Tipando con la librería "Typing"

Hemos estado "tipando" las definiciones de nuestras funciones usando los tipos básicos que ofrece Python. No obstante, desde la versión 3.5 existe la posibilidad de "tipar" las namedtuples y las funciones mediante la librería typing, pero con un mero efecto documental. Realmente no tiene influencia en la ejecución de los programas más allá de haber escrito bien la sintaxis:

Requisito:

```
from typing import NamedTuple, Tuple, List, Set, Dict, Union
```

Sintaxis para NamedTuple:

Nombre=NamedTuple("nombre",[('nombre_campo1, tipo), ('nombre_campo2, tipo), ….])

Ejemplo, para definir las tuplas del tipo Vuelo:

```
Vuelo=NamedTuple("Vuelo",[('destino',str),('precio',float),('num_plazas',int),
        ('num_pasajeros',int),('código',str), ('fecha',date),('duración',int),
        ('hora',time),('velocidad',float), ('escalas',List[str]),
        ('económico',bool)])
```

Tipando con la librería “Typing”

Sintaxis para TUPLAS:

Tuple[tipo1, tipo2, tipo3,…]

Ejemplo, en el uso de Tuple:

```
def f1(tupla: Tuple[int,float,Tuple[str,int]])->None:
    print(tupla)

f1((1,2.3,("hola",4))) Produce como salida → (1, 2.3, ('hola', 4))
```

Tipando con la librería “Typing”

Sintaxis para LISTAS:

List[tipo]

Ejemplos, en el uso de List:

```
def f2(lista: List[int])->None:
    print(lista)

f2([1,2,3]) Produce como salida → [1, 2, 3]
```

```
def f3(lista: List[Tuple[int,float]])->None: //También se puede omitir Tuple
    print(lista)

f3([(1,2.3),(4,5.6)]) Produce como salida → [(1, 2.3), (4, 5.6)]
```

Tipando con la librería “Typing”

Sintaxis para CONJUNTOS:

Set[tipo]

Ejemplos, en el uso de Set:

```
def f4(conjunto: Set[int])->None:
    print(conjunto)

f4({1,2,3}) Produce como salida → {1, 2, 3}
```

```
def f5(conjunto:Set[Tuple[int,float]])->None:
    print(conjunto)

f5({(1,2.3),(4,5.6)}) Produce como salida → [(1, 2.3), (4, 5.6)]
```

Tipando con la librería “Typing”

Sintaxis para Diccionarios:

Dict[tipo de la clave, tipo del valor]

Ejemplos, en el uso de Dict:

```
def f6(dicc:Dict[int,str])->None:
    print(dicc)

f6({1:"Manolo",2:"Paula"}) Produce como salida → {1: 'Manolo', 2: 'Paula'}
```

```
def f7(dicc:Dict[int,Set[Tuple[str,float]]])->None:
    print(dicc)

f7({1:{("Manolo",82.1),("Paula",53.2)},2:{("Ana",50.3),("Lucas",77.7)}})

      Produce como salida →{1:{('Paula',53.2), ('Manolo',82.1)},
                            2:{('Ana', 50.3), ('Lucas',77.7)}}
```

Tipando con la librería “Typing”

Sintaxis para que un mismo campo pueda recibir más de un tipo diferente:

Union [tipo1,tipo2,…]

Ejemplos, en el uso de Union:

```
def f8(dato:Union[str,int]):
    print(dato)

f8("Hola")     Produce como salida →Hola
f8(25)         Produce como salida →25
```

Diccionarios

Esquema para la construcción de diccionarios "complejos", cuyos valores son: un máximo, un mínimo, una suma, un promedio, una lista ordenada, con un filtro sobre un contenedor y en general que necesiten de un "cálculo o similar"

Hemos aprendido a construir diccionario en el que los *valores* son contadores o una lista o conjunto, pero ahora se plantea una *operación adicional* sobre los valores.

Estos ejercicios se resuelven generalmente, de una forma eficiente, en dos pasos:

1. Construir *un primer diccionario* en el que los *valores* sean listas o conjuntos. *(ver las diapositivas anteriores)*
2. Construir *un segundo diccionario* a partir del diccionario del punto anterior que será recorrido con *"items()"*.
   - Las *claves* del *segundo diccionario* serán las del primero (no hay que comprobar si ya está o no en el segundo diccionario)
   - Los *valores* se obtienen realizando la *operación adicional* de que se trate.

Diccionarios

Ejemplo (diccionario "complejo"):

Supongamos una lista con tuplas con datos de estudiantes con su edad y un equipo al que pertenecen.

estudiantes=

[(Ismael,19,E4), (Ruben,18,E2), (Lorena,20,E2), (Rocío,18,E1), (M.Mar,19,E1), (David,18,E3), (Mario,20,E3), (Daniel,20,E2), (Javier,17,E1), (Daniel,19,E4), (Javier,18,E1), (Adrián,18,E4), (Javier,21,E2), (Celia,22,E1), (David,23,E3), (Mario,19,E4), (Rocío,18,E4), (Javier,19,E1), (Carlos,20,E2), (Guillermo,20,E2), (José,20,E3), (Luis,19,E1), (Javier,21,E4), (Fernando,20,E1), (Pedro,18,E3), (Ana,20,E1), (Manuel,18,E4), (Gonzalo,17,E3)]

Cada tupla se ha creado con: Estudiante=namedtuple ("estudiante","nombre, edad, equipo")

Enunciado: se pide un diccionario que, a cada equipo le haga corresponder los tres estudiantes de mayor edad. De cada estudiante se quiere conocer el nombre y la edad.

