**Prólogo**

En la actualidad Python es uno de los lenguajes de programación con mayor proyección. Su facilidad de uso, su librería estándar y la cantidad de librerías adicionales que existen contribuyen a que sean muchos los desarrolladores de software que optan por su utilización para llevar a cabo sus proyectos.

Python es un lenguaje de propósito general, de alto nivel, interpretado y que admite la aplicación de diferentes paradigmas de programación, como son, por ejemplo, la programación procedural, imperativa y la orientación a objetos.

Dadas sus principales características, Python es un lenguaje ideal para el prototipado. Para diversos tipos de aplicaciones pueden construirse rápidamente prototipos, facilitando el desarrollo del modelo final en otros lenguajes que ofrezcan mayor rendimiento como es el caso de C y C++. Todo ello sin perder de vista el hecho de que Python puede utilizarse como un lenguaje de programación de más alto nivel.

**Capítulo 1.**

* Lenguaje de alto nivel, interpretado, de propósito general, multiplataforma y multiparadigma.
* Con este lenguaje podemos desarrollar software para aplicaciones científicas, para comunicaciones de red, para aplicaciones de escritorio con interfaz gráfica de usuario (GUI), para crear juegos, para smartphones y por supuesto, para aplicaciones web.
* Empresas y organizaciones del calibre de *Industrial Light & Magic*, *Walt Disney*, *la NASA*, *Google*, *Yahoo!*, *Ret Hat* y *Nokia* hacen uso intensivo de este lenguaje para desarrollar sus productos y servicios. Esto demuestra que Python puede ser utilizado en diversos tipos de sectores, con independencia de su actividad empresarial.
* Se trata de un lenguaje potente, flexible y con una sintaxis clara y concisa. Además, no requiere dedicar tiempo a su compilación debido a que es interpretado.
* Python es *open source*, cualquiera puede contribuir a su desarrollo y divulgación. Además, no es necesario pagar ninguna licencia para distribuir software desarrollado con este lenguaje. Hasta su intérprete se distribuye de forma gratuita para diferentes plataformas.
* Gracias a su licencia de software libre (*Python Software Foundation License*) es posible modificar el código fuente y desarrollar código derivado sin la necesidad de hacerlo open source. <http://www.python.org/>
* La versión del intérprete de Python es nativa para cada plataforma. En este sentido Python es similar a Perl o a Ruby y difiere de otros lenguajes como C++ y Objetive-C.
* Habitualmente, a los programas en Python se les denomina *scripts*. En realidad *script* es el término que se suele emplear para los ficheros de código fuente escritos en Python, pudiendo un programa contar con uno o más de estos *scripts*.
* Los programadores de Python suelen llamar indistintamente con este nombre tanto al lenguaje como al intérprete del mismo. Debemos tener esto en cuenta, debido a que es habitual escuchar “voy a instalar Python” o “la versión que tengo instalada de Python es la 3.2”. En estos casos se hace referencia directa al intérprete y no al lenguaje.
* La interacción con el intérprete se puede hacer directamente a través de la *consola*.
* Python carece de tipos propiamente dichos, es decir, es un lenguaje de *tipado dinámico*.
* Usa indentación para marcar las sentencias que corresponden al mismo bloque; además cada sentencia no necesita un punto y coma (;), como sí ocurre en lenguajes como C/C++, PHP y Java; basta con que cada sentencia vaya en una línea diferente.
* Para facilitar la programación, Python incluye una serie de estructuras de datos de alto nivel, como son, por ejemplo, las listas, los diccionarios, las cadenas de texto (*strings*), tuplas y conjuntos.
* Su *librería estándar* incorpora multitud de funciones que pueden ser utilizadas en diversos ámbitos.
* En Python existe un recolector de basura (*garbage collector*), esto significa que no es necesario pedir y liberar memoria, de forma explícita, para crear y destruir objetos. El intérprete lo hará automáticamente cuando sea necesario y el recolector se encargará de gestionar la memoria para evitar los temidos **memory leaks**.
* Facilidad para interactuar con otros lenguajes de programación (C/C++).
* Genera ficheros binarios llamados bytecode (ficheros con extensión *.pyc*), que al final serán los ficheros que serán ejecutados por el intérprete; lo anterior por motivos de eficiencia. La generación del bytecode es automática y el programador no debe preocuparse por este proceso. El intérprete es lo suficientemente inteligente para volver a generar el bytecode cuando es necesario, habitualmente, cuando el fichero de código fuente correspondiente cambia.
* Es posible generar ficheros binarios listos para su ejecución, sin necesidad de contar con el intérprete. Esto suele darse en sistemas Windows ya que por defecto no tienen instalado el intérprete de Python. Para salvar este obstáculo contamos con programas como py2exe ([*http://www.py2exe.org*](http://www.py2exe.org)), que se encarga de ejecutar un binario para Windows (*.exe*) a partir de un fichero fuente escrito en Python.
* Ver depurador pdb.

