GOBIERNO DE ESPAÑA
MINISTERIO DE HACIENDA Y FUNCIÓN PÚBLICA
INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 1. Introducción a Python. Sintaxis básica

Autor: Jesús Moreno Jabato

Contenido

1. Introducción a Python

1.1. Lenguaje de programación Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Interpretado quiere decir (a diferencia de otros lenguajes compilados como C ++, C # o VB) que el código que escribimos se convierte a código máquina (lenguaje que entiende el ordenador o dispositivo de forma nativa) de forma directa.

Javascript es otro lenguaje interpretado. Java es un lenguaje compilado a través de un lenguaje intermedio (bytecode) que después se interpreta a través de una máquina virtual de Java.

Un lenguaje compilado traduce el código que escribimos mediante un paso adicional que se denomina compilación, cuyo objetivo final es la creación de un ejecutable. En un lenguaje interpretado, no existe el ejecutable como tal. La principal ventaja de los lenguajes interpretados es que el ciclo de desarrollo, es decir, el tiempo que pasa desde que escribes el código hasta que lo pruebas y ejecutas es menor que en un lenguaje compilado.

Python posee eficientes estructuras de datos de alto nivel y un enfoque sencillo pero efectivo de la programación orientada a objetos. La sintaxis, los tipos de datos dinámicos y el hecho de ser un lenguaje interpretado hacen de Python un lenguaje ideal para la creación de scripts y el desarrollo rápido de aplicaciones en todo tipo de áreas y plataformas.

Lenguaje de alto nivel significa que no trabajamos directamente con las instrucciones que entiende la máquina (registros de almacenamiento, direcciones de memoria y pilas de llamada). Trabajamos con variables, procedimientos, objetos, etc… que son más fáciles de entender para los humanos. El lenguaje ensamblador es de bajo nivel, próximo a lo que las máquinas entienden de forma nativa.

El intérprete de Python y la biblioteca estándar están disponibles en forma de código fuente o código ejecutable para los principales sistemas operativos en el sitio web oficial de Python, https://www.python.org/ y puede distribuirse libremente. Ese sitio web también contiene información sobre muchos módulos, programas y herramientas para Python, así como documentación adicional.

El intérprete de Python puede ampliarse fácilmente con nuevas funciones y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python es también adecuado como lenguaje de extensión de aplicaciones, por ejemplo en FME se usa.

1.2. Historia

El origen del lenguaje Python se remonta a principios de los noventa cuando un investigador holandés llamado Guido van Rossum, que trabajaba en el centro de investigación CWI (Centrum Wiskunde & Informatica) de Ámsterdam es asignado a un proyecto que consistía en el desarrollo de un sistema operativo distribuido llamado Amoeba.

El CWI utilizaba un lenguaje de programación llamado ABC. En lugar de emplear este lenguaje para el proyecto Amoeba, Guido decide crear uno nuevo que pueda superar las limitaciones y

problemas con los que se había encontrado al trabajar en otros proyectos con ABC. Así pues, es esta la principal motivación que dio lugar al nacimiento de Python.

La primera versión del lenguaje ve la luz en 1991, pero no es hasta tres años después cuando decide publicarse la versión 1.0. Inicialmente el CWI decidió liberar el intérprete del lenguaje bajo una licencia open source propia, pero en septiembre de 2000 y coincidiendo con la publicación de la versión 1.6, se toma la decisión de cambiar la licencia por una que sea compatible con la licencia GPL (GNU General Public Licence). Esta nueva licencia se denominará Python Software Foundation Licence y se diferencia de la GPL al ser una licencia no copyleft. Este hecho implica que es posible modificar el código fuente y desarrollar código derivado sin la necesidad de hacerlo open source.

Hasta el momento solo se han liberado tres versiones principales, teniendo cada una de ellas diversas actualizaciones. En lo que respecta a la versión 2, la última en ser liberada fue la 2.7, en julio de 2010 y se a modo de documentación se debe tener en cuenta porque hay muchos productos programados para la versión 2. En el momento de edición de este curso, la versión cuenta con la actualización 3.9.6, liberada en junio de 2021.

