

Introducción a Python

Cátedra Visión por Computadoras

Características de Python

Características de Python

- Simple
- Fácil de aprender
- Libre y de código abierto
- Lenguaje de alto nivel
- Portable
- Interpretado
- Orientado a objetos
- Extensible
- Embebible
- Con muchas librerías

Instalación y uso del intérprete

Ubuntu

```
sudo apt update && sudo apt install python3
```

- Abrir una terminal, por ejemplo gnome-terminal
- En la terminal escribir python3 y luego presionar enter

```
python3
Python 3.9.10 (main, Feb 22 2024, 13:54:07)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola Mundo")
Hola Mundo
>>>
```

- Para salir del intérprete presionamos
 - ▶ Ctrl + d
 - ▶ exit() y enter

Intérprete interactivo

IPython

- Existe un intérprete interactivo llamado IPython
- Características principales:
 - ▶ Resaltado de líneas y errores con colores
 - ▶ Autocompletado de variables, módulos y atributos usando el tabulador (Tab)

Para instalarlo ejecutamos los siguientes comandos

```
apt-get update && apt-get install ipython3
```

Lo ejecutamos desde una consola con:

```
ipython3
```

Eligiendo un editor

- gedit
- kate
- vim
- emacs
- PyCharm

Usando un archivo fuente

- Crear un archivo **hola.py**
- Agregarle la línea siguiente y guardarlo

```
print("Hola Mundo")
```

- Abrir una terminal
- Nos movemos al directorio con el comando (donde user es nuestro usuario):

```
mkdir /home/user/borrar  
cd /home/user/borrar/
```

- Ejecutamos el programa con

```
python hola.py
```

Práctica

- Instalar las librerías
- Instalar IPython
- Elegir un editor de texto
- Crear el programa **hola.py** que imprima “Hola Mundo”
- Ejecutarlo desde una consola

Conceptos Básicos

Comentarios

```
# Esto es un comentario  
print('Hola Mundo') # Esto es otro comentario
```


Conceptos Básicos

Números

- Tenemos dos tipos de números
- enteros, por ejemplo 2
- flotantes, por ejemplo 3.23, 2.45 o 23.5e-4
- ambos son inmutables

```
entero0 = 2  
entero1 = -2  
flotante0 = 3.23  
flotante1 = 23.5e-4
```

Conceptos Básicos

Cadenas (objeto de clase str)

- Comillas simples
- Comillas dobles
- Comillas triples
- También son inmutables

```
'Cadena con comillas simples'  
"Cadena con comillas dobles"  
'''Cadena con múltiples líneas  
y 3 comillas simples.'''  
"""Cadena con múltiples líneas  
y 3 comillas dobles."""
```

Conceptos Básicos

Cadenas - .format()

- Método usado para construir cadenas desde otra información.

```
edad = 20
nombre = 'JuanP'
print('{} tiene {} años'.format(nombre, edad))
print('Apenas {1} años tiene {0}'.format(nombre, edad))
```

JuanP tiene 20 años
Apenas 20 años tiene JuanP

```
print(nombre + ' tiene ' + str(edad) + ' años')
```

JuanP tiene 20 años

```
print('{0:.3f}'.format(1.0/3)) # imprime 0.333
```

0.333

Conceptos Básicos

Cadenas f-strings (introducidas en python 3.6)

- f-strings (o formatted string literals) son una forma moderna y más legible de formatear cadenas

```
edad = 20
nombre = 'JuanP'
mensaje = f"Hola, me llamo {nombre} y tengo {edad} años."
print(mensaje)
```

Hola, me llamo JuanP y tengo 20 años

```
a = 5
b = 10
print(f"La suma de {a} y {b} es {a + b}.")
```

La suma de 5 y 10 es 15.

Conceptos Básicos

Variables

- Son identificadores usados para guardar cosas y luego poder usarlas.

