netmind

WeKnowIT

Roche

Python

INTRODUCCIÓN A PYTHON

1. Características y propiedades del lenguaje
2. Cómo desarrollar código Python de calidad
3. IDE
4. Librería estándar
5. Tipos de datos
6. Estructuras de control de flujo, bucles
7. Listas
8. Archivos
9. Expresiones regulares

1 Características y propiedades del lenguaje

¿Qué es Python?

- Python es un lenguaje de programación orientado a objetos creado en 1989
  - 1.0 - 1994
  - 2.0 - 2000
  - 3.0 - 2008
  - 3.4 - 2014
  - 3.5 - 2016
  - 3.6 - 2017
  - https://www.python.org/

Características del lenguaje

- Interpretado (también se puede compilar)
- Código legible (compárar con Perl por ejemplo)
- Multiparadigma: permite crear programas con más de un estilo de programación
- Programación Orientada a objetos
- Programación Imperativa
- Programación Funcional
- Tipado Dinámico (muérete Java)
- Multiplataforma (verdad .NET??)
- Prototipado Rápido (como en C no?)

Instalación

- Ubuntu 16.04: https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-16-04-server
- Windows: https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-windows-10
  - https://www.python.org/downloads/release/python-362/
- MacOSX: https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-macos

Instalando Python

- Accede al sitio:
  - https://www.python.org/downloads/release/python-362/
- Descarga e instala Python 3.6
- Añade el directorio de python al path
- Añade el directorio scripts al path
- Abre un terminal y escribe:
  - python --version

2 Cómo desarrollar código Python de calidad

El Zen de Python

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente. A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés. (Obviamete el tipo era holandés...)
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Guía de estilo PEP8

- Un PEP (Python Enhancement Proposal) es un documento de diseño que proporciona información a la comunidad de Python o describe una nueva característica para Python o sus procesos o entorno.
- El PEP debe proporcionar una especificación técnica concisa de la característica y una justificación para la característica.
- PEP8: es la guía de estilo está definida en:
  - https://www.python.org/dev/peps/pep-0008/
  - Proporciona convenciones de codificación para el código Python que comprende la biblioteca estándar en la distribución principal de Python.

Principios SOLID

- Publicados por Robert C. Martin, en su libro Agile Software Development: Principles, Patterns, and Practices
- Se trata de cinco principios de diseño orientado a objetos que nos ayudarán a crear mejor código, más estructurado, con clases de responsabilidad más definida y más desacopladas entre sí:
  - **S**ingle Responsibility: Responsabilidad uńica.
  - **O**pen/Closed: Abierto/Cerrado.
  - **L**iskov substitution: Sustitucioń de Liskov.
  - **I**nterface segregation: Segregacioń de interfaz.
  - **D**ependency Inversion: Inversioń de dependencia.

3 IDEs

IDE's

- Eric: https://eric-ide.python-projects.org/
- Pycharm: https://www.jetbrains.com/pycharm/
- LiClipse (eclipse plugin): http://www.liclipse.com/

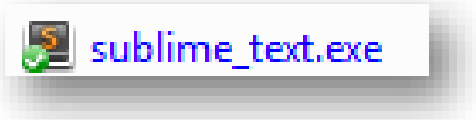
- Sublime 3: https://www.sublimetext.com/3

Instalando Sublime 3

Descargar y ejecutar sublime Text

- Acceder a: https://www.sublimetext.com/3
- Escoger la versión portable → descargar

- Descomprimir
- Acceder al directorio Sublime
- Ejecutar



AddOns Sublime

- Package Control
  - https://packagecontrol.io/installation
- Anaconda
  - https://realpython.com/blog/python/setting-up-sublime-text-3-for-full-stack-python-development/
  - Ctrl+shift+p
  - Package control: Install package
  - Anaconda
  - http://damnwidget.github.io/anaconda/
- Emmet
  - http://docs.emmet.io/cheat-sheet/

El intérprete

- El intérprete nos va a permitir ir ejecutando comandos de manera interactiva es ir viendo los resultados de cada sentencia
  - Python
  - >>>

4 Librería estándar

Biblioteca Estándar

- Son un conjunto de módulos algunos hechos en C y otros en Python que se incorporan al intérprete
- Incluye una serie de funcionalidades comunes que no se necesitan importar
  - https://docs.python.org/3.6/library/index.html

Biblioteca Estándar

- Funciones incorporadas:
  - https://docs.python.org/3.6/library/functions.html

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Biblioteca Estándar

Constantes predefinidas:

- https://docs.python.org/3.6/library/constants.html

- False
- True
- None
- NotImplemented
- Note
- Ellipsis
- __debug__

Creando el primer programa Python

Comenzando a programar

- Creamos un nuevo proyecto (carpeta)
- Creamos un nuevo fichero main.py
- Añadimos el contenido:

```
print("Hola Mundo!")
```

