Python fácil

Arnaldo Pérez Castaño

Alfaomega

marcombo
ediciones técnicas

Python fácil

Python fácil

Arnaldo Pérez Castaño

Alfaomega

marcombo
ediciones técnicas

Diseño de la cubierta: *NDENU DISSENY GRÀFIC*

Datos catalográficos

Pérez, Arnaldo
Python fácil
Primera Edición

Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-622-661-2

Formato: 17 x 23 cm Páginas: 284

Python fácil
Arnaldo Pérez Castaño
ISBN: 978-84-267-2212-6, edición en español publicada por MARCOMBO, S.A., Barcelona, España

Primera edición: Alfaomega Grupo Editor, México, abril 2016

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720, Ciudad de México.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-622-661-2

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México – Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile
Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

A mi amor, mi esperanza, mi dulce reina del frío invierno, mi Irinita

CONTENIDO

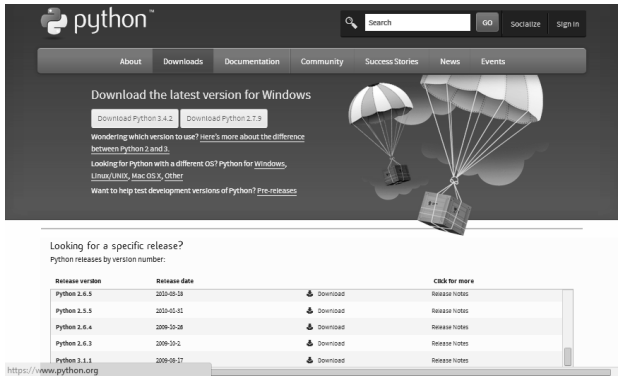
CAPÍTULO 1.
Introducción

Un interrogante común para muchas personas que se adentran en el mundo de la programación es la siguiente: ¿qué es Python? Dando respuesta a esta cuestión, Python es un lenguaje de programación que ha adquirido considerable popularidad entre programadores, aficionados y estudiantes por su alto nivel de expresividad, sus códigos compactos y elegantes, su sencillez y su capacidad para crear tanto aplicaciones de escritorio como aplicaciones web. Grandes empresas como Google o la NASA utilizan Python extensivamente en sus proyectos.

El lenguaje fue creado a comienzos de los noventa como sucesor del lenguaje ABC. Su creador Guido Van Rossum es un científico de la computación nacido en los Países Bajos y el nombre del lenguaje proviene de la serie de televisión del Reino Unido *Monty Python,* de la cual Guido es fanático. Actualmente es uno de los lenguajes que cuenta con mayor soporte en el mundo entero con versiones públicas que se lanzan cada seis meses aproximadamente. En este libro trataremos con la versión 3.1 y todos los ejemplos que se expongan se supondrán implementados en dicha versión.

1.1 Instalando Python

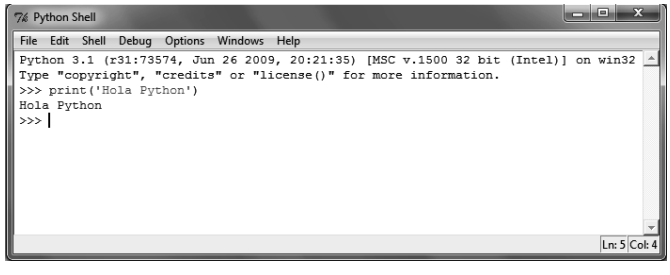
Para instalar Python primeramente debe descargar el paquete de instalación de Windows desde el sitio oficial de la Python Software Foundation https://www.python.org/downloads/.



El paquete seleccionado para ser usado en este libro es *python-3.1.msi* para Windows y la carpeta por defecto para la instalación se define en la raíz del disco del sistema.

