

# Python Inicial

---

Introducción al lenguaje de programación Python 3

# Presentémonos !

---

- Nombre
- Experiencia
- Estudios

# Instalando Python en Windows

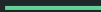
---

Y cómo setear la variable de entorno **path**

# Hagamos un “Hola Mundo” en Python.

Primero lo primero.

- abrir una consola de comandos.
- ingresar el comando python.
- mostrar por pantalla el texto “Hola Mundo”



# Características principales

---

conociendo un poco más este maravilloso lenguaje

# Python

Creado por Guido van Rossum  
en 1991

- Lenguaje **interpretado / scripting**
- tipado **dinámico**
- Fuertemente **tipado**
- **multiplataforma**
- **multiparadigma**

# Python

Lo bueno

- **sintaxis** sencilla y clara
  - Curva de **aprendizaje**
  - gran **comunidad** online
  - gran cantidad de **frameworks** y **librerías** para todo tipo de aplicaciones.
-



# Python

Lo malo

- Tiempos de ejecución
  - Python 2 o Python 3 ?
  - tipado dinámico
-

# Tipos básicos

---

Enteros, coma flotante, strings, bools y números complejos

# Tipos básicos

1. int
  2. float
  3. string
  4. bool
  5. complex
-

# Tipos básicos

a = 5

b = 2.13

c = "Hola"

d = True

e = 5+3j

1. Declarar las variables en el intérprete, aplicar la función **type()** a cada una de ellas y ver que devuelve.
2. Probar las siguientes sentencias:
  - a. `type(a) is int`
  - b. `type(b) is complex`
  - c. `type(c) is str`
  - d. `type(d) is bool`
  - e. `type(e) is float`

# Operaciones aritméticas

---

suma - resta - multiplicación - división - división entera -  
módulo - potencia

# Operaciones aritméticas

En un script de python, crear las siguientes variables:

a = 4

b = 3.14

c = "Soy un simple String"

d = False

e = 5 + 3j

luego, mostrar por pantalla las siguientes operaciones. Compruebe el **type** de cada resultado:

1. a + b

2. a \* b

3. b \* b

4. a \*\* a

5. a \*\* b

6. c + c

7. c \* a

8. a / 3

9. a // 3

10. a % 3

# Operaciones con strings

Python resuelve de manera muy simple las operaciones más comunes entre strings, por ejemplo:

1. **Concatenación:**

`"hola" + " mundo"`

2. **Casteo:**

`str(6), str(3+1j), str(True)`

3. **Multiplicación por un entero:**

`"hola " * 3`

# Operaciones con floats

Realizar la siguientes operaciones y analizar los resultados:

- $0.1 + 0.2$
- $a = 0$   
for i in range(0, 10):  
     $a = a + 0.1$



# Tipado dinámico en acción

a = 5

b = 3.14

c = "soy un texto"

1. Guardar en una variable **aux** el resultado de a + b
2. ejecutar type(aux)
3. ejecutar aux = c
4. ejecutar de nuevo type(aux)

# Operadores lógicos

---

and - or - not

# Operadores lógicos

Realizar las siguientes operaciones:

1. `not True`
2. `True and False`
3. `True or False`
4. `not True and not False`

luego, realizar estas:

1. `bool("")`
2. `bool("Soy un string")`
3. `bool(0)`
4. `bool(1)`
5. `5 and True`
6. `7 and False`
7. `"" and True`
8. `"soy un simple string" and True`
9. `"A" and "B"`

# lógica de cortocircuito

Realizar las siguientes operaciones:

1. `7/0`
2. `True or 7/0`
3. `False and 7/0`
4. `7/0 and False`

# Operadores de comparación

---

mayor - menor - igual - mayor o igual - menor o igual -  
distinto

# Operadores de comparación

Declarar las siguientes variables:

a = 16

b = 16.0

c = "a"

d = "abc"

Luego realizar las siguientes operaciones:

1. a == b

2. c == d

3. c > d

4. c < d

5. a > c

6. a == c

# controles de flujo

---

if - while - for

# if

el if no lleva paréntesis, se pueden agregar pero se recomienda no hacerlo

```
if condición :
```

```
    ...
```

```
else:
```

```
    ...
```

```
_____
```



# while

Lo mismo aplica para el while, la condición no debería estar encerrada entre paréntesis.

prestar atención a la indentación

`while` condición:

...

