Introducción a la programación con Python

LABI
Facultad de Ingeniería de la UBA

Segundo cuatrimestre, 2019

Plan

¿Qué es Python?

- Lenguaje de programación de propósito general e interpretado
- Utiliza tipado fuerte y dinámico
- Más simple de leer y mantener que otros lenguajes
- En general posee un menor rendimiento que otros lenguajes

¿Qué es Python?

- Lenguaje de programación de propósito general e interpretado
- Utiliza tipado fuerte y dinámico
- Más simple de leer y mantener que otros lenguajes
- En general posee un menor rendimiento que otros lenguajes

¿Qué es Python?

- Lenguaje de programación de propósito general e interpretado
- Utiliza tipado fuerte y dinámico
- Más simple de leer y mantener que otros lenguajes
- En general posee un menor rendimiento que otros lenguajes

¿Qué es Python?

- Lenguaje de programación de propósito general e interpretado
- Utiliza tipado fuerte y dinámico
- Más simple de leer y mantener que otros lenguajes
- En general posee un menor rendimiento que otros lenguajes

¿Por qué Python?

La popularidad de Python ha ido creciendo a lo largo de los años, en especial a partir de 2008 con Python 3.0, permitiendo que hoy en día se utilice en varios campos:

- Data analytics - Data Science
- Aplicaciones en Machine learning e inteligencia artificial
- Sistemas embebidos - Micropython
- Programación Web - Back end

¿Por qué Python?

La popularidad de Python ha ido creciendo a lo largo de los años, en especial a partir de 2008 con Python 3.0, permitiendo que hoy en día se utilice en varios campos:

- Data analytics - Data Science
- Aplicaciones en Machine learning e inteligencia artificial
- Sistemas embebidos - Micropython
- Programación Web - Back end

¿Por qué Python?

La popularidad de Python ha ido creciendo a lo largo de los años, en especial a partir de 2008 con Python 3.0, permitiendo que hoy en día se utilice en varios campos:

- Data analytics - Data Science
- Aplicaciones en Machine learning e inteligencia artificial
- Sistemas embebidos - Micropython
- Programación Web - Back end

¿Por qué Python?

La popularidad de Python ha ido creciendo a lo largo de los años, en especial a partir de 2008 con Python 3.0, permitiendo que hoy en día se utilice en varios campos:

- Data analytics - Data Science
- Aplicaciones en Machine learning e inteligencia artificial
- Sistemas embebidos - Micropython
- Programación Web - Back end

¿Por qué Python?

La popularidad de Python ha ido creciendo a lo largo de los años, en especial a partir de 2008 con Python 3.0, permitiendo que hoy en día se utilice en varios campos:

- Data analytics - Data Science
- Aplicaciones en Machine learning e inteligencia artificial
- Sistemas embebidos - Micropython
- Programación Web - Back end

Intérprete de Python

- El intérprete traduce y ejecuta línea a línea código de alto nivel a diferencia de los compiladores en donde se traduce todo el programa para luego ejecutarlo
- Es más fácil depurar el código pero siempre se necesita del código fuente para ejecutar el programa
- Se puede descargar el último intérprete de Python desde https://www.python.org/downloads/

Intérprete de Python

- El intérprete traduce y ejecuta línea a línea código de alto nivel a diferencia de los compiladores en donde se traduce todo el programa para luego ejecutarlo
- Es más fácil depurar el código pero siempre se necesita del código fuente para ejecutar el programa
- Se puede descargar el último intérprete de Python desde https://www.python.org/downloads/

Intérprete de Python

- El intérprete traduce y ejecuta línea a línea código de alto nivel a diferencia de los compiladores en donde se traduce todo el programa para luego ejecutarlo
- Es más fácil depurar el código pero siempre se necesita del código fuente para ejecutar el programa
- Se puede descargar el último intérprete de Python desde https://www.python.org/downloads/

Ejecución de script de Python - Hola Mundo

Es posible ejecutar programas de python de 2 formas:

- Ejecutando las sentencias una a la vez a través de la consola
- Mediante scripts: archivos de texto que el interprete de Python ejecuta de una corrida

Ejecución de script de Python - Hola Mundo

Es posible ejecutar programas de python de 2 formas:

- Ejecutando las sentencias una a la vez a través de la consola
- Mediante scripts: archivos de texto que el interprete de Python ejecuta de una corrida

Ejecución de script de Python - Hola Mundo

Es posible ejecutar programas de python de 2 formas:

- Ejecutando las sentencias una a la vez a través de la consola
- Mediante scripts: archivos de texto que el interprete de Python ejecuta de una corrida

Ejecución de script de Python - Hola Mundo

Para abrir el interprete solo es necesario escribir *Python* en una consola.
Probemos las siguientes sentencias:

- 5+6
- 7*90
- print("Hola mundo")

Ejecución de script de Python - Hola Mundo

Generalmente, en python se ejecutan scripts:

- Abrir un archivo de texto
- Escribir print("Hola mundo")
- Guardarlo con el nombre *hola_mundo.py*
- Desde una consola escribir *Python hola_mundo.py*

Anaconda

Es un conjunto de herramientas que incluye interpretes actualizados de los lenguajes Python y R.

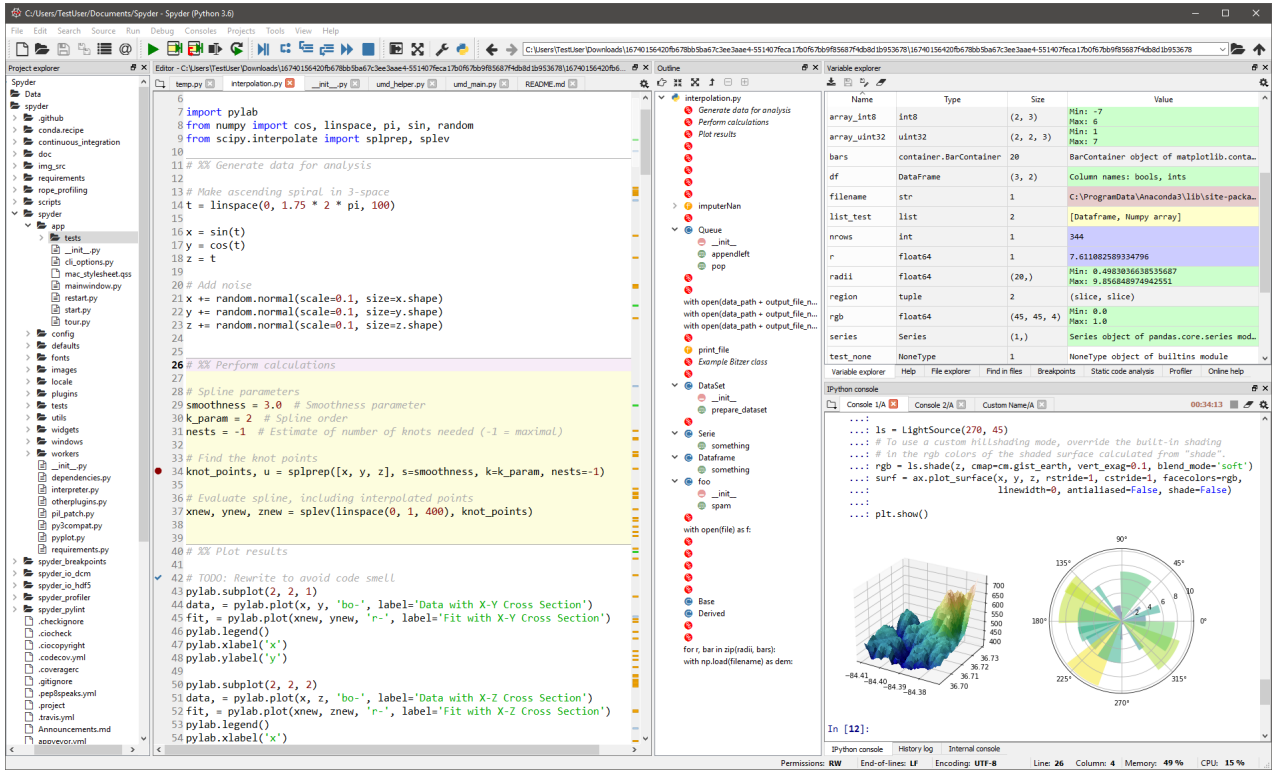
- Incluye bibliotecas científicas para el análisis de datos como numpy, scipy, matplotlib, etc.
- Incluye el IDE Spyder
- Incluye Jupyter Notebook
- Posee un manejador de ambientes de Python

Guía de instalación de Anaconda en linux

https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart

Anaconda: Spyder

Spyder es un IDE para desarrollo en Python con consola interactiva y explorador de variables.



