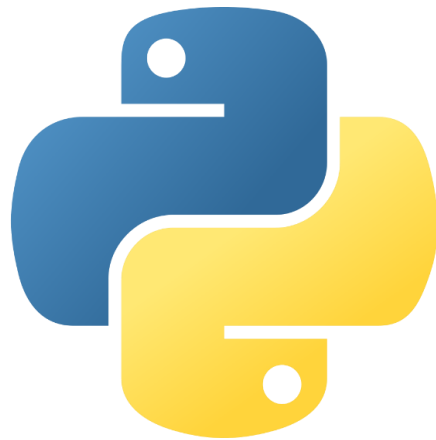


Introducción a python



pythonTM

¿Qué es Python?

- Python es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender, y fue creado a principios de los noventa por Guido van Rossum en los Países Bajos.
- Es un lenguaje de alto nivel, que permite procesar fácilmente todo tipo de estructuras de datos, tanto numéricas como de texto.
- Es software libre, y está implementado en todas las plataformas y sistemas operativos habituales.

- Las características del lenguaje de programación Python se resumen a continuación:
- Es un lenguaje interpretado, no compilado que usa tipado dinámico, fuertemente tipado (el tipo de valor no cambia repentinamente).
- Es multiplataforma, lo cual es ventajoso para hacer ejecutable su código fuente entre varios sistemas operativos.

Desventajas

- Curva de aprendizaje: La “curva de aprendizaje cuando ya estás en la parte web no es tan sencilla”.
- Hosting: La mayoría de los servidores no tienen soporte a Python, y si lo soportan, la configuración es un poco difícil.
- Librerías incluidas: Algunas librerías que trae por defecto no son del gusto de amplio de la comunidad, y optan a usar librerías de terceros.