Entre las características de las primeras versiones de Python cabe destacar:

- El soporte de la orientación a objetos.
- El manejo de excepciones.
- Soporte de estructuras de datos de alto nivel como las listas y los diccionarios.
- Desde su desarrollo inicial se tuvo en cuenta que el código escrito en este lenguaje fuera fácil de leer y de aprender, sin que esto suponga renunciar a características y funcionalidades avanzadas.

Muchos se preguntan el origen del nombre de Python. Gudo van Rossum decidió darle este nombre en honor a la serie de televisión Monty Python's Flying Circus, de la cual era fan. Esta es una serie cómica protagonizada por el grupo de humoristas Monty Python, famoso por películas como La vida de Brian o El sentido de la vida.

Figura 1. Fotograma de La vida de Bryan

Desde el principio de su diseño, se pretendía que Python fuera un lenguaje que resultara divertido de utilizar, de ahí que en el nombre influyera la mencionada serie. También resulta curioso que, tanto en tutoriales, como en ejemplos de código, se suelen utilizar referencias a los Monty

Python. Por ejemplo, en lugar de emplear los tradicionales nombres de variables foo y bar, se suele utilizar spam y egss, en referencia a sketchs de este grupo de cómicos.

El desarrollo y promoción de Python se lleva a cabo a través de una organización, sin ánimo de lucro, llamada Python Software Foundation, que fue creada en marzo de 2001. Entre las actividades que realiza esta organización destacan el desarrollo y distribución oficial de Python, la gestión de la propiedad intelectual del código y documentos realizados, así como la organización de conferencias y eventos dedicados a poner en contacto a todas aquellas personas interesadas en este lenguaje de programación.

Python es open source y la Python Software Foundation invita a cualquiera que quiera hacerlo a contribuir al desarrollo y promoción de este lenguaje de programación.

1.3. Características

- Lenguaje interpretado.
- Tipado dinámico.
- Organizado y extensible. (Clases, módulos y paquetes).
- Fuertemente tipado. (conversión explicita de tipos).
- Multiplataforma.
- Orientado a objetos.
- Propósito general.
- Lenguaje interactivo. Línea de comandos.

1.4. Ventajas

- Sintaxis simple, clara y sencilla.
- Sintaxis cercana al lenguaje natural.
- Tipado dinámico.
- Gestor de memoria.
- Variedad de librerías disponibles.
- Potencia del lenguaje.
- Gran implantación en entornos de investigación o Inteligencia Artificial.

1.5. Desventajas

- No adecuado para aplicaciones críticas.
- Cuidado con las indentaciones – tabulaciones.
- Poca implantación en servicios web

1.6. Aplicaciones Python

Los script de Python tienen la gran ventaja de que son ejecutables en cualquier máquina que tenga instalado el intérprete, por lo que tanto en ejecución en local como en remoto pueden usarse.

La versatilidad del uso de módulos hace que se pueda realizar prácticamente lo que se quiera imaginar con los resultados esperados.

Python en AI (Inteligencia Artificial) se aplica con librerías como TensorFlow, Keras o Scikitl para la aplicación de algoritmos de regresión, clasificación o clustering.

Python en Big Data se combina en el entorno de Hadoop para Análisis de datos. Pydoop sobre HDFS o las librerías Dask o PySpark son ejemplos de implementaciones.

Python en investigación destaca con los entornos de Pandas, NumPy, Matplotlib o Seaborn estando Python muy influenciado por MATLAB.

Python en desarrollo web en el Backend en el que la referencia es el entorno Django. También es empleado para web scraping (recolección de otras webs). Empresas como Instagram o aplicaciones de descarga de Torrent son algunas de las aplicaciones.

2. Manos a la obra

2.1. IDES vs Editores de texto

El código, los símbolos e instrucciones que dictan cómo se va a comportar una máquina al ejecutar un programa puede ser un dolor de cabeza para cualquiera que no tenga conocimientos y se enfrente a un fragmento cualquiera, incluso puede serlo hasta para los programadores experimentados que no acostumbran a comentar el código.

