejemplo:

```
month_names = ['Januari', 'Februari', 'Maart',      # These are the
               'April',   'Mei',      'Juni',       # Dutch names
               'Juli',    'Augustus', 'September',  # for the months
               'Oktober', 'November', 'December']   # of the year
```

Las líneas continuas implícitas pueden llevar comentarios. No es importante la sangría de las líneas de continuación.

- Una línea lógica que contiene sólo espacios, tabulaciones, saltos de página y posiblemente un comentario, es ignorada.
- A excepción del comienzo de una línea lógica o en los literales de cadenas, los caracteres de espacio en blanco, tabulación y formfeed pueden utilizarse indistintamente para separar tokens. Los espacios en blanco se necesitan entre dos tokens sólo si su concatenación podría interpretarse de otra manera como un token diferente.
- **Un bloque en Python viene determinado por el nivel de sangría aplicada al código,** así como del delimitador de comienzo de bloque (:). De tal manera que todas aquellas líneas lógicas que tengan la misma sangría estarán dentro del mismo bloque. El sangrado es obligatorio y en caso de utilizarlo mal generará un error de sintaxis. Las recomendaciones PEP indican que el sangrado de cada bloque se debe hacer con cuatro espacios en blanco más que el bloque anterior, pero nada nos impide usar tabuladores o cualquier otro número de espacios en blanco siempre que no se mezclen ni se cambie el número de espacios.

```
def perm(l):
    # Compute the list of all permutations of l
    if len(l) <= 1:
        return [l]
    r = []
    for i in range(len(l)):
        s = l[:i] + l[i+1:]
        p = perm(s)
        for x in p:
            r.append(l[i:i+1] + x)
    return r
```

Comentarios

Un comentario comienza con un carácter de almohadilla (#) que no es parte de un literal de cadena y termina al final de la línea física. Los comentarios son ignorados por la sintaxis y se utiliza para documentar o explicar el código fuente.

Identificadores

Un identificador es una secuencia de uno o más caracteres asignada por el programador a un elemento del programa (constante, variable, método, clase, paquete...).

Un identificador en Python comienza con una letra (de la A a la Z o de la a a la z) o con un guion bajo (_) seguido de cero o más letras, guiones bajos y números. Python distingue mayúsculas de minúsculas en el nombrado de los identificadores y Python **NO** permite signos de puntuación como @, $ y %, excepto el guion bajo (_).

Se recomienda nombrar las clases comenzando por una letra mayúscula cada palabra sin espacios y el resto de los identificadores por una letra minúscula separando las palabras con guiones bajos, excepto aquellos

que funcionen como constantes que se escribirán en mayúsculas y los identificadores que se definan en las clases como privados empezarán por el guion bajo. Ejemplos:

```
numero_palabras, encontrar_media, imprimir_linea
NUMERO_ALUMOS, LINEAS_POR_PAGINA
_nombre_persona,
```

Palabras reservadas

Los siguientes identificadores se utilizan como palabras reservadas, o palabras clave del lenguaje, y no pueden utilizarse como identificadores ordinarios. Deben escribirse exactamente como están escritas aquí:

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

Representación de los datos (Literales)

Literales

Los literales son las formas de escribir (notaciones) que utilizamos para los valores constantes de algunos tipos, son la representación escrita de los datos.

Literales numéricos

Hay tres tipos de literales numéricos: números enteros, números de punto flotante y números imaginarios. No hay literales complejos (los números complejos pueden formarse sumando un número real y un número imaginario: 4 + 3j).

- **Enteros**. (enteros, octales 0o, binarios 0b y hexadecimales 0x)
  7 2147483647 0o177 0b100110111 3 79228162514264337593543950336 0o377 0xdeadbeef 100000000000 100_000_000_000 0b_1110_0101
- **Coma flotante.** (notación tradicional o notación científica)
  3.14 10. .001 1e100 3.14e-10 0e0 3.14_15_93
- **Imaginarios.**
  3.14j 10.j 10j .001j 1e100j 3.14e-10j 3.14_15_93j

Literales de bytes

Los literales de bytes siempre se prefijan con 'b' o 'B'; y producen una instancia del tipo **bytes** en lugar del tipo **str**, por lo que solo pueden contener caracteres ASCII. Los bytes con un valor numérico de 128 o mayor deben ser expresados con secuencias numéricas (\xxxx). Además, pueden ser prefijados con una letra 'r' o 'R', tales cadenas se llaman raw strings y consideran las barras inversas como caracteres literales.

