


WALC UTEC 2017

 **AGAMOS LA DIFERENCIA**



*Universidad Tecnológica
de El Salvador*





1. Fundamentos de Python - pt1

Una breve introducción al lenguaje, sintaxis y uso

OFlores - RCriollo - MZennaro

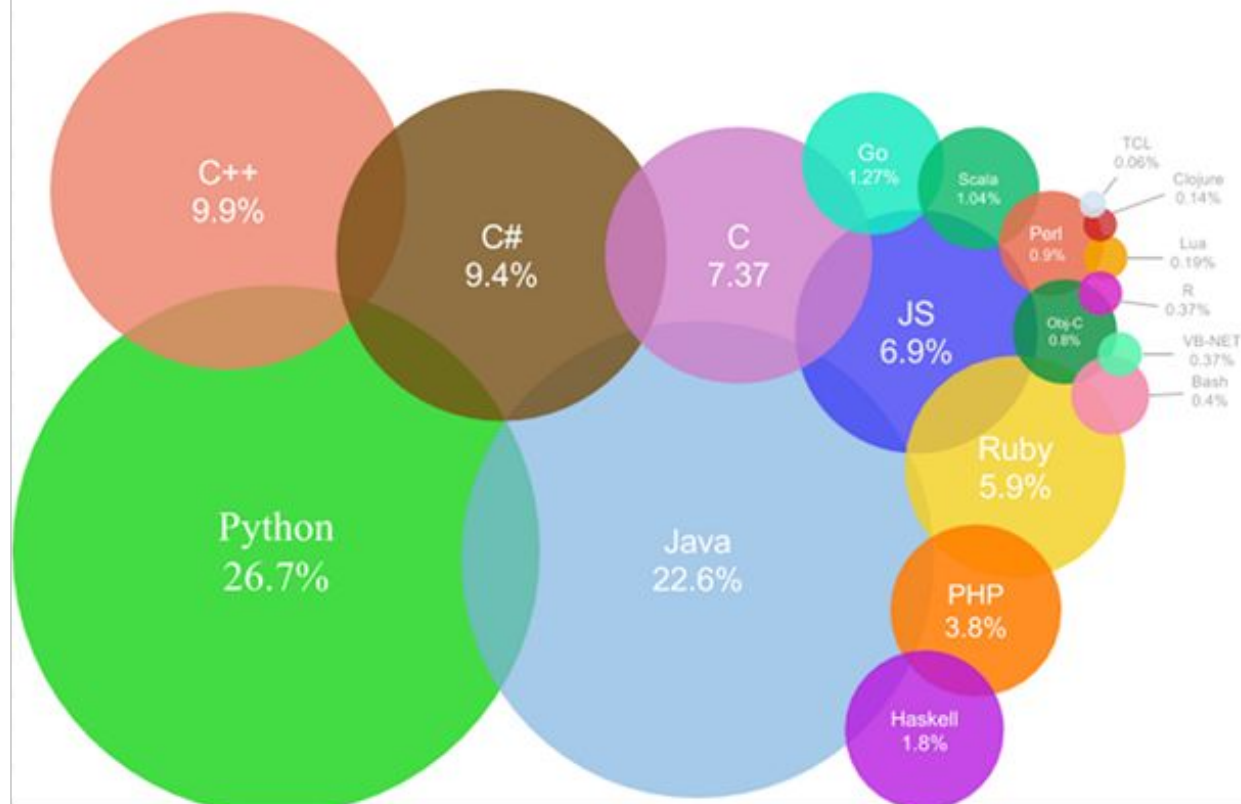
TRACK 6 - IoT - WALC 2017
San Salvador - El Salvador

Contenido



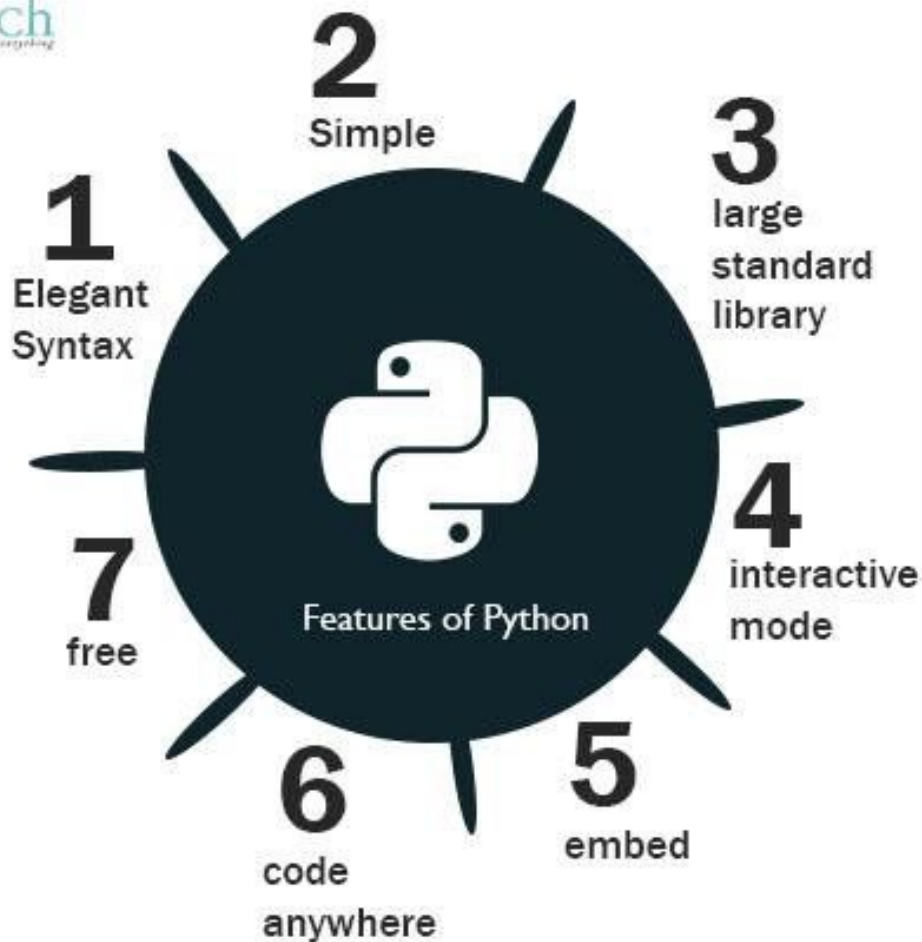
- Conceptos básicos
- Características del lenguaje
- Instalación - Anaconda & Jupiter
- Aspectos de Sintaxis
- Demostraciones y ejercicios

