mcpp_taller5_john_caro

September 4, 2016

1 Taller 5

Métodos Computacionales para Políticas Públicas - URosario

Entrega: viernes 9-sep-2016 11:59 PM

[John Alexander Caro Becerra] [jhonalexbc@gmail.com]

1.1 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.)

1.1.1 1

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

```
In [123]: def orden(diccionario, descendente):
              '''
              descendente debe ser valores como False o True, en donde True indica
              '''
```

```
            ordenado = sorted(diccionario.values(), reverse = descendente)
            return ordenado
```

```
In [124]: sexo = {"Masculino": 1, "Femenino":2, "Indefinido":3}
```

```
In [125]: sexo
```

```
Out[125]: {'Femenino': 2, 'Indefinido': 3, 'Masculino': 1}
```

```
In [126]: orden(sexo,False)
```

```
Out[126]: [1, 2, 3]
```

```
In [127]: orden(sexo,True)
```

```
Out[127]: [3, 2, 1]
```

```
In [128]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '
```

```
In [129]: nums
```

```
Out[129]: {1: 'Bogotá', 2: 'Medellín', 3: 'Suma Caridad', 4: 'Azeroth', 5: 'Rasganc
```

```
In [130]: orden(nums,False)
```

```
Out[130]: ['Azeroth', 'Bogotá', 'Medellín', 'Rasganorte', 'Suma Caridad']
```

```
In [131]: orden(nums,True)
```

```
Out[131]: ['Suma Caridad', 'Rasganorte', 'Medellín', 'Bogotá', 'Azeroth']
```

1.1.2 2

Escriba una función que agregue una llave a un diccionario.

```
In [132]: def agregación(diccionario, llave_nueva, valor_llave_nueva):
              diccionario2 = diccionario
              diccionario2[llave_nueva] = valor_llave_nueva
              return diccionario2
```

```
In [133]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '
```

```
In [134]: agregación(nums, 6, "Reinos del Este")
```

```
Out[134]: {1: 'Bogotá',
           2: 'Medellín',
           3: 'Suma Caridad',
           4: 'Azeroth',
           5: 'Rasganorte',
           6: 'Reinos del Este'}
```

```
In [135]: agregación(nums, 7, "Kalindor")

Out[135]: {1: 'Bogotá',
           2: 'Medellín',
           3: 'Suma Caridad',
           4: 'Azeroth',
           5: 'Rasganorte',
           6: 'Reinos del Este',
           7: 'Kalindor'}

In [136]: nums

Out[136]: {1: 'Bogotá',
           2: 'Medellín',
           3: 'Suma Caridad',
           4: 'Azeroth',
           5: 'Rasganorte',
           6: 'Reinos del Este',
           7: 'Kalindor'}
```

1.1.3 3

Escriba un programa que concatene los siguientes tres diccionarios en uno nuevo: 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 [137]: dicc1 = {1:10, 2:20}
          dicc2 = {3:30, 4:40}
          dicc3 = {5:50,6:60}

In [138]: def pegar_dicc(diccionario1, diccionario2):
              diccionario = diccionario1.update(diccionario2)
              return diccionario

In [139]: pegar_dicc(dicc1, dicc2)
          pegar_dicc(dicc1, dicc3)

In [140]: dicc1

Out[140]: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

1.1.4 4

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

```
In [141]: def determinacion_dicc(diccionario, llave):
              diccionario = diccionario
              if diccionario.get(llave) == None:
                  print("No existe esta llave")
              else:
                  print("Existe esta llave y su valor correspondiente es:")
                  return diccionario.get(llave)
```

```
In [142]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '

In [143]: nums

Out[143]: {1: 'Bogotá', 2: 'Medellín', 3: 'Suma Caridad', 4: 'Azeroth', 5: 'Rasganc

In [144]: determinacion_dicc(nums, 8)

No existe esta llave
```

1.1.5 5

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

```
In [145]: def elementos_dicc(diccionario):
              diccionario = diccionario
              for i, h in diccionario.items():
                  print(i , h)

In [146]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '

In [147]: elementos_dicc(nums)

1 Bogotá
2 Medellín
3 Suma Caridad
4 Azeroth
5 Rasganorte
```

1.1.6 6

Escriba una función que genere un diccionario con los números enteros entre 1 y n en la forma (x: x**2). Ejemplo: n = 5 Resultado esperado: 1: 1, 2: 4, 3: 9, 4: 16, 5: 25

```
In [148]: def nums_cuadrado(num_final):
              diccionario = {}
              for j in range(1,num_final + 1):
                  diccionario[j] = j ** 2
              return diccionario


In [149]: n = 10
          nums_cuadrado(n)

Out[149]: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

1.1.7 7

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

```
In [150]: def sum_llaves_dicc(diccionario):
              from functools import reduce
              diccionario = diccionario
              llaves = list(diccionario.keys())
              suma = reduce((lambda x, y:x + y), llaves)
              return suma
```

```
In [151]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '
```

```
In [152]: nums
```

```
Out[152]: {1: 'Bogotá', 2: 'Medellín', 3: 'Suma Caridad', 4: 'Azeroth', 5: 'Rasganc
```

```
In [153]: sum_llaves_dicc(nums)
```

```
Out[153]: 15
```

```
In [154]: asdf = {1:0,50:0,51:0}
          sum_llaves_dicc(asdf)
```

```
Out[154]: 102
```

1.1.8 8

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

```
In [155]: def sum_valores_dicc(diccionario):
              from functools import reduce
              diccionario = diccionario
              valores = list(diccionario.values())
              suma = reduce((lambda x, y:x + y), valores)
              return suma
```