Ejemplos:

```
b'Espa\xc3\xb1a'.decode("UTF-8")     # España
b'Espa\xc3\xb1a'.decode("Latin1")    # España
rb'spam\xddegg'                      # spam\xddegg
```

Literales Cadena

Las cadenas (tipo **str**) en Python pueden ser encerrados entre comillas simples (') o dobles (") pero no se pueden mezclar los delimitadores. También pueden estar encerrados en grupos de tres comillas simples o dobles (a las que generalmente se les llama cadenas de tres comillas) para dividir la cadena en varias líneas físicas. El carácter de la barra inversa (\) se utiliza para escapar los caracteres que de otra manera tienen un significado especial, como la línea nueva, la barra inversa en sí misma, o el carácter de comillas. Son valores inmutables. Se pude acceder a un carácter mediante [] indexados en cero.

```
print('Hola Mundo')
mi_cadena = """Clase que representa
una Persona"""
print(mi_cadena)
print("hola \n \" mundo \" ")
print('hola \n \" mundo \" ')

cadena = "Hola"
cadena[2] ="a"  # error son inmutables
```

Los literales de cadena pueden ser prefijados con una letra 'r' o 'R'; tales cadenas se llaman raw strings y consideran las barras inversas como caracteres literales. A menos que un prefijo 'r' o 'R' esté presente, las secuencias de escape en literales de cadena y bytes se interpretan según reglas similares a las usadas por C estándar. Las secuencias de escape reconocidas son:

Secuencia de escape	Significado
\newline	Barra inversa y línea nueva ignoradas
****	Barra inversa (\)
\'	Comilla simple (')
\"	Comilla doble (")
\a	ASCII Bell (BEL)
\b	ASCII Retroceso (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Retorno de carro (CR)
\t	ASCII Sangría horizontal (TAB)
\v	ASCII Sangría vertical (VT)
\ooo	Carácter con valor octal *oo*
\xhh	Carácter con valor hexadecimal *hh*

Las secuencias de escape que sólo se reconocen en los literales de cadena son:

Secuencia de escape	Significado
\N{name}	El carácter llamado *name* en la base de datos de Unicode
\uxxxx	Carácter con valor hexadecimal de 16 bits *xxxx*
\Uxxxxxxxx	Carácter con valor hexadecimal de 32 bits *xxxxxxxx*

Se permiten múltiples literales de cadenas adyacentes (delimitados por espacios en blanco), utilizando diferentes convenciones y su significado es el mismo que su concatenación. Por lo tanto, **"hola" 'mundo'** es equivalente a **"holamundo"**. Esta característica puede ser utilizada para reducir el número de barras inversas necesarias, para dividir largas cadenas convenientemente a través de largas líneas, o incluso para añadir comentarios a partes de las cadenas, por ejemplo:

```
re.compile("[A-Za-z_]"           # letter or underscore
           "[A-Za-z0-9_]*"       # letter, digit or underscore
    )
```

Téngase en cuenta que esta característica se define a nivel sintáctico, pero se implementa en el momento de la compilación, por lo que el operador "+" debe ser usado para concatenar **expresiones** de cadena al momento de la ejecución.

Cadenas formateadas

Una cadena formateada o f-string es un literal de cadena que se prefija con 'f' o 'F'. Estas cadenas pueden contener campos de reemplazo, que son expresiones delimitadas por llaves {}. Mientras que otros literales de cadena siempre tienen un valor constante, las cadenas formateadas son realmente expresiones evaluadas en tiempo de ejecución. Ver más adelante en la sección cadenas el nuevo modelo **format**.

Tipos de datos

En cualquier lenguaje de programación de alto nivel se manejan tipos de datos. Los tipos de datos definen un conjunto de valores que tienen una serie de características y propiedades determinadas. Los tipos de datos básicos de Python son los booleanos, los numéricos (enteros, punto flotante y complejos), el tipo nulo y las cadenas de caracteres. Los tipos de datos complejos los veremos en unidades posteriores.

Tipado fuerte vs Tipado débil vs No tipado

- Tipado fuerte indica que el lenguaje no permite operaciones entre tipos que no sean los especificados, así no se podrá pasar un float a un parámetro int o sumar una cadena y un número.
- Por contra, débil permite este tipo de operaciones, en la que se pasa o usa otro tipo al especificado.
- Por último, no tipado indica que el tipo del parámetro se determina en tiempo de ejecución.