Most Popular Coding Languages of 2016



The 9 Most In-Demand Programming Languages 2017





Lenguaje PYTHON?

Python es un lenguaje de alto nivel e "interpretado dinámicamente", lo que significa que se ejecuta sobre la marcha sin la necesidad de ser compilado. Esto está en contraste con los lenguajes de programación de "bajo nivel", que primero deben convertirse en código máquina (es decir, compilados) antes de que puedan ejecutarse. Con Python, uno no necesita preocuparse por la compilación y solo puede escribir código, evaluarlo, repararlo, re-evaluarlo, etc. en un ciclo rápido, convirtiéndolo en una herramienta muy productiva.

web oficial: python.org

Python es un lenguaje de programación fácil de aprender y poderoso. Tiene estructuras de datos de alto nivel eficientes y un enfoque simple pero efectivo para la programación orientada a objetos. La elegante sintaxis y el tipado dinámico de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para el desarrollo de scripts y de aplicaciones rápidas en muchas áreas en la mayoría de las plataformas.



Características notables de Python:




- Utiliza una sintaxis elegante, lo que facilita la lectura de los programas que escribe.
- Ideal para el desarrollo de prototipos y otras tareas de programación ad-hoc - placas como LoPy
- Se ejecuta en cualquier lugar, incluidos Mac OS X, Windows, Linux y Unix - placas para IoT
- Es software libre en dos sentidos. No cuesta nada descargar o usar Python, o incluirlo en su aplicación. Python también se puede modificar y redistribuir libremente, porque si bien el idioma está protegido por derechos de autor, está disponible bajo una licencia de código abierto.
- Hay una variedad de tipos de datos básicos disponibles: números (enteros largos de punto flotante, complejo y de longitud ilimitada), cadenas (tanto ASCII como Unicode), listas y diccionarios.
- Python es compatible con la programación orientada a objetos con clases y herencia múltiple.
- El lenguaje admite el aumento y la captura de excepciones
- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción

Características notables de Python:



- Los archivos de python tienen la extensión .py.
- Son archivos de texto que son interpretados por el compilador.
- Para poder ejecutar programas en Python, necesitamos en primer lugar el intérprete de python, y en segundo lugar el código que queremos ejecutar.
- Python dispone de multitud de “plugins” (llamados habitualmente módulos, paquetes, bibliotecas o librerías) desarrollados por terceros, que permiten realizar multitud de funciones, tales como descargar vídeos de youtube, navegar por internet, dibujar gráficos, etc.
- Python es “case sensitive”, lo que quiere decir que diferencia mayúsculas de minúsculas, y por lo tanto python y Python, no son lo mismo.
- La agrupación de instrucciones se hace por sangría en vez de llaves de apertura y cierre
- No es necesario declarar variables ni argumentos.

como se ve un programa en Python?



```
REFRAIN = '''
%d bottles of beer on the wall,
%d bottles of beer,
take one down, pass it around,
%d bottles of beer on the wall!
'''
bottles_of_beer = 99
while bottles_of_beer > 1:
    print REFRAIN % (bottles_of_beer, bottles_of_beer, bottles_of_beer - 1)
    bottles_of_beer -= 1
```

Instalar Python

La última versión de Python es la 3, pero tiene el problema de que no es compatible del todo con los programas escritos para la versión 2.

Python con Anaconda

Vamos a usar una distribución de Python gratuita llamada Anaconda.

Aunque está disponible sólo en inglés, esta distribución incluye multitud de utilidades que nos facilitarán el trabajo y es ideal para empezar. Para instalarla, descargar la versión de Anaconda para tu sistema operativo desde su página web: <https://anaconda.org>



Anaconda Navigator

File Help

ANACONDA NAVIGATOR

Upgrade Now Sign in to Anaconda Cloud

Home

Environments

Projects (beta)

Learning

Community

Documentation

Developer Blog

Feedback

Twitter YouTube GitHub

Create Clone Import Remove

Search Environments

root

py4data-2.7

Create new environment

Environment name New environment name

☒ Python ☐ R

Python version 3.6 3.5 2.7

Installed

Channels Update index... Search ...