**Capítulo 2**

**Conceptos Básicos**

* **Objeto:** Los objetos son la representación tangible de una clase en el mundo de la programación. Una clase es la especificación de un objeto, una especie de guía para construirlo. Encontrando analogía con la vida real, la clase puede verse como el plano para la construcción de un edificio y este sería entonces el objeto. Es por ellos que se dice que un objeto es una instancia o ejemplar de una clase y, lógicamente, de una misma clase se pueden crear varios objetos al igual que de un mismo plano pueden crearse diferentes edificios, todos con las mismas características. La clase es la base teorica y el objeto es la realización de esa base teorica. Si la clase contiene indicaciones para determinados comportamientos, atributos o datos, entonces el objeto deberá tener todos estos elementos.
* **Todo es un objeto en Python**. Las funciones, los tipos predefinidos, los módulos y hasta los ficheros son representados en Python por medio de objetos. Todo en Python es un objeto y por ende todo contiene métodos y propiedades que son afines con su propósito particular. El hecho de que la abstracción de datos en Python se lleve a un ciento por ciento por medio de objetos propicia que todo el intercambio de datos en el lenguaje se produzca entre objetos. Considerando que las funciones también son objetos, todo el código de un programa en Python puede verse como un conjunto de objetos.
* **Objeto**: Componente que se aloja en memoria y que tiene asociados una serie de valores y operaciones que pueden ser realizadas con él. Los datos que manejamos en el lenguaje cobran vida gracias a estos objetos. Un objeto en Python puede ser una cadena de texto, un número real, un diccionario o un objeto propiamente dicho, según el paradigma OOP, creado a partir de una clase determinada.
* Habitualmente, un programa en Python puede contener varios componentes. El lenguaje nos ofrece cinco tipos de estos componentes claramente diferenciados. El primero de ellos es el *objeto*. Por otro lado tenemos las *expresiones*, entendidas como una combinación de valores, constantes, variables, operadores y funciones que son aplicadas siguiendo una serie de reglas. Estas expresiones se suelen agrupar formando *sentencias*, consideradas estas como las unidades mínimas ejecutables de un programa. Por último, tenemos los *módulos* que nos ayudan a formar grupos de diferentes sentencias.
* Python cuenta con una serie de objetos integrados (*build-in*). Contamos con números, cadenas de texto, booleanos, listas, diccionarios, tuplas, conjuntos y ficheros. Además, contamos con un tipo de objeto especial llamado **None** que se emplea para asignar un valor nulo.

**Tipado dinámico**

* En Python al declarar una variable, no se puede indicar su tipo. En tiempo de ejecución, el tipo será asignado a la variable, empleando una técnica conocida como *tipado dinámico*.
* Función *type()* 🡪 Para ver de qué tipo es una variable en un momento determinado en el programa en ejecución. Como argumento recibe el nombre de la variable en cuestión y devuelve el tipo precedido de la palabra clave *class*.
* El tipo de un objeto determina las operaciones y atributos que posee y si puede ser alterado.
* Un objeto que puede ser modificado se dice que es mutable y, en caso contrario, se dice que es inmutable.