Anaconda: Jupyter Notebook

- Jupyter Notebook es una aplicación web para desarrollo colaborativo en Python.
- Brinda la posibilidad de ejecutar código Python de forma remota.
- Tiene capacidad de escribir fórmulas utilizando LaTeX

Variables

Definición: Es el espacio de memoria designado para guardar un cierto tipo de información.

Python no requiere definir el tipo de dato de la variable.
El tipado dinámico nos permite reasignar una variable de un tipo a otra variable de un tipo distinto

Variables

Código

```
x = 5
y = 2.7
z = "FIUBA"
x = "LABI"
x = y
```

Tipos de datos

Los siguientes tipos de datos son soportados de forma nativa por el intérprete:

- **Boolean:** *True*, *False*
- **Entero:** 3, 15 , -6
- **Punto flotante:** 3.14159
- **Cadena de caracteres (*String*):** "Laboratorio ABIerto"

Tipos de datos

Los siguientes tipos de datos son soportados de forma nativa por el intérprete:

- **Tuplas:** Secuencia ordenada (indexada) de elementos inmutable (sus elementos no pueden ser reasignados).
  (1, 2.3, "LABI")
- **Listas:** Secuencia ordenada (indexada) de elementos mutable (sus elementos pueden ser reasignados).
  [1, 2.3, "LABI"]

En ambos casos se utiliza el operador [] para acceder a ellos.

Tipos de datos

Los siguientes tipos de datos son soportados de forma nativa por el intérprete:

- **Diccionarios:** Colección de pares *etiqueta - valores* (*key - values*).
  { 1: "LABI", "d": 3.14 }
- **Conjunto:** Colección no ordenada (y por lo tanto no puede ser indexada).
  {1,3,7,2}

Tipos de datos

Código

```
x = (1 , 5.23 , "LABI")
x[0]
y = [2 , -4, "LABI", 2.36]
y[2]
z = { 1: "LABI", "d": 3.14 }
z[1]
z["d"]
v = {1,3,5,0}
```

¿Qué pasa si tratamos de acceder a un elemento de un conjunto o si se repiten elementos dentro del mismo?

Comandos básicos

- **Comentarios:**
  - '#' : No se interpreta lo ingresado a continuación del carácter
  - '###' : Se utiliza para comentar bloques
- **Imprimir por pantalla:**
  - *print()* : Compatible con versiones de Python mayores a 3.0
  - *print* : Utilizada hasta la versión 2.7 de Python

Código

```
print("LABI")
print(22)
print "FIUBA"
print 3.14
```

Comandos básicos

- **Conocer el tipo de una variable:** *type()*

Código

```
x = 23
type(x) # Devuelve 'int'
```

- **Conocer la longitud de un string, lista o tupla:** *len()*

Código

```
x = "FIUBA"
len(x) # Devuelve 5
y = (1,3,"LABI",2.3)
len(y) # Devuelve 4
```

Comandos básicos

- **Los tipos enteros y punto flotante aceptan las operaciones:**
  - suma: +
  - resta: -
  - multiplicación: *
  - división: /
  - resto: %
  - modulo: *abs()*
  - redondeo: *round()*
  - potenciación: ** ó *pow()*
- El operador '+' también puede utilizarse con tipos *string* para concatenar.

Práctica I

Apliquemos lo aprendido hasta ahora:

- Sumar todos los elementos de una lista *lista* suponiendo que tiene *n* elementos e imprimir el resultado por pantalla. Utilizar variables e indexación de arreglos.
  Ejemplo: *lista* = [1,2,3,4,5,6] y *n* = 6
- Definir 1 diccionario que asigne las letras vocales minúsculas a números. Utilizar este diccionario para definir una variable que tenga el valor *"Hola mundo"* pero reemplazando las vocales por sus correspondientes valores.

