1. Introducción al lenguaje de programación.

1.1. ¿Qué es Python?

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible. Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.

Lenguaje interpretado o de script

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilados).

Fuertemente tipado

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o string) no podremos tratarla como un número (sumar la cadena “9” y el número 8). En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores.

Multiplataforma

El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios. Orientado a objetos

Orientado a objetos

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones

entre los objetos. Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

Uso de Python

Al ser un lenguaje de propósito general, aplicaciones en todos los campos científico-tecnológicos:

Análisis de datos.
Aplicaciones de escritorio.
Bases de datos relacionales / NoSQL
Buenas prácticas de programación / Patrones de diseño.
Concurrencia.
Criptomonedas / Blockchain.
Desarrollo de aplicaciones multimedia.
Desarrollo de juegos.
Desarrollo en dispositivos embebidos.
Desarrollo móvil.
Desarrollo web.
DevOps / Administración de sistemas / Scripts de automatización.
Gráficos por ordenador.
Inteligencia artificial.
Internet de las cosas.

Machine Learning.
Programación de parsers / scrapers / crawlers.
Programación de redes.
Propósitos educativos.
Prototipado de software.
Seguridad.
Tests automatizados.

Python2 Python3

Versión	Fecha
PYTHON 1.0	ENERO 1994
PYTHON 1.5	DICIEMBRE 1997
PYTHON 1.6	SEPTIEMBRE 2000
PYTHON 2.0	OCTUBRE 2000
PYTHON 2.1	ABRIL 2001
PYTHON 2.2	DICIEMBRE 2001
PYTHON 2.3	JULIO 2003
PYTHON 2.4	NOVIEMBRE 2004

PYTHON 2.5	SEPTIEMBRE 2006
PYTHON 2.6	OCTUBRE 2008
PYTHON 2.7	JULIO 2010
PYTHON 3.0	DICIEMBRE 2008
PYTHON 3.1	JUNIO 2009
PYTHON 3.2	FEBRERO 2011
PYTHON 3.3	SEPTIEMBRE 2012
PYTHON 3.4	MARZO 2014
PYTHON 3.5	SEPTIEMBRE 2015
PYTHON 3.6	DICIEMBRE 2016
PYTHON 3.7	JUNIO 2018
PYTHON 3.8	OCTUBRE 2019
PYTHON 3.9	OCTUBRE 2020

El cambio de Python 2 a Python 3 se perdió la compatibilidad en muchas de las estructuras del lenguaje. la necesidad de aplicar estas modificaciones en beneficio del rendimiento y expresividad del lenguaje de programación. Este cambio implicaba que el código escrito en Python 2 no funcionaría (de manera inmediata) en Python 3.En enero de 2020 finalizó oficialmente el soporte a la versión 2.7 del lenguaje de programación Python.

1.2. Instalación de Python

Primero comprueba si tu ordenador ejecuta la versión 32 bits de Windows o la de 64, en "Tipo de sistema" en la página de "Acerca de". Para llegar a esta página, intenta uno de estos métodos: Presiona la tecla de Windows y la tecla Pause/Break al mismo tiempo Abre el Panel de Control desde el menú de Windows, después accede a Sistema & y Seguridad, luego a Sistema Presiona el botón de Windows, luego accede a Configuración > Sistema > Acerca de Puedes descargar Python para Windows desde la siguiente web

Da clic en el enlace "Latest Python 3 Release -Python x.x.x". Si tu ordenador ejecuta la versión de 64 bits de Windows, descarga Windows x86-64 executable installer. De lo contrario, descarga Windows x86 ejecutable installer. Después de descargar el instalador, deberías ejecutarlo (dándole doble click) y seguir las instrucciones.

