TEMA II

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"Un sueño no se hace realidad por arte de magia, necesita sudor, determinación y trabajo duro"
Colin Powell

Tipos de datos Mapas (diccionarios)

Son tipos de datos que nos permiten guardar valores, a los que se puede acceder por medio de una clave. Son tipos mutables y los campos no tienen asignado orden. El tipo de dato almacenado puede ser cualquiera.

Para declararlos utilizamos su constructor dict

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}

>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == == e
True
```

Yo suelo utilizar la segunda opción

```
listaCapitales={ "Francia":"París","Italia":"Roma","España":"Madrid"}
print(listaCapitales["Italia"])
```

Una vez definido el diccionario si ejecuto la instrucción midiccionario[clave]=valor, si la clave existe se modifica su valor y si no existe se añade un elemento nuevo con esas claves y valor.

Operaciones básicas

- len(): Devuelve número de elementos del diccionario.

```
>>> len(a)

3
```

- Indexación: Podemos obtener el valor de un campo o cambiarlo (si no existe el campo nos da una excepción KeyError):

```
>>> a["one"]

1

>>> a["one"]+=1

>>> a

{'three': 3, 'one': 2, 'two': 2}
```

- del():Podemos eliminar un elemento, si no existe el campo nos da una excepción KeyError:

```
>>> del(a["one"])

>>> a

{'three': 3, 'two': 2}
```

- Operadores de pertenencia: key in d y key not in d.

```
>>> "two" in a

True
```

Para copiar diccionarios vamos a usar el método copy():

```
>>> a = dict(one=1, two=2, three=3)

>>> b = a.copy()

>>> a["one"]=1000

>>> b

{'three': 3, 'one': 1, 'two': 2}
```

Las claves no tienen por qué ser de tipo string, todos los elementos (clave – valor) no tienen por qué ser del mismo tipo

```
mitupla=["España", "Francia", "Reino Unido","Alemania"]
midiccionario={mitupla[0]:"Madrid", mitupla[1]:"París", mitupla[2]:"Londres", mitupla[3]:"Berlín"}
print(midiccionario)
```

Puedo automatizar más este proceso

```
tPaises=["Francia","Alemania", "Italia"]
tCapitales=["Paris","Berlin","Roma"]
miDicc={}
for p,c in zip(tPaises,tCapitales):
    miDicc[p]=c
print(miDicc)
```

M[étodos principales de diccionarios]

dict1.clear dict1.get dict1.pop dict1.update

dict1.copy dict1.items dict1.popitem dict1.values

dict1.fromkeys dict1.keys dict1.setdefault

Para recorrer un diccionario podemos optar por recorrer claves, valores o ambos

```
for p in miDicc:
    print(p)
```

Recorro las claves

```
for p in miDicc:
    print(miDicc[p])
```

Recorro los valores

Puedo utilizar los métodos keys y values para determinar que quiero recorrer o bien el método items para obtener ambos a la vez (clave y valor)

miDicc.keys() me retorna las claves

miDicc.values() me retorna los valores

miDicc.items() retorna ambos valores

Ejemplo: Utiliza esos tres métodos para poner un ejemplo de como recorrer claves, valores y ambos.

Estos métodos me retornan objetos de tipo dictviews (itemViews, KeyViews..)

```
>>> dict1 = dict(one=1, two=2, three=3)
>>> i = dict1.items()
>>> i
dict_items([('one', 1), ('two', 2), ('three', 3)])
```

A este tipo de datos se les pueden aplicar las funciones len(), in

Los valores del diccionario pueden ser a su vez listas, tuplas o incluso otro diccioanrio.

Ejemplo: Declara un diccionario con al menos uno de sus valores siendo una tupla

```
midiccionario={23:"Jordan", "Nombre":"Michael", "Equipo":"Chicago", "anillos":{"temporadas":[1991,1992,1993,1996,1997,1998]}}
print(midiccionario["anillos"])
```