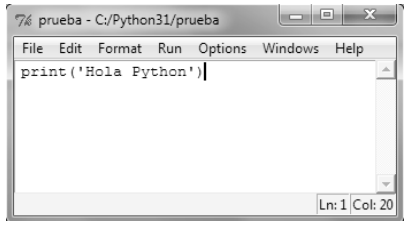
Una vez instalado el paquete usted puede acceder al Ambiente de Desarrollo Integrado (IDLE según sus siglas en inglés) que instala el paquete.

El IDLE contiene un intérprete que permite fácilmente ejecutar sentencias, realizar pruebas y crear pequeñas funciones. También ofrece la opción de depurar código y un visor de pila en el menú *Debug.*

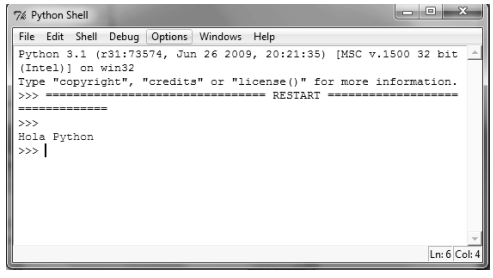


Es posible crear módulos mediante la ruta *File->New Window*; los módulos serán analizados en detalle próximamente. Solo para que el lector comprenda en este punto, un módulo es básicamente una unidad que empaqueta funcionalidad.

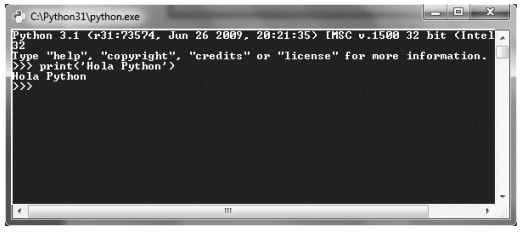
El módulo que se observa a continuación, de nombre prueba.py, realiza un llamado a la función *print* con el texto 'Hola Python'. Este módulo puede ejecutarse a través del menú *Run->Run Module*.



Luego de ejecutar el módulo anterior.



El paquete también viene acompañado de una consola a modo de intérprete.



Finalmente una amplia documentación que contiene detalles de funciones, clases, etc. acompaña al paquete.

1.2 Características

Actualmente Python es un lenguaje que goza de gran aceptación, y no solo entre estudiantes y aficionados, sino que ahora también se comienza a utilizar en ámbitos científicos y en el procesamiento de grandes volúmenes de información. Algunas de sus características distintivas son las siguientes:

- Es un lenguaje multiparadigma, soporta y favorece la programación orientada a objetos y tiene vestigios de la programación funcional y la estructurada.

- Tiene una sintaxis limpia y reducida que propicia la creación de códigos muy legibles y compactos.
- Es gratuito y libre, un caso claro de Open Source Software - Gratuito/Libre y Software de Fuente Abierta. En otras palabras, pueden distribuirse libremente copias del *software*, puede leerse su código fuente, llevar a cabo cambios, usar partes del mismo en nuevos programas libres, y, de manera general, se puede acometer cualquier acción que se desee con los códigos fuente. Se basa en la idea de una comunidad que comparta conocimiento y esta comunidad resulta un pilar fundamental en los avances que tiene el lenguaje día a día.
- Es multiplataforma, portable. Dado que el lenguaje es Open Source es soportado en diversas plataformas por lo que el código que se desarrolle en una determinada plataforma será compatible y ejecutable en otras plataformas. A pesar de esto, se debe ser lo suficientemente precavido para evitar la inclusión de características con dependencia de sistema en el código (librerías o módulos que operen solo en un sistema en particular). Python puede utilizarse sobre Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE y PocketPC.
- Es un lenguaje interpretado. Los programas desarrollados en lenguajes compilados como C o C++ se traducen de un lenguaje fuente a otro lenguaje comprensible por un ordenador (código binario, secuencias de ceros y unos) empleando un programa conocido como compilador. Cuando se ejecuta un programa, el *software* encargado de esta tarea guarda el código binario en la memoria del ordenador e inicia la ejecución desde la primera instrucción. Cuando se emplea un lenguaje interpretado como Python, no existen compilaciones separadas ni pasos de ejecución, simplemente se ejecuta el programa desde el código fuente. Intrínsecamente, Python convierte el código fuente a una representación intermedia conocida como *bytecodes* y luego lo traduce a un lenguaje nativo en el ordenador para finalmente ejecutarlo. Es por ello que de alguna forma es mucho más sencillo que otros lenguajes. He ahí su carácter portable, la mera copia del código de un programa en Python a cualquier otro sistema resultará en el mismo programa, considerando por supuesto la existencia de los módulos, librerías de los que hace uso el programa en cada sistema.
- Administración automática de memoria.
- En general, es fácil de aprender.