Una cosa para tener en cuenta: Durante la instalación, verás una ventana de "Setup". Asegúrate de marcar las casillas "Add Python 3.6 to PATH" o "Add Python to your environment variables" y hacer click en "Install Now", como se muestra aquí (puede que se vea un poco diferente si estás

instalando una versión diferente):

Cuando la instalación se complete, verás un cuadro de diálogo con un enlace que puedes seguir para saber más sobre Python o sobre la versión que has instalado. Cierra o cancela ese diálogo. Comprobar que se instalo Python con el siguiente instrucción `Python --version`

Anaconda

Anaconda es un conjunto de herramientas, orientadas en principio a la ciencia de datos, pero que podemos utilizarlas para desarrollo general en Python (junto con otras librerías adicionales).

Miniconda

Instalador mínimo que trae por defecto Python y un pequeño número de paquetes útiles.

PIP Gestión de paquetes

La instalación limpia de Python ya ofrece de por sí muchos paquetes y módulos que vienen por defecto. Es lo que se llama la librería estándar. Pero una de las características más destacables de Python es su inmenso numero de paquetes disponibles en el Python

Package Index (PyPI)

Para gestionar los paquetes que tenemos en nuestro sistema se utiliza la herramienta pip, una utilidad que también se incluye en la instalación de Python.

Desinstalar y actualizar paquetes

Instrucciones que usaríamos para cada una de estas operaciones:

paquete pandas utilizando pip

```
$ pip install pandas
$ pip uninstall pandas
$ pip install pandas --upgrade
```

Existen dos formas de ejecutar código Python. Podemos escribir líneas de código en el intérprete y obtener una respuesta del intérprete para cada línea (sesión interactiva) o bien podemos escribir el código de un programa en un archivo de texto y ejecutarlo. A la hora de realizar una sesión interactiva instalar y utilizar iPython "ipynb", en lugar de la consola interactiva de Python.

Cuando vamos a trabajar con Python debemos tener instalado, el intérprete del lenguaje ,nos permitirá ejecutar nuestro código. Pero normalmente poder escribir programas algo más largos, por lo que también necesitaremos un editor. Un editor es un programa que nos permite crear archivos de código extensión *.py, que luego son ejecutados por el intérprete.

Hay otra herramienta interesante dentro del entorno de desarrollo que sería el depurador. por su nombre inglés debugger. Es el módulo que nos permite ejecutar paso a paso nuestro código y visualizar qué está ocurriendo en cada momento. Se suele usar normalmente para encontrar fallos (bugs) en nuestros programas y poder solucionarlos.

intérprete

Para hacer una prueba inicial del intérprete vamos a retomar el primer programa. Modo interactivo: ejecutando la orden en tiempo real
Modo script: escribiendo el código en Python y ejecutando después con el intérprete

«Hello, World». Para ello escribimos lo siguiente en el intérprete y pulsamos la tecla ENTER:

```
>>> print("Hello, World")
```

Python ruta/hm.py

./hm.py

```
# esto es una cadena
```

```
c = “Hola Mundo”
```

```
print(c)
```

```
# y esto es un entero
```

```
e = 23
```

```
# podemos comprobarlo con la función type
```

```
type(c)
```

```
type(e)
```

Entornos virtuales

Cuando trabajamos en distintos proyectos, no todos ellos requieren los mismos paquetes ni siquiera la misma versión de Python. La gestión de estas situaciones no es sencilla si únicamente instalamos paquetes y manejamos configuraciones a nivel globa Como su propio nombre indica se trata de crear distintos entornos en función de las necesidades de cada proyecto, y esto nos permite establecer qué versión de Python usaremos y qué paquetes instalaremos.

jupyter Notebook

Jupyter Notebook es una aplicación «open-source» que permite crear y compartir documentos que contienen código, ecuaciones, visualizaciones y texto narrativo. Podemos utilizarlo para propósito general aunque suele estar más enfocado a ciencia de datos: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización o «machine-learning».

