

UNIVERSIDAD DE SALAMANCA

ADMINISTRACIÓN DE SISTEMAS

Python

Autor:

Diego MARTÍN

Profesora:

Dr. Vivian LÓPEZ

10 de abril de 2014

Índice

1. Introducción	2
1.1. ¿Por qué usar Python?	2
1.2. Contras	3
1.3. ¿Quién usa Python?	4
1.4. ¿Qué se hace con Python?	4
1.5. Fortalezas técnicas de Python	5
1.6. ¿Cómo se ejecuta un programa en Python?	5
2. Números	6
2.1. ¿Cómo se almacena un elemento en memoria?	7
3. Cadenas de texto	7
3.1. Manipulación de cadenas	7
3.2. Secuencias de escape	8
3.3. Formateo avanzado de cadenas	9
3.4. Consideraciones especiales	9
4. Listas y diccionarios	9
4.1. Diccionarios	11
5. Tuplas, archivos y otras estructuras de datos	12
5.1. ¿Para qué sirve tener tipos inmutables y mutables	12
5.2. Índices. Las mejoras de Python	13
5.3. Ficheros	14
6. Condicionales	14
6.1. Reglas de sintaxis	15

The Zen of Python

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one – and preferably only one – obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than **right** now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea – let's do more of those!

1. Introducción

1.1. ¿Por qué usar Python?

Para aquel que no aún no conozca las virtudes de este lenguaje, y qué ventajas proporciona para sus labores como programador, puede que al principio se muestre reacio a adoptar esta tecnología. Sin embargo, puede que estos principios de desarrollo hagan cambiar de opinión hasta al más acérrimo a otros lenguajes.

- Python es legible y coherente.

‘Legible, por tanto mantenible’, es una de las premisas del lenguaje. Se aleja de lenguajes como Perl en ese aspecto. Además es un lenguaje orientado a objetos, lo cual aumenta su reusabilidad enormemente.

Muchos dicen que Python *encaja en tu cerebro*. Todo es comprensible cuando se

dominan unos pocos conceptos clave. Además, el enfoque general es muy minimalista.

- Potenciar la productividad del programador

Un código en Python ocupa 1/5 de su equivalente en C o Java. Además se elimina el proceso de compilación, dado que es un lenguaje interpretado¹.

- Portabilidad

Todos los programas en Python son multiplataforma *per se* o requieren muy pocos cambios para trasladar un programa a otro sistema. Esto se aplica también a las interfaces gráficas de usuario.

- Bibliotecas de soporte

Python cuenta con una gran cantidad de recursos de serie, permitiendo conseguir tareas de nivel de aplicación con la funcionalidad con la que se cuenta desde el momento en el que se instala. Además, existe una gran cantidad de código desarrollado por terceros y la posibilidad de crear módulos personales para solucionar tareas.

- Integración

Python se integra con lenguajes y tecnologías como C, C++, Java, COM, Corba, .NET, SOAM, XML-RPC

- Disfrútalo

Python es un lenguaje pensado para hacer al programador la tarea de desarrollar software mucho más amigable y entretenida.

1.2. Contrás

El principal problema de Python es su rendimiento. Al ser un lenguaje interpretado y no compilado, no es tan potente como C o C++. Sin embargo actualmente se ha mejorado notablemente su rendimiento gracias a la traducción del código a un *bytecode*, que es realmente el conjunto de sentencias utilizado por la PVM (Python Virtual Machine). Generalmente es un lenguaje rápido, y cuando es necesario optimizar el rendimiento al máximo, se puede realizar una traducción a C y traducirlo posteriormente a código

¹A lo largo de la historia del desarrollo de software se han dado situaciones en las que era difícil encontrar suficientes programadores para implementar ciertas aplicaciones. Actualmente los equipos se ven forzados a conseguir las mismas tareas con menos gente. En ambos escenarios, Python permite conseguir más con menos.

máquina, por lo que se ejecutaría a velocidades nativas. También es posible escribir estas partes directamente en C y enlazarlas al resto del código.