Tipado estático vs Tipado dinámico

- El tipado estático establece en la definición de una variable o constante el tipo y es inamovible.
- El tipado dinámico hace que el tipo del contenido de una variable o constante pueda cambiar durante la ejecución.

Booleanos

Para tratar las expresiones boolenas, el lenguaje incorpora dos objetos que representarán verdadero (True) y falso (False), pero no existe un tipo como tal.

Nulo

El lenguaje no tiene un valor u objeto que represente el concepto de nulo en una Base de datos, o en otros lenguajes como C, pero existe el objeto **None** que representará la no existencia o el vacío dependiendo del contexto.

Numéricos

Dentro de los números que puede manejar Python se encuentra los enteros (**int**: -5), los decimales (**float**: 2.3) y los irracionales: 3j.

Podemos realizar conversiones entre los diferentes tipos con las funciones: **hex**() para convertir a hexadecimal, **oct**() para octal, **bool**() para booleana, **int**() para entero y **float**() para números decimales y **complex**() para complejos.

La precisión matemática que proporciona el tipo **float** no es adecuada para todos los entornos, por lo que si es necesario se puede utilizar una librería con una precisión mayor: **decimal**.

Se puede hacer uso de funciones matemáticas de forma directa: **abs**, **round**, **min**, **max**, **sum** o de la librería math (**ceil**, **floor**, **trunc**, etc.) importándola antes con **import math**.

```
c1 = 4 + 3j
print(type(c1))
c2 = complex(2, -3)
print(c1 + c2)
print(c1 - c2)
print(3 * c1)
print(c1 * c2)
print(c1 == c2)
print(c1.conjugate())
print(c1 / c2)
print(abs(c1))
print(c1.real)
print(c1.imag)
```

```
# comparadores en secuencia: a mayor que dos y menor que cinco
print(2 < a < 5)
```

Cadenas

Las cadenas en Python (tipo **str**) se inicializan con cualquiera de los siguientes métodos: “Cadena”, ‘Cadena’ o “”” Cadena “””. En este último caso se podrá partir la cadena en varias líneas y se tendrá en cuenta los retornos de carro, en los otros dos métodos deberemos escapar los retornos de carro.

Hay que tener en cuenta que Python trabaja de base con UTF-8 con lo que la representación interna de toda cadena se realiza en esta codificación, por lo que es prescriptivo convertirlo a otra codificación si lo requiriese la salida.

En caso que necesitemos caracteres Unicode con la secuencia de escape deberemos anteponer a la definición de la cadena u, y si necesitamos acceso a los bytes utilizaremos antes de la cadena r (raw) para cadenas en crudo Unicode o b (byte) para ASCII.

Bajo Python no existe el concepto de carácter como otros lenguajes, se implementa con una cadena de longitud uno. El acceso a un carácter individual se realizará mediante índice y la notación corchete, sabiendo que los índices empiezan en 0 y terminan en **len(cadena)-1**.

Las cadenas en Python son objetos **inmutables**, por lo que no permiten modificaciones directas del contenido y cualquiera que se realice con una de sus funciones será necesario asignarla a una variable para no perder dicho cambio.

Podemos conseguir una subcadena simplemente usando la sintaxis corchete [inicio: fin], teniendo en cuenta que fin nunca se incluirá en el resultado y que si usamos valores negativos significa que empezamos desde el final. También consideraremos que la omisión de uno de los parámetros significa: si omitimos el primero indica índice cero y si omitimos el último, indica **len(cadena)**.

```
print("la" in "!Hola")      # True
pal="Una palabra"
print(pal, pal[0],pal[-2])  # Una palabra U r
print(pal[0:2])             # Un
print(pal[4:])              # palabra
print(pal[-3:-1])           # br
print("Una" "Palabra")      # UnaPalabra
pal[3]="b"              # 'str' object does not support item assignment
print(len(pal))
print(pal*2)                 # Una PalabraUna Palabra
```

Para finalizar, los operadores que podemos usar con las cadenas con la concatenación (+), innecesaria para solo cadenas ya que el lenguaje concatena dos cadenas consecutivas, y el operador repetición (*) que repite una cadena un número de veces.

Formato de cadenas

Para formatear una cadena podemos usar dos métodos, uno más antiguo y desaconsejado similar a las cadenas de formato de la función **printf** del C, y el nuevo mecanismo basado en la función **format()**.