Podemos verlo como un intérprete de Python (contiene un «kernel» que permite ejecutar código) con la capacidad de incluir documentación en formato Markdown, lo que potencia sus funcionalidades y lo hace adecuado para preparar cualquier tipo de material vinculado con lenguajes de programación. Aunque su uso está más extendido en el mundo Python, existen muchos otros «kernels» sobre los que trabajar en Jupyter Notebook

---

Datos

Los programas están formados por código y datos. Pero a nivel interno de la memoria no son más que una secuencia de bits. La interpretación de estos bits depende del lenguaje de programación, que almacena en la memoria no sólo el puro dato sino distintos metadatos. Cada parte de memoria contiene realmente un objeto, de ahí que se diga que en Python todo son objetos. Y cada objeto tiene, al menos, los siguientes campos:

- Un tipo del dato almacenado.
- Un identificador único para distinguirlo de otros objetos.
- Un valor consistente con su tipo.

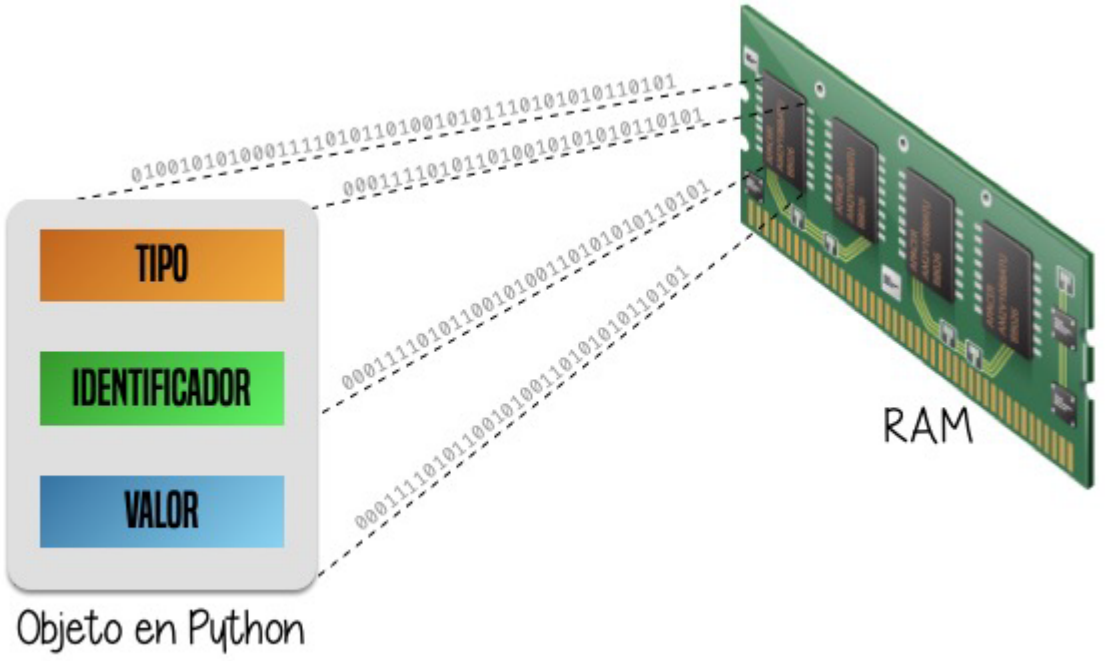

Tabla 1: Tipos de datos en Python

Nombre	Tipo	Ejemplos
Booleano	bool	`True`, `False`
Entero	int	`21`, `34500`, `34_500`
Flotante	float	`3.14`, `1.5e3`
Complejo	complex	`2j`, `3 + 5j`
Cadena	str	`'tfn'`, `'''tenerife - islas canarias'''`
Tupla	tuple	`(1, 3, 5)`
Lista	list	`['Chrome', 'Firefox']`
Conjunto	set	`set([2, 4, 6])`
Diccionario	dict	`{'Chrome': 'v79' , 'Firefox': 'v71'}`

Palabras reservadas

```
help("keywords")
```

```
>>>help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not

>>>
```

Python Indentation

La sangría se refiere a los espacios al comienzo de una línea de código. Mientras que en otros lenguajes de programación la sangría en el código es solo para facilitar la lectura, la sangría en Python es muy importante. Python usa sangría para indicar un bloque de código.