1.3. ¿Quién usa Python?

Con una comunidad de entre 500,000 y un millón de usuarios, Python cuenta con una comunidad alrededor muy grande. Compañías como Google o Yahoo! utilizan el lenguaje en sus servicios en red, y para HP, Seagate o IBM es una herramienta de utilidad a la hora de realizar pruebas de hardware² Proyectos como el microordenador Raspberry Pi promocionan y utilizan Python como lenguaje principal de programación a la hora de enseñar a nuevos programadores y al utilizar el dispositivo.

1.4. ¿Qué se hace con Python?

Administración de sistemas Crear interfaces para servicios de un sistema operativo es muy sencillo en Python, lo cual lo convierte en una herramienta idónea para administradores de sistemas a la hora de trabajar con árboles de directorios, búsqueda de ficheros, lanzamiento de otros programas, procesamiento paralelo ...

Interfaces Gráficas de Usuario Crear interfaces en Python es muy simple. El entorno trabaja de forma natural con objetos diseñados para crear IGUs, como Tk Tkinter, que se adapta a la estética de cada plataforma, wxPython, basada en C++ o PythonCard.

Scripting Es fácil realizar tareas de red con Python, y es muy utilizado para CGI (*Common Gateway Interface*), FTP, sockets o HTML. Módulos como HTMLGen permiten generar código y otros como win32all insertan código Python como JavaScript. Además, utilizades como Zope, WebWare o Quixote permiten el desarrollo de sitios web grandes de forma rápida y sencilla.

Prototipado Debido a que el desarrollo en Python es muy ágil, muchos desarrolladores aprovechan esta característica para generar ‘esbozos’ de sus proyectos para evaluarlos antes de crear la versión definitiva en otro lenguaje. Aprovechando además la integración con otros lenguajes, en ocasiones partes del prototipo en Python son integradas en el resto del código.

Programación numérica, juegos, trabajo con imágenes, inteligencia artificial, etcétera

²La lista completa se puede encontrar en python.org

1.5. Fortalezas técnicas de Python

- Es un lenguaje orientado a objetos.
- Es libre (licencia de la Python Software Foundation, compatible con la GPL).
- Muy soportado.
- Espíritu de comunidad. Guido van Rossum, creador de Python es el *Benevolent Dictator For Life of Python* (Benevolente dictador vitalicio de Python, orquesta a un equipo de 1000 personas encargados de mejorar el lenguaje. Los cambios siguen un proceso de mejora formal, que es analizado por Guido. Esta metodología hace que el desarrollo sea mucho más conservador que en otros lenguajes.

- Portabilidad

- Potencia

Es un lenguaje dinámicamente tipado.

Gestión automática de la memoria, utilizando un contador de referencias

Tipos predefinidos que cubren la mayoría de las estructuras de datos utilizadas.

Gran variedad de herramientas incluidas en el lenguaje.

Gran cantidad de bibliotecas y código de terceros

- Se combina con facilidad con otros lenguajes
- Es fácil de utilizar y aprender.
- Sintaxis clara.

1.6. ¿Cómo se ejecuta un programa en Python?

Para ejecutar un programa en Python es necesario únicamente un intérprete y una biblioteca de soporte. El código es compilado a *bytecode*, un código de bajo nivel e independiente de la plataforma utilizada. Un archivo .pyc es generado, y es interpretado por la máquina virtual de Python del sistema³. Existen además implementaciones especiales,

³Hay varias implementaciones de la máquina virtual: CPython está escrita en ANSI C, Jython, que cuenta con clases de Java que compilan el código de Python a *bytecode* de Java, dirigiéndolo a la *Java Virtual Machine* del sistema o Python.NET, que se integra con C#.

como JIT Python, que incorpora un compilador *Just In Time*, o binarios ‘congelados’ (*frozen binaries*), que empaquetan todo lo necesario para que un programa se ejecute (máquina virtual, código, etcétera).

En todo código se deberá indicar la ruta del intérprete. Utilizando el fichero `/usr/bin/env python` obtenemos la ruta se encuentre donde se encuentre el intérprete, por lo que evitamos modificar el código a la hora de cambiar de sistema.