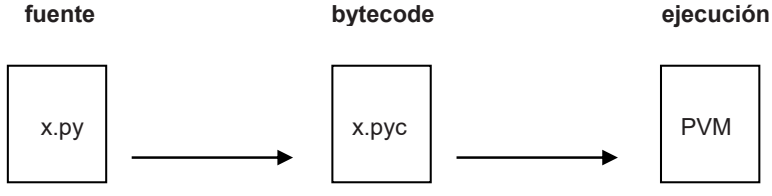
Durante las próximas secciones se abordarán temas que ayudarán a comprender mejor algunas de las particularidades mencionadas previamente. También se describirá el entorno de trabajo que se utilizará en el transcurso de este libro para desarrollar los diferentes códigos de ejemplos.

1.3 Máquina Virtual

Desde un punto de vista general, un programa en Python es simplemente un fichero compuesto por un conjunto de sentencias del lenguaje. Este fichero, que no es más que un fichero de texto plano con extensión .py, puede crearse con cualquier editor de texto y luego ser provisto de un conjunto de sentencias. Una vez que se haya definido este conjunto es necesario indicar a Python que ejecute el código, lo cual se traduce en ejecutar cada sentencia en el fichero de arriba hasta abajo. Esta acción puede llevarse a cabo a través de un comando en la consola de Python o simplemente mediante un botón *Run* (*Ejecutar*) en el entorno de desarrollo utilizado.

Cuando finalmente se realiza la acción de ejecutar el código sucede que es compilado a una forma intermedia llamada *bytecode* y luego éste es suministrado a la Máquina Virtual de Python (PVM según sus siglas en inglés) que es el motor de ejecución de Python.

Bytecode es una representación intermedia del código fuente, es una traducción del código a un formato de bajo nivel que no es binario sino una especificación del propio lenguaje y que es independiente de la plataforma. El *bytecode* generado suele almacenarse en el disco duro como un fichero con extensión *.pyc*, *c* de compiled y se almacena con el objetivo de acelerar la ejecución del programa que para ejecuciones sucesivas reutilizará este *bytecode* generado y evitará, de ser posible, el paso de la compilación. Para conocer si puede evitarse la etapa de compilación se revisan las marcas de tiempo del fichero fuente y del fichero *bytecode*, de ser distintas se procede a la compilación. Luego se suministra el *bytecode* a la Máquina Virtual de Python (PVM)

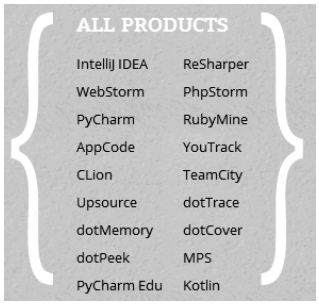


La PVM consiste básicamente en un ciclo que ejecuta todas las instrucciones contenidas en el fichero .pyc y forma parte del sistema instalado en el paquete de Python, es el último paso del conocido intérprete de Python.