Ejemplo
if 5 > 2:
print("cinco es mayor que dos!")
Python dará un error si omite la sangría:

Ejemplo
Error de sintaxis:
if 5 > 2:
print("cinco es mayor que dos!")

Python Built in Functions conjunto de funciones integradas.

Función	Descripción
ABS()	DEVUELVE EL VALOR ABSOLUTO DE UN NÚMERO
ALL()	DEVUELVE VERDADERO SI TODOS LOS ELEMENTOS EN UN OBJETO ITERABLE SON VERDADEROS
ANY()	DEVUELVE TRUE SI ALGÚN ELEMENTO EN UN OBJETO ITERABLE ES VERDADERO
ASCII()	DEVUELVE UNA VERSIÓN LEGIBLE DE UN OBJETO. REEMPLAZA LOS CARACTERES QUE NO SON ASCII CON UN CARÁCTER DE ESCAPE
BIN()	DEVUELVE LA VERSIÓN BINARIA DE UN NÚMERO
BOOL()	DEVUELVE EL VALOR BOOLEANO DEL OBJETO ESPECIFICADO
BYTEARRAY()	DEVUELVE UNA MATRIZ DE BYTES

BYTES()	DEVUELVE UN OBJETO DE BYTES
CALLABLE()	DEVUELVE TRUE SI EL OBJETO ESPECIFICADO ES INVOCABLE, DE LO CONTRARIO FALSE
CHR()	DEVUELVE UN CARÁCTER DEL ESPECIFICADO CÓDIGO UNICODE.
CLASSMETHOD()	CONVIERTE UN MÉTODO EN UN MÉTODO DE CLASE
COMPILE()	DEVUELVE EL CÓDIGO FUENTE ESPECIFICADO COMO UN OBJETO, LISTO PARA EJECUTARSE
COMPLEX()	DEVUELVE UN NÚMERO COMPLEJO
DELATTR()	ELIMINA EL ATRIBUTO ESPECIFICADO (PROPIEDAD O MÉTODO) DEL OBJETO ESPECIFICADO
DICT()	DEVUELVE UN DICCIONARIO (ARRAY)
DIR()	DEVUELVE UNA LISTA DE LAS PROPIEDADES Y MÉTODOS DEL OBJETO ESPECIFICADO
DIVMOD()	DEVUELVE EL COCIENTE Y EL RESTO CUANDO ARGUMENTO1 SE DIVIDE POR ARGUMENTO2
ENUMERATE()	TOMA UNA COLECCIÓN (POR EJEMPLO, UNA TUPLA) Y LA DEVUELVE COMO UN OBJETO DE ENUMERACIÓN
EVAL()	EVALÚA Y EJECUTA UNA EXPRESIÓN
EXEC()	EJECUTA EL CÓDIGO (U OBJETO) ESPECIFICADO
FILTER()	UTILICE UNA FUNCIÓN DE FILTRO PARA EXCLUIR ELEMENTOS EN UN OBJETO ITERABLE
FLOAT()	DEVUELVE UN NÚMERO DE COMA FLOTANTE
FORMAT()	DA FORMATO A UN VALOR ESPECIFICADO
FROZENSET()	DEVUELVE UN OBJETO FROZENSET

