EAE Business School

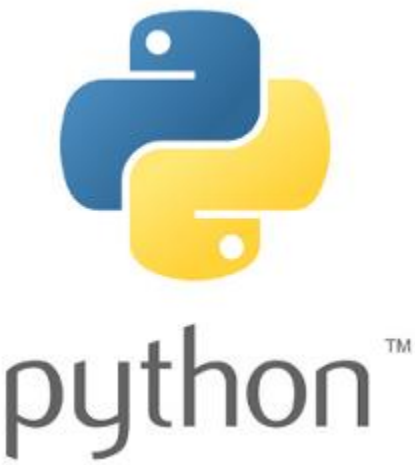
Python

Máster en Business Intelligence e Innovación Tecnológica

Índice de contenidos

Que es Python?

- Python es uno de los lenguajes de programación más extendidos de la actualidad.
- Es un lenguaje de scripting creado por Guido van Rossum en 1991.
- Es un lenguaje sencillo, mucho más accesible que otros más complejos como C o Java.
- Utiliza tipado dinámico, lo que permite asignar distintos tipos de datos a sus variables.
- Existe un gran soporte y una gran cantidad de librerías disponibles, lo que lo convierten en un lenguaje muy potente con un carácter “vale para todo y para todos”.
- Es Open Source.
- Python es uno de los lenguajes más utilizados en ámbitos como web o BigData.



Hello World!

```
print "Hola Mundo!"
```

Podéis intentarlo desde vuestra MV, instalando Python directamente en vuestro PC (Windows, Mac o Linux) o desde cualquier intérprete online, por ejemplo:
https://repl.it/languages/python

Ejecución de una aplicación Python

Python es un lenguaje de scripting que se ejecuta en un formato de línea de comandos. Por lo tanto, la forma clásica de ejecución es desde una Shell:

```
python myApp.py
```

Uso de variables

- Python permite definir variables sin especificar su tipo, que lo tomarán desde el valor que le asignemos.

```
my_ent = 2

print my_ent
```

Los saltos de línea identifican el final de la sentencia

```
my_ent = 2
my_real = 3.14
my_bool = True
my_string = "cadena de texto“
print my_ent," ", my_real, " ",my_bool, " ", my_string
```

Separamos los distintos elementos con “,”

Comentarios

- Es muy común utilizar comentarios para documentar el comportamiento del código y hacerlo más legible y entendible. Los escibimos con el carácter #

```
# comentario del código
my_real = 3.14
```

- Para comentarios de más de una línea, los escribimos en un bloque iniciado y acabado por tres comillas dobles

```
"""este comentario tiene
varias líneas
"""
my_real = 3.14
```

Operaciones matemáticas

- Las operaciones matemáticas se realizan con los símbolos clásicos

```
suma = 72 + 23
resta = 108 - 204
multiplicacion = 108 * 0.5
division = 108 / 9
print suma," ", resta, " ",multiplicacion, " ", division
```

Entrada desde consola

- La función raw_input sirve para introducir cualquier valor desde la consola manualmente

```
value = raw_input("Introduce un valor")
print value
```

Operadores de comparación

- Igual a (==)
- No es igual a (!=)
- Menor que (<)
- Menor o igual que (<=)
- Mayor que (>)
- Mayor o igual que (>=)

```
bool1 = (1==1)
bool2 = (1!=2)
bool3 = (2<3)
bool4 = (2<=3)
bool5 = (3>2)
bool6 = (4>=2)
```

- Todos estos comparadores devolverán un valor booleano: True o False
- Podemos combinar varios operadores mediante “or” y “and”
- Un valor booleano se puede invertir con el operador “not”

```
bool1 = (1==1)
bool2 = (0>2)
bool3 = bool1 or bool2
bool4 = bool2 and bool1
bool5 = not bool1
print bool1, bool2, bool3, bool4, bool5
```

If y Else

- Son evaluadores de condiciones que forman la base de cualquier programa

```
ocho = 8
nueve = 9
if ocho > nueve:
    print "¡No me muestro!"
else:
    print "¡Me muestro!"
```

Es muy importante mantener la indentación: tab!

```
Traceback (most recent call last):
  File "python", line 4
    print "¡No me muestro!"
        ^
IndentationError: expected an indented block
```

If y Else

- Podemos añadir flujos más complejos mediante “elif”, que permite evaluar tantas condiciones como sean necesarias:

```
if 8 > 9:
    print "¡No me muestro!"
elif 8 < 9:
    print "¡Me muestro!"
else:
    print "¡Yo tampoco me muestro!"
```

Bucles while

- Ejecutan todas las operaciones introducidas mientras se cumpla una condición

```
recuento = 0

while recuento <= 9:
    print "Hola, Soy un bucle while y el recuento es", recuento
    recuento = recuento + 1
```