Name	T	Description	Version
✓ _license			1.1
✓ alabaster		Configurable, python 2+3 compatible sphinx theme.	0.7.10
		Simplifies package management and deployment of anaconda	4.4.0
		Anaconda.org command line cli...	1.6.3
		Reproducible, executable project directories.	0.6.0
			0.22.0
		A abstract syntax tree for python with inference support.	1.4.9
		Community-developed python li...	1.3.2
		Utilities to internationalize and localize python applications	2.4.0
✓ backports			1.0

190 packages available (C:\Users\omar.flores\AppData\Local\Continuum\Ar

Python en Jupyter

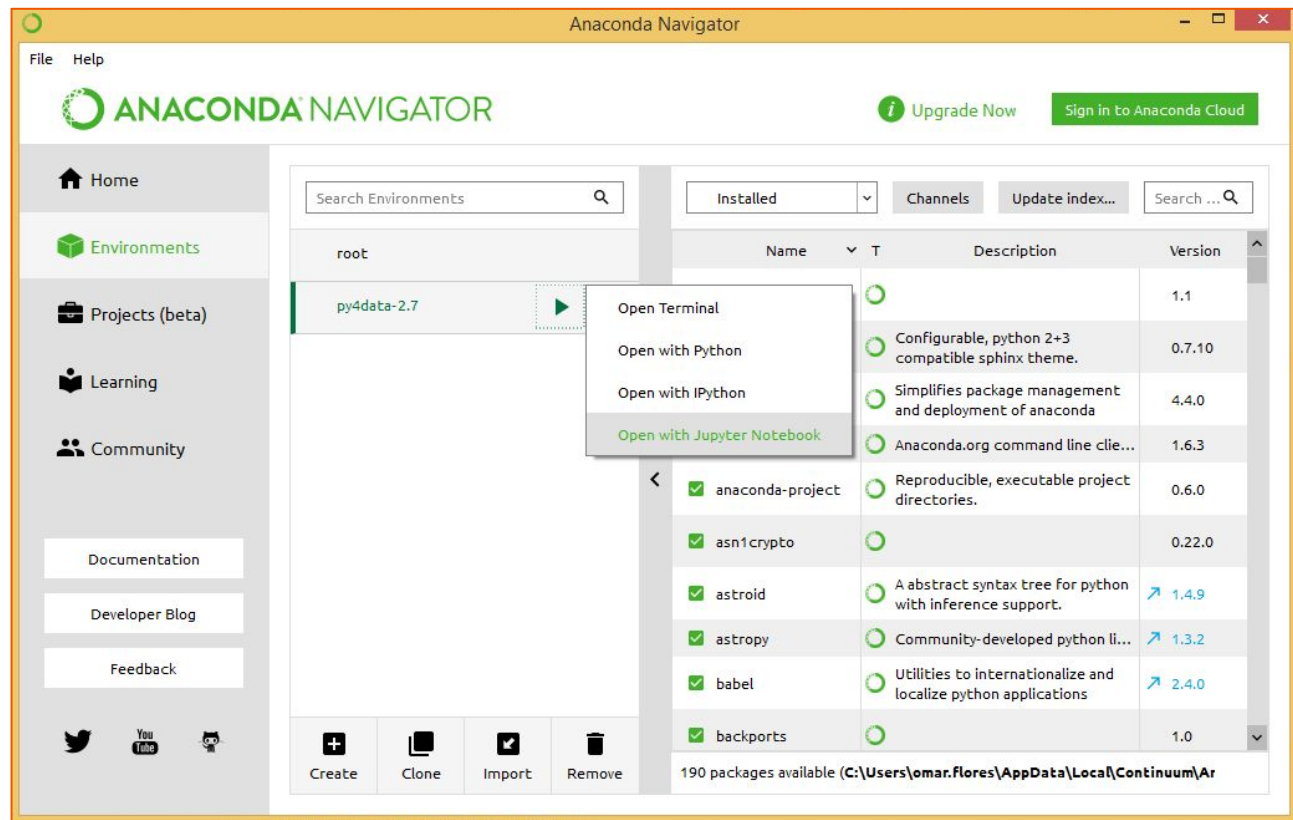
Después de instalar la distribución de Anaconda, ya podemos acceder también a los notebook de Jupyter.

El **Jupyter Notebook** es un entorno interactivo web de ejecución de código en los que, por ejemplo, puedes incluir gráficas que ayuden en el análisis e explicación de tus datos. Utilizados para facilitar la explicación y reproducción de estudios y análisis.

