PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

*Una TUPLA es una **SECUENCIA INMUTABLE** de datos*

```
tupla_1 = (1, 2, 4, 8)
tupla_11 = 1, 2, 4, 8
print(tupla_1)
print(tupla_11)
```

(1, 2, 4, 8)
(1, 2, 4, 8)

Una tupla se escribe como una serie de valores ***entre paréntesis***. O ***separando*** los valores por ***comas***

```
contador = 3
lista = ['a','b','c']
tupla2 = (True, contador, lista, .28 *2)
tupla21 = True, contador, lista, .28 *2

print(tupla2)
print(tupla21)
```

(True, 3, ['a', 'b', 'c'], 0.56)
(True, 3, ['a', 'b', 'c'], 0.56)

Cada elemento de una tupla puede ser de ***diferente tipo,*** incluso ***variables o expresiones***

Una tupla vacía se crea con ***()***
ES NECESARIA COMA FINAL para crear tuplas de ***un solo elemento***

```
tupla_vacia = ()

tupla_un_elemento = (1, )
print(type(tupla_un_elemento))

tupla2_un_elemento = (1)
print(type(tupla2_un_elemento))

tupla3_un_elemento = 1.,
print(type(tupla3_un_elemento))
```

<class 'tuple'>
<class 'int'>
<class 'tuple'>

```
tupla = tuple("cadena")
print(tupla)

lista = [1,2,3]
tupla = tuple(lista)
print(tupla)

tupla = tuple(range(0,10,2))
print(tupla)
```

La función integrada ***tuple*** crea una tupla a partir de un iterable

('c', 'a', 'd', 'e', 'n', 'a')
(1, 2, 3)
(0, 2, 4, 6, 8)

```
tup = 1, 2, 3
a, b, c = tup
# desempaqueta la tupla en a, b y c
```

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

Se accede a los elementos de una tupla mediante ***índices*** o ***rangos*** como en las listas

```
tupla = (1, 10, 100, 1000)

print(tupla[0])
print(tupla[-1])
print(tupla[1:])
print(tupla[:-2])

for elem in tupla:
    print(elem)
```

```
1
1000
(10, 100, 1000)
(1, 10)
```

Las tuplas son elementos ***INMUTABLES***:
No se permite añadir, borrar o modificar sus elementos

```
tupla.append(10000)
del tupla[0]
tupla[1] = -10
```

AttributeError: 'tuple' object has no attribute 'append'
TypeError: 'tuple' object doesn't support item deletion
TypeError: 'tuple' object does not support item assignment

Se permite:

- ⇨ la función ***len(***tupla***)*** para obtener el número de elementos de la tupla
- ⇨ el operador **+** para concatenar tuplas
- ⇨ el operador ***** para multiplicar las tuplas
- ⇨ los operadores ***in*** y ***not in***
- ⇨ ***comparación tuplas*** elemento a elemento empezando por el índice 0

- ⇨ ***min(***tupla***)*** obtiene el menor elemento de la tupla
- ⇨ ***max(***tupla***)*** obtiene el mayor elemento de la tupla
- ⇨ ***tupla.count(***x***)*** devuelve el nº de apariciones de x en la tupla
- ⇨ ***tupla.index(***x***)*** devuelve índice de la 1ª aparición de x en la tupla

```
tupla = (1, 10, 100)

t1 = tupla + (1000, 10000)
t2 = tupla * 3

print(len(t2))
print(t1)
print(t2)
print(10 in tupla)
print(-10 not in tupla)
print(t1 < t2)
```

```
9
(1, 10, 100, 1000, 10000)
(1, 10, 100, 1, 10, 100, 1, 10, 100)
True
True
False
```

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

Un diccionario es conjunto de datos mutable y no secuencial.
Almacena pares de la forma:

{clave:valor, clave:valor, …, clave: valor}

- ⇨ cada clave debe de ser ***única***
- ⇨ una clave puede ser un dato de ***cualquier tipo*** pero no una lista
- ⇨ un diccionario ***no es*** una lista: ***NO GUARDA ORDEN DE SUS DATOS***
- ⇨ la función ***len***() aplica también para los diccionarios

```
agenda = {
    'Antonio': 666777888,
    'Vera': 678876654
}

print(agenda['Antonio'])                # Se accede a un valor mediante su clave
agenda['Vera'] = 555434343              # Si existe esa clave, modifica su valor
agenda['Pedro'] = 985554433             # Si no existe la clave, crea un nuevo par
agenda.update({'Elena': 985554433})     # Añade o modifica el par indicado
del(agenda['Antonio'])                  # Elimina el par de clave 'Antonio'
borrado = agenda.pop('Pedro')           # Extrae del diccionario el valor de esa clave
                                        # Se puede recuperar el valor
agenda.popitem()                        # Extrae del diccionario el último elemento
                                        # Se puede recuperar el valor
```

```
agenda.clear()        # elimina todos los elementos
print(len(agenda))    # salida 0

del agenda            # elimina el diccionario
print(agenda)         # error 'agenda' is not defined
```

Utilizar una clave inexistente provoca un ***error.***
Emplear los operadores ***in / not in*** ayuda a evitarlo

in -> Devuelve True si el valor está presente como clave en el diccionario
not in -> Devuelve True si el valor no está presente como clave en el diccionario

```
if 'Ana' in agenda:
    print('Ana', '->', agenda['Ana'])
```

```
diccionario = {
        'Antonio': 666777888,
        'Vera': 678876654,
        'Cuqui': 654665544
}
for key in diccionario.keys():
    print(key, "->", diccionario[key])
```

```
Antonio -> 666777888
Vera -> 678876654
Cuqui -> 654665544
```

El método ***keys()*** proporciona una ***lista*** con todas las claves del diccionario

```
for nombre, telefono in diccionario.items():
    print(nombre, "->", telefono)
```

```
Antonio -> 666777888
Vera -> 678876654
Cuqui -> 654665544
```

El método ***items()*** proporciona una ***lista de tuplas,*** cada tupla es un par clave, valor

```
for telefono in diccionario.values():
    print(telefono)
```

```
666777888
678876654
654665544
```

El método ***values()*** devuelve los valores del diccionario

```
for key in sorted(diccionario.keys()):
    print(key, "->", diccionario[key])

for nombre, telefono in sorted(diccionario.items()):
    print(nombre, "->", telefono)
```

```
Antonio -> 666777888
Cuqui -> 654665544
Vera -> 678876654
```

La función ***sorted()*** devuelve una ***lista*** ordenada con los elementos del iterable que se le pasa como parámetro

```
copia = terminos.copy()
# copia el diccionario terminos en copia
```