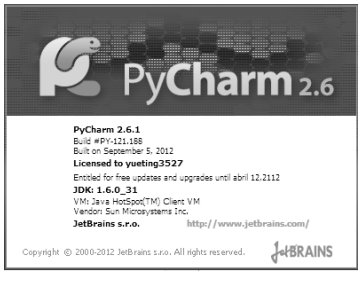
1.4 Entornos de Desarrollo Integrados

Un Entorno de Desarrollo Integrado (IDE según sus siglas en inglés, *Integrated Development Environment*) es un programa que incluye un editor de texto, uno o varios compiladores, depuradores y, en algunos casos, sistemas para desarrollar interfaces gráficas. Es una herramienta que contribuye a facilitar y humanizar la tarea del programador ofreciendo un ambiente cómodo para desarrollar aplicaciones.

El entorno utilizado en este libro corresponde a un producto de JetBrains, empresa líder en el desarrollo de herramientas de este estilo. Una lista con todas las herramientas de la compañía puede encontrarse en su sitio oficial http://www.jetbrains.com.



Entre los productos que el autor ha utilizado y recomienda al lector se encuentran WebStorm (desarrollo web, HTML, CSS, Node.js), ReSharper (Visual Studio), PhpStorm (desarrollo web, PHP) y finalmente PyCharm que será el IDE empleado en todos los códigos de este libro.



PyCharm provee muchas facilidades para desarrollar aplicaciones: incluye autocompletamiento (siempre que es posible), permite crear proyectos vacíos o siguiendo plantillas para proyectos Django, Google App Engine, etc., y también incluye soporte para crear código HTML y JavaScript.

El entorno de trabajo posee un panel de salida que de manera predeterminada aparece en la parte inferior y donde es posible visualizar las impresiones realizadas en el código.

Este panel será visto durante los siguientes capítulos para mostrar los resultados de los diferentes ejemplos del libro.

1.5 Sintaxis básica

Python es un lenguaje que propicia la creación de código legible y compacto. Tiene la característica de ser altamente dinámico por lo que su sintaxis carece de la declaración del tipo de variables, lo cual puede resultar en diversas ocasiones en beneficio de una sintaxis clara y concisa. Se encuentra muy cercano a la forma en que nosotros los seres humanos realizamos órdenes a otros, por ejemplo suponiendo que alguien desee, de manera imperativa, orientar a otra persona que imprima un cartel que diga 'Hola Python', entonces en un lenguaje como Python se procedería de la siguiente forma:

print ('Hola Python')

En este caso *print* es el tipo de orden o comando, *print* contiene la descripción de la orden y cómo debe ejecutarse mientras que 'Hola Python' es aquello que utiliza la orden para realizarse, es un prerrequisito.

```
def suma(self,*matrices):
    for i in range(self.filas):
        fila = []
        for j in range(self.columnas):
            temp = self.elems[i][j]
            for m in matrices:
                temp += m.elems[i][j]
            fila.append(temp)
        yield  fila
```

Python es un lenguaje basado en la *identación*, no utiliza bloques de instrucciones encerrados entre llaves ({ }) como los lenguajes de la familia C, Java o JavaScript, sino que solo se basa en la identación a nivel de funciones, clases, etc. La identación es lo que se conoce comúnmente como sangría, o sea, separar el texto del margen izquierdo mediante la introducción de espacios o tabuladores para así darle un orden visual y lógico al código. Afortunadamente PyCharm delimita mediante líneas blancas las divisiones lógicas de la identación y favorece así la identificación de los límites de identación. Obsérvese el código anterior.

1.6 Módulos

Los módulos son objetos contenedores que organizan de manera recursiva el código en Python; se dice de manera recursiva porque, al ser objetos, un módulo puede contener objetos y también otros módulos. Cada módulo tiene un espacio de nombres asociado que se puede ver como el nombre del propio módulo.

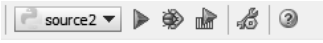
1.6.1 Sentencia import

La palabra clave utilizada para importar un módulo es *import* y una vez que se importa pueden utilizarse todos los objetos que en este se contienen. En el siguiente ejemplo se importa el módulo *math,* que contiene funciones y constantes matemáticas. Para que el lector comience a conocer el lenguaje debe saber que en Python todo se considera un objeto, eso incluye a las funciones.