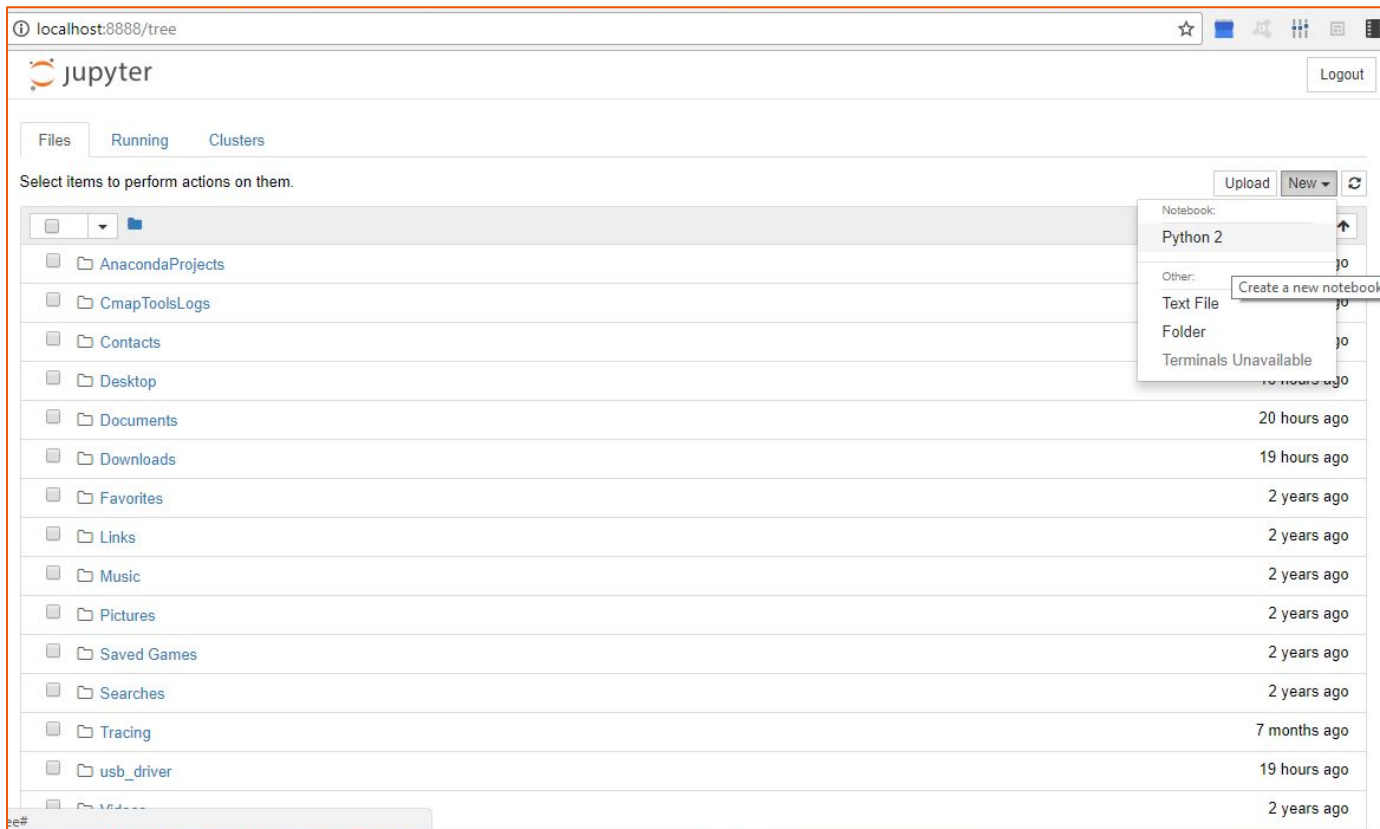
Para trabajar con ellos se realiza directamente desde el navegador. Estos notebook se pueden almacenar e intercambiar o mostrar en páginas web.



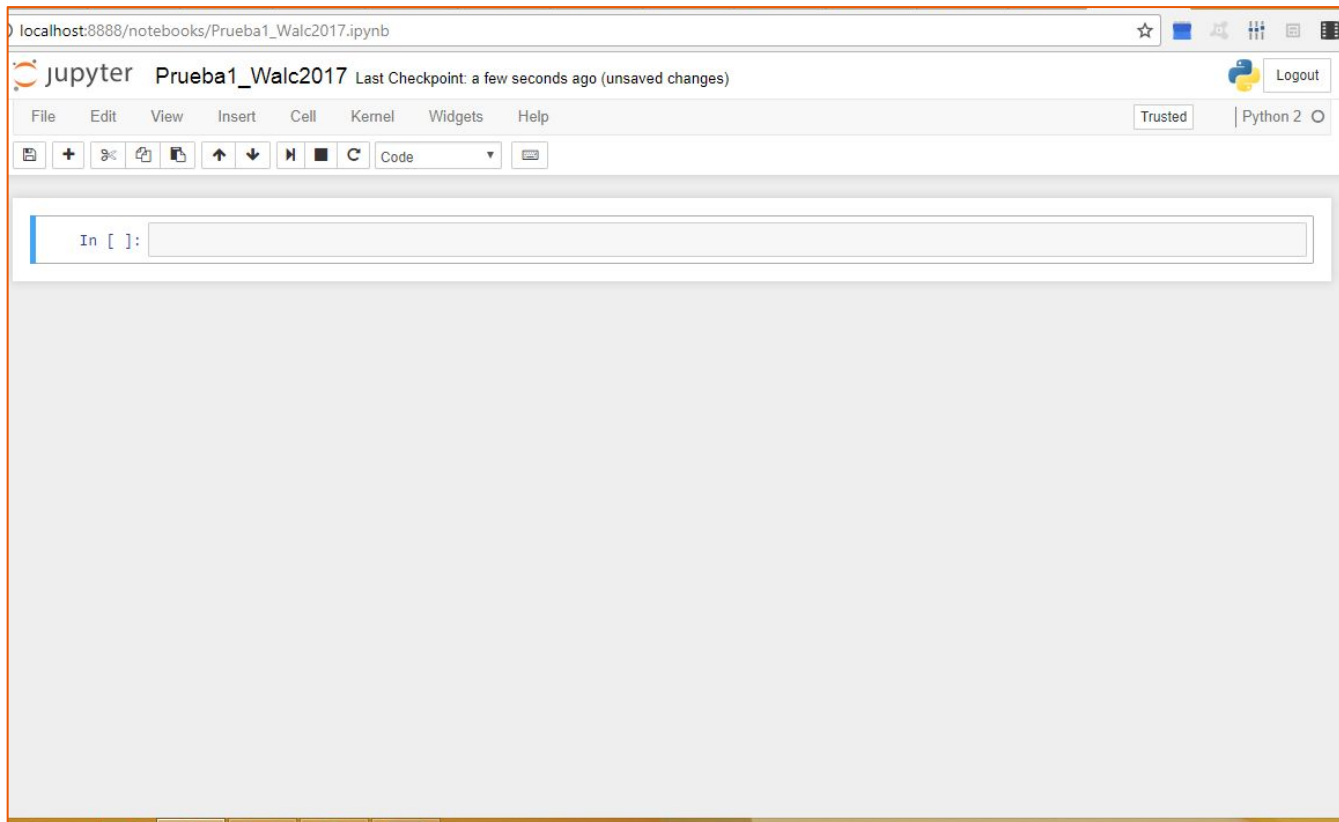
Pruébalo !