Así el mecanismo antiguo sería: “%s es una %s” % (a, b). Este formato utiliza las reglas de sintaxis de **printf** de C añadiendo la posibilidad de que entre el % y el símbolo correspondiente de tipo exista un nombre que se use como clave de un diccionario para el valor en vez de la posición. Podemos aprender más sobre este formato en la siguiente dirección:

https://docs.python.org/es/3/library/stdtypes.html#old-string-formatting

El nuevo mecanismo implica el uso de llaves (**{}**) para indicar en la cadena la posición del parámetro, anteponiendo la letra **f** a la cadena o llamando al método **format**. Se puede utilizar números, claves o dejar vacías las llaves, pero no se deben mezclar. En caso de usar números se puede utilizar el orden que deseemos, no siendo obligatorio que sea secuenciales o incrementales. También permite centrar el valor anteponiendo:

^Tamaño, justificarlo a la derecha **:>tamaño** o a la izquierda **:<Tamaño**. Para la impresión de números podemos rellenar con ceros anteponiendo el carácter que necesitamos.

```
pal="Una palabra"
print("{}".format(pal))
print("{} es {}".format("A", "b"))
print("{0} es {1}".format("A", "b"))
print("{1} es {0}".format("A", "b"))
print("{:<10} {:^10} {:>10} {:0>5}".format("Ab", "CDE", "eF",
23))
# Ab            CDE             eF 00023
# Formato recomendado con el prefijo f
width = 10
precision = 4
value = decimal.Decimal("12.34567")
print(f"result: {value}")
print(f"result: {value =}")  # escribe number_var = valor
print(f"result: {value:{width}.{precision}}")  # nested fields
# 'result:      12.35'
today = datetime(year=2017, month=1, day=27)
print(f"{today:%B %d, %Y}")  # using date format specifier
#  'January 27, 2017'
```

Forzado de tipos

Python es un lenguaje con tipado fuerte pero dinámico, con lo que el tipo de cada variable se establece en tiempo de ejecución, pero tiene un tipo. Una de las mayores críticas que sufren los lenguajes con tipado dinámico son los problemas que introducen al poder el variar el tipo de una variable durante el desarrollo de la aplicación y no detectarse en el momento del análisis sintáctico errores en los usos de los tipos. Cada lenguaje a introducido sus propios mecanismos, pero bajo Python se han creado las anotaciones.

Las anotaciones permiten a otras herramientas (IDEs, analizadores estáticos como mypy o pylint) realizar las mismas tareas que en los lenguajes tradicionales se realizan en el análisis sintáctico. Pero hay que tener claro que, si no usamos esas herramientas, las anotaciones no serán más que meras clarificaciones, ya que el lenguaje no impone restricciones en el tipo de las variables durante la ejecución, pudiendo cambiar si así lo deseamos.

La estructura general de una anotación es:

```
nombre_variable: tipo [valor]
```

Veamos ejemplos a usar.

```
age: int = 1
x_1: float = 1.0
x_2: bool = True
x_3: str = "test"
x_4: bytes = b"test"
```

Los tipos anteriores los usaremos inmediatamente en nuestros programas, los siguientes se utilizarán a lo largo de diferentes capítulos, pero se introducen aquí para tener una referencia completa en un único lugar.

```
x: list[int] = [1]    # Listas
x: set[int] = {6, 7}  # Conjuntos
x: dict[str, float] = {"field": 2.0}  # Diccionarios
x: tuple[int, str, float] = (3, "yes", 7.5) # Tuplas

x: list[int | str] = [3, 5, "test", "fun"] #Valores optatativos
```

Las funciones y métodos que veremos más adelante también deben usar las anotaciones.

```
def plus(num1: int, num2: int) -> int:   # Tipos y devolviendo
      return 0

def show(value: str, excitement: int = 10) -> None:
      pass       # En este caso la función no devuelve nada
```

Cuando se vea el capítulo de POO, las clases son otro tipo más y las clases también pueden ser anotadas.

```
class BankAccount:
    def __init__(self, account_name: str,
                 initial_balance: int = 0) -> None:
        self.account_name: str = account_name
        self.balance: int = initial_balance


    def deposit(self, amount: int) -> None:
        self.balance += amount

account: BankAccount = BankAccount("Alice", 400)
def transfer(src: BankAccount,
             dst: BankAccount,
             amount: int) -> None:
    src.withdraw(amount)
    dst.deposit(amount)
```

Variables y Constantes

En programación, una variable o una constante está formada por un espacio en memoria y un nombre simbólico (un identificador) que está asociado a dicho espacio. Esa área contiene una cantidad de información, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. La diferencia principal es que la variable permite cambiar el contenido almacenado, mientras que la constante una vez definida, su valor nunca cambiará.