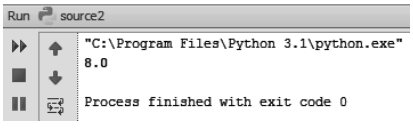
```
import math

print(math.pow(2,3))
```

Para ejecutar el código anterior debe presionarse el botón *Run* o *Ejecutar,* que se encuentra en la parte superior de la interfaz gráfica de PyCharm.



Una vez presionado, se ejecutará el código, el cual debe estar en un archivo de Python previamente creado en el menú *File* o *Archivo*. El panel de salida mostrará los resultados.



En este caso se ha utilizado la función *pow(x, y)* del módulo *math* que devuelve el resultado de elevar el número *x* a la potencia *y*. Una sentencia similar a *import* también puede encontrarse en otros lenguajes de programación.

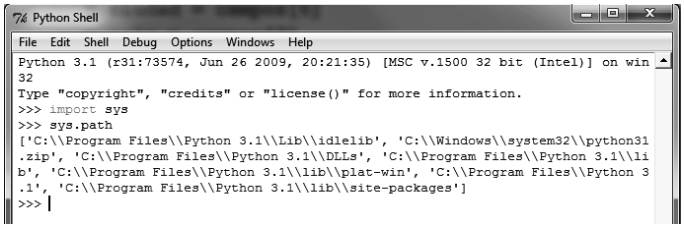
C# using System.Text

C++ #include<math.h>

Java import java.util

Los módulos ofrecen varias ventajas entre ellas la más notable es la reutilización de código, ya que como se mencionó anteriormente, un módulo sirve como contenedor de funcionalidad. Además de módulos, Python también incluye otro tipo de contenedor conocido como paquetes. Un paquete es un módulo de Python que contiene otros módulos y/o paquetes. La diferencia entre un paquete y un módulo radica en que el primero contiene un atributo __path__ que representa la ruta en el disco duro donde está almacenado el paquete. Desde un punto de vista físico, los ficheros con extensión .py son módulos, mientras que cualquier directorio que contenga un archivo con nombre __init__.py representa un paquete. Así se puede resumir que los módulos son ficheros y los paquetes pueden ser ficheros o directorios con ficheros.

El *Python Path* indica las rutas donde se buscarán los módulos, dicha ruta puede consultarse por medio de la variable *path* del módulo *sys* (sistema).

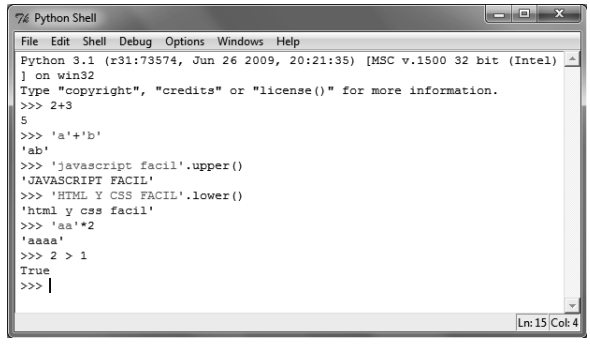


Un punto relevante a destacar en el *Python Path* reside en el hecho de que la estructura de datos utilizada para almacenar las cadenas es una lista y las listas

son objetos **mutables** (pueden sufrir cambios)**.** Esto se traduce en que si manipulamos el Python Path podemos indicar nuevas rutas a Python para que busque módulos y paquetes.

1.7 Modo intérprete vs. modo script

En Python existen dos modos para ejecutar sus códigos: el modo intérprete o interactivo y el modo script. El primero resulta bastante útil cuando se desea probar códigos pequeños, funciones, operadores u operaciones del lenguaje, etc. En este caso, el intérprete de Python interpreta y ejecuta cada sentencia y retorna un resultado, en caso de existir.