```
In [156]: sexo = {"Masculino": 1, "Femenino":2, "Indefinido":3}
```

```
In [157]: sexo
```

```
Out[157]: {'Femenino': 2, 'Indefinido': 3, 'Masculino': 1}
```

```
In [158]: sum_valores_dicc(sexo)
```

```
Out[158]: 6
```

```
In [159]: asdf = {"a":1, "b":2, "c":105, "d":2}
```

```
In [160]: asdf
```

```
Out[160]: {'a': 1, 'b': 2, 'c': 105, 'd': 2}
```

```
In [161]: sum_valores_dicc(asdf)
```

```
Out[161]: 110
```

1.1.9 9

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

```
In [162]: def sum_items_dicc(diccionario):
              from functools import reduce
              diccionario = diccionario

              valores = list(diccionario.values())
              suma_val = reduce((lambda x, y:x + y), valores)

              llaves = list(diccionario.keys())
              suma_llav = reduce((lambda x, y:x + y), llaves)

              suma_total = suma_val + suma_llav
              print("La suma de las llaves es: ",suma_llav),
              print("\nLa suma de los valores es: ",suma_val)
              print("\nLa suma Total es: ")
              return suma_total
```

```
In [163]: dicc= {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 10
```

```
In [164]: sum_items_dicc(dicc)
```

```
La suma de las llaves es:  55

La suma de los valores es:  385

La suma Total es:


Out[164]: 440
```

1.1.10 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 [165]: def mapeeando(lista1, lista2):
              diccionario = {}
              for a in range(len(lista1)):
                  llave = lista1[a]
                  valor = lista2[a]
                  diccionario[llave] = valor
              return diccionario
```

```
In [166]: lista1=["a","b","c"]
          lista2=[1,2,3]
          mapeeando(lista1, lista2)
```

```
Out[166]: {'a': 1, 'b': 2, 'c': 3}
```

1.1.11 11

Escriba una función que elimine una llave de un diccionario.

```
In [167]: def eliminacion(diccionario, llave):
              diccionario2 = diccionario
              del diccionario2[llave]
              return diccionario2

In [168]: nums = {1: "Bogotá", 2: "Medellín", 3: "Suma Caridad", 4: "Azeroth", 5: '

In [169]: nums

Out[169]: {1: 'Bogotá', 2: 'Medellín', 3: 'Suma Caridad', 4: 'Azeroth', 5: 'Rasganc

In [170]: eliminacion(nums,1)

Out[170]: {2: 'Medellín', 3: 'Suma Caridad', 4: 'Azeroth', 5: 'Rasganorte'}
```

1.1.12 12

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

```
In [171]: def min_max(diccionario):
              diccionario2 = diccionario
              minimo = min(list(diccionario2.values()))
              maximo = max(list(diccionario2.values()))
              print("El minimo y maximo del diccionario es: ")
              return minimo, maximo


In [172]: dicc= {1: 1, 2: 4, 3: 9, 4: 16, 5: 5985, 6: 36, 7: 49, 8: 64, 9: 81, 10:

In [173]: dicc

Out[173]: {1: 1, 2: 4, 3: 9, 4: 16, 5: 5985, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

In [174]: min_max(dicc)

El minimo y maximo del diccionario es:


Out[174]: (1, 5985)
```

1.1.13 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.

1.1.14 Primera forma: creando el verctor de número de letras de las palabras

```
In [175]: sentence = "the quick brown fox jumps over the lazy dog"
          words, word_lengths = sentence.split(),[]
          [word_lengths.append(len(word)) for word in words if word != "the"]

Out[175]: [None, None, None, None, None, None, None]

In [176]: sentence

Out[176]: 'the quick brown fox jumps over the lazy dog'

In [177]: words

Out[177]: ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

In [178]: word_lengths

Out[178]: [5, 5, 3, 5, 4, 4, 3]
```

1.1.15 Segunda forma: solo mostrando el número de letras en las palabras

```
In [179]: sentence = "the quick brown fox jumps over the lazy dog"
          words, word_lengths = sentence.split(),[]
          [len(word) for word in words if word != "the"]

Out[179]: [5, 5, 3, 5, 4, 4, 3]
```

1.1.16 14

```
In [180]: a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

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

```
In [181]: [x for x in a if x%2 == 0]

Out[181]: [4, 16, 36, 64, 100]
```

1.1.17 15

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

```
In [182]: from functools import reduce
          reduce((lambda x,y: x*y), a )

Out[182]: 13168189440000
```

1.1.18 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 [183]: lados = [1, 2, 3, 4, 5, 6]

In [184]: lados

Out[184]: [1, 2, 3, 4, 5, 6]

In [185]: combinaciones=[(dado_2,dado_1) for dado_1 in lados for dado_2 in lados]

In [186]: len(combinaciones)

Out[186]: 36

In [187]: type(combinaciones)

Out[187]: list

In [188]: type(combinaciones[0])

Out[188]: tuple

In [189]: combinaciones[0]

Out[189]: (1, 1)

In [190]: combinaciones

Out[190]: [(1, 1),
           (2, 1),
           (3, 1),
           (4, 1),
           (5, 1),
           (6, 1),
           (1, 2),
           (2, 2),
           (3, 2),
           (4, 2),
           (5, 2),
           (6, 2),
           (1, 3),
           (2, 3),
           (3, 3),
           (4, 3),
           (5, 3),
           (6, 3),
           (1, 4),
           (2, 4),
```

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