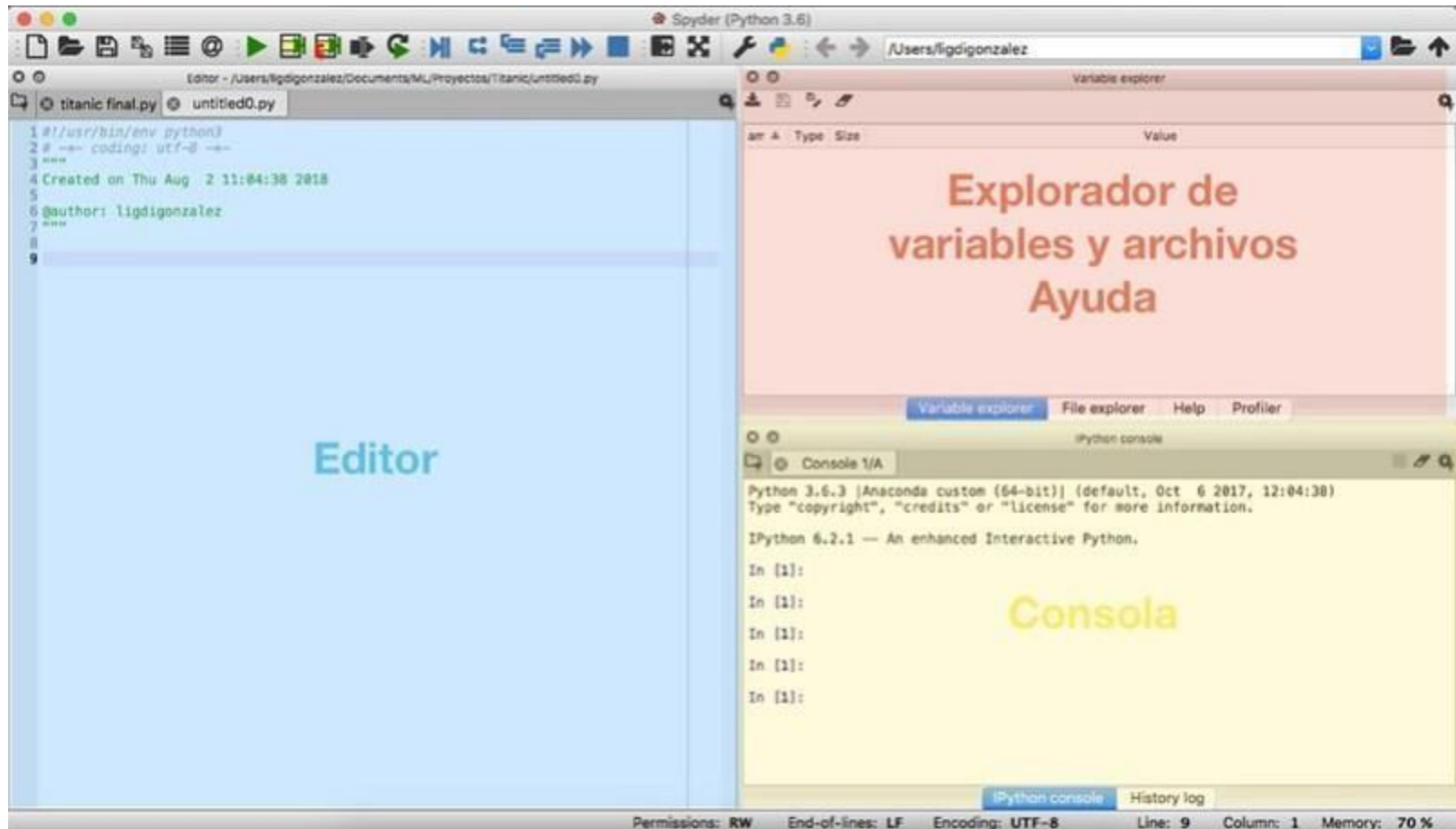
Versiones

Versión	Fecha de lanzamiento
Python 3.6	Diciembre 2016
Python 3.7	Junio 2018
Python 3.8	Octubre 2019
Python 3.9	Octubre 2020
Python 3.10	Octubre 2021
Python 3.11	Octubre 2022
<i>Python 3.12</i>	<i>Octubre 2023 ?</i>

