Taller 5

Métodos Computacionales para Políticas Públicas - URosario

Entrega: viernes 6-mar-2020 11:59 PM

Juan Sebastián Gómez

juansebastian.gomezm@urosario.edu.co

Instrucciones:

- Guarde una copia de este *Jupyter Notebook* en su computador, idealmente en una carpeta destinada al material del curso.
- Modifique el nombre del archivo del *notebook*, agregando al final un guión inferior y su nombre y apellido, separados estos últimos por otro guión inferior. Por ejemplo, mi *notebook* se llamaría: mcpp_taller5_santiago_matallana
- Marque el *notebook* con su nombre y e-mail en el bloque verde arriba. Reemplace el texto "[Su nombre acá]" con su nombre y apellido. Similar para su e-mail.
- Desarrolle la totalidad del taller sobre este *notebook*, insertando las celdas que sea necesario debajo de cada pregunta. Haga buen uso de las celdas para código y de las celdas tipo *markdown* según el caso.
- Recuerde salvar periódicamente sus avances.
- Cuando termine el taller:
  1. Descárguelo en PDF. Si tiene algún problema con la conversión, descárguelo en HTML.
  2. Suba los dos archivos (.pdf -o .html- y .ipynb) a su repositorio en GitHub antes de la fecha y hora límites.

(Todos los ejercicios tienen el mismo valor.)

1

Escríba una función que ordene (de forma ascedente y descendente) un diccionario según sus valores.

In [1]:

```
def ascendente (x):
    lista_a = list(x.items())
    lista_a.sort(key = lambda x :x[1])
    return lista_a

def descendente (x):
    lista_d = list(x.items())
    lista_d.sort(key=lambda x: x[1], reverse=True)
    return lista_d

dic = {'a':1,'b':2,'c':3,'d':6,'e':15, 'f':8}

def main(x):
    tipo_orden = input("Como quiere organizar su diccionario: ")
    print(tipo_orden)
    if tipo_orden == 'ascendente':
        return ascendente(x)
    elif tipo_orden == 'descendente':
        return descendente(x)
```

In [2]:

```
main(dic)
```

```
ascendente
```

Out[2]:

```
[('a', 1), ('b', 2), ('c', 3), ('d', 6), ('f', 8), ('e', 15)]
```

In [3]:

```
main(dic)
```

```
descendente
```

Out[3]:

```
[('e', 15), ('f', 8), ('d', 6), ('c', 3), ('b', 2), ('a', 1)]
```

2

Escriba una función que agregue una llave a un diccionario.

In [4]:

```
dic_1={}

def addkey(d,key, val):
    d[key]= d.get(key, val)
    return(d)
```

In [5]:

```
addkey(dic_1,'juan', 5)
```

Out[5]:

```
{'juan': 5}
```

In [6]:

```
print(dic_1)
```

```
{'juan': 5}
```

3

Escriba un programa que concatene los siguientes tres diccionarios en uno nuevo:

In [7]:

```
dicc1 = {1:10, 2:20}
dicc2 = {3:30, 4:40}
dicc3 = {5:50,6:60}
 # Resultado esperado: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

In [8]:

```
dic_nuevo = {}
def merge_dictionaries ():
    dic_nuevo.update(dicc1)
    dic_nuevo.update(dicc2)
    dic_nuevo.update(dicc3)
    return(dic_nuevo)
```

In [9]:

```
merge_dictionaries()
```

Out[9]:

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

In [10]:

```
print(dic_nuevo)
```

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

4

Escriba una función que verifique si una determinada llave existe o no en un diccionario.

In [11]:

```
dic_2={'a':2, 'b':4, 'c':6, 'd':8, 'e':10}
dic_3={'f':5,'g':13,'h':17, 'i':23, 'j':32}

# Se diseña una función para verificar la llave en diccionarios que estén cargados

def key_exists (d, key):
    if key not in d:
        print('Key not found')
    else:
        return 'key found'
```

In [12]:

```
#Se verifica si en el diccionario 2 existe la Llave 'c'
key_exists(dic_2,'c')
```

Out[12]:

```
'key found'
```

In [13]:

```
#Se verifica si en el diccionario 2 existe la Llave 'g'
key_exists(dic_2,'g')
```

```
Key not found
```

In [14]:

```
#Se verifica si en el diccionario 3 existe la Llave 'i'
key_exists(dic_3,'i')
```

Out[14]:

```
'key found'
```

In [15]:

```
#Se verifica si en el diccionario 3 existe la Llave 'a'
key_exists(dic_3,'a')
```

```
Key not found
```

5

Escriba una función que imprima todos los pares (llave, valor) de un diccionario.

In [16]:

```
dic_4={'a':2, 'b':4, 'c':6, 'd':8, 'e':10, 'f':5, 'g': 13, 'h':24, 'i':27}

def pares(d):
    for k,v in d.items():
        parejas = (k,v)
        print (parejas)
```

In [17]:

```
pares(dic_4)
```

```
('a', 2)
('b', 4)
('c', 6)
('d', 8)
('e', 10)
('f', 5)
('g', 13)
('h', 24)
('i', 27)
```

6

Escriba una función que genere un diccionario con los números enteros entre 1 y n en la forma (x: x**2).

In [18]:

```
n = int(input('Especifique un rango: '))
print(n)
d = {}

def gen_ent_dic(n):
    for x in range(1, n+1):
        v = x ** 2
        d[x] = v
    return d
```

```
7
```

In [19]:

```
gen_ent_dic(n)
```

Out[19]:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}
```

7

Escriba una función que sume todas las llaves de un diccionario. (Asuma que son números.)

c = {1 : 2, 2: 3} sum_llaves(c)

In [20]:

```
c = {1 : 2, 2: 3}
def sum_llaves(d):
    x = sum(d.keys())
    return x
```

In [21]:

```
sum_llaves(c)
```

Out[21]:

```
3
```

In [22]:

```
# Se usa la función con el diccionario 'd' del punto 6 para corroborar
sum_llaves(d)
```

Out[22]:

```
28
```

8

Escriba una función que sume todos los valores de un diccionario. (Asuma que son números.)

