UFCD 10793 - Fundamentos de Python

Notebook 05 - Tuplos, Dicionários e Conjuntos - Python

Sílvia Martins

AEFHP 2022

Os `tuplos dicionários e conjuntos` são estruturas mais complexas e dotadas de funcionalidades predefinidas que facilitam a implementação em Python de soluções algorítmicas que implicam organização e agrupamento de dados.
Podem-se definir `tuplos de tuplos` , `listas de tuplos` , `dicionários com listas e tuplos` .

Tuplos

A manipulação de `tuplos` é muito parecida à das listas, sendo que os valores dos seus elementos `são imutáveis` . Estes valores iniciais mantém-se durante toda a execução do programa. Nos tuplos utilizam-se os `parentesis curvos ()` para delimitar os seus elementos ao invés dos parêntesis retos.
Os dados também podem ser heterogéneos quanto ao tipo. Os tuplos são aconselhados para a

Os tuplos são indicadas para dados heterogéneos, que contenham informações diferentes. Podemos dizer que a função das listas é ordenar dados numa sequência, enquanto que a dos tuplos é fornecer-lhes uma estrutura.

Síntaxe: `nome_tuplo=(valores iniciais separados por vírgulas)`

Um tuplo vazio representa-se por `t=()` , sendo t o nome do tuplo, ou ainda `tuple()`

Exemplos:

In [ ]:

```
a = (1, 2, 3, 4, 5)
a+a
```

In [ ]:

```
b = (6, 7)
a+b
```

In [ ]:

```
b*2
```

In [ ]:

```
t = 123, 543, 'Olá!'
t[0]
```

In [ ]:

```
# tuplo aninhados ou tuplo de tuplo
u = t, (8, 9)
u
```

In [ ]:

```
monthsOfYear = ('Jan', 'Feb', 'Mar', 'Apr', 'May','Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
print(monthsOfYear[0])
print(monthsOfYear[-1])
```

Considera o seguinte tuplo, que agrupa o `nº de um produto, o nome, opreço unitário, a taxa de IVA e a data da última atualização de stock` :

In [ ]:

```
prod = (100, 'Caneta de feltro n234', 1.20, 0.23, '12/03/2022')
print(prod)
```

Há ainda a possibilidade de nomear os elementos de um tuplo. Dessa forma, facilita a sua aplicação, pois permite o uso diretamente pelo índice. Isso é feito com a utilização de uma função chamada `namedtuple()` da biblioteca `collections` do Python. Para mais informação consulta o site [docs.python.org (https://docs.python.org/3/library/collections.html#collections.namedtuple)](https://docs.python.org/3/library/collections.html#collections.namedtuple) **Exemplos:**

Criar um tuplo usando referências a campos de um subtuplo (podíamos usar também uma lista):

Função integrada para tuplos com campos nomeados, com a seguinte sintaxe:

`collections.namedtuple(typename, field_names, *, rename=False, defaults=None, module=None)`

Retorna uma nova subclasse de tupla chamada typename. A nova subclasse é usada para criar objetos semelhantes a tuplas que possuem `campos` acessíveis por pesquisa de atributos, além de serem indexáveis e iteráveis. As instâncias da subclasse também possuem uma docstring útil (com typename e field_names) e um método **repr**() útil que lista o conteúdo do tuplo num formato name=value.

In [ ]:

```
from collections import namedtuple
Paises = namedtuple('Paises', ['sigla', 'nome'])
Pais = Paises('PT', 'Portugal')
print(Pais)
print(Pais.sigla)
print(Pais.nome)
```

Criar um tuplo de 3 dados - número, nome e categoria - sobre 4 funcionários:

Número	Nome	Categoria
100	Rui Silva	A
110	João Vaz	B
120	Ana Faia	C
130	Rita Martins	D

In [ ]:

```
Pessoal=()  #inicializar o tuplo
F=4
Num=100
Cat=65     #código ASCII do carater 'A'
for i in range (F) :
    Nome = input (f"Nome do funcionário {i+1} ->")
    Pessoal += (Num, Nome, chr(Cat))  # aqui os parêntesis são opcionais
    Num += 10
    Cat += 1
print(Pessoal)
```

Tuplos - empacotamento e desempacotamento

Os elementos de um tuplo podem ser acedidos de uma forma implícita na atribuição (`desempacotamento`)

In [ ]:

```
x,y=(9,10)
print(x)
print(y)
```

Um tuplo ser implicitamente criado apenas separando os elementos por vírgulas (`empacotamento`).