Pruébalo !



Pruébalo !



Pruébalo !



localhost:8888/notebooks/Prueba1_Walc2017.ipynb

jupyter

Prueba1_Walc2017

Last Checkpoint: 4 minutes ago (unsaved changes)

Python 2

Logout

FileEditViewInsertCellKernelWidgetsHelp

TrustedPython 2

Code

run cell, select below

In []:

```
REFRAIN = '''
%d bottles of beer on the wall,
%d bottles of beer,
take one down, pass it around,
%d bottles of beer on the wall!
'''

bottles_of_beer = 99
while bottles_of_beer > 1:
    print REFRAIN % (bottles_of_beer, bottles_of_beer, bottles_of_beer - 1)
    bottles_of_beer -= 1
```

Python Tutor !



<http://pythontutor.com>

VISUALIZE Python, Java, JavaScript, TypeScript, Ruby, C, and C++

Python Tutor, created by [Philip Guo \(@pgbovine\)](#), helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of source code.

Using this tool, you can write [Python 2](#), [Python 3](#), [Java](#), [JavaScript](#), [TypeScript](#), [Ruby](#), [C](#), and [C++](#) code in your web browser and visualize what the computer is doing step-by-step as it executes.

Over 3.5 million people in over 180 countries have used Python Tutor to visualize over 30 million pieces of code, often as a supplement to textbooks, lectures, and online tutorials.

[Start visualizing your code now](#) (or try [live programming](#))



Python Tutor !



Write code in Python 2.7

(drag lower right corner to resize code editor)

```
1 |
```

Print output (drag lower right corner to resize)

Frames

Objects

Probando diferentes Versiones !



Versión 2.X

```
print 'Hello, World!'
print('Hello, World!')
print "text", ; print 'print more text on the same line'
```

Versión 3.X

```
print('Hello, World!')

print("some text,", end="")
print(' print more text on the same line')
```

Probando diferentes Versiones !

Write code in Python 2.7 ▼

(drag lower right corner to resize code editor)

```
1 REFRAIN = '''
2 %d bottles of beer on the wall,
3 %d bottles of beer,
4 take one down, pass it around,
5 %d bottles of beer on the wall!
6 '''
7 bottles_of_beer = 99
→ 8 while bottles_of_beer > 1:
9     print REFRAIN % (bottles_of_beer, bottles_of_beer, bottles_of_beer-1)
10    bottles_of_beer -= 1
```

Elementos básicos de Python



De una manera muy informal, un programa de Python

"hace algo con cosas"

Las "cosas" son **objetos** y usted especifica qué quiere hacer con ellos usando declaraciones/instrucciones.

Python incorpora por defecto muchos tipos de objetos, pero se pueden crear tipos de objetos personalizados utilizando clases.

Objetos

Los objetos son esencialmente sólo piezas de memoria, con valores y conjuntos de operaciones asociadas. Como veremos, todo es un objeto en Python.

Elementos básicos de Python



Objetos.

Por ejemplo, para crear una cadena de caracteres "Internet de las cosas", simplemente escriba:

```
1 # expresion para crear un cadena de caracteres
2 'internet of things'
```

```
'internet of things'
```

```
1 # Este objeto es almacenado en memoria, con la siguiente direccion
2 hex(id('internet of things'))
```

```
'0x10f82be88'
```

Elementos básicos de Python

Objetos.

Cada objeto tiene un ****tipo**** que le define el tipo de objeto y las cosas que el programa puede hacer con objetos de ese tipo.

```
In [10]: 1 type('internet of things')
```

```
Out[10]: str
```

El tipo de objeto define que tipo de operaciones el objeto soporta, o, en otras palabras, qué operaciones se pueden realizar sobre el valor/dato de ese objeto.

