Python UNO: https://t.me/uno_py

python.org

python

gnuno.com.ar

/ignagarcia

/alexandermelian

ES UN LENGUAJE DE PROGRAMACIÓN...

01

DE ALTO NIVEL

Muy cercano al lenguaje humano

02

INTERPRETADO

El código se traduce a medida que se ejecuta

Esto lo vuelve tan sencillo y hermoso!

03

MULTIPARADIGMA

Imperativo,
Funcional y
Orientado a Objetos

04

CREADO POR
Guido van Rossum

Publicado en el 91

EN DONDE SE USA?

WEB

Con Frameworks como Flask y Django

MACHINE LEARNING

Con Bibliotecas como Tensorflow, Keras, ScyKit

DATA SCIENCE Y BIG DATA

Con Bibliotecas como Pandas, NumPy y Pydoop entre otras

APPS DE CONSOLA O TERMINAL

Muchas Bibliotecas para cada gusto


QUEREMOS EL CÓDIGO!

HOLA MUNDO!

TAN SIMPLE COMO...

```
print("Hola mundo!")
```

No más ;

Time to CODE!

1. Instalar py
2. Crear un algo.py
3. Ejecutar en consola py algo.py

ENTENDAMOS SU SINTAXIS

VARIABLES

Son de Tipado Dinámico, no necesitan palabras reservadas para declarar

```
saludo = "Hola mundo!" # str
saludo = 5 # int
```

COMENTARIOS

```
# Comentario de linea
'''
Comentario
Multilinea
3 comillas simples
'''
```

ENTENDAMOS SU SINTAXIS

OPERADORES

Matemáticos

```
+ - * / %
15 // 2 # division entera 7
3 ** 2 # potencia 9
```

Lógicos

```
and  or  not
```

De Comparación

```
< > == <= >= !=
```

De Pertenencia

```
"Hola" in "Hola Mundo" # True
"Chau" not in "Hola" # True
```


A PARTIR DE AHORA...
TENGAN CUIDADO CON LOS TABS, ES MUCHO MUY IMPORTANTE

ENTENDAMOS SU SINTAXIS

IF - ELSE - ELIF

No es obligatorio usar `( )`

```
if a == 5:
    print("es 5")
elif a == 3:
    print("es 3")
else:
    print("no se que es")
```

Los bloques se abren con `:` y se cierran cambiando la indentación

OJO CON LOS TABS!

PD: No existe el switch()

ENTENDAMOS SU SINTAXIS

BUCLES!

for

Ejecutará para cada elemento dentro del rango

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

for i in range(1, a, 2):
    print(i)