**Números enteros, reales y complejos.**

* En lo que respecta a los números complejos, aquellos formados por una parte *real* y otra *imaginaria*, la asignación sería la siguiente:

*>>> num\_complejo = 3.2 + 7j*

Siendo también válida la siguiente expresión:

*>>> num\_complex = 5J + 3*

La parte imaginaria aparece representada por la letra *j*, siendo también posible emplear la misma letra en mayúscula.

* En Python 3 los tipos *int* y *long* han sido integrados en un único tipo *int*.
* En Python 3 los reales se representan por un único tipo *doublé*, lo cual permite un amplio rango, el lenguaje emplea para su representación un bit para el signo (*positivo o negativo*), 11 para el exponente y 52 para la mantisa.
* También es posible expresar un número real utilizando notación científica. Simplemente necesitamos añadir la letra *e*, que representa el exponente, seguida del valor para él mismo. La siguiente expresión sería válida para representar al número :

*>>> num\_real = 0.5e-7*

* En realidad, aunque Python cuenta con el tipo integrado *bool*, este no es más que una versión personalizada del tipo *int*.
* Si necesitamos trabajar con números reales que tengan una precisión determinada, por ejemplo, dos cifras decimales, podemos utilizar la clase *Decimal*. Esta viene integrada en la librería básica que ofrece el intérprete e incluye una serie de funciones, para, por ejemplo, crear un número real con precisión a través de una variable de tipo *float*.

**Sistemas de representación**

* Python puede representar los números enteros en los sistemas decimal, octal, binario y hexadecimal. La representación decimal es la empleada comúnmente y no es necesario indicar nada más que el número en cuestión.
* Para representar un número en binario nos basta con anteponer los caracteres 0b. El número 7 se representaría en binario de la siguiente forma:

*>>> num\_binario = 0b111*

* Para el octal necesitamos anteponer los caracteres *0o*. El número 8 quedaría representado utilizando la siguiente sentencia:

*>>> num\_octal = 0o10*

* En lo que respecta al sistema hexadecimal, el carácter necesario, tras el número 0, es la letra x. Un ejemplo sería la representación del número 255 a través de la siguiente sentencia:

*>>> num\_hex = 0xff*

**Operadores**

|  |  |
| --- | --- |
| **Expresión con Operador** | **Operación** |
| *a* ***+*** *b* | Suma |
| *a* ***–*** *b* | Resta |
| *a* ***\**** *b* | Multiplicación |
| *a* ***%*** *b* | Resto |
| *a* ***/*** *b* | División Real |
| *a* ***//*** *b* | División Entera |
| *a* ***\*\**** *b* | Potencia |
| *a* ***|*** *b* | OR (bit) |
| *a* ***ˆ*** *b* | XOR (bit) |
| *a* ***&*** *b* | AND (bit) |
| *a* ***==*** *b* | Igualdad |
| *a* ***!=*** *b* | Desigualdad |
| *a* ***or*** *b* | OR (lógica) |
| *a* ***and*** *b* | AND (lógica) |
| ***not*** *a* | Negación (lógica) |

**Conjuntos**

* Definir y operar con conjuntos matemáticos (unión, intersección, creación de subconjuntos, diferencia) también es posible en Python. La función para crear un conjunto se llama *set()* y acepta como argumentos una serie de valores pasados entre comas, como si se tratara de una cadena de texto. Por ejemplo, la siguiente línea de código define un conjunto de tres números diferentes:

*>>> conjunto = set(‘846’)*

* Un conjunto también puede ser definido empleando llaves ({}) y separando los elementos por comas.

*>>> conjunto = {8, 4, 6}*

* Si creamos un conjunto con valores repetidos, estos serán automáticamente eliminados, es decir, no formarán parte del conjunto (teoría de conjuntos).
* Los conjuntos se construyen mediante *set(items)* donde *items* es cualquier objeto iterable, como listas o tuplas. Los conjuntos no mantienen el orden ni contienen elementos duplicados.
* Se suelen utilizar para eliminar duplicados de una secuencia, o para operaciones matemáticas como intersección, unión, diferencia y diferencia simétrica.