Control de flujo

Python incluye varías estructuras para controlar el flujo de ejecución de un programa según el resultado de evaluar una condición de tipo boolean. En este curso veremos brevemente:

- Sentencia if
- Sentencia while
- Sentencia for

Operadores de comparación

Python permite los siguientes operadores para realizar operaciones de comparación que permiten evaluar una condición:

- Igual: ==
- No igual: !=
- Menor que: <
- Menor o igual que: <=
- Mayor que: >
- Mayor o igual que: >=

Operadores de comparación

Código

```
2 == 2 # Devuelve True
2 == 3 # Devuelve False
2 != 3 # Devuelve True
2 < 2 # Devuelve False
2 <= 2 # Devuelve True
```

Operador de pertenencia

El operador *in* y *not in* se utilizan para saber si un elemento pertenece o no a una lista, tupla, conjunto o diccionario (solo para etiquetas).

Código

```
5 in [1,2,3,4,5] # Devuelve True
5 not in [1,2,3,4] # Devuelve True
'5' in ('1','2','3','4',5) # Devuelve False
'5' in {'1':10 ,'5':4} # Devuelve True
5 in {1,2,5,7} # Devuelve True
```

Operador de pertenencia con diccionarios

Código

```
dicc = {'a':1, 'b':2, 3:'c', 4:'d'}
'a' in dicc.keys()
1 in dicc.keys()
2 in dicc.values()
'e' in dicc.values()
```

Sentencia If

Sentencia If: Permite evaluar una condición y que al devolver un valor de tipo boolean se siga un camino u otro.

Código

```
if 'condicA':
    # código en caso de que la condición A haya devuelto True
elif 'condicB':
    # código en caso de que la condición B haya devuelto True
.
.
else:
    # código en caso de que todas las condiciones anteriores hayan devuelto False
```

Sentencia While

Sentencia While: Permite ejecutar un conjunto de sentencias mientras una condición devuelva True

Código

```
while 'condic':
    # sentencias que se ejecuta si la condición devolvió True
```

Sentencia For

Sentencia For: Permite iterar sobre los elementos de una lista, tupla, diccionario o string.

Código

```
elem = [elem1, elem2, elem3, ....elemN]
for nombre_var_iterac in elem:
    # sentencias que se ejecutan a medida que se itera sobre
    # la lista. En cada iteración, nombre_var_iterac toma el
    # valor de un elemento de elem empezando por elem1
    # y terminando por elemN
```

Practica II

- La siguiente matriz tiene las notas de cursada de los alumnos de la materia *Introducción a la ingeniería electrónica*. Cada fila representa un alumno distinto y en las columnas se indica las notas de los distintos TPs. Se está buscando un ayudante de segunda para el próximo cuatrimestre por lo que se pide indicar qué alumnos obtuvieron al menos un 10 como nota. Para ese caso además indicar el promedio.

$$M = \begin{bmatrix} 7 & 8 & 5 & 6 & 7 \\ 9 & 5 & 3 & 7 & 6 \\ 5 & 6 & 10 & 7 & 7 \\ 10 & 10 & 3 & 4 & 10 \\ 9 & 9 & 6 & 6 & 7 \end{bmatrix}$$

Practica III

- La siguiente matriz tiene las notas de cursada de los alumnos de la materia *Introducción a la ingeniería electrónica*. Cada fila representa un alumno distinto y en las columnas se indica las notas de los distintos TPs. Se pide indicar que alumnos promocionan la materia. Teniendo en cuenta que una materia se promociona si ninguna de las notas es menor a 4 y el promedio es mayor a 7.

$$M = \begin{bmatrix} 7 & 8 & 5 & 6 & 7 \\ 9 & 10 & 3 & 7 & 10 \\ 5 & 6 & 10 & 7 & 7 \\ 10 & 10 & 3 & 4 & 10 \end{bmatrix}$$

Funciones

Definición: Subrutina con un objetivo definido que puede ser llamada desde otras partes del algoritmo que se esté programando.

- Python no requiere que se informe el tipo de lo que devuelve la función y los tipos de los parámetros que se ingresan.
- Se requiere una correcta indentación para que la función pueda ser interpretada. De otro modo el interprete devuelve un error.