- Abrimos una consola de terminal en el directorio y ejecutamos el código

```
python main.py
```

5 Tipos de datos

Tipos de Datos Básicos

- String
  - "Esto es un string“
  - 'Esto es un string‘
- Booleano (False, True, None)
  - Operaciones Booleanas :
  - and, or, not
- Comparaciones
  - <
  - <=
  - >
  - >=
  - ==
  - !=
  - is #identidad
  - is not #no es idéntico

Tipos de Datos Básicos

- Tipos numéricos
  - int
  - float
  - complex
- Operadores Aritméticos
  - x + y
  - x - y
  - x * y
  - x / y
  - x // y división entera
  - x % y

Tipos de Datos Básicos

- Funciones aritméticas
  - abs(x)
  - int(x)
  - float(x)
  - complex(re, im)
  - c.conjugate() conjugar un número complejo c
  - divmod(x, y) (x // y, x % y)
  - pow(x, y) x elevado a y
  - x ** y x elevado a y

Tipos de Datos Básicos

- Funciones manejo de bits
  - x | y bitwise or of x and y
  - x ^ y bitwise exclusive or of x and y
  - x & y bitwise and of x and y
  - x << n x shifted left by n bits (1)(2)
  - x >> n x shifted right by n bits (1)(3)
  - ~x the bits of x inverted

Formateando Strings

- Se logra usando el método str.format()
  - https://docs.python.org/3/library/string.html#format-string-syntax
- Se puede insertar en el string el texto a reemplazar entre {}

  'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')

- En la versión 2, se usaba el operador %
  - '%d' % (42,)
  - '{:d}'.format(42)
- Más información en https://pyformat.info/

Formateando Strings

- Existen un conjunto de flags de conversión
  - '!s' llama a str() sobre el valor
  - '!r' llama a repr()
  - '!a' llama a ascii()
- Ejemplos

```
"Harold's a clever {0!s}"        # Calls str() on the argument first
"Bring out the holy {name!r}"    # Calls repr() on the argument first
"More {!a}"                      # Calls ascii() on the argument first
```

Formateando Strings

Tuplas

- Las tuplas son un nuevo tipo de dato que permite la agrupación de datos
  - http://es.diveintopython.net/odbchelper_tuple.html

- Es una lista inmutable

  tup1 = (50, "Hola")

- Se utiliza principalmente para devolver más de un dato en las funciones
  - Acceder a un valor:

  tup1[0]

  - Borrar una tupla:

  del tup1

  - Sumar tuplas:

  tup3 = tup1 + tup2

Listas

- Son otro tipo de secuencias en Python, semejante a un array
  - http://es.diveintopython.net/odbchelper_list.html
- Se manejan de una manera similar a un array
- Declaración:

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = list(range(4,10,2))
```

- Acceso:
  - list1[2]=27
  - list1[2] # Busca desde el principio (0)
  - list1[-2] # Busca desde el final
  - list1[1:] # Permite rango
  - list1[1:3] # Permite rango
- Borrado:

```
del list[2]
```

Listas

- Acciones sobre listas:
  - len([1, 2, 3]) #Longitud
  - [1, 2, 3] + [4, 5, 6] # Concatenación
  - ['Hi!'] * 4→ ['Hi!', 'Hi!', 'Hi!', 'Hi!'] # Repetición
  - 3 in [1, 2, 3] → True # Búsqueda
  - for x in [1, 2, 3]: print x, → 1 2 3 # Iteración

Diccionarios

- Son listados de conjuntos de clave-valor (un hash)

  dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

- Acceso y Actualización :

  dict['Name']

- Borrado:

  del dict['Name']

- Longitud:

  len(dict)

Jugando con listas, tuplas y diccionarios

6 Estructuras de control de flujo, bucles

Estructuras de Control

- Existen las siguientes expresiones:
  - If
  - if else
  - if elif else
  - while
  - for

  - No hay switch

Estructuras de Control: if

```
var1 = 100
if var1:
   print ("1 - Got a true expression value")
```

Estructuras de Control: if else

```
if var1:
   print ("1 - Got a true expression value")
   print (var1)
else:
   print ("1 - Got a false expression value")
   print (var1)
```

Estructuras de Control: if elif else

```
if var == 200:
   print ("1 - Got a true expression value")
   print (var)
elif var == 150:
   print ("2 - Got a true expression value")
   print (var)
elif var == 100:
   print ("3 - Got a true expression value")
   print (var)
else:
   print ("4 - Got a false expression value")
   print (var)
```

Estructuras de Control: while

```
count = 1
while (count < 9):
    print ('The count is:', count)
    count = count + 1
```

Estructuras de Control: while