Convención de nombres

- El primer carácter debe ser una letra, minúscula o mayúscula
- El resto puede ser letras, números o el carácter _
- Son sensitivos a mayúsculas o minúsculas
- Ejemplos válidos: **i**, **E**, **hola_1_3**
- Ejemplos no válidos: **2i**, **hola-1**

Conceptos Básicos

Indentación

- Los espacios en blanco al inicio de una línea son importantes
- Esto se llama indentación
- Estos espacios definen un nivel de indentación y todo lo que esté en ese nivel forma un bloque que se ejecuta junto

```
i = 5
# Error en la siguiente línea!
# Notar un espacio al inicio de la línea
print('El valor es', i)
print('Repito, el valor es', i)
```

Si corremos el programa de arriba obtenemos un error parecido al siguiente:

```
IndentationError: unexpected indent
```

Conceptos Básicos

Indentación

- La indentación es similar al uso de las llaves de C/C++
- La recomendación oficial del lenguaje es usar 4 espacios para indentar

```
a = True
if(a is True):
    print('Es verdadero')
    a = False
else:
    print('No es verdadero')
    a = True
```

Operadores

Suma

3 + 5 = 8

'a' + 'b' = 'ab'

Resta

5 - 4 = 1

4 - 5 = -1

-4.3 = -4.3

Multiplicación

3 * 5 = 15

3 * 'b' = 'bbb'

Potencia

2 ** 3 = 8

División

13 / 3 = 4.33333333333

División y redondeo al entero menor

13 // 3 = 4

-13 // 3 = -5

9 // 1.8 = 4.0 *#igual que antes pero resultado en un float*

Módulo

13 % 2 = 1

Más Operadores

Shift a la izquierda

3 << 2 = 12

Shift a la derecha

16 >> 2 = 4

AND

3 & 5 = 1

OR

3 | 5 = 7

XOR

3 ^ 5 = 6

Muchos más Operadores

```
# Menor que
3 < 2  -> False
2 < 3  -> True

# Mayor que
3 > 2  -> True
2 > 3  -> False

# Menor o igual que
3 <= 2  -> False
2 <= 2  -> True

# Mayor o igual que
3 >= 2  -> True
3 >= 3  -> True

# Igual que
3 == 3  -> True

# Distinto que
3 != 3  -> False
```

Los últimos operadores

```
# not booleano
not True -> False
# and booleano
True and True -> True
True and False -> False
# or booleano
False or False -> False
True or False -> True
```

Operación y asignación en la misma sentencia

```
a = 2
a = a * 3 # (-> a = 6)
```

Esto puede ser escrito como:

```
a = 2
a *= 3 # (-> a = 6)
```

Operadores y Expresiones

Asociatividad

- Para asociar operaciones usamos paréntesis
- $2 + 3 + 4$
- $2 + (3 + 4)$

Control de Flujo

La sentencia if

- Se utiliza para chequear condiciones
- Las palabras reservadas que usa son: **if**, **elif** y **else**.

```
numero = 23
candidato = int(input('Ingrese un entero: '))

if candidato == numero:
    # El bloque comienza aquí
    print('Felicitaciones, adivinaste.')
    print('(pero no ganaste ningún premio!')')
    # El bloque termina aquí
elif candidato < numero:
    # Otro bloque
    print('No, es un poco mayor')
    # En un bloque se puede hacer lo que una quiera ...
else:
    print('No, es un poco menor')
    # Este bloque se ejecuta si: candidato > numero

print('Hecho')
# Esta última sentencia siempre se ejecuta,
# después de que el if se ejecute.
```

Control de Flujo

La sentencia while

- Permite ejecutar un bloque de código varias veces
- Puede tener una cláusula opcional **else**

```
numero = 23
intentos = 10

while intentos > 0:
    candidato = int(input('Ingrese un entero: '))
    intentos -= 1

    if candidato == numero:
        print('Felicitaciones, adivinaste.')
        # bucle while termina sin ejecutar el else
        break
    elif candidato < numero:
        print('No, es un poco mayor')
    else:
        print('No, es un poco menor')
else:
    print("Se agotaron los intentos") #Si no hubo break

print('Hecho')
```

Control de Flujo

El bucle for

- La sentencia **for**..**in** itera sobre una secuencia de objetos, o sea sobre cada elemento de la secuencia
- Una secuencia es una colección ordenada de elementos

```
for i in range(1, 5):  
    print(i)
```

```
1  
2  
3  
4
```

Funciones

Funciones

- Son piezas de código reutilizable
- Se definen con la palabra clave **def**

```
def hola_mundo():  
    # bloque perteneciente a la función  
    print('Hola Argentina')  
    # final de la función  
# llamada a la función  
hola_mundo()  
# llamada desde f-strings  
print(f'hola_mundo() }, yo de nuevo')
```

Hola Argentina

Hola Argentina, yo de nuevo

Parámetros de las funciones

- Los parámetros son variables que le pasamos a las funciones para que esta haga algo con ellos
- Se establecen entre los paréntesis de la declaración de la función
- Se separan con comas
- Los valores que pasamos como parámetros a la función se llaman argumentos