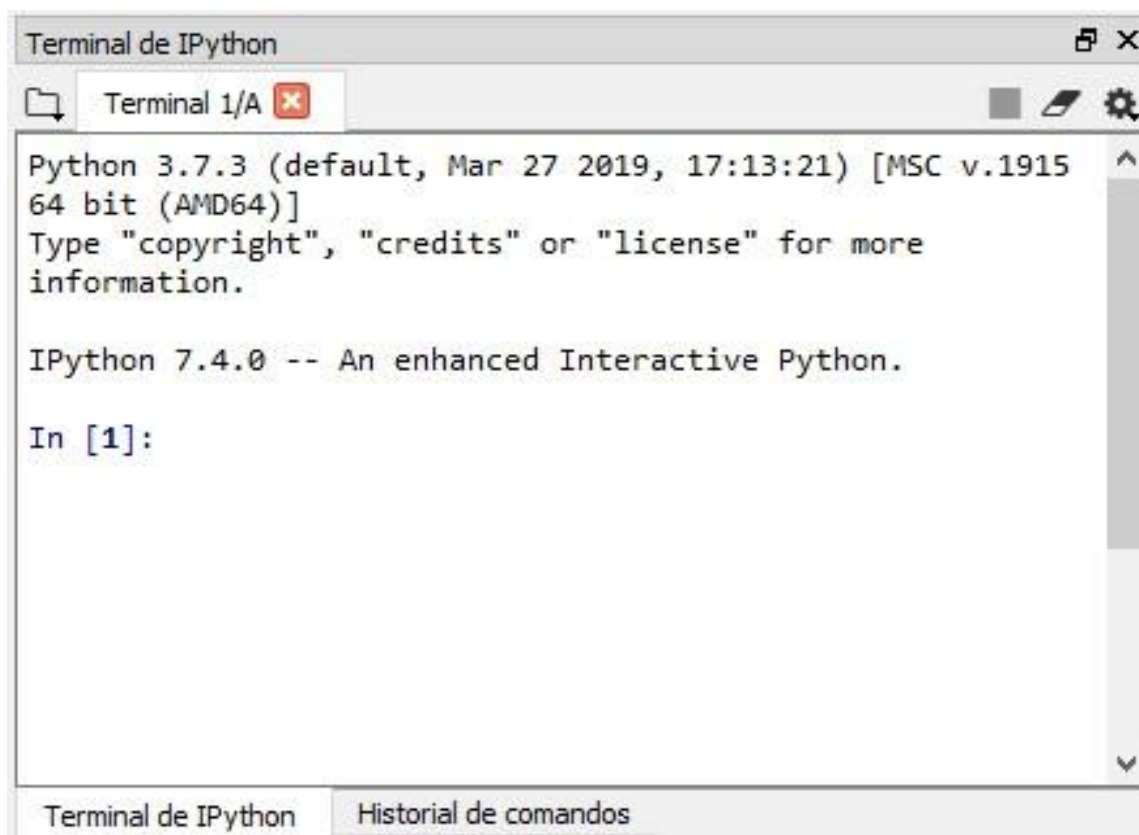
Como comenzamos

- Para poder utilizar Python lo tienes que tener instalado en tu ordenador.
- Hay diferentes maneras de hacerlo, hay varias distribuciones.
- Elige la versión que corresponde a tu sistema operativo e instálala siguiendo las instrucciones.
- Spyder es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) que permiten escribir scripts de Python e interactuar con el software de Python desde una interfaz única.

Spyder



Consola



The image shows a screenshot of a terminal window titled "Terminal de IPython". The window has a tab labeled "Terminal 1/A" and standard window controls (minimize, maximize, close). The terminal content displays the Python 3.7.3 startup message, including the version, date, and architecture (64 bit (AMD64)). It also shows the IPython 7.4.0 banner, which describes it as an enhanced interactive Python environment. The prompt "In [1]:" is visible, indicating the start of a new code block. At the bottom of the window, there are two tabs: "Terminal de IPython" and "Historial de comandos".

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915  
64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more  
information.  
  
IPython 7.4.0 -- An enhanced Interactive Python.  
  
In [1]:
```


“Shell”

- Aprender el idioma.
- Experimentar con la biblioteca.
- Prueba de tus propios módulos.
- Dos variaciones: IDLE (GUI), python (línea de comando)
- Escriba declaraciones o expresiones en la consola

