Introducción a la Programación con Python 3

Miguel A. Astor

Ecoanova 2019

Agenda

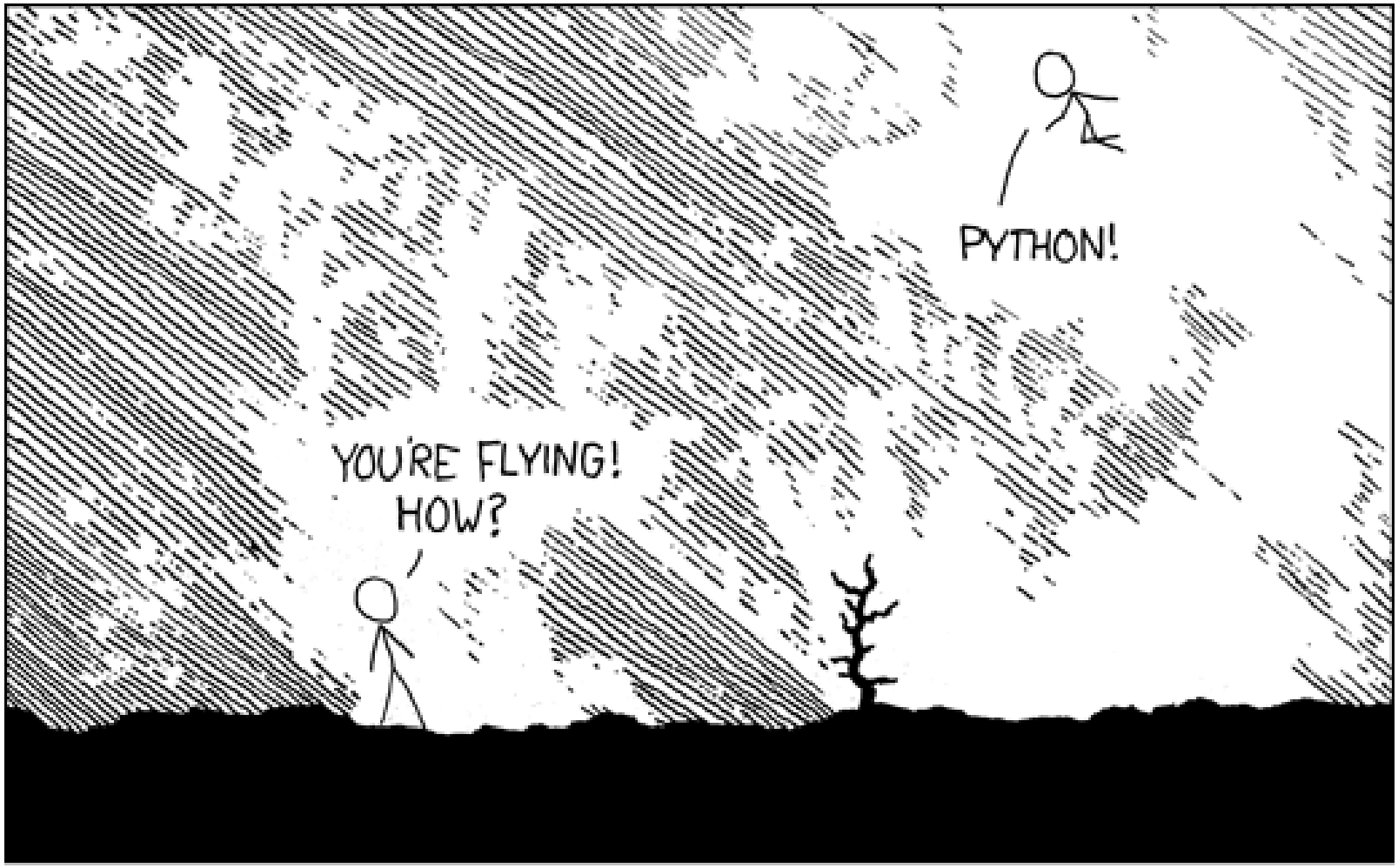
Introducción

¿Que es Python?

Python es un lenguaje de programación:

- Interpretado
- De tipos dinámicos
- Indentado
- Multi-paradigma:
  - Procedimental
  - Orientado a objetos
  - Funcional
- Con “baterías incluidas”

¿Por que Python?



¡Python Hace que Programar sea Divertido Otra Vez!

Breve historia de Python

- Creado por Guido van Rossum
  - Publicado en 1991
- En el año 2000 se publica Python 2 (actualmente 2.7)
- En el 2008 se publica Python 3 (actualmente 3.8)

Guido van Rossum

Intérpretes de Python

Existen múltiples implementaciones de Python para varios dominios:

CPython	Implementación de referencia
Jython	Python sobre la JVM
IronPython	Integración con .NET
PyPy	Python en Python
Pyjs	Traductor de Python a JavaScript
MicroPython	Python como SO para microcontroladores

El interprete de Python

El interprete de CPython puede ejecutar código de forma interactiva con un ciclo REPL (*Read-Eval-Print Loop* - Ciclo de Lectura, Evaluación y Ejecución) o cargar y ejecutar código desde *scripts*.