- Una variable es un contenedor de valores (generalmente un puntero a una dirección de memoria).
- Las variables son objetos.
- Las reglas de nombrado dictan que deben empezar por letra siempre y que pueden contener números y subrayado (_). Las reglas de estilo de nombrado de variables indican que serán siempre en minúsculas y separadas por un subrayado.
- Las variables no son tipadas, el tipo del contenido de una variable viene determinado en tiempo de ejecución por el contenido, no en tiempo de compilación. Por tanto, no es necesario especificar un tipo a la hora de definir una variable.
- No es necesario definir de forma explícita una variable, se define en el momento del primer uso cuando se le asigne valor, pero vamos a inicializar todas las variables que necesitemos antes de usarlas al comienzo del bloque correspondiente.
- Una variable puede ser eliminada con la función **del nombre_variable**.
- No se pueden usar las palabras clave de la sintaxis como nombres de variables.
- El contenido de una variable es inmutable, por lo que cuando cambiamos su valor se crea un nuevo objeto con el nuevo valor, se cambia el puntero al nuevo objeto y se libera la memoria del valor (objeto) anterior.
  Existen tipos de datos inmutables: números, cadenas, tuplas que una modificación implicará una creación del nuevo objeto y una asignación a la variable; y tipos mutables que se modifica el objeto directamente: diccionario, lista, conjunto.

- Se pueden definir varias variables en una misma línea usando el operador coma (,) y usar el operador empaquetar (*) en la asignación.
- No existen los valores constantes, se usa la nomenclatura en mayúsculas para indicar un valor que no cambia, pero no se implementa ningún control.

```
a, b = 3.5, "Hola"
c = d = 3
a, c = c, a
e, *f = 3, 4, 5, 6   # el asterisco es el resto de valores
*g, h = 7, 8, 9   # el asterisco es el resto de valores
del b
print(a, c , d, e, f, g, h) # 3 3.5 3 3 [4, 5, 6] [7, 8] 9
a, b  = b, a  # intercambio de variables
VALOR_CTE = 23  # es solo nomenclatura no se controla
print(VALOR_CTE)
VALOR_CTE = 22
print(VALOR_CTE)
```

Operadores y expresiones

Un operador es un token que realiza alguna operación sobre uno o más operandos. Los operadores se clasifican según su tipo.

Operador de concatenación de cadenas de caracteres

Una de las operaciones más básicas cuando se trabaja con cadenas de caracteres es la concatenación. Esto consiste en unir dos cadenas en una sola, siendo el resultado una nueva cadena.

La forma más simple de concatenar dos cadenas en Python es utilizando el operador de concatenación +:

```
>>> hola = 'Hola'
>>> python = 'Pythonista'
>>> hola_python = hola + ' ' + python  # concatenamos 3 strings
>>> print(hola_python)
Hola Pythonista
```

Operadores lógicos o booleanos

A la hora de operar con valores booleanos, tenemos a nuestra disposición los operadores and, or y not.

> Las operaciones and, or y not realmente no devuelven True o False, sino que devuelven uno de los operandos como veremos en el cuadro de abajo.

Operación	Resultado	Descripción
a or b	Si a se evalúa a falso, entonces devuelve b, si no devuelve a	Solo se evalúa el segundo operando si el primero es falso
a and b	Si a se evalúa a falso, entonces devuelve a, si no devuelve b	Solo se evalúa el segundo operando si el primero es verdadero
not a	Si a se evalúa a falso, entonces devuelve True, si no devuelve False	Tiene menos prioridad que otros operadores no booleanos

Ejemplos:

```
>>> x = True
>>> y = False
>>> x or y
True
>>> x and y
False
>>> not x
```

```
False
>>> x = 0
>>> y = 10
>>> x or y
10
>>> x and y
0
>>> not x
True
```

Operadores de comparación

Los operadores de comparación se utilizan, como su nombre indica, para comparar dos o más valores. El resultado de estos operadores siempre es True o False.

Operador	Descripción
>	Mayor que. True si el operando de la izquierda es estrictamente mayor que el de la derecha; False en caso contrario.
>=	Mayor o igual que. True si el operando de la izquierda es mayor o igual que el de la derecha; False en caso contrario.
<	Menor que. True si el operando de la izquierda es estrictamente menor que el de la derecha; False en caso contrario.
<=	Menor o igual que. True si el operando de la izquierda es menor o igual que el de la derecha; False en caso contrario.
==	Igual. True si el operando de la izquierda es igual que el de la derecha; False en caso contrario.
!=	Distinto. True si los operandos son distintos; False en caso contrario.