`#!/usr/bin/env python`

También es posible trabajar directamente con la línea de comandos, y el lenguaje es compatible con tuberías y redirección de las diferentes entradas y salidas.

2. Números

Trabajar con valores numéricos algo habitual en, si no la totalidad de las aplicaciones en cualquier lenguaje, en una inmensa mayoría. Python incorpora tipos predefinidos para simplificar las tareas más habituales. Tipos de números:

- Enteros (`int`)
- Real en coma flotante (`float`) y de doble precisión (`float`)
- Como diferencia con otros lenguajes, Python soporta de forma nativa operaciones con números complejos.

El límite de tamaño para un número en Python lo determina la memoria, y debido al carácter dinámico del lenguaje, no es necesario definir el tipo de número. Operaciones con octales y hexadecimales son triviales también. La conversión es automática y se realiza siempre (a menos que se especifique de forma explícita). La representación es variable, utilizándose las funciones `str()` y `repr()` para cambiar de la forma utilizada por el ordenador a una representación más ‘humana’.

Los operadores de C han sido integrados en Python respetando la precedencia de cada uno. Se incluyen algunas novedades y mejoras en algunos operadores. Destacan:

- Creación de funciones sin nombre (‘anónimas’): `lambda <argumentos>: <expresión>`
- `is` para determinar el tipo de objeto.
- Operador `//` para división con truncado.
- `oct()`, `hex()`, `str()`, `repr()`... para conversiones de tipo.
- Nuevas operaciones con arrays y otras colecciones de datos (se verán más tarde)

2.1. ¿Cómo se almacena un elemento en memoria?

Aprovechando que hemos cubierto el primer grupo de tipos de datos, es importante conocer cómo funciona una variable declarada en Python. Toda variable almacena una referencia a un elemento en memoria, por lo que las mismas no tienen un tipo definido. Esto no significa que los elementos en memoria no tengan tipos definidos. Al contrario, es importante conocer las características de cada uno, dado que la conversión entre tipos no es automática. Funciones como `str()` permiten realizar la transformación. El contador de referencias es el encargado de la gestión de la memoria, liberando el espacio ocupado por un bloque de datos cuando se pierde la última referencia.

3. Cadenas de texto

Una cadena de texto en Python es un tipo especial de secuencia (las veremos posteriormente). No existe el tipo `char` como en C y, a diferencia de este, Python cuenta con un potente conjunto de utilidades para manipular cadenas. Las cadenas de texto son elementos inmutables, por lo que una vez creados no se podrán modificar (es necesario crear una nueva cadena con los cambios si se desea alterar uno de estos elementos). Ejemplos de literales `string`:

- Cadena vacía: `cadena = ''`
- Cadena vacía (con dobles comillas): `cadena = ""`
- Bloque vacío (un bloque permite representar cadenas de texto de varias líneas):
`bloque = """..."""`
- Cadena en bruto: `bruto = r'\temp\haus'`
- Unicode: `u'universal'`
- Formateo: `"Resultados: %d" % resultados`

3.1. Manipulación de cadenas

Operadores

- Concatenación: `+`

- Repetición: `*`
- Índice: `cadena[indice]`
- Longitud: `len(cadena)`
- Búsqueda de elementos: `cadena.find('Waldo')`
- Iteración: `for x in cadena`
- Longitud: `len()`

Métodos

- `cadena.capitalize()`
- `cadena.center()`
- `.encode()`, `.endswith()`, `.expandtabs()` ...
- `.isalnum()`, `.isdigit()`, `.istitle()`, `.isupper()`
- ...

El soporte de Unicode está integrado en el lenguaje, y siempre que se combinen con otros tipos de cadenas (por ejemplo, en una concatenación) el resultado será una cadena Unicode. Dado que una cadena es un conjunto ordenado de caracteres pueden ser manipuladas mediante índices.

3.2. Secuencias de escape

- `\newline`

3.3. Formateo avanzado de cadenas

Se puede utilizar cualquier especificados de formato de C en Python, así como alguno nuevo.