Por ejemplo, los objetos de tipo **str** soportan operaciones como **uppercase**.

```
In [17]: 1 'internet of things'.upper()
```

```
Out[17]: 'INTERNET OF THINGS'
```


Elementos básicos de Python

Cuando se dice que todo en Python es un objeto, incluye el caso de las funciones. Una función será de tipo `function`. Las funciones son ciudadanos de primera clase.

```
In [49]: 1 # declaracion de una funcion
         2 def square(x):
         3     return x**2
         4
         5 type(square)
```

Out[49]: `function`

```
In [22]: 1 # para invocar o llamar la funcion, y obtener la potencia del numero 2
         2 square(2)
```

Out[22]: `4`

Python provee muchos tipos de datos <https://docs.python.org/3/library/stdtypes.html> como los que se ven a continuación:

Tipos de datos en Python



```
In [24]: 1 # Booleanos: Verdadero, Falso  
2 False
```

Out[24]: False

```
In [25]: 1 # Tipos Numericos: int, float, complex  
2 4.534876
```

Out[25]: 4.534876

```
In [26]: 1 # Tipos de secuencias de datos: str, list, tuple, range  
2 [1, 4, 6, 'spam'] # lista de objetos
```

Out[26]: [1, 4, 6, 'spam']

```
In [27]: 1 # Tipos mapa de datos / Mapping : dict  
2 {'name': 'Abdus', 'surname': 'Salam', 'year': 1979}
```

Out[27]: {'name': 'Abdus', 'surname': 'Salam', 'year': 1979}

...existen muchos más que se verán a medida se utilicen.

Expresiones en Python



Una **expresion** es una frase o linea de codigo que produce un valor. La linea de codigo mas simple contiene **literales** e **identificadores** (por ejemplo el nombre de una variable). Usando literales, identificadores, y operadores (arithmeticos, booleanos, comparacion, ...)...se pueden construir lineas o expresiones que generan nuevos objetos a partir de otras ya existentes.

```
1 # tres objetos
2 a = 2
3 b = 5.23
4 c = True
5
6 # Una expresion que genera un valor
7 a + b + c
```

8.23

Es de notar que el objeto booleano `c` a sido **forzado** automaticamente por el interprete de Python a su equivalente numero, 1.

Ahora que tenemos Objetos, tenemos que especificar que Hacer con ellos.

Sentencias en Python

En terminos simples, las sentencias son las cosas que escribimos para indicarle a Python lo que los programas deben hacer. Python es un lenguaje **procedimental**, basado en sentencias.

b.1 Sentencia de asignacion ► para crear referencias

Anteriormente se menciona, como crear objetos usando expresiones **literales**, sin embargo, si se desea utilizar esos objetos en una parte posterior dentro del nuestro script/programa, es necesario una forma de acceder a ellos, lo que llamaremos **referencia**. Una **referencia** es un nombre que refiere a la locacion en memoria del valor (object). Para crear tal referencia, se hace uso de una **sentencia de asignacion**. Con esto se **vincula** una **variable** (un nombre) que contiene una referencia al objeto. El nombre de referencia es a libre eleccion, pero debe ser algo significativo al objeto y que no sea una palabra reservada de Python.

```
1 help('keywords')
```

```
1 # una sentencia de asignacion
2 track6 = 'internet of things' # ligamos el nombre de variable track6 a la cadena
```

```
1 # ahora la variable/nombre track6
2 track6
```

```
1 # y la referencia al espacio en memoria
2 hex(id(track6))
```

Asignaciones en Python

```
1 # asignacion Tuple
2 name, surname, phone_number = ('John', 'Cheng', '56.46.67')
3 name
```

```
1 # de hecho No es necesario los parentesis
2 name, surname, phone_number = 'John', 'Cheng', '56.46.67'
3 name
```

```
1 # asignacion multiple
2 spam = ham = 'lunch'
3 print(spam)
4 print(ham)
```

```
1 # asignacion argumentada
2 spams = 23
3 spams += 42
4 print(spams)
```

Referencia compartida

Inmutabilidad

```
1 a = 23
2 b = a
3
4 a = 99
```

Cual es el valor de a y de b?

```
1 print(a)
2 print(b)
```

Mutabilidad

```
1 a = [1, 2]
2 b = a
3 print a
4 print b
```

```
1 a[1] = 'spam'
```

Cual seria el valor de b?

```
1 print(b)
```

Referencia compartida



```
1  Objetos Inmutables:  
2  
3  int  
4  float  
5  decimal  
6  complex  
7  bool  
8  string  
9  tuple  
10 range  
11 frozenset  
12 bytes  
13
```

:

```
1  Objetos Mutables:  
2  list  
3  dict  
4  set  
5  bytearray
```

En Resumen !



en pocas palabras, en Python:

- todo es un **objeto** (incluidas las funciones)
- todo objeto tiene un **tipo**
- se puede acceder a muchos tipos **pre-cargados** (str, float, list, tuple, dict, ...)
- se pueden combinar objetos para producir nuevos valores, usando **expresiones**
- todo lo anterior son "cosas"
- y para realizar acciones utiles con esas cosas, se usan las **sentencias**

IF en Python



```
1 value = 34
2 if value < 30:
3     print("Value lower than 30")
4 elif value > 30 and value < 50:
5     print("Value between 30 and 50")
6 else:
7     print("Value higher than 50")
```

```
1 # otra forma de escribirlo
2 value = 34
3 if value > 30:
4     level = 'higher'
5 else:
6     level = 'lower'
```

escrito en la forma "Pythonica":

```
1 # mas "natural"
2 'higher' if value > 30 else 'lower'
```