La segunda opción se basa en la idea de un conjunto de sentencias que conformen un *script* o fichero. En este caso se interpretan y ejecutan las sentencias en su totalidad y no una a una como sucede con el modo intérprete. En el IDE de JetBrains PyCharm trabajamos siempre en modo *script*, definiendo un conjunto de sentencias y obteniendo como resultado la ejecución de todas las sentencias del archivo, como un todo. En modo *script* es posible guardar los ficheros que representan el código del programa mientras que en el modo interactivo evidentemente no existe esta posibilidad. El intérprete puede ser útil para llevar a cabo experimentos pero para desarrollar un programa siempre debe utilizarse el modo *script*.

1.8 Diferentes versiones

El mantenimiento y desarrollo de Python es guiado por Guido Van Rossum junto a un equipo de desarrolladores del núcleo del lenguaje. Guido tiene la última palabra en lo que respecta a la inclusión de librerías y lo que se añade o no en el lenguaje; es, como se dice popularmente, el Dictador Benévolo de por Vida. La propiedad intelectual de Python pertenece a la Python Software Foundation, una organización sin ánimo de lucro encargada de promover el lenguaje.

Los cambios propuestos para Python son detallados en documentos llamados Propuestas de Ampliaciones de Python (en inglés *Python Enhancement Proposals*). Estas son debatidas por los desarrolladores y la comunidad de Python y finalmente aprobadas o rechazadas por Guido. Muchas personas contribuyen a mejorar el lenguaje a través de discusiones, reportes de errores, creación de librerías, etc.

Nuevas versiones de Python pueden introducir cambios así como facilitar el uso del lenguaje y añadirle posibilidades.

1.9 Relación con otros lenguajes

En esta sección se realizará una comparación entre Python y algunos de los lenguajes más populares de la actualidad. El objetivo de esta comparación es que el lector pueda sacar conclusiones así como conocer las ventajas y desventajas que cada uno posee.

1.9.1 PHP

La siguiente tabla que asume varios criterios comparativos resume las diferencias entre los lenguajes de PHP y Python en torno a los criterios tenidos en cuenta.

Criterio	PHP	Python
Popularidad del lenguaje	Mayor	Menor
Discusiones del lenguaje	Menor	Mayor
Tipado	Débilmente	Dinámico
Sitios desarrollados con el lenguaje	Facebook, Wikipedia	Google, YouTube
Diseñado para	Desarrollo web	Propósito general
Usabilidad	Sigue un patrón clásico, usabilidad media	Lenguaje legible y usable
Fácil de aprender	No tanto si se comienza de cero	Genial para novatos, estudiantes

El límite principal que presenta PHP es que es un lenguaje para la web; en cambio, Python es de propósito general, puede hacerse uso del lenguaje en marcos de trabajo web como Django y también es posible desarrollar aplicaciones de escritorio utilizando PyQt o Tkinter.

1.9.2 Java

Al igual que en la sección anterior en esta se presenta una tabla comparativa, esta vez entre los lenguajes de Python y Java.

Criterio	Java	Python
Tipado	Estático	Dinámico
División de código	Llaves	Identación
Usabilidad	Sigue un patrón clásico, usabilidad media	Lenguaje legible y usable
Fácil de aprender	Fácil	Genial para novatos, estudiantes
Diseñado para	Propósito general	Propósito general

Python y Java son ambos lenguajes de propósito general y ambos emplean una máquina virtual para ejecutar sus códigos. Java sigue un enfoque sintáctico similar a aquellos de los lenguajes de la familia C, mientras que Python es altamente dinámico y nunca requiere la declaración del tipo de una variable.

1.9.3 CSharp

Finalmente se realiza una comparación entre Python y un miembro de la plataforma .NET, que comparte varias similitudes con Python; este lenguaje es CSharp.