- `%s`
- `%d`
- `%r`

3.4. Consideraciones especiales

Una cadena no se convierte automáticamente a número o a cualquier otro tipo de datos, es necesario hacerlo explícitamente.

4. Listas y diccionarios

Existen tres categorías generales en Python: números, secuencias y mapas. Cada una de ellas tiene unas propiedades especiales y operaciones asociadas a él. Hasta ahora hemos visto las dos primeras (números y secuencias, dado que las cadenas de caracteres son un tipo de estas). La categoría restante, los mapas.

Las listas son la colección de elementos ordenada más flexible de Python. Pueden contener cualquier elemento, dado que

son meramente colecciones de referencias, y permiten anidar más listas dentro de ellas. Una ventaja de Python es que la mayoría de las operaciones que en otros lenguajes deben ser implementadas por el programador se encuentran disponibles de serie en el lenguaje. Las propiedades de estas soluciones son:

- Son mutables, de longitud variable y orden definido.
- Se accede por índices.
- colecciones de referencias. Obtener un elemento es tan rápido como en C (de hecho son arrays de C dentro del intérprete). Mientras que no se especifique explícitamente, los elementos son referencias y no copias.

Ejemplos de listas:

```
#!/usr/bin/env python  
L1 = [];  
L2 = [ 'abc' , 'is' , 'as' , 'easy' , 'as' ];  
L2.append( '123' );
```

Operaciones con listas:

- Operaciones básicas: `len()`, `['Monty'] *4` (multiplicación).

- `for x in [1,2,3]: print x`
- `[1,2] + list("34")` ⁴.
- Modificación: `.sort()`, `.extend()`

4.1. Diccionarios

Los diccionarios son el tipo más flexible que viene integrado en Python. Son colecciones sin orden de elementos, cuya distinción respecto las listas es que los elementos son de tipo valor-clave. Pertenecen a la categoría de mapas mutables, y se accede a ellos mediante una clave. No tienen un orden particular⁵. Las claves son localizaciones simbólicas de ítems en un diccionario. Son colecciones de longitud variable, heterogéneas y arbitrariamente anidables. La sintaxis de sus elementos es `{clave:valor}`. Ejemplos de diccionarios:

- `D='spam':1, 'eggs':2;`
- Operaciones: `.copy()`, `.keys()`, `.has_key()`, `.values()`, `.get()`
- Acceso a elementos: tal y como en un array: `D['spam']`
- Concatenación: `D.update(d1)`

⁴Importante, utilizar operadores con distintos tipos de operados generará un error.

⁵De hecho, Python ordena aleatoriamente los elementos para conseguir un acceso más rápido

Son muy utilizados como estructuras de datos para una multitud de casos.