```
>>> print ("Hello, world")
```

```
Hello, world
```

```
>>> x = 12**2
```

```
>>> x/2
```

```
72
```

```
# Esto es un comentario de una sola línea
```

```
"""Y este es un comentario
```

```
de varias líneas"""
```

Programa

- Un programa python está formado por instrucciones que acaban en un caracter de “salto de línea”.
- El punto y coma “;” se puede usar para separar varias sentencias en una misma línea, pero no se aconseja su uso.
- Una línea empieza en la primera posición, si tenemos instrucciones dentro de un bloque de una estructura de control de flujo habra que hacer una indentación.
- La indentación se puede hacer con espacios y tabulaciones pero ambos tipos no se pueden mezclar. Se recomienda usar 4 espacios.
- La barra invertida “\” al final de línea se emplea para dividir una línea muy larga en dos o más líneas.
- Las expresiones entre paréntesis “()”, llaves “{}” y corchetes “[]” separadas por comas “,” se pueden escribir ocupando varias líneas.
- Cuando el bloque a sangrar sólo ocupa una línea ésta puede escribirse después de los dos punto.

Variables

- Una variable es un espacio para almacenar datos modificables, en la memoria de un ordenador. En Python, una variable se define con la sintaxis:
- `nombre_de_la_variable = valor_de_la_variable`

Numeros

- entero = 7
- print (entero, type(entero))
- i = 12
- print(i) #12
- print(type(i)) #<class 'int'>
- b = int(1.6)
- print(b) #1

Clase	Tipo	Notas	Ejemplo
int	Números	Número entero con precisión fija.	42
long	Números	Número entero en caso de overflow.	42L ó 456966786151987643L
float	Números	Coma flotante de doble precisión.	3.1415927
complex	Números	Parte real y parte imaginaria j .	(4.5 + 3j)

Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search
- "escapes: \n etc, \033 etc, \if etc"
- 'single quotes' ""triple quotes"" r"raw strings"

Variables

- nombre_de_la_variable = valor_de_la_variable
- No necesitan declarar
- Necesitan inicializar(initialize)
 - use of uninitialized variable raises exception
- No tipo

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```
- ***Todo es una*** "variable":
 - Even functions, classes, modules

Ejemplo

- $\text{monto_bruto} = 175$
- $\text{tasa_interes} = 12$
- $\text{monto_interes} = \text{monto_bruto} * \text{tasa_interes} / 100$
- $\text{tasa_bonificacion} = 5$
- $\text{importe_bonificacion} = \text{monto_bruto} * \text{tasa_bonificacion} / 100$
- $\text{monto_neto} = (\text{monto_bruto} - \text{importe_bonificacion}) + \text{monto_interes}$

Comentarios

- Un archivo, no solo puede contener código fuente. También puede incluir comentarios
- (notas que como programadores, indicamos en el código para poder comprenderlo mejor).
- Los comentarios pueden ser de dos tipos: de una sola línea o multi-línea y se expresan de la siguiente manera:
 - `#` Esto es un comentario de una sola línea
 - `"""Y este es un comentario`
 - `de varias líneas"""`
 - `mi_variable = 15 # Este comentario es de una línea también`

Tuplas

- **Una tupla es una variable que permite almacenar varios datos inmutables (no pueden ser modificados una vez creados) de tipos diferentes:**
- `mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)`
- Se puede acceder a cada uno de los datos mediante su índice correspondiente, siendo 0 (cero), el índice del primer elemento:
- `print mi_tupla[1]` # Salida: 15
- También se puede acceder a una porción de la tupla, indicando (opcionalmente) desde el índice de inicio hasta el índice de fin:
- `print mi_tupla[1:4]` # Devuelve: (15, 2.8, 'otro dato')
- `print mi_tupla[3:]` # Devuelve: ('otro dato', 25)
- `print mi_tupla[:2]` # Devuelve: ('cadena de texto', 15)
- Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:
- `print mi_tupla[-1]` # Salida: 25
- `print mi_tupla[-2]` # Salida: otro dato

Lists

- **Una lista es similar a una tupla con la diferencia fundamental de que permite modificar los datos una vez creados**
- `mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]`
- A las listas se accede igual que a las tuplas, por su número de índice:
- `print mi_lista[1]` # Salida: 15
- `print mi_lista[1:4]` # Devuelve: [15, 2.8, 'otro dato']
- `print mi_lista[-2]` # Salida: otro dato
- Las lista NO son inmutables: permiten modificar los datos una vez creados:
- `mi_lista[2] = 3.8` # el tercer elemento ahora es 3.8
- Las listas, a diferencia de las tuplas, permiten agregar nuevos valores:
- `mi_lista.append('Nuevo Dato')`

Diccionarios

- **Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor:**
- `mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, \`
- `'clave_7': valor_7}`
- `print mi_diccionario['clave_2']` # Salida: `valor_2`
- Un diccionario permite eliminar cualquier entrada:
- `del(mi_diccionario['clave_2'])`
- Al igual que las listas, el diccionario permite modificar los valores
- `mi_diccionario['clave_1'] = 'Nuevo Valor'`

Estructuras de Control de Flujo

- Una estructura de control, es un bloque de código que permite agrupar instrucciones de manera controlada.
- Tenemos 2 tipos de estructuras de control:
 - Estructuras de control condicionales
 - Estructuras de control iterativas

INDENTACION

- ¿Qué es la indentación? En un lenguaje informático, la indentación es lo que la sangría al lenguaje humano escrito (a nivel formal). Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, requieren una indentación.
- No todos los lenguajes de programación, necesitan de una indentación, aunque sí, se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. Pero en el caso de Python, la indentación es obligatoria, ya que de ella, dependerá su estructura.
- **Una indentación de 4 (cuatro) espacios en blanco, indicará que las instrucciones indentadas, forman parte de una misma estructura de control.**
-

Se agrupan por indentación

In Python:

```
for i in range(20):
    if i%3 == 0:
        print (i)
    if i%5 == 0:
        print ("Bingo!")
print ("---")
```

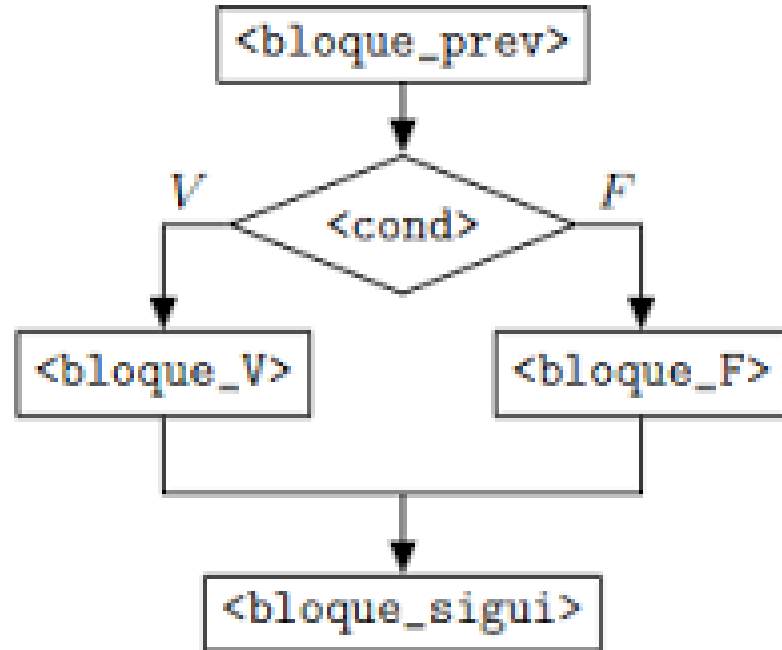
In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"); }
    }
    printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

Estructuras de control Condicionales