In [23]:

```
c = {1 : 2, 2: 3}
def sum_vals(dic):
    x = sum(dic.values())
    return x
```

In [24]:

```
sum_vals(c)
```

Out[24]:

```
5
```

9

Escriba una función que sume todos los ítems de un diccionario. (Asuma que son números.)

In [25]:

```
e ={2:1, 3:2, 4:5}
```

In [26]:

```
def sum_items(d):
    d_new = {sum_llaves(d): sum_vals(d)}
    return d_new
sum_items(e)
```

Out[26]:

```
{9: 8}
```

In [27]:

```
print(type(sum_items(e)))
```

```
<class 'dict'>
```

10

Escriba una función que tome dos listas y las mapee a un diccionario por pares. (El primer elemento de la primera lista es la primera llave del diccionario, el primer elemento de la segunda lista es el valor de la primera llave del diccionario, etc.)

In [28]:

```
lista_a = [2,4,6,8,10,12,14,16,18]
lista_b = [1,3,5,7,9,11,13,15,27]

# El diccionario se construye del tamañode x, porque es el número de llaves que hab
rán
# Se hizo con dos diccionarios de igual tamaño, porque el ejercicio no pregunta sob
re los casos donde las listas son de diferente tamaño
dic = {}
def join_lists_to_dic(x,y):
    for i in range (0, len(x)):
        dic[x[i]] = y[i]
    return dic
```

In [29]:

```
join_lists_to_dic(lista_a,lista_b)
```

Out[29]:

```
{2: 1, 4: 3, 6: 5, 8: 7, 10: 9, 12: 11, 14: 13, 16: 15, 18: 27}
```

In [30]:

```
print(dic)
```

```
{2: 1, 4: 3, 6: 5, 8: 7, 10: 9, 12: 11, 14: 13, 16: 15, 18: 27}
```

11

Escriba una función que elimine una llave de un diccionario.

In [31]:

```
dic_ex_11 = {'a':2, 'b':4, 'c':6, 'd':8, 'e':10, 'f':5, 'g': 13, 'h':24, 'i':27}

def elim_key(d,k):
        d.pop(k)
        return(d)
elim_key(dic_ex_11,'d')
```

Out[31]:

```
{'a': 2, 'b': 4, 'c': 6, 'e': 10, 'f': 5, 'g': 13, 'h': 24, 'i': 27}
```

In [32]:

```
# Corroborando que la llave fue eliminada del diccionario
print(dic_ex_11['d'])
```

```
---------------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
<ipython-input-32-b42cf2c55568> in <module>
      1 # Corroborando que la llave fue eliminada del diccionario
----> 2 print(dic_ex_11['d'])

KeyError: 'd'
```

12

Escriba una función que arroje los valores mínimo y máximo de un diccionario.

In [33]:

```
dic_ex_12 = {'a':2, 'b':4, 'c':6, 'd':8, 'e':10, 'f':5, 'g': 13, 'h':24, 'i':27}

def minimax(dicc):
    val_min = min(dicc.values())
    val_max = max(dicc.values())
    return val_min, val_max
minimax(dic_ex_12)
```

Out[33]:

```
(2, 27)
```

13

sentence = "the quick brown fox jumps over the lazy dog" words = sentence.split() word_lengths = [] for word in words: if word != "the": word_lengths.append(len(word))

Simplifique el código anterior combinando las líneas 3 a 6 usando list comprehension. Su código final deberá entonces tener tres líneas.

sentence = "the quick brown fox jumps over the lazy dog" words = sentence.split() word_lengths = [] for word in words: if word != "the": word_lengths.append(len(word))

In [34]:

```
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = [len(word)for word in words if word != "the"]
```

In [35]:

```
print(word_lengths)
```

```
[5, 5, 3, 5, 4, 4, 3]
```

14

Escriba UNA línea de código que tome la lista `a` y arroje una nueva lista con solo los elementos pares de `a`.

In [36]:

```
lista_a = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

In [37]:

```
lista_a_par = list(filter(lambda x: x%2 == 0, lista_a))
```

In [38]:

```
lista_a_par
```

Out[38]:

```
[2, 4, 6, 8, 10, 12, 14]
```

15

Escriba UNA línea de código que tome la lista `a` del ejercicio 14 y multiplique todos sus valores.

In [39]:

```
from functools import reduce

result = reduce(lambda x,y: x*y, lista_a)

print(result)
```

```
1307674368000
```

16

Usando "list comprehension", cree una lista con las 36 combinaciones de un par de dados, como tuplas: [(1,1), (1,2),...,(6,6)].

In [40]:

```
lista_combinaciones = [(x,y) for x in range(1,7) for y in range(1,7)]
```

In [41]:

```
print(lista_combinaciones)
```

```
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 1), (2, 2), (2,
3), (2, 4), (2, 5), (2, 6), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3,
6), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (5, 1), (5, 2), (5,
3), (5, 4), (5, 5), (5, 6), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6,
6)]
```