Si hay menos de tres, los que haya y si empatan, cualquiera de ellos.

Diccionarios

Paso 1. *Construcción del primer diccionario*: "a cada equipo le hacemos corresponder una lista con los estudiantes de dicho equipo". Como se quiere el nombre y la edad, los valores estarán formados por tuplas con dichos elementos.

Esquema 1

```
dic_aux=dict()
for e in estudiantes:
    if e.equipo not in dic_aux:
        dic_aux[e.equipo ]=[(e.nombre, e.edad)]
    else:
        dic_ aux[e.equipo ]+=[(e.nombre, e.edad)]
```

Esquema 2

```
dic_aux=dict()
for e in estudiantes:
    if e.equipo not in dic_aux:
        dic_aux[e.equipo ]=list()  o [ ]
dic_ aux[e.equipo ]+=[(e.nombre, e.edad)]
```

El resultado es de este primer diccionario es:

{'E4': [('Ismael', 19), ('Daniel', 19), ('Adrián', 18), ('Mario', 19), ('Rocío', 18), ('Javier', 21), ('Manuel', 18)],
'E2': [('Ruben', 18), ('Lorena', 20), ('Daniel', 20), ('Javier', 21), ('Carlos', 20), ('Guillermo', 20)],
'E1': [('Rocío', 18), ('M.Mar', 19), ('Javier', 17), ('Javier', 18), ('Celia', 22), ('Javier', 19), **('Luis', 19)**, ('Fernando', 20), ('Ana', 20)],
'E3': [('David', 18), ('Mario', 20), ('David', 23), ('José', 20), ('Pedro', 18), ('Gonzalo', 17)] }

Diccionarios

Primer diccionario (*se visualiza en esta dispositiva para mejor comprensión del paso 2*)

{*'E4'*: [*('Ismael', 19)*, *('Daniel', 19*), *('Adrián', 18)*, *('Mario', 19)*, *('Rocío', 18)*, *('Javier', 21)*, *('Manuel', 18)*],
'E2': [*('Ruben', 18)*, *('Lorena', 20)*, *('Daniel', 20)*, *('Javier', 21)*, *('Carlos', 20)*, *('Guillermo', 20)*],
'E1': [*('Rocío', 18)*, *('M.Mar', 19)*, *('Javier', 17)*, *('Javier', 18)*, *('Celia', 22)*, *('Javier', 19)*, ***('Luis', 19)***,
('Fernando', 20), *('Ana', 20)*],
'E3': [*('David', 18)*, *('Mario', 20)*, *('David', 23)*, *('José', 20)*, *('Pedro', 18)*, *('Gonzalo', 17)*] }

Paso 2. *Construcción del segundo diccionario*: Mantenido las mismas *claves* realizamos la *operación adicional* *(los 3 de mayor edad)* sobre los *valores* de dic_aux.

```
res=dict()
for clave, valor in dic_aux.items():
    res[clave]=sorted (valor, key=lambda e:e[1], reverse=True)[:3]
```

El resultado del ejercicio es :

res→ {*'E4'*: [*('Javier', 21)*, ('Ismael', 19), *('Daniel', 19)*], *'E2'*: [*('Javier', 21)*, *('Lorena', 20)*, *('Daniel', 20)*],
'E1': [*('Celia', 22)*, *('Fernando', 20)*, *('Ana', 20)*], **'E3'**: [*('David', 23)*, *('Mario', 20)*, *('José', 20)*]}

Diccionarios

Variante1 del ejercicio: Supongamos que en vez de nombre y edad sólo quisieran el nombre.

Es claro que, en el primer diccionario, los valores deben tener el nombre y la edad, para poder ordenar por esta última. En este caso, nos apoyamos en una *lista auxiliar*, de la que escogeremos el nombre, con el siguiente esquema:

```
res=dict()
for clave, valor in dic_aux.items():
    list_aux=sorted(valor, key=lambda e:e[1], reverse=True)[:3]
    res[clave]=[nombre for nombre, edad in list_aux]
```

El resultado del ejercicio es :

res→{'E4': ['Javier', 'Ismael', 'Daniel'], 'E2': ['Javier', 'Lorena', 'Daniel'], 'E1': ['Celia', 'Fernando','Ana'], 'E3': ['David', 'Mario', 'José']}

Diccionarios

Variante2 del ejercicio: Supongamos además que los nombres, lo quieren en *orden alfabético*

En este caso bastaría con ordenar la lista de *valores*.

```
res=dict()
for clave, valor in dic_aux.items():
    list_aux=sorted(valor, key=lambda e:e[1], reverse=True)[:3]
    res[clave]=sorted([nombre for nombre, edad in list_aux])
```

El resultado del ejercicio es :

res→{*'E4'*: [*'Daniel'*, *'Ismael'*, *'Javier'*], *'E2'*: [*'Daniel'*, *'Javier'*, *'Lorena'*], **'E1'**: [*'Ana'*, *'Celia'*, *'Fernando'*], **'E3'**: [*'David'*, *'José'* , *'Mario'*]}

Ejercicio:

Proyecto Vuelo:

Realice el enunciado ***T2023_11_14_Vuelos.pdf***

- Atención a partir de ahora para "**tipar**" las namedtuple usamos *NamedTuple* y para los contenedores *Tuple*, *List*, *Set*, *Dict*. Por ello, modifique la *namedtuple* Vuelo para adaptarla al uso de *NamedTuple*
- Ejercicios del 12 al 15