Ejemplos:

```
>>> x = 9
>>> y = 1
>>> x < y
False
>>> x > y
True
>>> x == y
False
```

Consideraciones sobre los operadores de comparación

Los objetos de diferentes tipos, excepto los tipos numéricos, nunca se comparan igual. El operador == siempre está definido, pero para algunos tipos de objetos (por ejemplo, objetos de clase) es equivalente a **is**.

Las instancias no idénticas de una clase normalmente se comparan como no iguales a menos que la clase defina el método __eq__(). Las instancias de una clase no se pueden ordenar con respecto a otras instancias de la misma clase u otros tipos de objeto, a menos que la clase defina los métodos __lt__(), __gt__().

Los operadores de comparación se pueden concatenar.

Ejemplo:

```
>>> x = 9
>>> 1 < x and x < 20
True
>>> 1 < x < 20
True
```

Operadores aritméticos en Python

En cuanto a los operadores aritméticos, estos permiten realizar las diferentes operaciones aritméticas del álgebra: suma, resta, producto, división, … Estos operadores Python son de los más utilizados. El listado completo es el siguiente:

Operador	Descripción

+	Suma dos operandos.
-	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un float).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Ejemplos:

```
>>> x = 7
>>> y = 2
>>> x + y  # Suma
9
>>> x - y  # Resta
5
>>> x * y  # Producto
14
>>> x / y  # División
3.5
>>> x % y  # Resto
1
>>> x // y  # Cociente
3
>>> x ** y  # Potencia
49
```

Operadores a nivel de bits

Los operadores a nivel de bits actúan sobre los operandos como si fueran una cadena de dígitos binarios. Como su nombre indica, actúan sobre los operandos bit a bit. Son los siguientes:

Operación	**Descripción**
x | y	or bit a bit de x e y.
x ^ y	or exclusivo bit a bit de x e y.
x & y	and bit a bit de x e y.
x << n	Desplaza x n bits a la izquierda.
x >> n	Desplaza x n bits a la derecha.
~x	not x. Obtiene los bits de x invertidos.

Ejemplo: Supongamos que tenemos el entero 2 (en bits es 00010) y el entero 7 (00111). El resultado de aplicar las operaciones anteriores es:

```
>>> x = 2
>>> y = 7
>>> x | y
7
>>> x ^ y
5
>>> x & y
2
>>> x << 1
4
>>> x >> 1
1
>>> ~x
-3
```

Operadores de asignación

El operador de asignación se utiliza para asignar un valor a una variable. Como se ha mencionado en otras secciones, este operador es el signo =.

Además del operador de asignación, existen otros operadores de asignación compuestos que realizan una operación básica sobre la variable a la que se le asigna el valor. Por ejemplo, x += 1 es lo mismo que x = x + 1. Los operadores compuestos realizan la operación que hay antes del signo igual, tomando como operandos la propia variable y el valor a la derecha del signo igual.

Operador	Ejemplo	Equivalencia
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2
&=	x &= 2	x = x & 2
|=	x |= 2	x = x | 2
^=	x ^= 2	x = x ^ 2
<<=	x <<= 2	x = x << 2
>>=	x >>= 2	x = x >> 2

Operador morsa

Python ha creado un operador de morsa (:=) en la versión 3.8 que permite asignar valores a una variable como parte de una expresión. Se puede realizar gran cantidad de cosas interesantes, pero el mejor aspecto es la reducción de líneas necesarias para tareas muy comunes.

Ahora, en vez de escribir esto:

```
line = f.readline()
while line:
  line = f.readline()
```

Puedes escribir esto:

```
while line := f.readline():
```

Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (**list**, **tuple**, **dict**, **set** o **str**).

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.

Ejemplos:

```
>>> lista = [1, 3, 2, 7, 9, 8, 6]
>>> 4 in lista
False
>>> 3 in lista
True
>>> 4 not in lista
True
```

Operadores de identidad

Por último, los operadores de identidad se utilizan para comprobar si dos variables son, o no, el mismo objeto.

Operador	Descripción
is	Devuelve True si ambos operandos hacen referencia al mismo objeto; False en caso contrario.
is not	Devuelve True si ambos operandos no hacen referencia al mismo objeto; False en caso contrario.