---

# for

El for en Python se escribe de una manera bastante peculiar.

**Notar** que i toma los valores entre [x,y)

```
for i in range(x, y):
```

```
...
```

---

# for y while

break y continue

- **break**
    - Con esta sentencia podemos forzar la terminación de un bloque
  - **continue**
    - nos permite saltar una iteración y continuar con la siguiente
-

# Controles de flujo

1. Mostrar por pantalla los primeros 100 múltiplos de 3. Primero usando un **for** y luego modificarlo usando un **while**.
2. Martín, un joven programador, está en camino a su habitación para irse a dormir. En el trayecto a su dormitorio se cruza con la cocina y esto le genera dos preguntas, si **tiene hambre** y si **tiene ganas de comer**. Crear un algoritmo que le permita decidir a Martín qué hacer:
  - a. Si **no** tiene hambre ni ganas de comer:  
“se va a dormir”
  - b. Si tiene hambre, pero **no** ganas de comer:  
“se prepara un té”
  - c. si **no** tiene hambre, pero **si** ganas de comer:  
“se come una mandarina”
  - d. si tiene hambre **y** ganas de comer:  
“Se prepara un sanguiche”

# Funciones

---

Como se declaran y cómo utilizarlas

# Docstrings

---

Documentando el código python

# Funciones

Sintaxis

```
def nombre_funcion(parametro):
```

```
...
```

```
...
```

```
return
```

---

# Docstrings

Documentando las funciones

```
def funcion_dos():
```

```
    """ esta funcion se encarga de  
    algo mucho muy importante """
```

```
    ...
```

```
    return 0
```

---



# Funciones

1. Crear una función que reciba dos parámetros, los compare y que devuelva un string que diga:
  - a. “el parámetro A es mayor al B”
  - b. “el parámetro A es menor al B”
  - c. “Son iguales”
2. Crear una función que reciba un parámetro y que retorne:
  - a. “el parámetro es un número” si el tipo del parámetro es **int**, **float** o **complex**
  - b. “el parámetro no es un número” si es de cualquier otro tipo
3. Crear una función que reciba un parámetro y que haga **absolutamente** nada.

# Filosofía Python

---

convenciones de nombres y otras yerbas

# Filosofía Python

- PEP-8: <https://www.python.org/dev/peps/pep-0008/>
- PEP-20: <https://www.python.org/dev/peps/pep-0020/>
- import this

# FAQ

<https://docs.python.org/3.7/faq/design.html>

# Ejercicios

1. Crear una función que reciba un número **n** y muestre por pantalla las primeras **n potencias** de dos.
2. Crear una función que reciba un parámetro **n** y muestre por pantalla los primeros **n múltiplos** de 3.
3. Crear una función que muestre por pantalla:

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*

- La altura del “árbol” viene dada por un parámetro **n**

1. Crear una función que calcule el **factorial** de un número **n**.
2. Crear una función que muestre por pantalla los primeros **n** términos de la sucesión de **fibonacci**.
3. crear una función que decida si un número es **primo**.
4. Crear una función que muestre por pantalla los primeros 100 números primos.

# Colecciones

---

Tuplas listas y diccionarios

# Operaciones

Las más comunes

- Agregar elementos
  - Quitar elementos
  - Obtener elementos
-

# Tuplas - ( )

Inmutables

- `tupla = (1, 2, 3)`
  - `3 in tupla`
  - `tupla.index(2)`
  - `tupla[0]`
-



# Listas - [ ]

Mutable

- `lista = [1, 2, 3]`
  - `3 in lista`
  - `lista.index(2)`
  - `lista[1]`
  - `lista[2] = "Hola"`
  - `lista.append("1")`
  - `lista.insert(0, 6)`
  - `del lista[1]`
-

# Diccionarios - { }

Clave - valor

```
dict = {  
    'clave1' : valor1,  
    'clave2': valor2  
}
```

obtener un valor

agregar un valor

modificar un valor

eliminar un valor

obtener claves, valores, items

# Slicing

---

Creando subconjuntos a partir de otras colecciones

# Slicing

Obteniendo subconjuntos

- `a = [1,2,3]`
  - `a[:]`
  - `a[0:2]`
  - `a[:2]`
-

# Funciones built-in

---

Funcionalidades útiles que ofrece Python

# Veamos algunas

Usando los tipos básicos:

- int
- str
- bool
- float

misceláneas:

- help
- type
- print
- input

Con las colecciones:

- list
- dict
- tupe

Operando sobre colecciones:

- max
- min
- len
- range
- sorted

# El listado completo

<https://docs.python.org/3/library/functions.html>

# Entrada y salida de archivos

---

creando, leyendo y escribiendo archivos en el SO



# Apertura

Modos más comunes:

**R**ead - **W**rite - **A**ppend

```
file = open("file.txt", modo)
```

---

# Lectura

Tengan en cuenta que leer el archivo implica un movimiento en el buffer.

- `read()`
  - `read(5)`
  - `readline()`
  - `readlines()`
  - `for line in file`  
`print(line)`
-

# Escritura

El modo **W** sobrescribe el archivo.

el modo **A** agrega al final el mismo.

```
write("un poco de texto")
```

---

# Cerrado

Como regla general:

Siempre que se hace un **open()**  
se debe hacer un **close()**

```
a = open("nombre.txt", "r")
```

```
...
```

```
a.close()
```

---

# Cláusula with

Operando de manera prolija con  
archivos

```
with open("file.txt", "r") as archivo:  
    archivo.readlines()
```

---

# Ejercicios de colecciones y archivos

1. Crear una función que reciba una **colección** como parámetro.  
La función deberá **escribir** al final de un archivo externo una línea por cada elemento dentro de la colección.
2. Crear una función que **lea** un archivo del sistema y devuelva un **diccionario** a partir del contenido del archivo.

El archivo debe tener la siguiente estructura:

```
clave1: valor1
clave2: valor2

...
```

- \* Usar la función **split( )** para separar claves y valores.
- \* Usar la función **replace( )** para limpiar los enters.

# Módulos en python

---

Importando funcionalidades, bibliotecas, etc.

# sintaxis

Cómo importar otros módulos

```
import math
```

```
from math import *
```

```
from math import sqrt
```

```
from math import sqrt as raizCuadrada
```

---



# Importando módulos

Cómo podemos importar  
módulos ubicados fuera del  
directorio actual

```
import sys
```

```
sys.path.insert(1, 'ruta\absoluta')
```

```
import modulo_externo
```

---

# Algunos módulos interesantes

Incluidos en Python:

- math
- random
- itertools
- sys
- datetime
- http
- html

No incluidos, pero de uso extensivo:

- matplotlib
- numpy
- Django

# pip

Instalando módulos desde el  
repositorio de Python

```
pip install matplotlib
```

---

# Listado completo de módulos nativos

<https://docs.python.org/3/py-modindex.html>

# Funciones parte 2

---

Parámetros opcionales - valores por defecto - argumentos por nombre

# Argumentos por nombre

pasando valores usando el **nombre** del argumento

```
def suma(primerο, segundo):
```

```
    return primerο + segundo
```

- suma(primerο=1, segundo=5)
  - suma(segundo=5, primerο=1)
  - suma(1,5)
  - suma(5,1)
-

# valores por defecto

Los valores por defecto convierten al parámetro en **opcional**.

```
def suma(primerο, segundo=8):
```

```
    return primerο + segundo
```

- suma(1, 6)
  - suma(1)
-

# cantidad arbitraria de valores

el argumento será una **tupla** con los valores pasados a la función

```
def sumatoria(*operandos):  
    total = 0  
  
    for operando in operandos:  
        total += operando  
  
    return total
```

---



# argumentos como diccionarios (keyword arguments)

el argumento será un **diccionario** con los valores pasados a la función.

```
def guia_telefonica(**registros)
    for persona in registros:
        print(registros[persona])
```

---

# ¿se pueden usar juntos?

claro que si!

```
def super_funcion(a, b, c, d="", e=5, *f, **g):  
    pass
```

---

# Valores de retorno

podemos devolver un **único**  
valor

```
def foo(bar):  
    return bar
```

---

# Valores de retorno

y también podemos devolver  
una **tupla** de elementos

```
def foo_2(a, b, c):  
    return a, b, c
```

---

# Ejercicios de funciones

1. Crear una función que reciba una cantidad **arbitraria** de parámetros y devuelva la **productoria** de los elementos. puede asumir que sólo se pasarán números.
2. Crear una función que reciba una cantidad **arbitraria** de parámetros y que cuente cuántos de ellos son números enteros.
3. Crear una función que reciba argumentos como **diccionario** y devuelva dos valores:
  - a. La cantidad total de elementos.
  - b. La sumatoria de todos los valores

\* asumir que los parámetros se pasarán de la siguiente forma: `funcion(a=1, b=2, c=3, ... etc)`

# Enumerate

---

O dándole un índice a cualquier colección.

