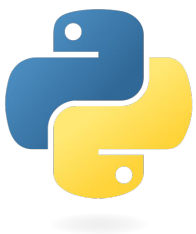


python™ Básico

Universidad Politécnica de Madrid

ETSI SISTE INFORMÁTI

CORE DUMPED_

Variables

En las variables guardamos valores.

```
x = 1
y = "Hello World"

x = y
print(x) # "Hello World"
```

String = Texto

```python
x = "Hello World"
x[0] # H
x[-1] # d
x[0:5] # Hello, no es necesario especificar ambos términos

for letra in x:
  print(letra) # Se puede recorrer las letras una a una

len(x) # Nos devuelve la longitud de la palabra

"Hello" (not) in x # Comprobamos si está o no en x
x.strip() # Elimina espacios inicio y final
x.replace('a', 'b') # sustituimos a por b
x.split(',') # separa por comas creando una lista
```

https://docs.python.org/3/library/stdtypes.html#string-methods

Listas

Almacenamos múltiples valores en ellas

```python
lista = ['Python', 'Java', 'C']
lista[1:] # [Java, C]
len(lista) # 3
'Python' in lista
lista.insert(2, 'Go') # ['Python', 'Java', 'Go', 'C']
lista.append('C++') # lista + [C++]
lista.remove('Java') # elimina la primera ocurrencia
lista.pop(1) # elimina el elemento en el índice indicado
['Python, 'C] sino el último
for i in range(len(lista)):
  print(lista[i]) # imprime los elementos uno a uno
lista.sort() # ordena los elementos alfabéticamente o de
menor a mayor
```

https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Tuplas

Lo mismo pero no se puede cambiar (valores o referencias)

```
tupla = (1, 2, 3)

tupla.count(2) # 1
tupla.index(1) # 0

a, b, c = tupla
```

https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences

Diccionario

```python
diccionario = {
  'alumna': 'Ana',
  'edad': 18,
  'curso': 'primero'
}
diccionario['alumna'] # Ana
diccionario.keys() # ['alumna', 'edad', 'curso']
diccionario.values() # ['Ana', 18, 'primero']
diccionario['edad'] = 19

'curso' in diccionario # True
diccionario['nota'] = 7.0
diccionario.pop('curso')

diccionario = {
  'alumna': {
    'nombre': 'Ana'
  },
  'alumno': {
    'nombre' : 'Alejando'
  }
}
diccionario['alumna']['nombre'] # Ana
```

Almacenamos pares clave-valor

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Condicionales

```
nota = 6

if nota < 0:
    print("Nota no válida")
elif nota >=5:
    print("Has aprobado :)")
else
    print("Has suspendido :(")
```

¿Qué pasa si saco un -1?

¿Y si he sacado un 10000?

¿Añadirías algo al código?

Bucles

Bucles

Bucles for / while

Tener cuidado con las modificaciones en listas, diccionarios mientras lo recorres y cambia los índices o la longitud.

Cuando usarlo:

- For: cuando sabes el número de iteraciones o te lo dice el elemento
- While: realmente en cualquier caso pero admitamoslo, el otro es más bonito :)

```python
for x in range(5):
  # range acepta varios parámetros (inicio, fin, paso)
  print(x)

lista = [1, 2, 3, 4]
for x in lista:
  print(x)


i = 0
while i < len(lista):
  print(lista[i])
  i+=1
  if i == 2:
    """
        Disclaimer: hay gente que considera muy mala práctica
        el uso de break y continue dentro de los bucles for y
        while, sin embargo si ayuda en la legibilidad usarlo.
    """
    break # Salimos del bucle

  if i == 1:
    continue # Para esta iteración y continua con la siguiente
```

Funciones

```python
def calcular_nota(nombre: str, nota: int) -> str:
    resultado = "suspendido"        # esto es un string

    if nota >= 5:                   # paso por copia
      resultado = "aprobado"

    return f"Hola, {nombre}, has {resultado}"

print(calcular_nota("Ana", 7))
```

Para estructurar y reutilizar código

Funciones - paso por copia

```python
def cambiar_nota(nota):
    nota = 10  # cambiamos el valor dentro de la función
    print(f"Nota dentro de la función: {nota}")

nota_original = 5
cambiar_nota(nota_original) # aquí saldría 10
print(f"Nota fuera de la función: {nota_original}") # aquí saldría 5
```