Ejemplos:

```
>>> x = 4
>>> y = 2
>>> lista = [1, 5]
>>> x is lista
False
>>> x is y
False
>>> x is 4
True
```

Prioridad de los operadores en Python

Al igual que ocurre en las matemáticas, los operadores en Python tienen un orden de prioridad. Este orden es el siguiente, de menos prioritario a más prioritario: asignación; operadores booleanos; operadores de comparación, identidad y pertenencia; a nivel de bits y finalmente los aritméticos (con el mismo orden de prioridad que en las matemáticas). Este orden de prioridad se puede alterar con el uso de los paréntesis ()

Expresiones

Una expresión es una combinación de valores, variables y operadores. La evaluación de una expresión produce un valor, esta es la razón por la que las expresiones pueden aparecer en el lado derecho de las sentencias de asignación. Un valor, por sí mismo, se considera como una expresión, lo mismo ocurre para las variables.

Expresa en Python las siguientes expresiones

- Mayor de edad y cinco años o más cotizados
- Una manzana es seleccionada si el color es amarillo o rojo y además el diámetro está entre 15 y 25 cm.
- La presión de un balón debe estar entre 3 y 5 bares y la circunferencia no debe pasar de 45 cm
- El número de bolas rojas serán tres si el de amarillas es dos, debe ser cinco si el de amarillas es cinco.
- Para seleccionar un jugador debe medir entre 180 y 210 cm. Si es más alto de 210cm tiene que pesar menos de 120kg. Si es menor de 180cm tiene que saltar más de 50cm.
- Para comprar un vehículo debe ser azul si es un deportivo o verde si es un 4x4, pero si quiero una moto la compraré roja o negra.
- La jubilación se produce a los 67 años con 35 cotizados. a partir de 65 con 40 cotizados y 20 o más continuos

Bibliotecas

La biblioteca estándar de Python es muy amplia y ofrece una gran cantidad de facilidades. La biblioteca contiene módulos incorporados que brindan acceso a las funcionalidades del sistema como entrada y salida de archivos que serían de otra forma inaccesibles para los programadores en Python, así como módulos escritos en Python que proveen soluciones estandarizadas para los diversos problemas que pueden ocurrir en el día a día en la programación. Veremos elementos de esta biblioteca a lo largo de todo el curso.

https://docs.python.org/es/3/library/index.html

Ejercicios

1. Escribe un programa que dé los “buenos días”.
2. Escribe un programa que calcule y muestre el área de un cuadrado de lado igual a 5.
3. Escribe un programa que calcule el área de un cuadrado cuyo lado se introduce por teclado.
4. Escribe un programa que lea dos números, calcule y muestre el valor de sus suma, resta, producto y división.
5. Escribe un programa que toma como dato de entrada un número que corresponde a la longitud de un radio y nos escribe la longitud de la circunferencia, el área del círculo y el volumen de la esfera que corresponden con dicho radio.
6. Escribe un programa que dado el precio de un artículo y el precio de venta real nos muestre el porcentaje de descuento realizado.
7. Escribe un programa que lea un valor correspondiente a una distancia en millas marinas y escriba la distancia en metros. Sabiendo que una milla marina equivale a 1.852 metros.
8. Calcular el consumo de un coche a los 100 kilómetros dados el gasto en dinero realizado, el precio del litro y el número de kilómetros recorridos.
9. Calcular el coste de la vida dados los precios de tres productos este año y el año pasado.
10. Escribir un programa que pregunte al usuario por el número de horas trabajadas y el coste por hora. Después debe mostrar por pantalla la paga que le corresponde.
11. Escribir un programa que lea un entero positivo, introducido por el usuario y después muestre en pantalla la suma de todos los enteros desde 1 hasta el número. La suma de los primeros enteros positivos puede ser calculada de la siguiente forma n(n+1)/2.
12. Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal, lo almacene en una variable y muestre por pantalla la frase Tu índice de masa corporal es <imc> donde <imc> es el índice de masa corporal calculado redondeado con dos decimales.
13. Escribir un programa que pida al usuario dos números enteros y muestre por pantalla la <n> entre <m> da un cociente <c> y un resto <r> donde <n> y <m> son los números introducidos por el usuario, y <c> y <r> son el cociente y el resto de la división entera respectivamente.
14. Una juguetería tiene mucho éxito en dos de sus productos: payasos y muñecas. Suele hacer venta por correo y la empresa de logística les cobra por peso de cada paquete así que deben calcular el peso de los payasos y muñecas que saldrán en cada paquete a demanda. Cada payaso pesa 112 g y cada muñeca 75 g. Escribir un programa que lea el número de payasos y muñecas vendidos en el último pedido y calcule el peso total del paquete que será enviado.
15. Imagina que acabas de abrir una nueva cuenta de ahorros que te ofrece el 4% de interés al año. Estos ahorros debido a intereses, que no se cobran hasta finales de año, se te añaden al balance final de tu cuenta de ahorros. Escribir un programa que comience leyendo la cantidad de dinero depositada en la cuenta de ahorros, introducida por el usuario. Después el programa debe calcular y mostrar por pantalla la cantidad de ahorros tras el primer, segundo y tercer años. Redondear cada cantidad a dos decimales.
16. Una panadería vende barras de pan a 3.49€ cada una. El pan que no es el día tiene un descuento del 60%. Escribir un programa que comience leyendo el número de barras vendidas que no son del día. Después el programa debe mostrar el precio habitual de una barra de pan, el descuento que se le hace por no ser fresca y el coste final total.