In [ ]:

```
67, 90
```

Os Tuplos permitem uma forma fácil de trocar o valor de 2 variáveis:

In [ ]:

```
x,y=(2,1)
x, y = y, x
print(x, y)
```

Iremos reformular o exemplo do pessoal de modo a proceder (registos):

- ao empacotamento dos dados num tuplo
- inserir cada tuplo numa lista
- imprimir a lista
- desenpacotar os dados de cada tuplo da lista

In [ ]:

```
Pessoal=()
ListaPessoal=[]
F=4
Num=100
Cat=65     #código ASCII do carater 'A'
for i in range (F) :
    Nome = input (f"Nome do funcionário {i+1} ->")
    c=chr(Cat)
    Pessoal = Num, Nome, c  # empacotamento dos dados no tuplo Pessoal
    ListaPessoal.append(Pessoal) #inserir cada tuplo na lista
    Num += 10
    Cat += 1
print(ListaPessoal)
for t in ListaPessoal :
    Nu, N, C  = t # desempacotamento dos dados no tuplo Pessoal
    print(f"número:{Nu} Nome: {N}  Cat: {C}")
```

Iremos reformular o exemplo do pessoal anterior de modo a imprimir cada tuplo e depois apenas o nome de cada pessoa:

- desenpacotar os dados de cada tuplo da lista

In [ ]:

```
print("Pessoal: ")
for t in ListaPessoal :
    print(t)
print("\nNome do pessoal:")
for t in ListaPessoal :
    Nu, N, C  = t # desempacotamento dos dados no tuplo Pessoal
    print(f"{N}")
```

Pode-se fazer de outra forma, recorrendo à posição de cada elemento no tuplo:

In [ ]:

```
print("Pessoal: ")
for t in ListaPessoal :
    print(t)
print("\nNome do pessoal:")
for i in range(len(ListaPessoal)) :
    print(f"{ListaPessoal[i][1]}")
```

In [ ]:

```
from collections import namedtuple
P= namedtuple("P",("Num", "Nome", "Cat"))
Fpt=()
F=4
N=100
c=65     #código ASCII do carater 'A'
for i in range (F) :
    No = input (f"Nome do funcionário {i+1} ->")
    Fpt += P(N, No, chr(c))  # empacotar
    N += 10
    c += 1
print(Fpt)
for i in range(0,len(Fpt),3):
    N, No, c = Fpt[i:i+3]
    print (N, No, c)
```

In [ ]:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

In [ ]:

```
#desconpactar
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Podemos usar a função *namedtuple* para, por exemplo ler registos de um ficheiro csv:

In [ ]:

```
import csv
from collections import namedtuple
RegistoEmp = namedtuple('RegistoEmp', ['nome','idade','cargo','departamento','salario'])
for emp in map(RegistoEmp._make, csv.reader(open("empregados.csv", "r"))) :
    print(emp.nome, emp.cargo)
```

Dicionários

Os dicionários (ou registos) são semelhantes a listas, mas com as posições nomeadas por um *referente* ou *chave* (`key`). O tipo dos dicionários é mutável, tal como o tipo das listas. Notacionalmente, usam-se chavetas para representar dicionários.

Recordar então que os dicionários são estruturas de dados com pares `<chave: valor>` , sendo que as chaves são únicas e cuja **sintaxe é:**

```
D={
Chave1: valor1,
Chave2: valor2,
...
ChaveN: valorN
}
```

In [ ]:

```
notas={'joao':10,'maria':14,'jack':9,'ana':18}
```

In [ ]:

```
notas['joao']
```

In [ ]:

```
notas.pop('joao')
```

In [ ]:

```
notas
```

In [ ]:

```
notas['john']=10
notas
```

As chaves, os valores, ou todo o conteúdo de um dicionário pode ser obtido usando, respctivamente, os métodos `keys` , `values` , ou `items` .

In [ ]:

```
notas.keys()
```

In [ ]:

```
notas.values()
```

In [ ]:

```
notas.items()
```

In [ ]:

```
for nome in notas.keys():
    if notas[nome]<10:
        print(nome+" RE")
    else:
        print(nome+" "+str(notas[nome]))
```

Criar um dicionário a partir de uma função:

In [ ]:

```
def f(x):
    return 2**x

def g(x):
    return x**2

dados={x:[f(x),g(x)] for x in range(6)}
dados
```

In [ ]:

```
for (i,[x,y]) in dados.items():
    print(x>y)
```

In [ ]:

```
#utilização de umdicionário para "seleção múltipla"

escalao=input("Insira a opção(A, B ou C): ").upper()
op={
    "A": 1000,
    "B": 900,
    "C": 850,
    }
print(op.get(escalao,'Opção não encontrada'))
print(f"Opção {escalao:2}: {op.get(escalao,0):4.2f}€")
```

Vale a pena rever e explorar os métodos específicos disponíveis para a manipulação de dicionários em W3Schools (https://www.w3schools.com/python/python_dictionaries_methods.asp)

Usar dicionários com tuplos:

- Criar um dicionário com tuplos constituídos pela descrição e preço.
- Imprimir os menus no ecrã
- Ordenar os tuplos pelo preço do menu (e converte o dicionario numa lista)
- imprimir nova lista
- imprimir nova lista formatada

Nota: Uma função lambda é uma pequena função anónima.Pode receber qualquer número de argumentos, mas apenas pode ter uma expressão.

In [ ]:

```
Menus={
    1:("Café e bolo de arroz", 1),
    2:("dois cafés e meia torrada", 4.5),
    3:("Meia de leite e tosta com manteiga", 3),
    4:("Galao com tosta mista", 5)
}
print("Menus disponíveis".center(45, "="))
for i in Menus :
    print(f"{i:<1}. {Menus[i][0].ljust(35)}{Menus[i][1]:>4} €")

print()
print("Menus disponíveis".center(45, "="))
Ordena=sorted(Menus.items(), key=lambda x:x[1][1], reverse=True) # ordena de acordo com o v
print(Ordena)
print()
for i in range (4):
    print (f"{Ordena[i][0]:<1} {Ordena[i][1][0]:<35} {Ordena[i][1][1]:>4} €")

print("Menus disponíveis".center(45, "="))
for i in Ordena :
    print (f"{i[0]:<1} {i[1][0]:<35} {i[1][1]:>4} €")

#imprimir o menu mais caro
print("\nMenu mais caro: ",Ordena[0][0], Ordena[0][1][0], Ordena[0][1][1],"€")
```

In [ ]:

```
#somar os valores dos preços dos menus
soma=0
for i in Menus :
    soma += Menus[i][1]

print(f"Soma dos valores dos menus: {soma:<1} €")
```

Conjuntos

Os conjuntos em *Python* (tal como é usual em matemática) correspondem a colecções de valores, sem ordem especificada, e sem duplicações. São tratados na linguagem de forma semelhante aos dicionários, mas sem chaves.São imutáveis e os seus elementos não podem ser indexados.Tem as mesmas propriedades que as definidas na teoria de conjuntos.

In [ ]:

```
a={0,1,2,0}
b=set()
c=set([3,4])

a
```

In [ ]:

```
c
```

In [ ]:

```
a.add(3)
a
```

Vale a pena explorar os métodos específicos disponíveis para a manipulação de conjuntos que, para além de `add` ilustrado acima, incluem as operações habituais sobre conjuntos como união, intersecção e complementação. Para isso consulta o site: realpython.com (https://realpython.com/python-sets/#operating-on-a-set) . Para visualizares os métodos e executa `dir(set)`

In [ ]:

```
a.intersection(c)
```

In [ ]:

```
dir(set)
```

Os conjuntos são também passíveis de definições por compreensão, como se ilustra de seguida.

In [66]:

```
{x for x in "as armas e os baroes assinalados" if x not in " "}
```

Out[66]:

```
{'a', 'b', 'd', 'e', 'i', 'l', 'm', 'n', 'o', 'r', 's'}
```

In [67]:

```
a={x for x in range (9)}
b={y for y in range (15) if y>3 and y<12}
c = {1, 50, 100}

print("a= ",a)
print("b= ",b)
print("c= ",c)
uniao=a|b
inter=a&b
dif=a-b
difs=a^b
difs3=a ^ b ^ c # operado da esquerda para a direira
print ("união de a e b = ", uniao)
print ("Intercecção de a e b = ", inter)
print ("Diferença de a e b = ", dif)
print ("Diferença simétrica de a e b = ", difs)
print ("Diferença simétrica de a,  b e c = ", difs3)
```

```
a=  {0, 1, 2, 3, 4, 5, 6, 7, 8}
b=  {4, 5, 6, 7, 8, 9, 10, 11}
c=  {1, 50, 100}
união de a e b =  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
Intercecção de a e b =  {4, 5, 6, 7, 8}
Diferença de a e b =  {0, 1, 2, 3}
Diferença simétrica de a e b =  {0, 1, 2, 3, 9, 10, 11}
Diferença simétrica de a,  b e c =  {0, 2, 3, 100, 9, 10, 11, 50}
```

Bibliografia

J. Carmo, A. Sernadas, C. Sernadas, F. M. Dionísio, C. Caleiro. (2014). Introdução à Programação em Mathematica* (3a edição). Lisboa:IST Press

Vasconcelos, J. (2015).Python - Algoritmia e Programação Web. Lisboa:FCA

Matthes. E.(2019). Python Crash Course - A Hands-On, Project-Based Introduction to Programming. San Francisco: USA

Carvalho. Adelaide. (2021). Práticas de Python - Algoritmia e programação. Lisboa:FCA

https://penseallen.github.io/PensePython2e/12-tuplas.html (https://penseallen.github.io/PensePython2e/12-tuplas.html)

https://docs.python.org/3/library/collections.html#collections.namedtuple (https://docs.python.org/3/library/collections.html#collections.namedtuple)

https://blog.betrybe.com/tecnologia/tuplas-em-python/ (https://blog.betrybe.com/tecnologia/tuplas-em-python/)

https://www.geeksforgeeks.org/namedtuple-in-python/ (https://www.geeksforgeeks.org/namedtuple-in-python/)

https://docs.python.org/3/library/csv.html#module-csv (https://docs.python.org/3/library/csv.html#module-csv)

https://docs.python.org/3/library/csv.html# (https://docs.python.org/3/library/csv.html#)