Funciones

```
def imprime_mayor(a, b):  
    if a > b:  
        print(a, 'es el mayor')  
    elif a == b:  
        print(a, 'es igual a', b)  
    else:  
        print(b, 'es el menor')  
  
# pasado de valores directamente  
imprime_mayor(3, 4)  
  
x = 5  
y = 7  
# paso de variables como argumentos  
imprime_mayor(x, y)
```

4 es el mayor

7 es el mayor

Valores por defecto de los argumentos

- Los valores por defecto se definen con el igual
- Cuando usamos la función estos parámetros pueden no estar definidos, en este caso se toman los argumentos por defecto
- Todos los parámetros con argumentos por defecto deben estar definidos al final de la lista de parámetros. Nunca pueden estar antes que parámetros sin argumentos por defecto.

```
def imprimir(mensaje, veces=1):  
    print(mensaje * veces)  
  
imprimir('Hola')  
imprimir('Argentina ', 5)
```

Hola

Argentina Argentina Argentina Argentina Argentina

Funciones

Keyword arguments

- Si tenemos funciones con muchos parámetros, podemos pasar los argumentos de los que usamos
- Para pasarlos usamos el nombre del parámetro y le damos el argumento que queremos
- También podemos cambiar el orden de los parámetros, siempre que usemos el nombre del parámetro

```
def func(a, b=5, c=10):  
    print('a es', a, 'y b es', b, 'y c es', c)  
  
func(3, 7)  
func(25, c=24)  
func(c=50, a=100)
```

```
a es 3 y b es 7 y c es 10  
a es 25 y b es 5 y c es 24  
a es 100 y b es 5 y c es 50
```

La sentencia return

- Se usa para salir de una función
- Y también para devolver un valor cuando volvemos de la función

```
def mayor(x, y):  
    if x > y:  
        return x  
    elif x == y:  
        return 'Los números son iguales'  
    else:  
        return y  
  
print(maximum(2, 3))
```

- Crear una función **adivinar** que permita adivinar un número generado en forma aleatoria
 - ▶ El número debe estar entre 0 y 100
 - ▶ Este número se genera adentro de la función
 - ▶ Además debe recibir un parámetro que sea la cantidad de intentos y en caso de que esta cantidad de intentos sea superada el programa debe terminar con un mensaje
 - ▶ Si el usuario adivina antes de superar el número de intentos máximo, se debe imprimir un mensaje con el número de intentos en los que adivinó
- Después de crear la función, llamarla en el mismo archivo
- Ejecutar el script desde la consola

Ayuda: código para generar un número aleatorio

```
import random  
numero = random.randint(0, 100)
```

Módulos

Módulos

- Los módulos sirven para reusar código ya escrito
- El método más simple para hacer un módulo es crear un archivo con la extensión **.py** con variables, funciones u objetos
- Para usar un módulo debemos importarlo

```
import sys

print('Los argumentos por línea de comando son:')
for i in sys.argv:
    print(i)
```

```
python3 usando_sys.py pasando 3 argumentos
Los argumentos por línea de comando son:
usando_sys.py
pasando
3
argumentos
```

Módulos

La sentencia from..import

- Sirve para importar en forma directa alguna variable, función u objeto que pertenece al módulo
- Nos evitamos escribir el nombre del módulo cada vez que usamos lo que importamos

```
import math
print("La raíz cuadrada de 16 es", math.sqrt(16)) #largo

from math import sqrt
print("La raíz cuadrada de 16 es", sqrt(16))      #corto
```


Módulos

Haciendo nuestros propios módulos

Creamos el archivo `mi_modulo.py`

```
def saluda():  
    print('Hola, este es mi_modulo.py saludando.')  
  
__version__ = '0.1'
```

Creamos el archivo `mi_modulo_demo.py`

```
import mi_modulo  
  
mi_modulo.saluda()  
print('Versión', mi_modulo.__version__)
```

```
Hola, este es mi_modulo saludando.  
Versión 0.1
```

Módulos

Haciendo nuestros propios módulos

```
def saluda():  
    print('Hola, este es mi_modulo.py saludando.')  
  
__version__ = '0.1'
```

Creamos el archivo mi_modulo_demo2.py

```
from mi_modulo import saluda, __version__  
  
saluda()  
print('Versión', __version__)
```

```
Hola, este es mi_modulo saludando.  
Versión 0.1
```