17. Escribir un programa que solicite al usuario ingresar un número con decimales y almacenarlo en una variable. A continuación, el programa debe solicitar al usuario que ingrese un número entero y guardarlo en otra variable. En una tercera variable se deberá guardar el resultado de la suma de los dos números ingresados por el usuario. Por último, se debe mostrar en pantalla el texto "El resultado de la suma es [suma]", donde "[suma]" se reemplazará por el resultado de la operación.
18. Escribir un programa que solicite al usuario el ingreso de una temperatura en escala Fahrenheit (debe permitir decimales) y le muestre el equivalente en grados Celsius. La fórmula de conversión que se usa para este cálculo es: Celsius = (5/9) * (Fahrenheit-32).
19. Escribir un programa que solicite al usuario el ingreso de dos palabras, las cuales se guardarán en dos variables distintas. A continuación, almacenará en una variable la concatenación de la primera palabra, más un espacio, más la segunda palabra. Mostrará este resultado en pantalla.
20. Escribir un programa que solicite al usuario el ingreso de un texto y almacene ese texto en una variable. A continuación, mostrar en pantalla la primera letra del texto ingresado. Luego, solicitar al usuario que ingrese un número positivo menor a la cantidad de caracteres que tiene el texto que ingresó (por ejemplo, si escribió la palabra "HOLA", tendrá que ser un número entre 0 y 4) y almacenar este número en una variable llamada índice. Mostrar en pantalla el carácter del texto ubicado en la posición dada por índice.
21. Escribir un programa que solicite al usuario que ingrese cuántos shows musicales ha visto en el último año y almacene ese número en una variable. A continuación, mostrar en pantalla un valor de verdad (True o False) que indique si el usuario ha visto más de 3 shows.
22. Escribir un programa que le solicite al usuario ingresar una fecha formada por 8 números, donde los primeros dos representan el día, los siguientes dos el mes y los últimos cuatro el año (DDMMAAAA). Este dato debe guardarse en una variable con tipo int (número entero). Finalmente, mostrar al usuario la fecha con el formato DD / MM / AAAA.
23. Escribir un programa para solicitar al usuario el ingreso de un número entero y que luego imprima un valor booleano dependiendo de si el número es par o no. Recordar que un número es par si el resto al dividirlo por 2, es 0.
24. Escribir un programa que le solicite al usuario su edad y la guarde en una variable. Que luego solicite la cantidad de artículos comprados en una tienda y la guarde en otra variable. Finalmente, mostrar en pantalla un valor de verdad (True o False) que indique si el usuario es mayor de 18 años de edad y además compró más de 1 artículo.
25. Escribir un programa que, dada una cadena de texto por el usuario, imprima True si la cantidad de caracteres en la cadena es un número impar, o False si no lo es.
26. Escribir un programa para pedir al usuario su nombre y luego el nombre de otra persona, almacenando cada nombre en una variable. Luego mostrar en pantalla un valor de verdad que indique si: los nombres de ambas personas comienzan con la misma letra o si terminan con la misma letra. Por ejemplo, si los nombres ingresados son María y Marcos, se mostrará True, ya que ambos comienzan con la misma letra. Si los nombres son Ricardo y Gonzalo se mostrará True, ya que ambos terminan con la misma letra. Si los nombres son Florencia y Lautaro se mostrará False, ya que no coinciden ni la primera ni la última letra.
27. Calcular las soluciones de las raíces de una ecuación de segundo grado, asumiendo que el discriminante es >= 0
28. Hacer un conversor de divisas: $, € y Libras