**Cadenas de texto**

* Básicamente, una cadena de texto o *string* es un conjunto inmutable y ordenado de caracteres.
* Para su representación y definición se pueden utilizar lanto comillas dobles (”), como simples (‘).
* Si necesitamos declarar un string que contenga más de una línea, podemos hacerlo utilizando comillas tríples en lugar de dobles o simples:

*>>> cad\_multiple = “ “ “ Esta cadena de texto*

*… tiene mas de una línea. En concreto, cuenta*

*… con tres líneas diferentes ” ” ”*

* Por defecto, en Python 3, todas las cadenas de texto son Unicode; a diferencia de la serie 2.x en la cual las cadenas de texto por defecto son ASCII.
* Otras de las novedades de Python 3 con referencia a las cadenas de texto es el tipo de estas que soporta. En concreto son tres las incluidas en esta versión: *Unicode*, *byte* y *bytearray*. El tipo *byte* solo admite caracteres en codificación ASCII y, al igual que los de tipo Unicode, son inmutables. Por otro lado, el tipo *bytearray* es una versión mutable del tipo *byte*.
* Para declarar un string de tipo byte, basta con anteponer la letra b antes de las comillas.

*>>> cad = b”cadena de tipo byte”*

*>>> type(cad)*

*<class ‘bytes’>*

* La declaración del tipo *bytearray* debe hacerse utilizando la función integrada que nos ofrece el interprete. Además, es imprescindible indicar el tipo de codificación que deseamos emplear. El siguiente ejemplo utiliza la codificación de caracteres *latin1* para crear un string de este tipo:

*>>> lat = bytearray(“texto”, ‘latin1’)*

* Consultar funciones y métodos para funciones. Métodos: *encode()*, *decode()*, *find()*, *replace()*, *strip()*, *lstrip()*, *rstrip()*, *upper()*, *lower()*, *capitalize()*, *split()*, *join()*. Funciones: *len()*.
* Tres formas de concatenar cadenas de texto:

*>>>* cad\_concat = *“Hola ” + var2 + “. Otra ” + var3*

*>>>* cad\_concat = *“Hola {0}. Otra {1}”.format(var2, var3)*

*>>>* cad\_concat = “Hola {var2}. Otra {var3}”.format(var2=var2, var3=var3)

* La concatenación entre strings y números también es posible, siendo para ello necesario el uso de funciones como *int()*, y *str()*. El siguiente ejemplo es un caso sencillo de cómo utilizar la función  *str()*:

*>>> num = 3*

*>>> cad\_concat = “Numero: ” + str(num)*

* Interesante resulta el uso del operador \* aplicado a cadenas de texto, ya que nos permite repetir un string n veces. Supongamos que deseamos repetir la cadena “Hola Mundo” cuatro veces. Para ello, bastará con lanzar la siguiente sentencia:

*>>> print(“Hola Mundo” \* 4)*

* Consultar el uso del operador *in* en cadenas.

**Tuplas**

* En Python una *tupla* es una estructura de datos que representa una colección de objetos, pudiendo estos ser de distintos tipos. Internamente, para representar una tupla, Python utiliza un array de objetos que almacena referencias hacia otros objetos.
* Para declarar una tupla se utilizan paréntesis, entre los cuales deben separarse por comas los elementos que van a formar parte de ella. En el siguiente ejemplo, crearemos una tupla con tres valores, cada uno con un tipo diferente:

*>>> t = (1, ‘a’, 3.5)*

* Debemos tener en cuenta que las tuplas son un tipo de dato inmutable, esto significa que no es posible asignar directamente un valor a través del índice.
* Dado que una tupla puede almacenar distintos tipos de objetos, es posible *anidar* diferentes tuplas:

*>>> t = (1, (‘a’, 3), 5.6)*

* Consultar en el libro concatenación de tuplas.
* Métodos para tuplas: *index()*, *count()*. Funcion: *len()*.

**Listas**