Preparar el código de forma correcta para ejecutarlo en un entorno seguro requiere de un ritual propio. Los sistemas operativos ya cuentan con editores de texto preinstalados, pero sus capacidades son muy limitadas. Es importante tener claro que elegir un editor adecuado es crucial a la hora de programar, así como que no todos son aconsejables para todos los niveles.

El mundo de los editores de texto es complejo, algunos cuentan con interfaces gráficas elaboradas que resultan más intuitivas y que permiten trabajar en grandes proyectos (aquí entrarían los grandes entornos de desarrollo integrado o IDEs). Por otra parte hay otros que son minimalistas y básicos en su concepción, que recompensarán a aquellos que tengan la paciencia de vérselas con una curva de aprendizaje elevada.

Algunos de estos editores están dirigidos a plataformas y lenguajes específicos (Swift con iOS por ejemplo), mientras que otros tienen un propósito más general cuyas funcionalidades se pueden ampliar mediante plugins que aumentan su capacidad de trabajo (Eclipse o NetBeans por ejemplo).

Como suponemos que estamos empezando a programar y que estamos en el punto de tener que decidir por un editor hay que conocer las diferencias entre IDE y editor de texto:

2.1.1. Editores de texto puros y duros

El Bloc de Notas de Windows, el Gedit de Linux o el TextEditor de Mac tienen sus limitaciones. Vale la pena comentar que son sólo para los más experimentados, para los que llevan toda una vida trabajando con ellos o para los más geeks. Vim o Emacs son otros ejemplos que documentamos aquí para tener un conocimiento más amplio pero que seguramente no lleguéis a recurrir a ellos.

Vim es complicado de usar. A simple vista parece un editor de texto en su mínima expresión, de esos que nos retrotraen a los tiempos en los que el ratón era una cosa que se usaba para hacerle compañía al teclado y que los menús eran algo que le pasaba a otra gente. Como los menús no están en el menú de Vim, se funciona con él a base de comandos de texto.

En cuanto a Emacs, se trata de un proyecto desarrollado por Richard Stallman durante los años 70 del siglo pasado en el MIT que se ha seguido continuando hasta el día de hoy. Es el competidor natural de Vim y también desprecia la interfaz en favor de comandos, que se pueden consultar en la Emacs Reference Card.

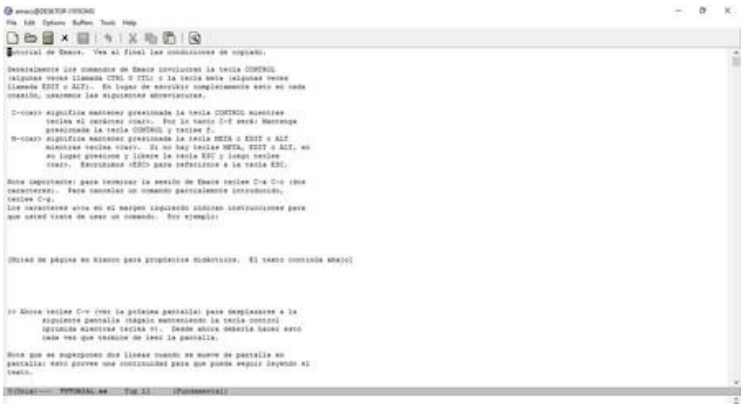

Figura 2. Ejemplo de Editor de Texto – Bloc de Notas

Visto desde fuera, tanto Vim como Emacs resultan intimidatorios para los usuarios que no están acostumbrados a ellos. Sin embargo, hay muchísimo potencial en ellos que permite escribir en una gran cantidad de lenguajes de programación y a gran velocidad.

Cualquier programa con una interfaz gráfica requiere que se interactúe con él en una combinación de pulsaciones en el teclado y movimientos del ratón. Hay elementos que piden que saquemos las manos del teclado y tengamos que coger el ratón, lo que resulta poco eficiente. Con Emacs y Vim se puede escribir sin soltar el teclado y conseguir un flujo de trabajo muy eficiente.