Lazos de Iteracion en Python

b.3 La sentencia de Iteracion ► para repetir acciones sobre una secuencia

```
: 1 # creamos una lista
2 ciudades = ['San Salvador', 'Santa Ana', 'San Vicente', 'Santo Tomas']
3
4 for i in ciudades:
5     print("Debo visitar {}".format(i))
```

```
: 1 a = 0; b = 10 # note el uso de una sola linea para la asignacion de 2 variables
2
3 while a < b:
4     print(a)
5     a +=1
```

IF y FOR en Python

Write code in Python 3.6

(drag lower right corner to resize code editor)

```
1 ip_address = ['176.149.135.210', '176.149.135.211', '176.149.135.212', '176.149.135.213']
2
3 for i in ip_address:
4     print("Trying to connect to {}".format(i))
5 |
```

Print output (drag lower right corner to resize)

```
Trying to connect to 176.149.135.210
Trying to connect to 176.149.135.211
Trying to connect to 176.149.135.212
Trying to connect to 176.149.135.213
```

Frames

Objects

Global frame

ip_address

i "176.149.135.213"

list

0	1	2
"176.149.135.210"	"176.149.135.211"	"176.149.135.212"

Definir funciones en Python



b.4 La sentencia `def` ► para definir una Funcion

```
1 def square(x):  
2     return x**2
```

b.5 La sentencia `call` ► ejecutar una funcion

```
1 square(8)
```

b.6 y muchos mas ...

Se puede ver un listado completo de todas las sentencias: https://docs.python.org/3/reference/simple_stmts.html

Definir funciones en Python



```
1 # Declaracion de una funcion
2 def f(a, b, c):
3     print(a, b, c)
```

```
1 # invocando la funcion
2 f(1, 2, 3)
```

```
1 # las funciones son objetos tambien
2 g = f
3 g(1, 2, 3)
```

Excepciones en Python

```
1 def f(a, b):  
2     return a / b
```

```
1 f(1, 2)
```

```
1 f(1, 0)
```

Capturando una excepcion

```
1 def f(a, b):  
2     try:  
3         x = a / b  
4     except ZeroDivisionError:  
5         print('Division by zero!')  
6     else:  
7         print(x)  
8     finally:  
9         print('I am in finally block')  
10        print("Let's continue!")  
11  
12 f(1, 0)
```

Módulos en Python



Usar módulos o por qué no reinventar la rueda

Un módulo son un programa que viene “empaquetado” para poder usado por otros programas de manera sencilla. Muchos programadores elaboran estos módulos, y luego los comparten gratuitamente a través de páginas como Github. Si alguien ya ha hecho el trabajo, ¿por qué no aprovecharlo?.

Python dispone de módulos para realizar multitud de operaciones, tales como realizar matemáticas avanzadas, descargar vídeos de Youtube, dibujar gráficos, programas juegos, etc. Algunos de estos módulos vienen incluidos de “serie” y otros es necesario descargarlos.

Gracias a estos módulos, escribiendo muy pocas líneas de código, pueden hacerse cosas complejas. En ocasiones parece mágico.

Módulos en Python



¿Cómo instalar un módulo?

Para instalar un módulo simplemente hay que copiar el archivo con la extensión .py y copiarlo en la carpeta en la que tengamos nuestro código fuente. No obstante, en ocasiones hay maneras más cómodas de realizar la instalación.

Queremos instalar el módulo `html5lib`.

Dado que tenemos instalado Anaconda, lo primero que hay que hacer es cerrar Anaconda (incluida la ventana tipo msdos con fondo negro que puede que tengamos abierta), y basta con abrir una línea de comandos ejecutando “Anaconda Prompt”:

En la ventana que se abra escribiremos:

```
conda install html5lib
```


Módulos en Python



Cada módulo, tiene operaciones (funciones) asociadas ya incluidas con lo que en caso de que necesitemos hacer algo, sólo tenemos que pedir que se realice esa operación, y no tenemos que realizarla nosotros paso a paso. No hay que reinventar la rueda.

Si el módulo está instalado, para cargarlo tenemos que usar el comando `import`. Para usar alguna de las funciones que están en el módulo, basta con poner el `nombre_del_modulo.nombre_funcion()`:

#	Cargamos	el	módulo
<code>import</code>			<code>modulo</code>

Para usar alguna de las funciones que están en el módulo.
`modulo.funcion()`

Módulos en Python



```
1 # importando un modulo de la libreria standard, usando la sentecia 'import'  
2 import math
```

```
1 type(math)
```

```
1 # para obtener una lista de todos los atributos del modulo importado  
2 dir(math)
```

```
1 # el uso de '.' permite acceder a un atributo  
2 math.pi
```

```
1 math.log(10)
```

```
1 math.tan (math.pi)
```

Alcance de una variable en Python

```
1 a = 2 # primero se asigna el valor 2 a la variable a
2 def f():
3     a = 3 # luego re-ligamos a con el valor de 3 en el cuerpo de una función
4     print('Valor de "a" dentro de la función: ', a)
```