Módulos

Paquetes

- Los paquetes se utilizan para organizar los módulos
- Un paquete es solamente una carpeta con un archivo llamado `__init__.py`

```
- <cualquier carpeta>/  
  - ingenierias_en_UTN/  
    - __init__.py  
    - FRC/  
      - __init__.py  
      - electronica/  
        - __init__.py  
        - ingresantes.py  
    - FRSF/  
      - __init__.py  
      - electronica/  
        - __init__.py  
        - docentes.py
```

```
import ingenierias_en_UTN  
print(ingenierias_en_UTN.FRC.electronica.ingresantes.2024)  
  
from ingenierias_en_UTN.FRSF import electronica  
print(electronica.docentes.2024)
```

Estructuras de datos

- Son estructuras para guardar datos agrupados
- Son usados para guardar una colección de datos relacionados
- Hay 4 estructuras de datos que vienen por defecto en python (built-in)
 - ▶ lista (list)
 - ▶ tupla (tuple)
 - ▶ diccionario (dict)
 - ▶ conjunto (set)

Listas

- Es una lista ordenada de items
- Los elementos de la lista se separan entre comas
- La lista de items se encierran con corchetes

Estructuras de Datos

```
shoplist = ['manzana', 'mango', 'zanahoria', 'banana']

print('Tengo que comprar ', len(shoplist), 'productos')

print('Los productos son:', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\n También tengo que comprar arroz.')
shoplist.append('arroz')
print('Mi lista de compra es', shoplist)

print('Voy a ordenar la lista ahora ')
shoplist.sort()
print('La lista ordenada de compras es ', shoplist)

print(' El primer producto que voy a comprar es ', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('Ya compré la ', olditem)
print('Mi lista de compras ahora es ', shoplist)
```

Tengo que comprar 4 productos.

Los productos son: manzana mango zanahoria banana

También tengo que comprar arroz.

Mi lista de compras es ['manzana', 'mango', 'zanahoria', 'banana', 'arroz']

Voy a ordenar la lista ahora

La lista ordenada de compras es ['manzana', 'banana', 'zanahoria', 'mango', 'arroz']

El primer producto que voy a comprar es manzana

Ya compré la manzana

Mi lista de compra es ahora ['banana', 'zanahoria', 'mango', 'arroz']

Tuplas

- Son similares a las listas, pero con menos funcionalidad, por lo tanto son más livianas
- Son inmutables, o sea que no se pueden modificar
- Los elementos de la tupla se separan entre comas
- y se encierran con paréntesis, aunque son opcionales

Estructuras de Datos

```
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

# parentheses not required but are a good idea
new_zoo = 'monkey', 'camel', zoo
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
      len(new_zoo)-1+len(new_zoo[2]))
```

Number of animals in the zoo **is** 3

Number of cages in the new zoo **is** 3

All animals in new zoo are ('monkey', 'camel', ('python',
'elephant', 'penguin'))

Animals brought **from** old zoo are ('python', 'elephant', '
penguin')

Last animal brought **from** old zoo **is** penguin

Number of animals in the new zoo **is** 5

Diferencias entre tuplas y listas

- Las listas se pueden modificar, las tuplas no
- Las listas ocupan más lugar que las tuplas
- Las listas tienen métodos para agregar, insertar, modificar, eliminar y ordenar valores, las tuplas no

Diccionarios

- Asocian claves (keys) con valores (values)
- Las claves deben ser únicas en un diccionario, no deben repetirse, para poder recuperar el valor
- Las claves deben ser objetos inmutables (incluso tuplas)
- Los valores pueden ser cualquier tipo de objeto
- Los distintos pares clave:valor de un diccionario se separan con comas
- El par clave:valor se separa con el carácter :
- Los diccionarios se definen (encierran) entre llaves

Diccionarios

- Asocian claves (keys) con valores (values)
- Las claves deben ser únicas en un diccionario, no deben repetirse, para poder recuperar el valor
- Las claves deben ser objetos inmutables (incluso tuplas)
- Los valores pueden ser cualquier tipo de objeto
- Los distintos pares clave:valor de un diccionario se separan con comas
- El par clave:valor se separa con el carácter :
- Los diccionarios se definen (encierran) entre llaves

Para pensar/investigar: ¿Para qué serviría una tupla como clave de un diccionario?