Criterio	CSharp	Python
Tipado	Estático, aunque incluye inferencia de tipos	Dinámico
División de código	Llaves	Identación
Usabilidad	Sigue un patrón clásico, usabilidad media	Lenguaje legible y usable
Fácil de aprender	Fácil	Genial para novatos, estudiantes
Diseñado para	Propósito general	Propósito general
Rendimiento	Se le atribuye un rendimiento ligeramente mejor	Rendimiento ligeramente menor
Multiparadigma	Orientado a objetos, Funcional, Estructurada	Orientado a objetos, Funcional, Estructurada

Ambos lenguajes son bastante fáciles de aprender, Python siempre con puntos adicionales en este apartado dado su alta legibilidad. Ambos son de propósito general y existen marcos de trabajo web bastante populares para cada lenguaje, ASP .NET MVC para CSharp y Django para Python. Son multiparadigma y soportan la programación funcional. En próximos capítulos veremos cómo Python brinda facilidades para hacer uso de este paradigma de programación.

1.10 Implementaciones del lenguaje

Una implementación de Python es el modelo de ejecución analizado en la sección 1.3 o una variación del mismo. Las implementaciones más conocidas de Python son CPython, Jython e IronPython.

CPython corresponde con la versión clásica de Python, la más actualizada, optimizada y completa de las implementaciones del lenguaje. Aquella que ha sido mencionada y será estudiada en este libro. CPython está conformada por un compilador, un intérprete y un conjunto de módulos escritos en C que pueden utilizarse en cualquier plataforma cuyo compilador C vaya de acuerdo con la especificación estándar ISO/IEC 9899:1990.

Jython es la implementación de Python para cualquier Máquina Virtual de Java (JVM según sus siglas en inglés) que esté acorde con Java 1.2 o superior. Con Jython es posible utilizar todas las librerías y marcos de trabajo de Java.

Finalmente IronPython es la implementación de Python para la CLR (*Common Language Runtime*), la máquina virtual de .NET. En analogía con Jython, IronPython permite hacer uso de todas las librerías y marcos de trabajo de la plataforma.

Ejercicios del capítulo

1. Responda V o F. Justifique en caso de ser falso:

a) Python no es un lenguaje multiparadigma.

b) Python utiliza llaves para delimitar bloques de código.

c) El *bytecode* siempre es generado sin importar si el código fuente ha sufrido cambios o no.

d) La ejecución del código fuente es llevada a cabo finalmente por la Máquina Virtual de Python (PVM).

e) Python posee una sintaxis clara la cual favorece la creación de código legible.

CAPÍTULO 2.
Elementos del lenguaje

La popularidad de Python viene dada sin duda alguna por algunas de sus características más llamativas. Entre estas particularidades cabe mencionar su expresividad, obtenida a través de una estructura sintáctica organizada, concisa, clara. El hecho de ser un lenguaje multiparadigma y de alto nivel, con una evidente inclinación hacia el paradigma de la programación orientada a objetos, también ha contribuido a su aceptación e inclusión como lenguaje de preferencia de muchos en todo el mundo. El objetivo de este capítulo será entrar en detalles en la sintaxis de Python, en la forma en la que se indican variables, funciones, se definen clases, se utilizan operadores y demás cuestiones que resultan elementos esenciales en un lenguaje de programación.

2.1 Estructura léxica

La estructura léxica de un lenguaje es el conjunto de reglas que permiten formar un programa en ese lenguaje. Esta estructura se encuentra apoyada en una gramática que sirve como formalismo de esa estructura y que define la sintaxis y, en caso de ser una gramática atributada, también la semántica. Mediante esta estructura se define qué se entiende por una variable válida en el lenguaje, cómo se forman las estructuras de bucle, las estructuras de control de flujo, etc.