- En Unix se ejecuta con los comandos python, python3 o python3.8

Instrucciones y expresiones

En python se distingue entre declaraciones y expresiones:

- Las declaraciones se ejecutan
  - `a = 9`
  - `while x < 89:`
  - `pass`
  - `l.append(5)`
- Las expresiones se evalúan
  - `5 + 4`
  - `89 is not None`
  - `24 & 0x04`
  - `a`

El sistema de tipos

Los datos en Python poseen las siguientes propiedades:

- Todo es un objeto.
  - Incluso las clases son objetos (!)
- Los identificadores no tienen un tipo definido
  - Un identificador es solo un nombre para una región de memoria
  - Esto se conoce como *duck-typing*

Tipos de datos elementales

Python posee los siguientes tipos atómicos:

Tipo	Descripción
`int`	Enteros de 32 bits con signo
`long`	Enteros de precisión arbitraria con signo
`float`	Reales de coma flotante de 64 bits
`complex`	Números imaginarios (dos `float` de 64 bits)

Además posee los siguientes tipos complejos:

Tipo	Descripción
`str`	Cadenas de caracteres ASCII o UNICODE
`list`	Lista ordenada de datos heterogeneos
`tuple`	Agrupación ordenada de datos heterogeneos
`dict`	Tabla de entradas clave-valor
`set`	Conjunto desordenado de elementos no repetidos
`frozenset`	`set` inmutable

Tipos de datos elementales

Existen dos tipos atómicos especiales además de los mencionados:

Booleano Puede ser `True` o `False`

None Indica una referencia que no esta asociada a un objeto

Funciones predefinidas

Se definen las siguientes funciones básicas en el lenguaje, entre muchas otras:

Función	Acción
`print`	Escribe a la salida estándar
`input`	Lee de entrada estándar
`type`	Retorna el tipo de un dato
`len`	Retorna la longitud de una secuencia
`dir`	Lista los métodos de un objeto
`str`	Convierte un objeto a `str`
`repr`	Convierte un objeto a un `str` evaluable*
`open`	Abre un archivo
`eval`	Evalúa un `str`
`help`	Muestra documentación

* Si puede

Funciones predefinidas

Función	Acción
`map`	Aplica una función a un grupo de iterables
`range`	Genera números en un rango
`zip`	Genera `tuple`'s concatenando iterables

Entrada/Salida estándar

Para leer de entrada estándar se utiliza la función `input`, y para escribir se utiliza la función `print`.

Input

Argumentos Un `str` que se imprime como *prompt* al usuario

Retorno Los datos leidos como un `str`

Print

Argumentos Un dato a imprimir. Se convierte a `str` automáticamente

Manejo de archivos

Los archivos se abren con la función `open`.

Open

Argumentos El nombre del archivo y el modo de lectura como `str`

Retorno Un objeto archivo

Manejo de archivos

Una vez abierto, un archivo es un objeto con los siguientes métodos:

`read(x)`	Lee X caracteres del archivo. Sin argumentos lee todo el archivo
`write(x)`	Escribe el dato X al archivo. Este se convierte a `str` automáticamente
`readline()`	Lee una sola linea del archivo
`flush()`	Vacía las escrituras a disco
`seek(x)`	Mueve el apuntador de lectura/escritura
`tell()`	Indica donde se encuentra el apuntador de lectura/escritura
`close()`	Cierra el archivo

Manejo de archivos

Para no olvidar el cerrar los archivos, estos se pueden trabajar de la siguiente forma:

```
with open(archivo, modo) as nombre:
    ...
```

Una vez que el control deje este bloque, el archivo abierto se cerrará automáticamente.

Programación estructurada

En Python existen las siguientes estructuras de control:

`if`	Estructura condicional
`for`	Iteración sobre contenedores
`while`	Ciclo con condición
`def`	Declara funciones
`return`	Retorna de una función
`yield`	Retorna de un generador

Para la sintaxis de declaración y uso de funciones véase el ejemplo `basics.py`.

Programación estructurada

La estructura `if` tiene las siguientes formas:

Forma 1

```
if condición:
    ...
elif condición:
    ...
else:
    ...
```

Forma 2

```
expresión if condición else expresión
```

Programación estructurada

La estructura `for` itera sobre listas, tuplas o generadores.

```
for identificador in secuencia:
    ...
```

La estructura `while` verifica una condición.

```
while condición:
    ...
```

Programación orientada a objetos

Python permite declarar clases e instanciarlas:

```
class Nombre(superclase):
    ...
    def método(self, ...):
        ...
    ...
```

Todo método de una clase debe tener la palabra `self` como primer argumento.