Se trata de editores muy personalizables que nos permiten elegir entre distintos resaltados de fuentes, colores del texto y demás, si bien es necesario tocar los archivos de configuración a mano, al menos en sus versiones para Linux.

2.1.2. Editores de texto empoderados

En esta categoría entrarían programas como Atom o Sublime Text. Atom es gratuito y teóricamente Sublime Text cuesta dinero, si bien puedes mantener la versión de evaluación ad infinitum.

Cuando abres Atom o Sublime Text por primera vez te da la impresión de estar ante una especie de bloc de notas sofisticado. Es cuando empiezas a trabajar con ellos te das cuenta de que sus funcionalidades van mucho más allá.

Para empezar, los dos vienen equipados de serie con resaltado de sintaxis para prácticamente cualquier lenguaje de programación con el que se pueda trabajar. Esto puede parecer algo trivial cuando estás empezando, pero una vez que conoces bien un lenguaje y tienes claro cómo debería ser el código escrito para él, es fundamental para que el código se pueda leer e interpretar en un golpe de vista.

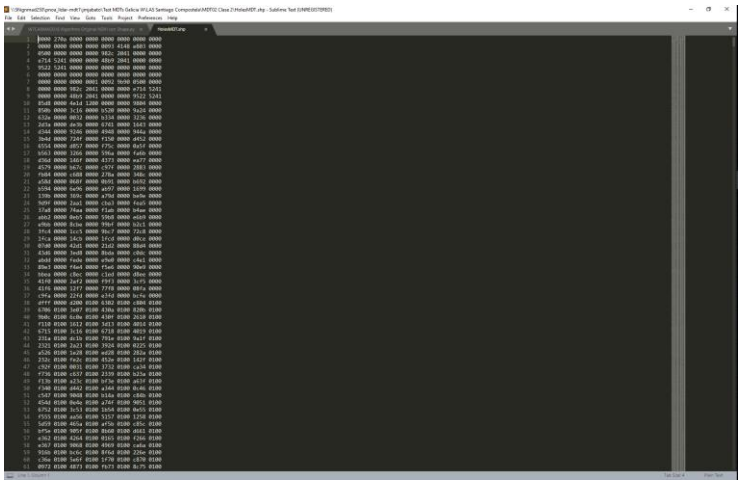

Figura 3. Ejemplo de Editor de Texto – Sublime Text

Tanto Atom como Sublime Text cuentan con un ecosistema de paquetes para extender sus capacidades. Ambos cuentan con un amplio panorama, ofreciendo soluciones que van desde mejorar el procesador de textos a plantillas HTML, integración con GitHub, herramientas de control de calidad del código y mucho más.

Son recomendables si ya se tiene algo de experiencia trabajando con un lenguaje concreto, ya que para terminar de afinar sus habilidades será necesario instalar algún paquete (por ejemplo uno para la ejecución de código y un depurador). Por tanto, hay que tener más o menos claro lo que se está buscando.

2.1.3. Entornos de desarrollo integrado o IDEs

Los IDEs suelen ser la primera experiencia con editores dedicados de la mayoría de programadores noveles. Se recomiendan siempre en todos los cursos de desarrollo de aplicaciones (en nuestro caso recomendamos PyCharm) o de páginas web, y casi siempre por el mismo motivo: son muy poderosos y concentran todo lo necesario para ejecutar un programa que el usuario escriba en su interfaz. También cuentan con resaltado de sintaxis por defecto.