2.1.1 Identación

A diferencia de otros lenguajes como los de la familia C, Python no utiliza llaves ({ }) para delimitar bloques de código, tampoco utiliza símbolos delimitadores de sentencias como el clásico punto y coma (;). En su lugar, para reconocer y delimitar bloques de código utiliza un sistema basado en espacios a la izquierda conocido como identación. La identación es básicamente como la sangría en tipografía, esto es, la inserción de espacios o tabuladores para mover un texto hacia la derecha. Los programas en Python deben seguir un orden jerárquico de identación para que su ejecución sea según lo esperado. Por ejemplo las sentencias que pertenezcan a un ciclo no pueden estar al mismo nivel de identación que la definición del ciclo. El siguiente ejemplo ilustra un caso en que la identación resulta errónea.

```
for i in lista:
print(i)
```

Considerando que la función *print* se encuentra definida al mismo nivel de identación que el ciclo *for* entonces se asume que esta no pertenece al bloque de instrucciones del ciclo, por ende es un ciclo sin instrucciones, lo cual se traduce en un error. La manera correcta de definir el bucle sería la siguiente:

```
for i in lista:
    print(i)
```

Es importante notar que las sentencias que tengan la misma connotación o jerarquía en el programa deben estar al mismo nivel de identación. Si en el ejemplo anterior se quisiera imprimir siempre i +1 una opción válida sería el siguiente código:

```
for i in lista:
    i = i + 1
    print(i)
```

Como las sentencias *i = i + 1* y *print (i)* se encuentran al mismo nivel de identación, entonces ambas se ejecutarán dentro del ciclo. Fíjese también en que el final de sentencia no va acompañado de un punto y coma sino de un cambio de línea.

2.1.2 Tokens

Los *tokens* son elementos esenciales que se definen en la gramática de un lenguaje. En el proceso de compilación estos elementos son extraídos por un componente conocido como lexicográfico y entregados al analizador sintáctico. Entre estos elementos figuran los identificadores, las palabras reservadas, los operadores, los literales y los delimitadores. Existen porciones de texto como los comentarios, que en el caso de Python aparecen precedidos del carácter # y son ignorados por el compilador. El compilador es un componente que se constituye de los analizadores previamente mencionados y de otras herramientas que contribuyen a que un programa en Python pueda ejecutarse en un ordenador.

2.1.3 Identificadores

Un identificador es un nombre utilizado para definir el nombre de una variable, función, clase, módulo u otro elemento del lenguaje. En Python los identificadores comienzan con una letra o un guion bajo (_) seguido por cero o más letras, guiones bajos o dígitos. Visto como una expresión regular un identificador puede ser cualquier expresión de: (_)?(a...z|A...Z)+(a...z|A...Z|_|0…9)*. Fíjese en que se han considerado tanto letras en mayúsculas como en minúsculas, Python es case sensitive, lo cual quiere decir que el identificador “a” es diferente del identificador “A”. Por convenciones en Python los identificadores de clases comienzan con mayúsculas y el resto en minúsculas, cuando un identificador comienza con guion bajo se supone que el elemento creado es privado. En el caso de que comience con dos guiones bajos, entonces por convención se supone que es fuertemente

privado y si termina también con dos guiones bajos entonces es un nombre especial definido en el lenguaje.

2.1.4 Literales

Los literales son representaciones sintácticas de valores primitivos soportados por un lenguaje de programación. Estos valores pueden ser enteros (Integer), coma flotante (Float), cadenas (String), binarios (Binary), etc. Considere el próximo código donde se muestran diferentes literales en Python y el tipo de valor al que se asocia.

```
2           # Integer
2.3         # Float
'Jazz'      # String
"Picasso"   # String
```

2.1.5 Delimitadores

Un delimitador puede cumplir, entre otras, la función de servir de organizador de código. A continuación una lista con los delimitadores de Python.

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

Las últimas dos filas contienen los conocidos como operadores de asignación incremental que no solo sirven como delimitadores sino también realizan una determinada operación.

2.1.6 Sentencias

Un programa en Python puede descomponerse en un conjunto de sentencias las cuales a su vez pueden ser descompuestas en sentencias simples y compuestas. Una sentencia simple, como pudiera ser, por ejemplo, una asignación es una sentencia que no contiene otras sentencias. Varias de estas sentencias pudieran aparecer en una misma línea separadas por el delimitador (;). Una sentencia compuesta como por ejemplo un ciclo es una sentencia que, de manera lógica y necesaria, requiere de otras sentencias en su cuerpo para cumplir una determinada función.