Cual sera el valor de a?

```
1 f()
```

Alcance de una variable en Python

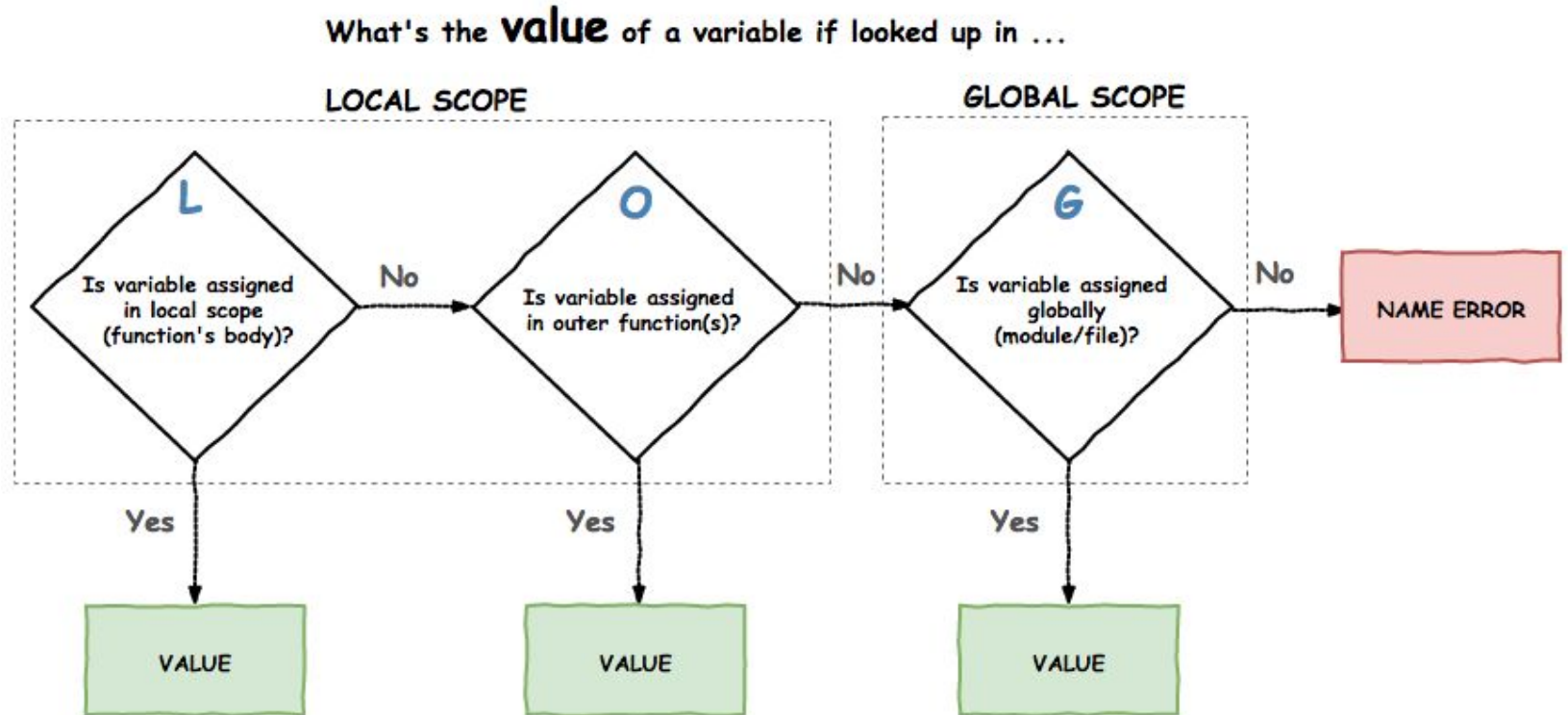
De otra manera

```
1 a = 2 # se asigna el valor de 2 a la variable a
2
3 def f():
4     print('Valor de "a" dentro de la funcion: ', a)
```

Cual sera el valor de a?

```
1 f()
```

Alcance de una variable en Python



Alcance de una variable en Python

Ejemplo 1: alcance global

```
a, b = 1, 1
for i in range(3):
    print(a)
```

Cual sera el valor de a?

```
1 a, b = 1, 1
2 for i in range(3):
3     print(a)
```

Ejemplo 2: alcance Local y Global

```
1 a, b = 1, 1
2
3 def f():
4     b, c = 2, 3
5     print(a, b, c)
```

Cuales seran los valores de a, b, c?

```
1 f()
```

Algunos desafíos de práctica



Problema 1 - Imprimir los números impares desde el 1 al 25, ambos inclusive

Write code in Python 3.6

(drag lower right corner to resize code editor)

```
1  ###Imprimir los numeros impares desde el 1 al 25, ambos inclusive
2  n = 1
3  h = ''
4  while n <= 25:
5      if n%2 != 0:
6          h += " %i" % n
7      n += 1
8  print (h)
9  |
```

Algunos desafíos de práctica



Problema 2 - Crear una función para validación de nombres de usuarios.
Dicha función, deberá cumplir con los siguientes criterios de aceptación:

El nombre de usuario debe contener un mínimo de 6 caracteres y un máximo de 12

Nombre de usuario con menos de 6 caracteres, retorna el mensaje "El nombre de usuario debe contener al menos 6 caracteres".

Nombre de usuario con más de 12 caracteres, retorna el mensaje "El nombre de usuario no puede contener más de 12 caracteres".

Nombre de usuario válido, retorna True.

Fin de la sesión: Fundamentos de Python.

Siguiente sesión: Intro to IoT

—