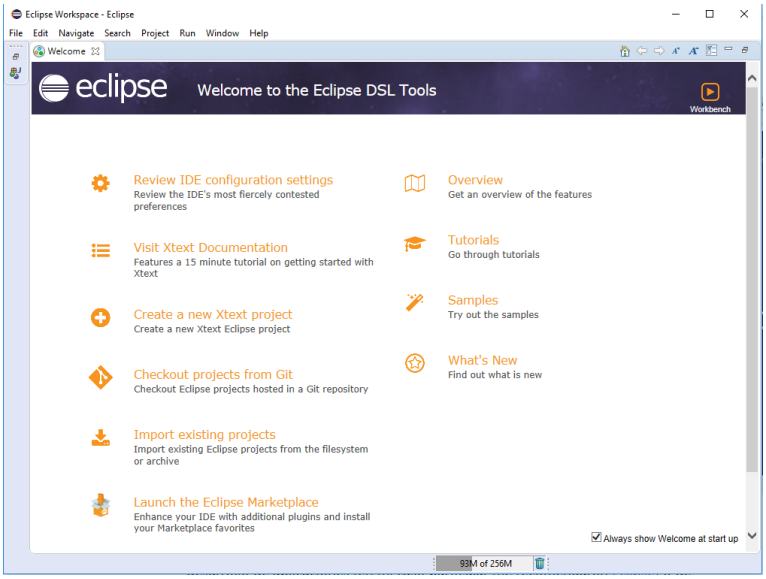

Figura 4. Ejemplo de IDE – Eclipse

Un IDE es una herramienta muy poderosa. La diferencia fundamental con los editores de texto como Atom es que su potencia puede abrumar al usuario. Muchos ofrecen extensiones para nuevos lenguajes de programación y frameworks que crean nuevos IDEs dentro de un IDE.

Por otra parte, para quienes se inician en el mundo de la programación suelen ser la mejor manera de empezar a escribir y ejecutar código. Ya no sólo por el resaltado de sintaxis, sino porque cuentan también con correctores de sentencias que detectan palabras mal escritas y con depuradores que señalan los errores en el código (que se presentan aunque la sintaxis de la instrucción sea correcta).

La depuración resulta esencial al tratar con programas largos y complejos. Gracias a esta característica se puede ver el código en acción mientras se ejecuta, lo que permite tomar medidas mucho más precisas sobre los errores en lugar de estar mirando a un montón de líneas en una pantalla esperando a que la solución se presente por arte de magia.

En este curso instalaremos y trabajaremos con el IDE PyCharm.

2.2. Intérprete Python

Antes de empezar con un IDE o a programar, es necesario que esté instalado el intérprete de Python en la máquina que estamos utilizando.

Como primero paso necesitaremos descargar la versión que nos interese de Python en el siguiente enlace: https://www.python.org/downloads/ y seguir los pasos para instalarlo.

En caso de que no queramos instalar el intérprete, para casos simples existen intérpretes online como estos:

https://www.programiz.com/python-programming/online-compiler/

https://paiza.io/es/languages/python3

Recomendamos instalar el intérprete ya que podremos trabajar mejor con las librerías y controlar las ejecuciones.

2.3. Pycharm

PyCharm es un IDE que se adapta a nuestras necesidades iniciales para programar en Python. Tiene una versión de pago, pero tenemos la versión gratuita también conocida como “Pycharm Community Edition”.

PyCharm tiene como ventajas que todo viene incluido en un mismo paquete y una vez realizado el proceso de configuración inicial estamos listos para empezar a programar.

Para instalar el framework debéis descargar la versión gratuita de PyCharm e instalar el programa. En primer lugar, tendremos que dirigirnos a la página oficial del proyecto en la URL https://www.jetbrains.com/pycharm/ donde nos encontraremos con el botón de descarga de PyCharm. Sólo tenemos que elegir nuestro sistema operativo y a descargar.

La instalación no debe tener complicaciones siguiendo lo que nos pide el instalador.

No es un IDE liviano y rápidamente lo comparamos con los editores de texto que necesitan menos recursos. Pero no nos olvidemos que PyCharm es un entorno de desarrollo completo, no un simple editor de texto.