GETATTR()	DEVUELVE EL VALOR DEL ATRIBUTO ESPECIFICADO (PROPIEDAD O MÉTODO)
GLOBALS()	DEVUELVE LA TABLA DE SÍMBOLOS GLOBAL ACTUAL COMO UN DICCIONARIO
HASATTR()	DEVUELVE TRUE SI EL OBJETO ESPECIFICADO TIENE EL ATRIBUTO ESPECIFICADO (PROPIEDAD/MÉTODO)
HASH()	DEVUELVE EL VALOR HASH DE UN OBJETO ESPECIFICADO
HELP()	EJECUTA EL SISTEMA DE AYUDA INTEGRADO
HEX()	CONVIERTE UN NÚMERO EN UN VALOR HEXADECIMAL
ID()	DEVUELVE EL ID DE UN OBJETO
INPUT()	PERMITIR ENTRADA DE USUARIO
INT()	DEVUELVE UN NÚMERO ENTERO
ISINSTANCE()	DEVUELVE TRUE SI UN OBJETO ESPECIFICADO ES UNA INSTANCIA DE UN OBJETO ESPECIFICADO
ISSUBCLASS()	DEVUELVE TRUE SI UNA CLASE ESPECÍFICA ES UNA SUBCLASE DE UN OBJETO ESPECÍFICO
ITER()	DEVUELVE UN OBJETO ITERADOR
LEN()	DEVUELVE LA LONGITUD DE UN OBJETO
LIST()	DEVUELVE UNA LISTA
LOCALS()	DEVUELVE UN DICCIONARIO ACTUALIZADO DE LA TABLA DE SÍMBOLOS LOCAL ACTUAL
MAP()	DEVUELVE EL ITERADOR ESPECIFICADO CON LA FUNCIÓN ESPECIFICADA APLICADA A CADA ELEMENTO

MAX()	DEVUELVE EL ELEMENTO MÁS GRANDE EN UN ITERABLE
MEMORYVIEW()	DEVUELVE UN OBJETO DE VISTA DE MEMORIA
MIN()	DEVUELVE EL ELEMENTO MÁS PEQUEÑO EN UN ITERABLE
NEXT()	DEVUELVE EL SIGUIENTE ELEMENTO EN UN ITERABLE
OBJECT()	DEVUELVE UN NUEVO OBJETO
OCT()	CONVIERTE UN NÚMERO EN OCTAL
OPEN()	ABRE UN ARCHIVO Y DEVUELVE UN OBJETO DE ARCHIVO
ORD()	CONVERTIR UN NÚMERO ENTERO QUE REPRESENTA EL UNICODE DEL CARÁCTER ESPECIFICADO
POW()	DEVUELVE EL VALOR DE X A LA POTENCIA DE 3 (LO MISMO QUE X * X * X):
PRINT()	IMPRIME EN EL DISPOSITIVO DE SALIDA ESTÁNDAR
PROPIEDAD()	OBTIENE, ESTABLECE, ELIMINA UNA PROPIEDAD
RANGE()	DEVUELVE UNA SECUENCIA DE NÚMEROS, COMENZANDO DESDE 0 E INCREMENTANDO EN 1 (POR DEFECTO)
REPR()	DEVUELVE UNA VERSIÓN LEGIBLE DE UN OBJETO
REVERSED()	DEVUELVE UN ITERADOR INVERTIDO
ROUND()	REDONDEA NÚMEROS
SET()	DEVUELVE UN NUEVO OBJETO ESTABLECIDO
SETATTR()	ESTABLECE UN ATRIBUTO (PROPIEDAD/MÉTODO) DE UN OBJETO

SLICE()	DEVUELVE UN OBJETO DE DIVISIÓN
SORTED()	DEVUELVE UNA LISTA ORDENADA
STATICMETHOD()	CONVIERTE UN MÉTODO EN UN MÉTODO ESTÁTICO
STR()	DEVUELVE UN OBJETO DE CADENA
SUM()	SUMA LOS ELEMENTOS DE UN ITERADOR
SUPER()	DEVUELVE UN OBJETO QUE REPRESENTA LA CLASE PADRE
TUPLA()	DEVUELVE UNA TUPLA
TYPE()	DEVUELVE EL TIPO DE UN OBJETO
VARS()	DEVUELVE LA PROPIEDAD __DICT__ DE UN OBJETO
ZIP()	DEVUELVE UN ITERADOR, DE DOS O MÁS ITERADORES