# Cómo lo usamos

Enumerate es una función  
**built-in**

```
a = dict()
```

```
enumerate(a)
```

---

# Packing y unpacking

---

Separando una colección en variables independientes



# Packing

Lo usamos anteriormente al pasar una cantidad **arbitraria** de valores.

```
def funcion(*args):  
    print(args)
```

---

# Unpacking

Es como el packing, pero al revés

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

```
def funcion(p1, p2, p3, p4):
```

```
    pass
```

- `funcion(a)`
  - `funcion(*a)`
-

# Unpacking

Sirve para **listas** y **diccionarios**

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

```
b = {"Pedro": 1234, "Juan": 6789}
```

```
def func1(*args):
```

```
    pass
```

```
def func2(**args):
```

```
    pass
```

- func1(a)
  - func1(\*a)
  - func2(b)
  - func2(\*\*b)
-

# Unpacking

y las **listas**

```
a, b, c = [1, 2, 3]
```

```
def func():
```

```
    return "A", "B", "C"
```

```
d, e, f = func()
```

---

```
for a,b,*c in enumerate([1,2,3]):
```

```
....
```

¿ se acuerdan de este caso de la  
clase anterior ?

**PEP- 3132:**

<https://www.python.org/dev/peps/pep-3132/>

---

# Ejercicios de packing y unpacking

1. Crear la función **esPrimo**, que recibe un número entero **A** y devuelve True o False según corresponda.
  - a. Para saber si un número es primo, basta con ver si éste es divisible por algún número entre **[2, A)** ¿recuerdan esta notación ?
2. Crear la función **Promedio**, que recibe una cantidad arbitraria de números y devuelve el promedio de todos ellos.
3. Crear una función que reciba una cantidad **arbitraria** de números enteros, y devuelva tres valores:
  - a. la cantidad de números primos recibidos.
  - b. el promedio de los números recibidos.
  - c. la sumatoria total.

# Excepciones

---

creando un contexto y capturando excepciones

# Estructura

manejo de excepciones

```
try:
```

```
    ...
```

```
except:
```

```
    ...
```

```
else:
```

```
    ...
```

```
finally:
```

```
    ...
```

---



# Lanzando excepciones

Manejando el flujo del programa

```
raise(Exception("UPS"))
```

---

# Ejercicios de excepciones

1. En un loop infinito, solicitar al usuario que ingrese un valor por pantalla. Haciendo uso del manejo de excepciones, informar por consola:
  - a. “Usted ha ingresado un valor numérico” si el usuario ingresó un número entero.
  - b. “Usted ha ingresado un valor NO numérico” para cualquier otro caso.
  - c. A su vez, si el usuario escribe el texto “SALIR”, detener el loop y terminar el programa.

# distribuyendo aplicaciones

---

exportando a un .exe

# Módulo con interfaz gráfica

auto-py-to-exe

- pip install `auto-py-to-exe`
  - correr el comando:  
`auto-py-to-exe`
-

# Otras alternativas

La idea es generar un único .exe de fácil distribución.

Si nuestro programa tiene interfaz gráfica, considerar  
--noconsole

- pyinstaller

exportar a exe:

- python -m pip install pyinstaller

- pyinstaller --onefile miscript.py

- pyinstaller --noconsole miscript.py

- py2exe

---

# Scripting

---

algunas funcionalidades interesantes

# Argumentos del programa

como podemos pasarle  
argumentos a un script de  
python

```
python miScript.py Hola Manuel
```

---

# sys

módulo para interactuar con el  
intérprete

```
import sys
```

```
print(sys.argv)
```

---



# OS

librería para invocar operaciones  
del sistema operativo

```
import os
```

```
print( os.listdir() )
```