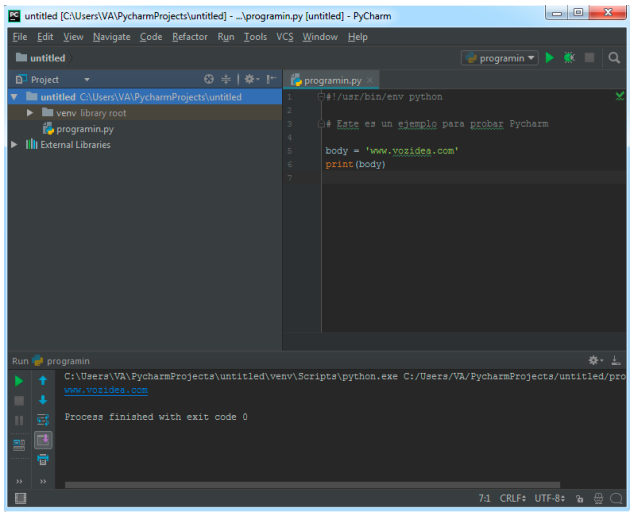

Figura 5. Ejemplo de vista PyCharm

2.3.1. Iniciando por primera vez PyCharm.

Durante el primer arranque de PyCharm, el programa nos guiará a través de un asistente en la configuración básica del IDE. Todas estas opciones también podremos configurarlas más adelante, si así lo deseamos.

En primer lugar, nos deja elegir entre una interfaz de usuario con un tema claro (IntelliJ) y un tema oscuro (Dracula).

Solo como recomendación, a la hora de programar y por temas de cuidar la vista, siempre es mejor contar con fondos oscuros. Por ello elegiremos el tema Dracula.

Este asistente también nos da la opción de instalar algunos plugins que se usan habitualmente: IdeaVim (emula al editor Vim), Markdown (soporte para sintaxis markdown), Bash support (soporte para código bash), R language support (soporte para código R). En nuestro caso no vamos a instalar ninguno.

El IDE PyCharm funciona con proyectos, es decir, tendremos que definir una carpeta o directorio donde se van a ir guardando los ficheros que generemos cuando progragremos y también tendremos que definir el intérprete, en nuestro caso es Python.

El intérprete puede escogerse entre el que viene instalado por defecto por PyCharm, por ejemplo “Virtualenv” o “Conda” o podremos escoger uno Existente, que si hemos instalado el Intérprete de Python según el paso 3.4, podremos buscar dónde está instalado y usarlo, habrá que seleccionar el fichero “python.exe”

2.4. "Hola Mundo"

En informática, un programa Hola mundo es el que imprime el texto «¡Hola, mundo!» en un dispositivo de visualización y no hace nada más.

Este programa suele ser usado como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico, y se lo considera fundamental desde el punto de vista didáctico.

El Hola Mundo se caracteriza por su sencillez, especialmente cuando se ejecuta en una interfaz de línea de comandos. En interfaces gráficas la creación de este programa requiere de más pasos.

El programa Hola Mundo también puede ser útil como prueba de configuración para asegurar que el compilador, el entorno de desarrollo y el entorno de ejecución estén instalados correctamente y funcionando.

Para obtener el Hola Mundo podemos hacerlo de 2 maneras:

1. Por comandos en la consola de Python.
   Hay que acceder a la carpeta donde está instalado Python y ejecutar el .exe de Python, que es la consola de Python.
   Introduce en la ventana que se abre el comando siguiente:

```
print ('Hola Mundo')
```

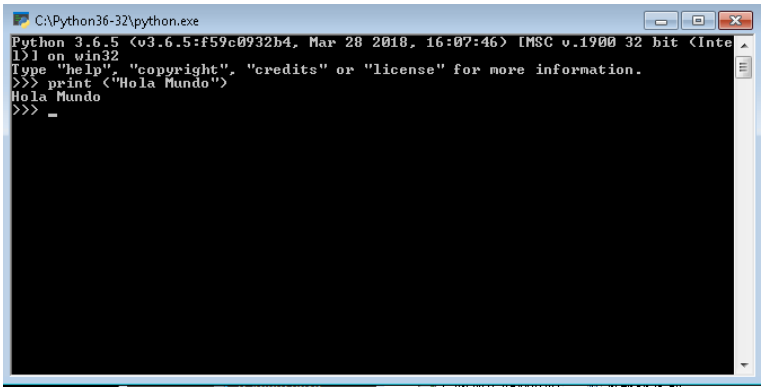

Figura 6. Ejemplo de vista de la Consola de Python

2. Crear fichero .py para ejecutarlo
   Crea una carpeta llamada “proyectos” en la unidad C:\
   Dentro de la carpeta crea un archivo de texto plano llamado “holamundo.py”. Lo puedes hacer con el mismo bloc de notas o en el proyecto de PyCharm con “Nuevo”
   Escriba lo siguiente dentro del fichero:

```
print ('Hola Mundo')
```