for i in "Hola!":
    print(i)
```

(desde, hasta, avance)

while

```
while a < 10:
    print(a)
    a += 1
```

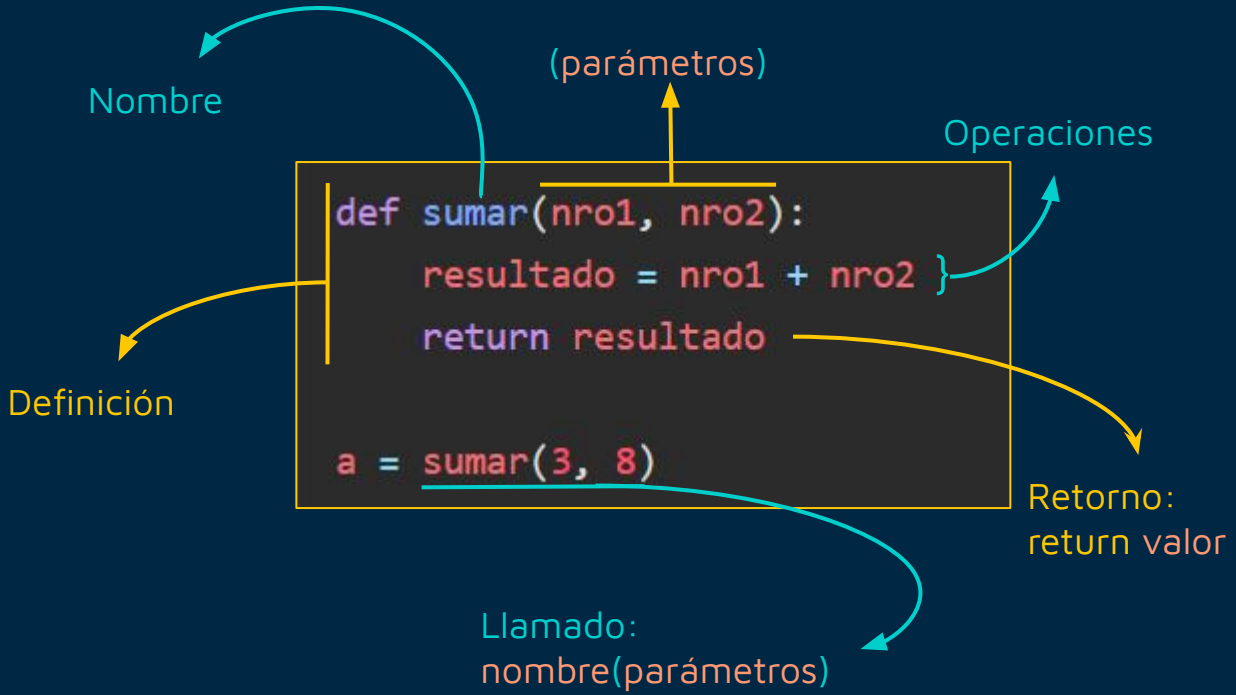
Siempre que la expresión de True se ejecutará

FUNCIONES

Bloque de código con un nombre, que puede recibir valores y devolver un valor

✓ **Divide y Vencerás:**
Permite separar un gran problema en muchos pequeños

✓ **Escribe solo 1 vez:**
Permite reutilizar el código



MANEJO DE STRINGS

Tamaño

```
cadena = "Hola mundo"
len(cadena) # 10
```

Operaciones

```
cadena + "!!!"
# Hola mundo!!!
```

Dividir en varias cadenas

```
cadena.split(" ")
# ['Hola', 'mundo']
```

Plantillas dinámicas

```
saludo = "Hola soy {}, tengo {} años"
saludo.format("Pepe", 22)
# Hola soy Pepe, tengo 22 años
```

las llaves se reemplazaran por
los parámetros de format

LISTAS

- Tiene orden
- Admite duplicados

Dentro de corchetes, separado por coma

```
lista = ["Hola", 24, 30.2, "Chau"]
```

```
len(lista) # 4
```

Cantidad de elementos

Obtener su valor

Índices desde 0 hasta len-1

```
lista[2] # 30.2

lista[1:2] # [24]
lista[1:] # [24, 30.2, 'Chau']
```

Obtener un subconjunto

```
lista.index(24) # 1
```

El índice con ese valor

LISTAS 2

Agregar Elementos

1 Elemento al final

```
lista.append("nuevo valor")
lista.extend(["Otra", "lista", 12])
lista.insert(2, "nuevo valor")
```

Se agrega el valor en el índice sin pisar

Varios elementos al final

Quitar Elementos

Quitar y guardar el último elemento

```
lista.pop() # Chau
lista.pop(1) # 24
```

Quitar y guardar el elemento de ese índice

TUPLAS

- Tiene orden
- Admite duplicados
- Es Inmutable!

Dentro de paréntesis, separado por coma

```
tupla = ("Hola", 24, 30.2, "Chau")
```

len, acceso y búsqueda igual que en listas

```
len(tupla) # 4
tupla.index(24) # 1
tupla[2] # 30.2
tupla[1:] # (24, 30.2, 'Chau')
```

Al ser inmutables no podemos agregar ni quitar elementos Peero...

Así podríamos extender la tupla