---

2. Cadenas

Las cadenas no son más que texto encerrado entre comillas simples (‘cadena’) o dobles (“cadena”). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación. Una cadena puede estar precedida por el carácter u o el carácter r, los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación Unicode y una cadena raw (del inglés, cruda). Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para las expresiones regulares, como veremos en el capítulo correspondiente. unicode = u”äóè”

raw = r”\n” También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma podremos escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter \n, así como las comillas sin tener que escaparlas. triple = “““primera linea esto se vera en otra linea”””

Las cadenas también admiten operadores como +, que funciona realizando una concatenación de las cadenas utilizadas como operandos y *, en la que se repite la cadena tantas veces como lo indique el número utilizado como segundo operando.

herdoc = ''' String Methods Python tiene built-in "conjunto de métodos integrados" que puede usar en cadenas capitalize() casefold() format() etc... '''

3. Números enteros y reales.

Números enteros

Son aquellos números positivos o negativos que no tienen decimales (además del cero).

En Python se pueden representar mediante el tipo int (de integer, entero) o el tipo long (largo). La única diferencia es que el tipo long permite almacenar números más grandes. Es aconsejable no utilizar el tipo long a menos que sea necesario, para no malgastar memoria.

El tipo int de Python se implementa a bajo nivel mediante un tipo long de C. Y dado que Python utiliza C por debajo, como C, y a diferencia de Java, el rango de los valores que puede representar depende de la plataforma.

En la mayor parte de las máquinas el long de C se almacena utilizando 32 bits, es decir, mediante el uso de una variable de tipo int de Python podemos almacenar números de -231 a 231 - 1, o lo que es lo mismo, de -2.147.483.648 a 2.147.483.647. En plataformas de 64 bits, el rango es de -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807.

El tipo long de Python permite almacenar números de cualquier precisión, estando limitados solo por la memoria disponible en la máquina. Al asignar un número a una variable esta pasará a tener tipo int, a menos que el número sea tan grande como para requerir el uso del tipo long.

Reales

Los números reales son los que tienen decimales. En Python se expresan mediante el tipo float.

En otros lenguajes de programación, como C, tenemos también el tipo double, similar a float pero de mayor precisión (double = doble precisión). Python, sin embargo, implementa su tipo float a bajo nivel mediante una variable de tipo double de C, es decir, utilizando 64 bits, luego en Python siempre se utiliza doble precisión, y en concreto se sigue el estándar IEEE 754: 1 bit para el

signo, 11 para el exponente, y 52 para la mantisa. Esto significa que los valores que podemos representar van desde ±2,2250738585072020 x 10-308 hasta ±1,7976931348623157×10308.

La mayor parte de los lenguajes de programación siguen el mismo esquema para la representación interna. Pero como muchos sabréis esta tiene sus limitaciones, impuestas por el hardware.

Por eso desde Python 2.4 contamos también con un nuevo tipo Decimal, para el caso de que se necesite representar fracciones de forma más precisa. Sin embargo, este tipo está fuera del alcance de este tutorial, y sólo es necesario para el ámbito de la programación científica y otros relacionados. Para aplicaciones normales poder utilizar el tipo float sin miedo, como ha venido haciéndose desde hace años, aunque teniendo en cuenta que los números en coma flotante no son precisos (ni en este ni en otros lenguajes de programación). Para representar un número real en Python se escribe primero la parte entera, seguido de un punto y por último la parte decimal

3. Operadores aritméticos.

Operadores

Veamos ahora qué podemos hacer con nuestros números usando los operadores por defecto. Para operaciones más complejas podemos recurrir al módulo math.