- Podemos romper la ejecución del bucle con la sentencia “break”

```
recuento = 0

while True:
        print recuento
        recuento += 1
        if recuento >=10:
                break
```

Bucles while

```
from random import randint

# genera un número del 1 al 10 inclusive
numero_aleatorio = randint(1, 10)

opciones_restantes = 3

#empieza el while
while opciones_restantes > 0:
    opcion = int(raw_input("Tu opcion: "))
    if opcion == numero_aleatorio:
        print "Ganaste"
        break

    opciones_restantes -= 1
else:
    print "Perdiste, era ", numero_aleatorio
```

Importación de la función de random

Inicio del while

Bloque de ejecución al final del while

Bucles for

- Son similares a while, pero se utilizan cuando queremos repetir el código *n* veces

```
print "Contando..."

for i in range(20):
    print i
```

En este caso, no tenemos que ir actualizando el valor de i

```
pasatiempos = []

# Agrega tu código a continuación
for i in range(3):
    pasatiempo = raw_input("Escribe tu pasatiempo " + str(i))
    pasatiempos.append(pasatiempo)

print pasatiempos
```

Bucles for

- También podemos realizar el bucle sobre las letras de una palabra

```
cosa = "spam!"

for c in cosa:
    print c,
```

La “,” indica al print que imprima en la misma línea

Listas

- Las listas son elementos de datos que contienen varios subelementos en su interior:

```
zoo_animals = [ "mono", "elefante", "tigre","mierda"]
```

- Podemos acceder a sus valores de forma individual mediante las claves:

```
zoo_animals = [ "mono", "elefante", "tigre","mierda"]
print zoo_animals[2]
```

El primer elemento de la lista es el 0!

- Podemos añadir nuevos elementos con la función “.append” y obtener su longitud con la función “len”:

```
zoo_animals = [ "mono", "elefante", "tigre","mierda"]
zoo_animals.append("rata")
print len(zoo_animals)
```

Funciones sobre listas

- En lugar de valores concretos, también podemos acceder a sub-listas:

```
letras = ['a', 'b', 'c', 'd', 'e']
slice = letras[1:3]
print slice
print letras
```

- Podemos buscar la posición concreta de cualquier elemento con la función index() y añadir un nuevo elemento en una posición concreta con la función insert():

```
animales = ["oso hormiguero", "tejon", "pato", "emu", "zorro del desierto"]
indice_pato = animales.index("pato")
animales.insert(indice_pato, "cobra")
print animales
```

Funciones sobre listas

- La función sort servirá para ordenarlas:

```
animales = ["zorro", "hormiga", "murcielago", "gato"]
animales.sort()
print animales
```

- Podemos buscar la posición concreta de cualquier elemento con la función index() y añadir un nuevo elemento en una posición concreta con la función insert():

```
animales = ["oso hormiguero", "tejon", "pato", "emu", "zorro del desierto"]
indice_pato = animales.index("pato")
animales.insert(indice_pato, "cobra")
print animales
```

- Podemos eliminar un elemento con la función remove()

```
animales = ["zorro", "hormiga", "murcielago", "gato"]
animales.remove("hormiga")
print(animales)
```

Diccionarios

- Las listas también pueden almacenar en sus elementos un par clave-valor:

```
animales_zoo = {'Gaviota' : 104, 'Perezoso' : 105, 'Piton de Birmania' : 106}
print animales_zoo
```

- Para añadir nuevos elementos, lo hacemos con la siguiente sintaxis:

```
animales_zoo['Koala'] = 107
print animales_zoo
```

- Para eliminarlos, utilizamos la sentendia del:

```
del animales_zoo['Koala']
print animales_zoo
```

Introducción

- Las funciones son bloques de código que se definen de una forma especial para hacerlos reaprovechables

```
def spam():
    print "SPAM!"

spam()
```

- Las funciones pueden incluir parámetros y devolver valores:

```
def cuadrado(n):
    squared = n**2
    print "%d al cuadrado es %d." % (n, squared)
    return cuadrado

cuadrado(10)
```

Funciones estándar

- Podemos utilizar funciones definidas de muchas maneras. Algunas vienen en el propio motor de Python, otras las podemos emplear importando su librería.

```
import math
print math.sqrt(25)
```

De este modo, podemos utilizar muchas librerías disponibles por internet!

- En lugar de la librería, podemos importar una única función

```
from math import sqrt

print sqrt(25)
```

Si en lugar de la función, introducimos “*”, importamos la librería al completo

De este modo utilizamos la función directamente, sin el math.