Funciones

Código

```
def 'nombre_func'('param1', 'param2 = paramValue2',....):
    # Código correspondiente a la función
    # return valor_a_devolver
```

Funciones: Ejemplo

Código

```
def suma(x= 2, y = 3):
        return x + y
print(suma(1,9))    # Imprime 10
print(suma())       # imprime 5
```

Práctica IV

- Escribir una función que resuelva el problema de la práctica II. La función debe recibir una matriz.
- Escribir una función que resuelva el problema de la práctica III. La función debe recibir una matriz y 2 enteros con los que se va a comparar.

Ayuda

- ¿Qué devuelve el intérprete al escribir [0 for x in range(5)]
- ¿Qué realiza el intérprete al escribir res = [0 for x in range(5)]

Práctica IV

- Escribir una función que resuelva el problema de la práctica II. La función debe recibir una matriz.
- Escribir una función que resuelva el problema de la práctica III. La función debe recibir una matriz y 2 enteros con los que se va a comparar.

Ayuda

- ¿Qué devuelve el intérprete al escribir [0 for x in range(5)]
- ¿Qué realiza el intérprete al escribir res = [0 for x in range(5)]

Paquetes

- Código escrito por otros desarrolladores que podemos incluir en nuestros scripts.
- Facilita el desarrollo de aplicaciones ya que permite la reutilización de código generado para cumplir una función.
- Python permite incluir sólo una parte de un modulo.

Paquetes

Código

import 'nombre_paquete'
import 'nombre_paquete' **as** 'otro_nombre_paquete'
from 'nombre_paquete' **import** 'nombre_funcionalidad' **as** 'otro_nombre_funcionalidad'

Paquetes: OS

OS es un paquete que nos permite interactuar con el sistema operativo sobre el que se está ejecutando el script

Código

```
import os
os.getcwd()
os.getpid()
```

Práctica V

- Escribir un script que defina una función la cual recibe el nombre de una variable de entorno. La función crea un archivo de texto con el nombre de esa variable de entorno y escribe en su interior el contenido de la variable. Desde otro script importar la función y probar su funcionamiento con el parámetro *"PATH"*

Ayuda

- Usar documentación. ¿Cómo se manejan los archivos en Python? ¿Cómo se encara un problema en el que no tenemos herramientas para resolverlo? *open("nmb.txt",'w')*
- Las variables de entorno son herramientas utilizadas por el sistema operativo. os.getenv('PATH')

Práctica V

- Escribir un script que defina una función la cual recibe el nombre de una variable de entorno. La función crea un archivo de texto con el nombre de esa variable de entorno y escribe en su interior el contenido de la variable. Desde otro script importar la función y probar su funcionamiento con el parámetro *"PATH"*

Ayuda

- Usar documentación. ¿Cómo se manejan los archivos en Python? ¿Cómo se encara un problema en el que no tenemos herramientas para resolverlo? *open("nmb.txt",'w')*
- Las variables de entorno son herramientas utilizadas por el sistema operativo. os.getenv('PATH')

Clases y objetos

- Son conceptos que pertenecen al paradigma de la programación orientada a objetos.
- Permiten modelar entidades del mundo real que son capaces de interactuar entre sí.
- La programación orientada a objetos es una forma de pensar el desarrollo de aplicaciones distinto a la programación estructurada.

Clases y objetos

Clase: Representación de una entidad del mundo real que posee propiedades y métodos.

Ejemplo de clase: Alumno_de_Fiuba

- Propiedades:
  - Nombre : String
  - Apellido : String
  - Padrón : Entero
- Métodos
  - cursar_materia
  - tomar_apuntes

Clases y objetos

__init__ Es la función que se ejecuta al momento de crear el objeto. El primer argumento siempre es el propio objeto.

Código

```
class alumno_Fiuba:
    def __init__(self, _nombre, _apellido, _padron):
        self.nombre = _nombre
        self.apellido = _apellido
        self.padron = _padron
    def imprimirt_padron(self):
        print(self.padron)
```

Clases y objetos

Objetos : Es la realización de una clase.

Ejemplo de un objeto de la clase Alumno_de_Fiuba:

- alumno.Nombre = "Franco"
- alumno.Apellido = "Nastasi"
- alumno.Padron = 100002

Clases y objetos

Código

```
alumno =alumno_Fiuba("Franco", "Nastasi",100002)
print(alumno.nombre)
print(alumno.apellido)
print(alumno.padron)
alumno.padron = 97825
print(alumno.padron)
alumno.imprimir_padron()
```