OPERADOR	DESSCRIPCION	EJEMPLO	RESULTADO
+	SUMA	C4+4	C=8
-	RESTA	C=4-2	C=2
-	NEGACIÓN	C-7	C=-7
*****	MULTIPLICACIÓN	C=2*5	C=10
**	POTENCIACIÓN	C=2**3	C=8
/	DIVISIÓN	C=7.5/2	C=3.75

OPERADOR	DESSCRIPCION	EJEMPLO	RESULTADO
//	DIVISIÓN ENTERA	C=7.5//2	C=3.0
%	MÓDULO	C=8%3	C=2

Tabla

Puede que tengas dudas sobre cómo funciona el operador de módulo, y cuál es la diferencia entre división y división entera. El operador de módulo no hace otra cosa que devolvernos el resto de la división entre los dos operandos. En el ejemplo, 7/2 sería 3, con 1 de resto, luego el módulo es 1.

La diferencia entre división y división entera no es otra que la que indica su nombre. En la división el resultado que se devuelve es un número real, mientras que en la división entera el resultado que se devuelve es solo la parte entera. No obstante hay que tener en cuenta que si utilizamos dos operandos enteros, Python determinará que queremos que la variable resultado también sea un entero, por lo que el resultado de, por ejemplo, 3 / 2 y 3 // 2 sería el mismo: 1. Si quisiéramos obtener los decimales necesitaríamos que al menos uno de los operadores fuera un número real, bien indicando los decimales.

3.1. Boleanos.

Como decíamos al comienzo del capítulo una variable de tipo booleano sólo puede tener dos valores: True (cierto) y False (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como veremos más adelante.

En realidad el tipo bool (el tipo de los booleanos) es una subclase del tipo int. Puede que esto no tenga mucho sentido para tí si no conoces los términos de la orientación a objetos, que veremos más adelante, aunque tampoco es nada importante. Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos, los llamados operadores lógicos o condicionales.

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	`r = True and False # r es False`
or	¿se cumple a o b?	`r = True or False  # r es True`
not	No a	`r = not True  # r es False`

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	`r = 5 == 3 # r es False`
!=	¿son distintos a y b?	`r = 5 != 3 # r es True`
<	¿es a menor que b?	`r = 5 < 3  # r es False`
>	¿es a mayor que b?	`r = 5 > 3  # r es True`

3.2. Operadores lógicos

Estos son operadores que actúan sobre las representaciones en binario de los operandores. Por ejemplo: Si veis una operación como 3 & 2, lo que estás viendo es un and bit a bit entre los números binarios 11 y 10 (las representaciones en binario de 3 y 2). El operador and (&), del inglés “y”, devuelve 1 si el primer bit operando es 1 y el segundo bit operando es 1. Se devuelve 0 en caso contrario. El resultado de aplicar and bit a bit a 11 y 10 sería entonces el número binario 10, o lo que es lo mismo, 2 en decimal (el primer dígito es 1 para ambas cifras, mientras que el segundo es 1 sólo para una de ellas). El operador or (|), del inglés “o”, devuelve 1 si el primer operando es 1 o el segundo operando es 1. Para el resto de casos se devuelve 0. El operador xor u or

exclusivo (^) devuelve 1 si uno de los operandos es 1 y el otro no lo es. El operador not (~), del inglés “no”, sirve para negar uno a uno cada bit; es decir, si el operando es 0, cambia a 1 y si es 1, cambia a 0.

Por último, los operadores de desplazamiento (<< y >>) sirven para desplazar los bits n posiciones hacia la izquierda o la derecha.

SÍMBOLO	DESCRIPCIÓN	EJEMPLO	BOOLEANO
==	IGUAL QUE	5 == 7	FALSE
!=	DISTINTO QUE	ROJO != VERDE	TRUE
<	MENOR QUE	8 < 12	TRUE
>	MAYOR QUE	12 > 7	TRUE
<=	MENOR O IGUAL QUE	16 <= 17	TRUE
>=	MAYOR O IGUAL QUE	67 >= 72	FALSE

4. Tipos de colecciones de datos.