Estructuras de Datos

```
# 'ab' is short for 'a'ddress'b'ook
ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}
print("Swaroop's address is", ab['Swaroop'])

# Deleting a key-value pair
del ab['Spammer']
print('\nThere are {} contacts\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'
if 'Guido' in ab:
    print("\nGuido's address is", ab['Guido'])
```

Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com

Contact Matsumoto at matz@ruby-lang.org

Contact Larry at larry@wall.org

Guido's address **is** guido@python.org

Secuencias

- Las listas, tuplas y strings son secuencias
- Pueden ser iteradas con un **for..in**
- Tienen test de membresía: **in** o **not in**
- Soportan operaciones de indexación
- Además soportan la operación de rebanar (slice)

Estructuras de Datos

```
shoplist = ['manzana', 'mango', 'zanahoria', 'banana']  
name = 'swaroop'
```

Indexing or 'Subscription' operation

```
print('Item 0 is', shoplist[0])  
print('Item 1 is', shoplist[1])  
print('Item 2 is', shoplist[2])  
print('Item 3 is', shoplist[3])  
print('Item -1 is', shoplist[-1])  
print('Item -2 is', shoplist[-2])  
print('Character 0 is', name[0])
```

Slicing on a list

```
print('Item 1 to 3 is', shoplist[1:3])  
print('Item 2 to end is', shoplist[2:])  
print('Item 1 to -1 is', shoplist[1:-1])  
print('Item start to end is', shoplist[:])
```

Slicing on a string

```
print('characters 1 to 3 is', name[1:3])  
print('characters 2 to end is', name[2:])  
print('characters 1 to -1 is', name[1:-1])  
print('characters start to end is', name[:])
```

Estructuras de Datos

```
Item 0 is manzana
Item 1 is mango
Item 2 is zanahoria
Item 3 is banana
Item -1 is banana
Item -2 is zanahoria
Character 0 is s
Item 1 to 3 is ['mango', 'zanahoria']
Item 2 to end is ['zanahoria', 'banana']
Item 1 to -1 is ['mango', 'zanahoria']
Item start to end is ['manzana', 'mango', 'zanahoria', '
    banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```


Conjuntos

- Es una colección no ordenada de objetos
- Son usados cuando importa la existencia de un objeto más que el orden o cuantas veces aparece
- Podemos hacer test de membresía, chequear si es un subconjunto o calcular intersección

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Referencias

- Cuando creamos una variable y le asignamos un objeto, el objeto es solo una referencia, no es el objeto
- La variable apunta a la dirección de memoria en donde se encuentra el objeto (binding)

Estructuras de Datos

```
print('Simple Assignment')
shoplist = ['manzana', 'mango', 'zanahoria', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist

# I purchased the first item, so I remove it from the list
del shoplist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that both shoplist and mylist both print
# the same list without the 'manzana' confirming that
# they point to the same object

print('Copy by making a full slice')
# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that now the two lists are different
```

Simple Assignment

```
shoplist is ['mango', 'zanahoria', 'banana']
```

```
mylist is ['mango', 'zanahoria', 'banana']
```

Copy by making a full **slice**

```
shoplist is ['mango', 'zanahoria', 'banana']
```

```
mylist is ['zanahoria', 'banana']
```

Práctica Listas

- Crear la siguiente lista:

$$\begin{bmatrix} 2 & 2 & 5 & 6 \\ 0 & 3 & 7 & 4 \\ 8 & 8 & 5 & 2 \\ 1 & 5 & 6 & 1 \end{bmatrix}$$

- Seleccionar el subarray [8 8 5 2].
- Poner la diagonal de la matriz en cero.
- Sumar todos los elementos del array.
- Setear los valores pares en 0 y los impares en 1.

Segmentando una imagen

- Crear un programa que lea un imagen en blanco y negro
- Aplique un umbral sobre los valores de los pixeles de la misma
- Guarde el resultado en otra imagen
- No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen

Segmentando una imagen

- Crear un programa que lea un imagen en blanco y negro
- Aplique un umbral sobre los valores de los pixeles de la misma
- Guarde el resultado en otra imagen
- No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen

Ayuda:

- Usar el template disponible en la filmina siguiente como base
- Antes de ejecutar el programa debemos instalar las OpenCV de la siguiente manera:

```
apt update && apt install libopencv-dev python3-opencv
```

Template

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('hoja.png', 0)

# Para resolverlo podemos usar dos for anidados
for row in img:
    for col in row:
        # Agregar código aquí

cv2.imwrite('resultado.png', img)
```