```
if condition:  
    statements  
[elif condition:  
    statements] ...  
else:  
    statements
```

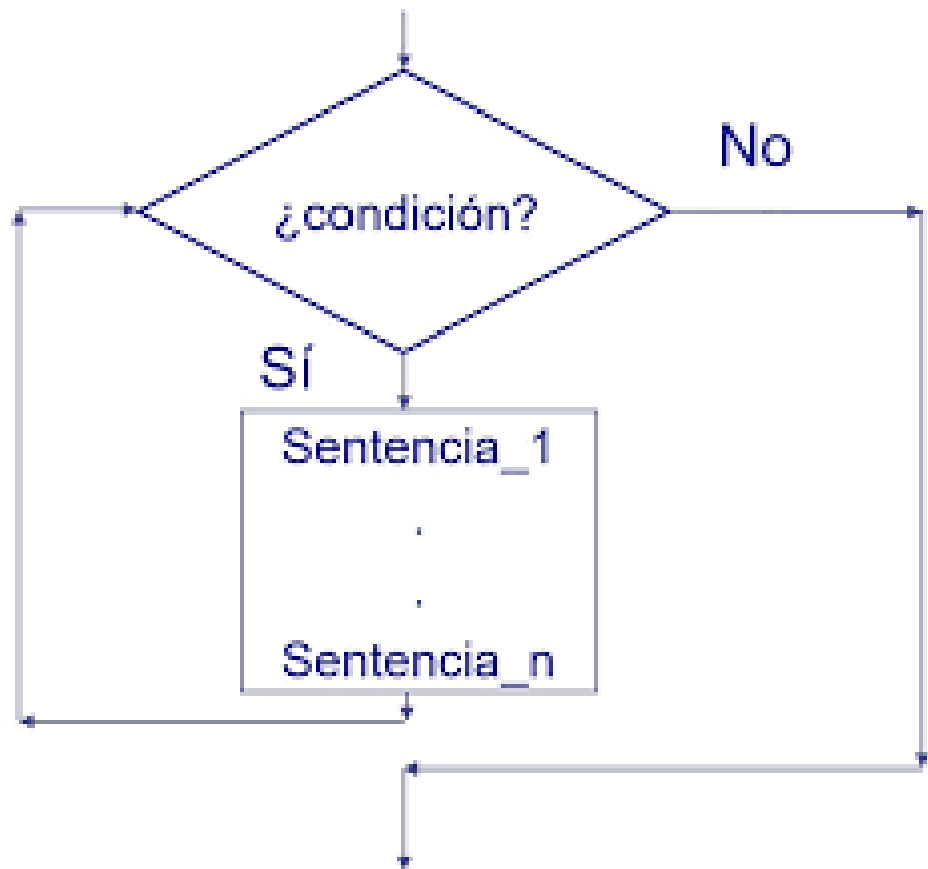


Ejemplo

- if compra <= 100:
 - print "Pago en efectivo"
- elif compra > 100 and compra < 300:
 - print "Pago con tarjeta de débito"
- else:
 - print "Pago con tarjeta de crédito"
- -----
- if total_compra > 100:
 - tasa_descuento = 10
 - importe_descuento = total_compra * tasa_descuento / 100
 - importe_a_pagar = total_compra - importe_descuento

Estructuras de control

while condition:
statements



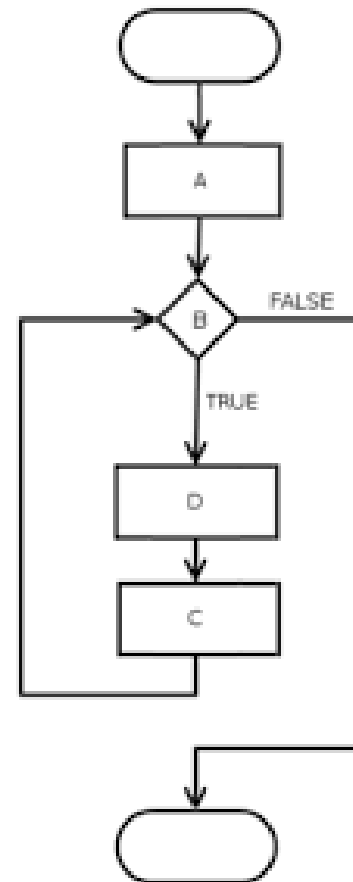
BUCLES

- anio = 2001
 - **while anio <= 2012:**
 - print "Informes del Año",
 str(anio)
 - anio += 1
- while True:
 - nombre = raw_input("Indique
 su nombre: ")
 - if nombre:
 - break

Estructuras de control

for *var* in *sequence*:
 statements

for(A;B;C)
D;



FOR

- Por cada nombre en mi_lista, imprimir nombre
 - mi_lista = ['Juan', 'Antonio', 'Pedro', 'Herminio']
 - for nombre in mi_lista:
 - print nombre
-
- Por cada color en mi_tupla, imprimir color
 - mi_tupla = ('rosa', 'verde', 'celeste', 'amarillo')
 - for color in mi_tupla:
 - print color

- Por cada año en el rango 2001 a 2013, imprimir la frase
- “Informes del Año año”
- # -*- coding: utf-8 -*-
- for anio in range(2001, 2013):
- print "Informes del Año", str(anio)

Encoding

- El encoding (o codificación) es otro de los elementos del lenguaje que no puede omitirse a la hora de hablar de estructuras de control.
- El encoding no es más que una directiva que se coloca al inicio de un archivo Python, a fin de indicar al sistema, la codificación de caracteres utilizada en el archivo.
- `# -*- coding: utf-8 -*-`
- utf-8 podría ser cualquier codificación de caracteres. Si no se indica una codificación de caracteres, Python podría producir un error si encontrara caracteres “extraños”:
- `print "En el Ñágara encontré un Ñandú"`
- Producirá un error de sintaxis: `SyntaxError: Non-ASCII character[...]`
- En cambio, indicando el encoding correspondiente, el archivo se ejecutará con éxito:
- `# -*- coding: utf-8 -*-`
- `print "En el Ñágara encontré un Ñandú"`

Funciones

- `def mi_funcion():`
- `# aquí el algoritmo`
- **Una función, no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:**
- `def mi_funcion():`
- `print "Hola Mundo"`
- `funcion()`
- **Cuando una función, haga un retorno de datos, éstos, pueden ser asignados a una variable:**
- `def funcion():`
- `return "Hola Mundo"`
- `frase = funcion()`
- `print frase`

Funciones / Procedimientos

EJEMPLO:

```
def name(arg1, arg2, ...):  
    """documentation""" # optional doc string  
    statements  
  
    return # para procedure  
    return expression # para function
```


Ejemplo Function