```
tupla2 = ("Nueva",)
tupla3 = tupla + tupla2
tupla3 # ('Hola', 24, 30.2, 'Chau', 'Nueva')
```

SETS

Fundamentado en Teoría de Conjuntos!

- Sin orden
- Sin duplicados!

Crear con llaves o set(lista o tupla)

```
set1 = set(["Hola", 24, 30.2, "Hola"])
# {'Hola', 24, 30.2}
set2 = {24, 30.2, "Hola", "Chau"}
```

```
len(set1) # 3
set1.add("Algo")
set1.discard("Algo")
set1.pop() # Al azar!
```

Algunas cosas no cambian

Otras si

Operaciones, se obtiene un nuevo set

```
set1.union(set2) # == set1 | set2
# {'Hola', 24, 30.2, "Chau"}

set1.intersection(set2) # == set1 & set2
# {24, 'Hola', 30.2}

set2.difference(set1) # == set1 - set2
# {'Chau'}
```

con _update se actualiza el set
ej: .union_update()

DICCIONARIOS

- Claves sin Duplicados
- Claves Inmutables

Crear con llaves o dict(keys=values)

```
dict1 = {"clave1": "valor1", 3: [7.8, "algo"], "clave1": "claverepe"}
# {'clave1': 'claverepe', 3: 7.8}
dict2 = dict(clave1="valor", clave2= 2, clave3= 7.8)
```

Si! como un array pero también con strings

```
len(dict1) # 2

dict1["clave1"] # 'claverepe'
dict1.get(3) # [7.8, "algo"]
dict1[3][0] # 7.8
```

Podemos eliminar al azar o seleccionando

```
dict1["clave3"] = "nuevo"

dict1.pop("clave3") # 'nuevo'
dict1.popitem() # al azar
```

DICCIONARIOS 2

Nos permiten iterar el Diccionario

pares clave-valor

solo claves

solo valores

```
items = dict1.items()
# dict_items([('clave1', 'valor1'), ...])
keys = dict1.keys()
# dict_keys(['clave1', ...])
values = dict1.values()
# dict_values(['valor1', ...])
```

```
for x in items:
    print(x)
```

por ejemplo..

Son "copias" por referencia!

osea...

```
print(values) # ['claverepe', 2, [7.8, 'algo']]
dict1["clave1"] = "nuevo"
print(values) # ['nuevo', 2, [7.8, 'algo']]
```

Y ESTOO!?!?

Los diccionarios son una forma de representar objetos, precisamente JSONs

Pero como dijimos al comienzo, python es **multiparadigma** y soporta objetos!

Por eso debemos utilizar las Clases para aprovechar los beneficios de POO

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

me = Person("Pepe", 22)
```

```
results= [
    {
      "final": True,
      "alternatives": [
        {
          "transcript": "Texto texto texto/// ",
          "confidence": 0.83
        }
      ]
    },
    {
      "final": True,
      "alternatives": [
        {
          "transcript": "Texto texto texto///",
          "confidence": 0.83
        }
      ]
    }
]
results[1]["alternatives"][0]["confidence"] # 0.83
```

OBJETOS

Definimos la clase con class Nombre:

init es nuestro constructor, aca definimos las propiedades

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return "Person({}, {})".format(self.name, self.age)

    def __str__(self):
        return "Nombre: {}\nEdad: {}".format(self.name, self.age)

    def cumpleanios(self):
        self.age += 1
```

self = objeto llamador

métodos de la clase, siempre el primer atributo es self

Crear/Instanciar

```python
me = Person("Pepe", 22)
```

usa __str__()

```python
print(me) # o str(me)
# Nombre: Pepe
# Edad: 22
print(repr(me))
# Person("Pepe", 22)
```

usa __repr__()

Llamar a método

```python
me.cumpleanios() # age = 23
```

Preguntas?

Comunidad Python UNO
https://t.me/uno_py

Comunidad Informática UNO
https://t.me/Informatica_UNO

Comunidad Algoritmia UNO
https://t.me/algoritmiaUNO