Funciona con tipos inmutables

(Los inmutables “no pueden cambiar su valor”)

Funciones - paso por copia

```python
def cambiar_nota(nota):
  nota = 10
  print(f"Nota dentro de la función: {nota}")

  return nota       # ojo!!!


nota_original = 5
nueva_nota = cambiar_nota(nota_original)
print(f"Nueva nota: {nueva nota}")      # ahora sí se cambia
```

```
Nota dentro de la función: 5
Nueva nota:                10
```

Funciona con tipos inmutables

(Los inmutables “no pueden cambiar su valor”)

Funciones - paso por referencia

```python
def modificar_lista(mi_lista):
    mi_lista.append(4)  # Modificamos la lista dentro de la función
    print(f"Lista dentro de la función: {mi_lista}")

# Lista original fuera de la función
mi_lista_original = [1, 2, 3]

# Llamamos a la función pasando la lista
modificar_lista(mi_lista_original)

# La lista original se ve afectada por el cambio
print(f"Lista fuera de la función: {mi_lista_original}")
```

```
Lista antes de la función:   [1, 2, 3]
Lista después de la función: [1, 2, 3, 4]
```

- **Objetos mutables** (listas, diccionarios, conjuntos) **se pasan por referencia**.
  - El cambio afecta al objeto en memoria dentro de una función.
  - ¡Cuidado con los ***scopes***!
- **Objetos inmutables** (enteros, cadenas, tuplas) **se pasan por copia**.
  - Si los modificas dentro de una función, el cambio **no** afecta al objeto original.

Clases y Objetos

Clase: plantilla para crear objetos.

- Ayuda a organizar tu código
- Guarda valores (atributos)
- Crea métodos para definir el comportamiento del objeto (funciones)

Objeto: son instancias de una clase:

- tienen sus propios atributos
- acciones que el objeto puede realizar

```python
class Person:
    age = 0

    def __init__(self, name, age):
        """
            Cuando creemos el objeto le podemos pasar parametros y guardarlos
              si queremos
        """
        self.name = name
        self.age = age

    def __str__(self):
        """
            Cuando hacemos print devolverá el formato que le indiquemos
        """
        return f"{self.name} tiene {self.age} años"

    def get_name(self): # Getter
        return self.name

    def set_name(self, nombre): # Setter
        self.name = nombre

persona = Person('Maria', 19)

print(persona) # Maria tiene 19 años
print(persona.get_name()) # Maria
persona.set_name('Lucía') # Cambiamos nombre a Lucía
print(persona.name) # Lucía
```

Sí, ya sé que no he explicado herencia muchach@s pero es que no me da tiempo para explicar todo

Excepciones

Capturan errores y permiten continuar el flujo de ejecución o detenerlo

```
try:
  print(3 / 0)
except:
  print("Ha ocurrido una excepción, continua la ejecución")

x = "Hello"

if not type(x) is int:
  # Esto para la ejecución
  raise TypeError('Solamente se permiten enteros')
```

Input / String Formatting

Tipos:

- Texto: str
- Números: int, float…
- list, tuple…
- dict
- bool
- …

```
x = input('Introduce un número: ')
y = input()

print(type(x)) # Es un string

x = int(x) # lo cambiamos a número
y = int(y) # la y también

print(x / y) # ya podemos realizar operaciones
```

Y sip, existen muchos más pero ya es mucho por hoy….

Librería Random

```python
import random

lista = [1, 2, 3]

random.randint(1, 3) # Devuelve un entero aleatorio entre [1, 3]

random.choice(lista) # Selecciona un elemento de la lista

random.shuffle(lista) # Mezcla aleatoriamente la lista, no devuelve nada
```