Programación orientada a objetos

Para construir una objeto se utiliza el nombre de su clase como constructor:

```
class Nombre(superclase):
    ...

objeto = Nombre(...)
```

Para definir el constructor se debe declarar el método `__init__`:

```
class Nombre(superclase):
    ...
    def __init__(self, param1, param2, ...):
        self.param1 = param1
        self.param2 = param2
        ...
    ...
```

Programación orientada a objetos

El constructor de una superclase se utiliza la siguiente sintaxis:

```
class Nombre(superclase):
    ...
    def __init__(self, param1, ...):
        super(superclase, self).__init__(param1, ...)
    ...
```

Programación orientada a objetos

Las clases pueden tener atributos de clase:

```
class Nombre(superclase):
    atributo1 = valor
    atributo2 = valor
    ...
```

Programación orientada a objetos

Las clases en Python poseen las siguientes propiedades:

- Toda clase hereda de `object` o de una subclase de `object`
- La herencia es simple
- Todos los métodos son virtuales
- No existen métodos o atributos privados
  - Por convención, si el nombre de un método o atributo comienza con "_", debe considerarse "privado".
- Las clases también son objetos

Funciones anónimas

La estructura `lambda` permite crear funciones sin nombre:

```
lambda param1, param2, ...: expresión
```

Las funciones anónimas pueden guardarse como variables, pasarse como parámetros de otras funciones, y por supuesto, evaluarse.

Funciones de orden superior

Una función que recibe una o más funciones se conoce como una función de orden superior. Python posee la siguientes funciones de orden superior predefinidas

```
map(función, *iterables) -> generador
zip(*iterable) -> generador
```

Generadores

Los generadores son objetos que crean o iteran sobre elementos.

```
def generador(...):
    ...
    yield elemento
    ...
```

Se pueden utilizar para crear iteradores de colecciones

```
class Contenedor(superclase):
    ...
    def __iter__(self):
        ...
        yield elemento
        ...
    ...
```

Cortes y compreción de listas

Las listas pueden indexarse por posición (se permiten indices negativos).
Las listas también pueden ser utilizadas por cortes a partir de una determinada posición:

```
lista[x:y]
lista[x:]
lista[:y]
```

La comprensión de listas permite crear listas mediante generadores:

```
lista = [expresión for variable in generador \
         if condición sobre variable]
```

Importando módulos

Python incluye una muy extensa biblioteca estándar.
Para importar módulos se utilizan las siguientes declaraciones:

```
import módulo
import módulo as nombre
from módulo import elemento
from módulo import elemento as nombre
```

Generación de números aleatorios

El módulo `random` incluye varios métodos para generar números aleatorios y funciones relacionadas:

```
import random as r

r.random()
r.randint(0, 100)
r.randrange(5, 100)
r.choice([1, 2, 3, 4])
r.shuffle([1, 2, 3, 4])
```

Interacción con el entorno y el sistema operativo

Para esta tarea Python define los módulos `sys` y `os`.

- El módulo `sys` da información sobre el entorno de ejecución de python
  - Posee un atribto `argv` que contiene los argumentos de linea de comandos como una lista
- El módulo `os` provee métodos para solicitar servicios al sistema operativo
  - Entrada/salida de archivos de bajo nivel
  - Manipulación de directorios
  - Manipulación de rutas
  - Planificación del proceso
  - Otras llamadas al sistema

Codificación y decodificación de texto JSON

JSON (*JavaScript Object Notation* - Notación de Objetos JavaScript) es un lenguaje que permite representar estructuras de datos con una sintaxis similar a la de JavaScript. Este lenguaje es ampliamente utilizado en la Web para compartir datos.
Python puede codificar `dict`'s a texto en sintaxis JSON y viceversa con el módulo estándar `json`:

```
import json

d = {...}
text = json.dumps(d)
d = jsom.loads(text)
```

Bases de datos SQLite 3

Si el intérprete fue compilado con soporte para SQLite, entonces se pueden manipular bases de datos con la biblioteca estándar. Véase el ejemplo db.py.

Clientes y servidores HTTP simples

Para realizar solicitudes a servidores HTTP, Python incluye el módulo `urllib2`. Su funcionamiento básico es:

```
request  = urllib2.Request(URL)
response = urllib2.urlopen(request)
```

La biblioteca estándar también incluye un servidor HTTP básico. Véase el ejemplo `http_server.py`

Tópicos adicionales

Extensión del lenguaje

Python puede usar módulos externos escritos en C/C++

Scripting de aplicaciones

Con #include <python.h>

Programación paralela

Creación de hilos, subprocesos y comunicación entre procesos

Interfaces gráficas

Con el módulo Tkinter de la BE

Expresiones regulares

Similares a las del lenguaje Perl

Conclusiones

- Python es un lenguaje facil de aprender y utilizar pero sumamente poderoso
- La biblioteca estándar de Python es muy completa y contiene módulos para desarrollar tareas muy complejas con solo la instalación base del lenguaje
- El ecosistema de herramientas de Python es tan grande como su comunidad de usuarios

Dos últimos ejercicios

1. Escriba `import this` en el intérprete
2. Lea el texto impreso en la pantalla y razone sobre su significado
3. Ejecute `import antigravity`
4. Diviértase ;)

Contactos

Prof. Miguel A. Astor

- mastor89@protonmail.com

¿Donde conseguir esta presentación y ejemplos?

- `https://github.com/miky-kr5/Presentations`

¿Preguntas?