Introducción a la POO

- La POO (Programación Orientada a Objetos) se basa en el concepto de clase, que es una estructura de datos que representa objetos del mundo real.
- Estos objetos pueden tener una serie de atributos, que son campos internos que definen el objeto.
- Los objetos también tienen funciones o métodos, que representan las operaciones que realizamos con el objeto en nuestra aplicación.
- Una *clase*, es una definición de uno de esos objetos.

Ejemplo

```
class Animal(object):
    def __init__(self, nombre):
        self.nombre = nombre



cebra = Animal("Tomasa")
print cebra.nombre
```

Declaración de la clase

Declaración de la función de inicialización. Self es un atributo estándar que refiere al propio objeto

Instanciación del objeto

- A partir de aquí, no trabajamos con variables separadas sino con objetos complejos que representan el concepto real: por ejemplo, con Animales.

Ejemplo

```
class Animal(object):
        """Crea animales lindos."""
        esta_vivo = True
        def __init__(self, nombre, edad):
                self.nombre = nombre
                self.edad = edad
        # ¡Agregá acá tu método!

        def descripcion(self):
            print self.nombre
            print self.edad

hipopotamo = Animal ("Juan",10)
hipopotamo.descripcion()
```

Object define la clase de la que hereda

La clase incluye atributos globales

La clase incluye un nuevo método descripcion

Herencia

- Una de las características de la POO es que una clase puede heredar de otra, de forma que ésta hereda todos los atributos y funciones de la primera.

```
class Cliente(object):
        """Produce objetos que representan clientes."""
        def __init__(self, id_cliente):
                self.id_cliente = id_cliente

        def ver_carrito(self):
                print "Soy una cadena que representa el contenido de tu carrito de compras!"

class ClienteHabitual(Cliente):
        """Para los clientes de la variedad repetido."""

        def ver_historial_pedidos(self):
                print "Soy una cadena que representa tu historial de pedidos!"

monty_python = ClienteHabitual("ID: 12345")
monty_python.ver_carrito()
monty_python.ver_historial_pedidos()
```

Introducción

- Los ficheros son uno de los elementos más utilizados en cualquier tipo de aplicaciones, especialmente cuando hablamos de aplicaciones que gestionan información.
- Ficheros de configuración, de almacenamiento de la información aplicativa, ficheros de extracción, ficheros de carga…hay infinidad de usos para los mismos.
- Este tipo de procesos son llamados I/O, en función de sus características de Entrada-Salida.
- Python incorpora una serie de funciones dedicadas a simplificar esta forma de trabajo, de forma que se pueda trabajar con ficheros con apenas unas sentencias.

```
mi_lista = [i**2 for i in range(1,11)]

f = open("salida.txt", "w")

for item in mi_lista:
    f.write(str(item) + "\n")

f.close()
```

Apertura de ficheros:

- Como en cualquier SO, para poder trabajar con ficheros primero los debemos abrir. Eso se hace con la función *open*, que recibe como parámetros el nombre del fichero y el modo de apertura.
- El modo de apertura es:
  - “r” para lectura
  - “w” para escritura
  - “r+” para lectura y escritura

```
f = open("salida.txt", "w")
```

Clausura de ficheros:

- Una vez hayamos acabado de trabajar con un fichero, no debemos de olvidarnos de cerrarlo para que los cambios se apliquen correctamente.
- Podremos hacerlo con la función close:

```
f.close()
```

Escritura de ficheros:

- Una vez abierto el fichero, podremos escribir texto de una forma sencilla con la función write.
- Recordad que para escribir saltos de línea, lo deberemos hacer con el texto “\n”

```
f.write(str(item) + "\n")
```

Lectura de ficheros:

- Un ficero abierto en modo lectura puede ser leído con la función *read:*

```
f.read()
```

- Si en lugar de leer todo el fichero, queremos hacerlo línea por línea podremos hacerlo con la función *readline:*

```
f.readline()
```

Lectura de ficheros:

- Si queremos abrir un fichero y leerlo línea a línea, podremos hacerlo del siguiente modo:

```
f = open('texto.txt','r')
for line in f.xreadlines():
    print line
f.close()
```

Ejemplo escritura:

```
mi_lista = [i**2 for i in range(1,11)]

mi_archivo = open("salida.txt", "r+")

for j in mi_lista:
    mi_archivo.write(str(j))
    mi_archivo.write("\n")

mi_archivo.close()
```

Ejemplo lectura total:

```
mi_archivo = open("salida.txt","r")
print(mi_archivo.read())
mi_archivo.close()
```

Ejemplo lectura parcial:

```
mi_archivo = open("texto.txt","r")
print(mi_archivo.readline())
print(mi_archivo.readline())
print(mi_archivo.readline())
mi_archivo.close()
```