---

# Sqlite3

---

Python y bases de datos

# Importando la librería y conectándose a una base

```
import sqlite3
```

```
conn = sqlite3.connect("database.sqlite")
```

```
cursor = conn.cursor()
```

# Creando una tabla. Haciendo un commit.

```
cursor.execute("CREATE TABLE personas (TEXT nombre, NUMERIC edad)")
```

```
conn.commit()
```

# Realizando un INSERT. Commiteando los cambios.

```
personas = (
```

```
    ("Pablo", 30),
```

```
    ("Jorge", 41),
```

```
    ("Pedro", 27)
```

```
)
```

```
for nombre, edad in personas:
```

```
    cursor.execute("INSERT INTO personas VALUES (?, ?)", (nombre, edad))
```

```
conn.commit()
```

# Ejecutando un QUERY y obteniendo los datos.

```
cursor.execute("SELECT * FROM personas")
```

```
personas = cursor.fetchall()
```

```
print(personas)
```

Por último cerrar la conexión.

```
conn.close()
```

# HTTP - requests

---

Usando el módulo requests para interactuar con servicios web



# Descargar módulo requests

```
pip install requests
```

# Listado de APIs públicas

<https://github.com/public-apis/public-apis>

La que vamos a usar (CAT FACTS):

<https://cat-fact.herokuapp.com/facts/random>

# Haciendo un GET a la api

```
import requests
```

```
requests.get('https://cat-fact.herokuapp.com/facts/random')
```

Métodos **http** :

- **GET**
- **POST**
- DELETE
- HEAD
- PUT
- CONNECT
- OPTIONS
- PATCH
- TRACE

# interactuando con la respuesta de la API

`print(response)`                      # ¿ Qué **tipos** son response, response.text, etc?

`print(response.text)`

`print(response.headers)`

`print(response.json())`

# Validando el estado de la respuesta

```
response = response.get(...)
```

```
if response:
```

```
    print("Respuesta OK")
```

```
else:
```

```
    print("Ocurrió un error")
```

Posibles **status** en la respuesta del servidor:

- 1XX - Information
- 2XX - Success
- 3XX - Redirect
- 4XX - Client Error (Te la mandaste)
- 5XX - Server Error (Se la mandó el server)

# Documentación oficial

<http://es.python-requests.org/es/latest/>

# Tkinter

---

Creando una interfaz gráfica de usuario (GUI)

# Estructura básica

```
from tkinter import *
```

```
window = Tk()
```

```
window.title("Hello GUI World")
```

```
window.geometry('600x400')
```

```
window.mainloop()
```



# Agregando un label + botón

```
title = Label(window, text="GUI 1.0", font=("Arial Bold", 24))  
title.grid(column=0, row=0)
```

```
def clicked():  
    pass
```

```
btn = Button(window, text="Buscar", command=clicked)  
btn.grid(column=0, row=2)
```

# Agregando un área de texto

```
from tkinter import scrolledtext
```

```
...
```

```
txt = scrolledtext.ScrolledText(window,width=40,height=15)
```

```
txt.grid(column=1,row=1)
```

# Mostrando un alerta (messagebox)

```
from tkinter import messagebox
```

```
...
```

```
messagebox.showinfo('Message title', 'Message content')
```

# Documentación oficial

<https://docs.python.org/3/library/tk.html>

# Ejercicio integrador: requests + tkinter

Realizar una pantalla con tkinter que conste de un **botón** y un **área de texto** (más los **labels** que considere pertinentes).

Al presionar el botón, realizar una petición **GET** a la API de cats-facts usada anteriormente.

De la respuesta obtenida, nos interesan los campos **text** y **\_id**.

*\* Recordar que `response.json()` nos devuelve un diccionario con toda la información.*

Si la respuesta fue **errónea**, mostrar un alerta que indique que “ocurrió un error”.

Si la respuesta fue **correcta**, agregar en el **área de texto** y en un **archivo externo** la información obtenida con la siguiente estructura:

```
_id + ': ' + text + '\n'
```

*\* Por ejemplo: 591f98803b90f7150a19c238: In 1987 cats overtook dogs as the number one pet in America.*

*\* Para escribir en el área de texto: `sct.insert(END, texto)`*

# Eso fue todo !

---

Gracias por tomar el curso