```
def mi_funcion(nombre, apellido):  
    nombre_completo = nombre, apellido  
    print nombre_completo
```

```
# esta es una función básica de suma  
def suma(a, b):  
    return a + b
```

```
result = suma(1, 2)  
# result = 3
```

- # función sin parámetros o retorno de valores
- def diHola():
- print("Hello!")
- diHola() # llamada a la función, 'Hello!' se muestra en la consola
- # función con un parámetro
- def holaConNombre(name):
- print("Hello " + name + "!")
- holaConNombre("Ada") # llamada a la función, 'Hello Ada!' se muestra en la consola
- # función con múltiples parámetros con una sentencia de retorno
- def multiplica(val1, val2):
- return val1 * val2
- multiplica(3, 5) # muestra 15 en la consola

CLASES Y OBJETOS (POO)

Clases

Podemos definir una clase vagamente como una Plantilla o modelo para crear a partir de ella ciertos Objetos. Esta plantilla es la que contiene la información; características y capacidades que tendrá el objeto que sea creado a partir de ella.

Así a su vez los objetos creados a partir de ella estarán agrupados en esa misma clase.

Objetos

- Cuando instanciamos una clase...
- Atributos y métodos de un objeto:
- atributos son las características del objeto normalmente almacenadas en variables declaradas en la clase.
- Y los métodos podrían verse como funciones del objeto que puede realizar y que a veces necesitan de ciertos argumentos para poder operar con sus parámetros

- `Class Nombrede Clase (object):` #Declaramos la clase nuestra que hereda de `Object`
- `def __init__ (self, parámetros):` #Constructor de la clase
- #Declaración de atributos

Ejemplo Class

- `#!/usr/bin/env python`
- `# -*- coding: utf-8 -*-`
- `class Humano(): #Creamos la clase Humano`
- `def __init__(self, edad, nombre): #Definimos el atributo edad y nombre`
- `self.edad = edad # Definimos que el atributo edad, sera la edad asignada`
- `self.nombre = nombre # Definimos que el atributo nombre, sera el nombre asig`
- `Persona1 = Humano(31, "Pedro") #Instancia`

- `#!/usr/bin/env python`
- `# -*- coding: utf-8 -*-`
- `class Humano():` #Creamos la clase Humano
- `def __init__(self, edad, nombre, ocupacion):` #Definimos el parametro edad , nombre y ocupacion
- `self.edad = edad` # Definimos que el atributo edad, sera la edad asignada
- `self.nombre = nombre` # Definimos que el atributo nombre, sera el nombre asig
- `self.ocupacion = ocupacion` #DEFINIMOS EL ATRIBUTO DE INSTANCIA OCUPACION
-
- `#Creación de un nuevo método`
- `def presentar(self):`
- `presentacion = ("Hola soy {}, mi edad es {} y mi ocupación es {}")` #Mensaje
- `print(presentacion.format(self.nombre, self.edad, self.ocupacion))` #Usamos FORMAT
- `Persona1 = Humano(31, "Pedro", "Desocupado")` #Instancia
- `#Llamamos al método`
- `Persona1.presentar()`

Modules

- En Python, cada uno de nuestros scripts .py se denominan módulos. Estos módulos, a la vez, pueden formar parte de librerías. Una librería (o paquete) es una carpeta que contiene archivos
- .py. Pero, para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado `__init__.py`. Este archivo, no necesita contener ninguna instrucción. De hecho, puede estar completamente vacío. Los paquetes, a la vez, también pueden contener otros subpaquetes

```
.  
└─ paquete  
    ├── __init__.py  
    ├── modulo1.py  
    └─ subpaquete  
        ├── __init__.py  
        ├── modulo1.py  
        └─ modulo2.py
```

- El contenido de cada módulo, podrá ser utilizado a la vez, por otros módulos. Para ello, es necesario importar los módulos que se quieran utilizar. Para importar un módulo, se utiliza la instrucción `import`, seguida del nombre del paquete más el nombre del módulo (sin el `.py`) que se desee importar.
- La instrucción:
- `import` seguida de `nombre_del_paquete.nombre_del_modulo`,
- nos permitirá hacer uso de todo el código que dicho módulo contenga.