TEST

Necesito saber si el string x empieza por “Hoy voy a”

```
mi_string = "Hoy voy a comer lentejas"

resultado = x.startswith("Hoy voy a")
print(resultado)
```

```
mi_string = "Hoy voy a comer lentejas"
if "Hoy voy a" in mi_string:
    print("El string empieza por 'Hoy voy a'")
else:
    print("El string no empieza por 'Hoy voy a'")
```

```
import re

mi_string = "Hoy voy a comer lentejas"

if re.search(r"Hoy voy a", mi_string):
    print("El string empieza por 'Hoy voy a'")
else:
    print("El string no empieza por 'Hoy voy a'")
```

```
mi_string = "Hoy voy a comer lentejas"

if (mi_string[0] == "H" and
    mi_string[1] == "o" and
    mi_string[2] == "y" and
    mi_string[3] == " " and
    mi_string[4] == "v" and
    mi_string[5] == "o" and
    mi_string[6] == "y" and
    mi_string[7] == " " and
    mi_string[8] == "a"):
    print("El string comienza con 'Hoy voy a'")
else:
    print("El string no comienza con 'Hoy voy a'")
```

¿Cómo se importan las funciones de otros archivos en el mismo directorio?

```
form nombre_archivo import nombre_funciones
```

```
import nombre_archivo
```

```
from nombre_archivo import nombre_funcion as nombre
```

```
from nombre_archivo import *
```

```
import nombre_archivo.nombre_funcion
```


Comenta
tu código, plis

Problema

En un pueblo hay *n* personas, etiquetados de 1 a *n*. Hay un rumor de que uno de ellos es el juez del pueblo.

Si el juez existe:

1. El juez no confía en nadie
2. Todo el mundo (salvo el juez) confía en el juez
3. Solo hay una persona que satisface la 1 y la 2

Te damos una lista trust donde trust[i] = [a_i, b_i] representa que a_i confía en b_i. Si no existe la relación entonces no existe esa confianza.

Devuelve quién es el juez del pueblo y -1 si no existe

```
Input: n = 2, trust = [[1,2]]
Output: 2
Example 2:

Input: n = 3, trust = [[1,3],[2,3]]
Output: 3
Example 3:

Input: n = 3, trust = [[1,3],[2,3],[3,1]]
Output: -1

Constraints:

1 <= n <= 1000
0 <= len(trust) <= 104
len(trust[i]) == 2
All the pairs of trust are unique.
ai != bi
1 <= ai, bi <= n
```

[Beginner] Ahorcado

Pasos:

- Pensad en que vais a necesitar
- Definid la parte lógica
- Dividid el problema en trocitos

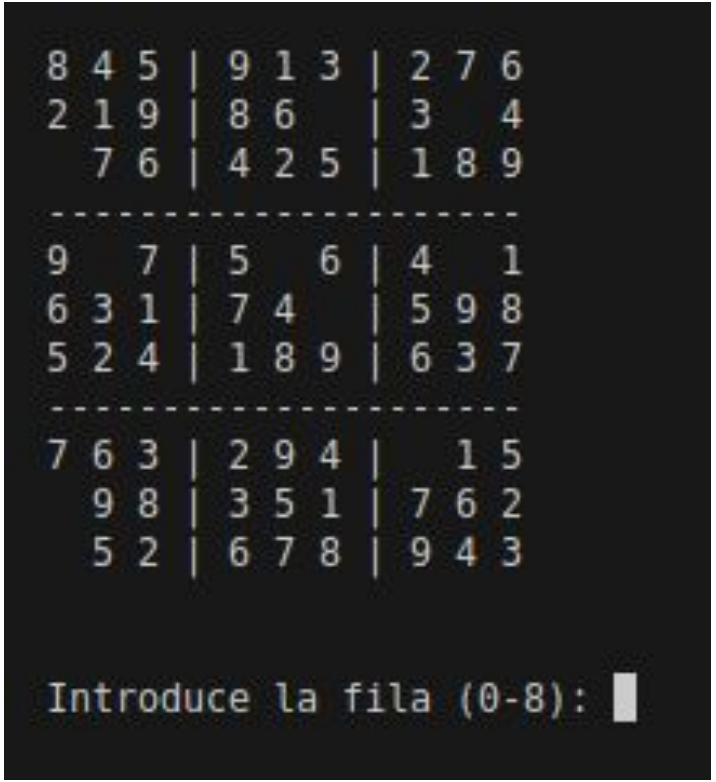
¡Mucho ánimo!

https://github.com/pandita-IA/TallerPython

[Challenge] Sudoku

Cosillas importantes:

- Lógica para creación del sudoku
- Comprobación de la creación



https://github.com/pandita-IA/TallerPython

[Challenge] Buscaminas

Cosillas Importantes:

- Expansión

https://github.com/pandita-IA/TallerPython