Abre PyCharm y ejecuta el fichero “holamundo.py”. Podrás con la opción “Run” o “Mayus + F10”

3. Elementos 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.

En la documentación de Python se puede consultar una descripción mucho más detallada y completa. https://docs.python.org/3/reference/lexical_analysis.html

El lenguaje Python está formado por elementos (tokens) de diferentes tipos:

- Palabras reservadas (keywords).
- Funciones integradas (built-in functions).
- Literales.
- Operadores.
- Delimitadores.
- Identificadores.

3.1. Líneas y espacios

Un programa de Python está formado por líneas de texto. Por ejemplo:

```
radio = 5
area = 3.14159242 * radio ** 2
print(area)
```

Se recomienda que cada línea contenga una única instrucción, aunque puede haber varias instrucciones en una línea, separadas por un punto y coma (;).

```
radio = 5; area = 3.14159242 * radio ** 2
print(area)
```

Por motivos de legibilidad, se recomienda que las líneas no superen los caracteres que ocupan una línea de pantalla, normalmente 80 caracteres. Si una instrucción supera esa longitud, se puede dividir en varias líneas usando el carácter barra (\):

```
radio = 5
area = 3.14159265358979323846 \
       * radio ** 2
print(area)
```

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

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

Los espacios no son significativos, da lo mismo uno o varios, excepto al principio de una línea. Los espacios al principio de una línea (sangrado) indican un nivel de agrupamiento. Ésta es una de las características de Python que lo distinguen de otros lenguajes, donde se utilizan las llaves { }).

3.2. Palabras reservadas (keywords)

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

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

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

3.3. Operadores

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

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

Los que más usaremos serán:

Símbolo	**Significado**	**Ejemplo**	**Resultado**
+	Suma	a = 10 + 5	a es 15
-	Resta	a = 12 – 7	a es 5
-	Negación	a = - 5	a es -5
*	Multiplicación	a = 7 * 5	a es 35
**	Exponente	a = 2 ** 3	a es 8
/	División	a = 12.5 / 2	a es 6.25
//	División entera	a = 12.5 // 2	a es 6.0
%	Módulo	a = 27 % 4	a es 3

Más información en https://www.programiz.com/python-programming/operators

3.4. Delimitadores

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

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

Más información en https://www.tutorialspoint.com/python/python_basic_operators.htm

3.5. Identificadores

Los identificadores son las palabras que se utilizan para nombrar elementos creados por el usuario u otros usuarios. Esos elementos pueden ser variables u objetos que almacenan información, funciones que agrupan instrucciones, clases que combinan ambos, módulos que agrupan los elementos anteriores, etc.

Los identificadores están formados por letras (mayúsculas y minúsculas), números y el carácter guion bajo (_).

Pueden ser caracteres Unicode, aunque normalmente se recomienda utilizar caracteres ASCII para evitar complicaciones a usuarios de oros países que utilizan juegos de caracteres diferentes.

El primer carácter del identificador debe ser una letra.

4. Bibliografía

Se ha tomado de referencias los siguientes enlaces y libros para los 3 primeros temas:

Libro Python 3 al descubierto, Arturo FERNANDEZ

http://www.mclibre.org/consultar/python/index.html

https://www.pythonmania.net/es/2016/02/29/las-principales-diferencias-entre-python-2-y-3-con-ejemplos/

https://www.codejobs.biz/es/blog/2013/03/03/historia-de-python

https://www.genbeta.com/a-fondo/ides-y-editores-que-diferencias-hay-entre-ellos-a-la-hora-de-escribir-codigo

http://www.vozidea.com/pycharm-el-mejor-ide-para-python