```
import modulo1  
import paquete.modulo1  
import paquete.subpaquete.modulo1
```

¿Cómo se hace un módulo?

- La importación de módulos debe realizarse al comienzo del documento, en orden alfabético de paquetes y módulos

- Imaginemos que construimos un script con las cuatro operaciones básicas: sumar, restar, multiplicar y dividir, y deseamos construir un paquete con dichas funciones:

```
basicas.py
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Sep 11 17:53:06 2020
4
5 @author: julio
6 """
7
8 def sumar(a,b):
9     return a+b
10
11 def restar(a,b):
12     return a-b
13
14 def multiplicar(a,b):
15     return a*b
16
17 def dividir(a,b):
18     return a/b
19
```

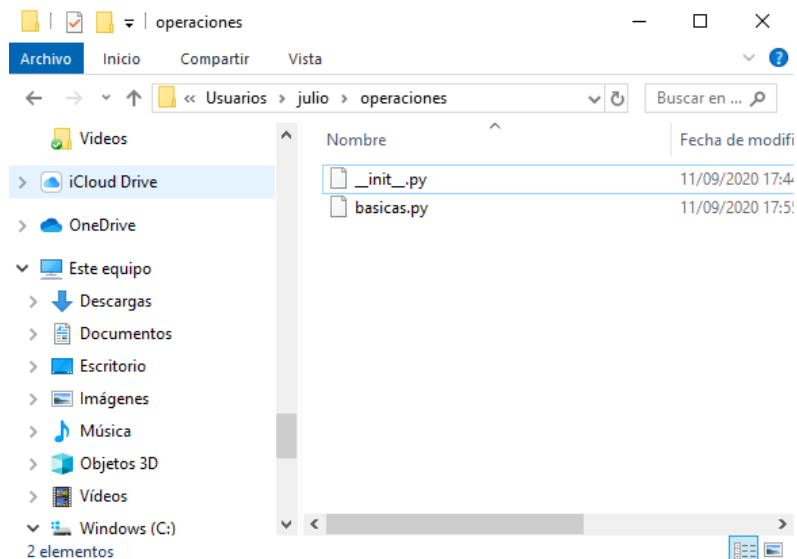
- En primer lugar, debemos conocer el directorio desde donde Python carga los paquetes que se
- importan. Para ello podemos usar la librería os y la instrucción `getcwd()`:

```
In [34]: import os
```

```
In [35]: os.getcwd()
```

```
Out[35]: 'C:\\\\Users\\'
```

- A continuación, creamos un directorio nuevo (de nombre, operaciones) e incluimos en él nuestro script (con nombre básicas.py) y un archivo adicional que estará vacío y que llamaremos
- `__init__.py`
- Puede ser hasta un fichero vacío



Librerias



Pillow

tqdm



matplotlib



SQLAlchemy

django

K Keras

NLTK 3.4



Importar

- Para trabajar con numpy y los arrays, importamos la librería a partir de alguna de las siguientes
- instrucciones:
- `import numpy` # Cargar el modulo numpy, o bien
- `import numpy as np` # Cargar el modulo numpy, llamándolo np, o bien
- `from numpy import *` # Cargar todas funciones de numpy

Capturar Exceptions

```
def foo(x):  
    return 1/x
```

```
def bar(x):  
    try:  
        print foo(x)  
    except ZeroDivisionError, message:  
        print "Can't divide by zero:", message
```

```
bar(0)
```

Try-finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close() # always executed
print "OK" # executed on success only
```

File Objects

- `f = open(filename[, mode[, buffersize]])`
 - mode can be "r", "w", "a" (like C stdio); default "r"
 - append "b" for text translation mode
 - append "+" for read/write open
 - buffersize: 0=unbuffered; 1=line-buffered; buffered
- methods:
 - `read([nbytes])`, `readline()`, `readlines()`
 - `write(string)`, `writelines(list)`
 - `seek(pos[, how])`, `tell()`
 - `flush()`, `close()`
 - `fileno()`

URLs

- <http://www.python.org>
 - official site
- <http://starship.python.net>
 - Community
- <http://www.python.org/psa/bookstore/>
 - (alias for <http://www.amk.ca/bookstore/>)
 - Python Bookstore