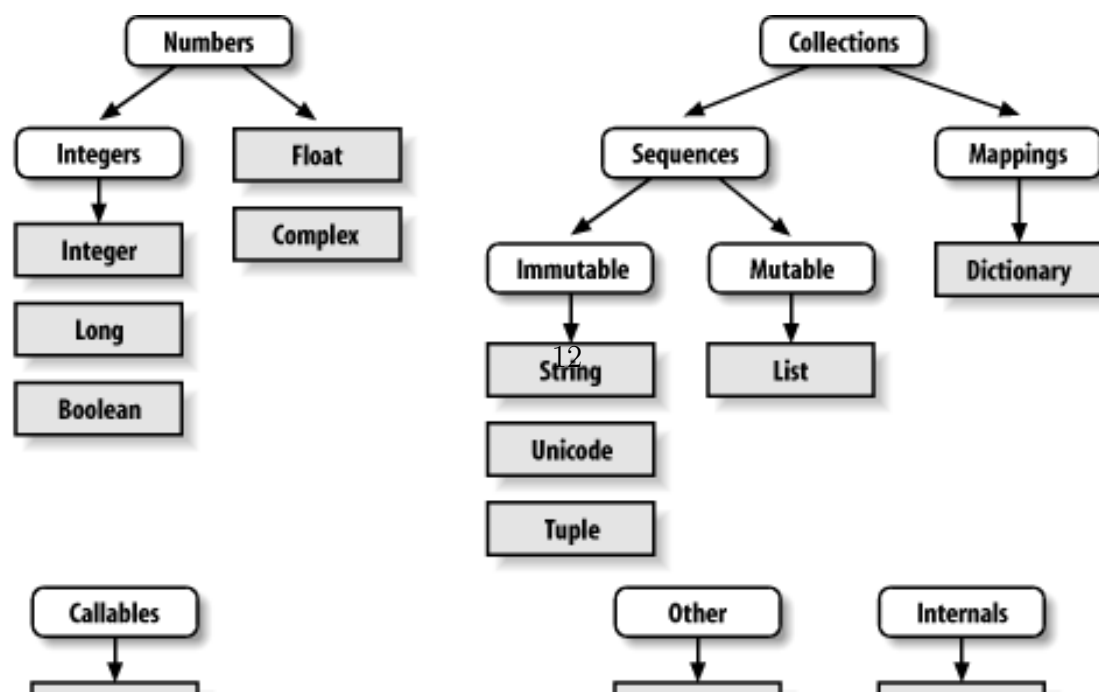
5. Tuplas, archivos y otras estructuras de datos

Las tuplas son colecciones equivalentes a las listas, pero de carácter inmutable. Declarar una tupla es muy parecido a la creación de listas:

```
#!/usr/bin/env python
()
t0 = (0,);
t1 = (0, 'Ni', 1.2, 3);
```

5.1. ¿Para qué sirve tener tipos inmutables y mutables

La principal ventaja de contar con ambos tipos es la flexibilidad de unos y las ventajas que proporciona contar con la garantía de que un elemento no se va a poder modificar.



5.2. Índices. Las mejoras de Python

Como hemos comentado a lo largo de este documento, Python incorpora mejoras al acceso a elementos en listas, tuplas y otro tipo de colecciones. Estos son nuevos operandos:

- `array[1:]` : Retorna una lista con los elementos desde el 1.
- `array[:4]` : Retorna una lista con los elementos hasta el 4.
- `array[1:3]` :Lista con los elementos desde 1 a 3 (no inclusive)
- `array[-1]` : Último elemento
- `array[:]` : Todos los elementos

tos del array

5.3. Ficheros

Para trabajar con un fichero es necesario almacenar en una variable un *descriptor* del mismo. Existen las funciones `open()`, `read()` o `write()` para manipular archivos de forma sencilla, así como métodos más específicos, como `readline()`

6. Condicionales

Un programa generalmente está compuesto por secuencias de control de flujo y repetición. En Python la primera categoría está protagonizada por la cláusula `if`. Ejemplo de una estructura condicional en Python:

```
if eggs :  
    buy ( 1 );  
elif milk  
    buy ( 6 );  
else :  
    buy ( 0 );
```

Importante: no existe la sentencia `switch` Esta característica, que a muchos procedentes de lenguajes como Pascal o C no gustará, es fácilmente reemplazable por un diccionario:

```
elegir='jamon';  
print{ 'leche': 1.25,  
        'jamon':1.99,  
        'huevos':0.99,  
        'bacon':1.10}[elegir]
```

6.1. Reglas de sintaxis

- Las sentencias se ejecutan una tras otra, a menos que se especifique lo contrario.
- Las sentencias y los bloques son detectados automáticamente: no se utilizan corchetes o cualquier otro tipo de delimitador. El indentado del código es lo que determina la pertenencia o no a un bloque.
- Líneas en blanco, espacios y comentarios son generalmente ignorados: especialmente en archivos. A la hora de trabajar con cadenas de caracteres se respetan. Los comentarios (marcados por un '#' inicial) siempre son ignorados.

7. Bucles

While:

```
while x :  
    print x,  
    x = x[1:];
```

Herramientas de control: **break**, **continue**, **pass** y **else**:

Referencias

[The Zen of Python, 2004] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.