```
count = 0
while count < 5:
   print (count, " is  less than 5")
   count = count + 1
else:
   print (count, " is not less than 5")
```

Estructuras de Control: for

```
for letter in 'Python':     # First Example
   print ('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
    print ('Current fruit :', fruit)
```

Estructuras de Control: for

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print ('Current fruit :', fruits[index])
```

Estructuras de Control: for

```
for num in range(10,20):  #to iterate between 10 to 20
   for i in range(2,num): #to iterate on the factors of the number
      if num%i == 0:      #to determine the first factor
         j=num/i          #to calculate the second factor
         print ('%d equals %d * %d' % (num,i,j))
         break #to move to the next number, the #first FOR
   else:                  # else part of the loop
      print (num, 'is a prime number')
```

Algo más complejo

- Crea un script que recorra una lista con los meses el año y los muestre por pantalla
  - En orden ascendente
  - En orden descendente
  - De 3 en 3
  - Mientras su número equivalente sea un número primo

7 Listas

Comprensión de Listas

- Veamos un ejemplo de construcción de un listado con este sistema

- Celsius = [39.2, 36.5, 37.3, 37.8]
- Fahrenheit = [((float(9)/5)*x + 32) for x in Celsius]

- lista = [(x,y,z) for x in range(1,30) for y in range(x,30) for z in range(y,30) if x**2 + y**2 == z**2]

Generando listas

- Genera una lista con los 10 primeros números sumatorios
  - sumatoria(1) = 1
  - ....
  - sumatoria(9) = 1+2+3+4+5+6+7+8+9 = 45
  - sumatoria(10) = 1+2+3+4+5+6+7+8+9+10= 55

8 Archivos

Archivos

- Python permite trabajar con archivos de una manera muy sencilla
  - https://docs.python.org/3/tutorial/inputoutput.html
- Para manejar los archivos necesitaremos la ruta y el modo de apertura
  - open(filename, mode)
  - mode:
    - 'r': solo lectura
    - 'w' sólo escritura (un archivo existente con el mismo nombre se borrará)
    - 'a' abre el archivo para añadir; los datos escritos en el archivo se añaden automáticamente al final.
    - 'r+' abre el archivo tanto para lectura como para escritura.
    - 'b' añadido al modo abre el archivo en modo binario: los datos se leen y escriben en forma de bytes de objetos.
- Escritura:

```
cadena="mi texto"
fobj_out = open("ad_lesbiam.txt","w")
fobj_out.write(cadena)
fobj_out.close()
```

Archivos: Lectura

- Lectura

```
fobj_in = open("ejemplo.txt")
for line in fobj_in:
    print line.rstrip()
fobj_in.close()
```

- En ambos casos es necesario abrir los archivos y cerrarlos

Archivos: lectura

- Como listado

```
lineas= open("sample.txt").readlines()
```

- Como string

```
texto = open("sample.txt").read()
print texto[16:34]
```

Borrar un archivo

- Se usa el método remove o unlink de la librería os
  - os.remove(path, *, dir_fd=None)

```
import os
os.remove("/path/<file_name>.<txt>")
```

  - Primero comprobar que es un archivo
    - os.path.isfile("/path/to/file")

- Para borrar directorios (vacíos)

  os.rmdir(path, *, dir_fd=None)

Leyendo y escribiendo archivos

- Escribe un script que copie el contenido de un archivo “sample.txt” en otro “new.txt”
- A continuación muestre el contenido de new.txt línea a línea
- Finalmente borre new.txt

- Haz lo mismo para una imagen

9 Expresiones regulares

Expresiones Regulares

- Las expresiones regulares nos van a permitir realizar matches entre distintos elementos para saber si cumplen o no con un determinado patrón
- Python posee una librería específica para ello: re
  - https://docs.python.org/3/library/re.html
- Por tanto debe ser importada con la orden import antes de usarla

```
import re
```

Expresiones Regulares

```
import re

if re.search("cat","A cat and a rat can't be friends."):
    print ("Some kind of cat has been found :-)")
else:
    print ("No cat has been found :-(")
```

Métodos de la librería

- re.search(pattern, string, flags=0)
- re.match(pattern, string, flags=0)
- re.fullmatch(pattern, string, flags=0)
- re.split(pattern, string, maxsplit=0, flags=0)
- re.findall(pattern, string, flags=0)
- re.finditer(pattern, string, flags=0)
- re.sub(pattern, repl, string, count=0, flags=0)
- re.escape(pattern)
- re.purge()

Expresiones Regulares

- Más información en
  - http://www.python-course.eu/re.php

Trabajando con expresiones regulares

- Escribe un script que lea un archivo “sample.txt” e identifique el número de líneas donde aparece un texto determinado (almacenado en una variable)

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es