Cisco Networking Academy® Mind Wide Open™

Práctica de laboratorio: Realice el desafío de Python

Este es un ejercicio opcional para probar su conocimiento de los principios básicos de Python. Sin embargo, recomendamos fervientemente que el estudiante complete estos ejercicios para prepararse para el resto de este curso. Si no sabe cómo resolverlos, fíjese en las lecciones de Python disponibles en la carpeta de Materiales del curso/tutoriales y demostraciones.

Responda las preguntas o complete las tareas detalladas a continuación; utilice el método específico descrito, si corresponde.

1) ¿Cuánto es 3 a la potencia de 5?

In [1]:

```
# Code cell 1
3**5
```

Out[1]:

```
243
```

2) Cree una variable, *'s'*, que contenga la cadena "¡Este curso es increíble!". Con la variable, divida la cadena en una lista.

In [2]:

```
# Code cell 2
s="¡Este Curso es Increíble!"
s=s.split(" ")
print("\t",s,"\n")
```

```
	 ['¡Este', 'Curso', 'es', 'Increíble!'] 
```

3) Dadas las variables altura y montaña, use .format() para imprimir la cadena siguiente:

'La altura del Monte Everest es de 8848 metros'.

In [3]:

```
# Code cell 3
mountain = "Mt. Everest"
height = 8848
print("\t","La altura del {} es de {} metros".format(mountain,height),"\n")
```

```
	La altura del Mt. Everest es de 8848 metros
```

4) Dada la lista anidada siguiente, use la indexación para tomar la palabra '"esto"'.

In [4]:

```
# Code cell 4
lst = ['a','b',[4,10,11],['c',[1,66,['this']],2,111],'e',7]
print("\t",lst[3][1][2],"\n")
```

```
	['this']
```

5) Dado el diccionario anidado siguiente, tome la palabra "eso". Este ejercicio es un poco más difícil.

In [5]:

```
# Code cell 5
d = {'k1':['val1','val2','val3',{'we':['need','to','go',{'deeper':[1,2,3,'that']}]}]}
print("\t",d['k1'][3]['we'][3]['deeper'][3],"\n")
```

```
	that
```

6) ¿Cuál es la diferencia principal entre una tupla y una lista?

Las tuplas son estáticas y las listas dinámicas.

7) Cree una función, GetDomain(), que tome el dominio del sitio web de correo electrónico de una cadena en la forma: 'user@domain.com'.

Por ejemplo, el paso de "user@domain.com (mailto:user@domain.com)" daría: domain.com

In [6]:

```
# Code cell 6
def GetDomain(email):
    return email.split('@')[-1]
GetDomain("user@domain.com")
```

Out[6]:

```
'domain.com'
```

8) Cree una función básica, findInternet(), que dé una devolución de True si la palabra 'Internet' se incluye en la cadena de entrada. No se preocupe por los casos de perímetro como la puntuación que se asocia con la palabra, pero tenga en cuenta el uso de mayúsculas. (Sugerencia: vea https://docs.python.org/2/reference/expressions.html#in (https://docs.python.org/2/reference/expressions.html#in))

In [7]:

```
# Code cell 7
def findInternet(st):
    return 'internet' in st.lower().split()
findInternet('The Internet Engineering Task Force was created in 1986')
```

Out[7]:

```
True
```

9) Cree una función, countIoT(), que cuente la cantidad de veces que la palabra "IdT" aparece en una cadena. Ignore los casos de perímetro pero tenga en cuenta el uso de mayúsculas.

In [8]:

```
# Code cell 8
def countIoT(st):
    count=0
    for word in st.lower().split():
        if word== 'iot' :
           count +=1
    return count
countIoT('I don\'t know how to spell IoT ! Is it IoT or iot ? What does iot mean anyway?
```

Out[8]:

```
4
```

10) Utilice las expresiones lambda y la función filter() para filtrar las palabras de una lista que no comiencen con la letra 'd'. Por ejemplo:

```
sec = [“datos”, “sal”, “diario”, “gato”, “perro”]
```

debe ser filtrado a:

```
['datos', 'diario']
```

In [9]:

```
# Code cell 9
seq = ['data','salt' ,'dairy','cat', 'dog']
list(filter(lambda word: word[0]=='d',seq))
```

Out[9]:

```
['data', 'dairy', 'dog']
```

11) Utilice las expresiones lambda y la función map() para convertir una lista de palabras a mayúsculas. Por ejemplo:

sec = [“datos”, “sal”, “diario”, “gato”, “perro”]

debe ser:

[“DATOS”, “SAL”, “DIARIO”, “GATO”, “PERRO”]

In [83]:

```
# Code cell 10
seq = list(map(lambda word: word.upper(), seq))
print(seq)
```

```
['DATA', 'SALT', 'DAIRY', 'CAT', 'DOG']
```

12) Imagine un termostato inteligente conectado a la puerta que pueda detectar, además de la temperatura, el momento en el que las personas entran o salen de la casa.

Escriba una función que, cuando la temperatura sea inferior a 20 ºC y haya personas en la casa (codificado como valor booleano que se envía como parámetro a la función), inicie la calefacción mediante la devolución de la cadena "calefacción encendida". Cuando la temperatura llegue a 23 grados o no haya personas en la casa, la función devuelve la cadena "calefacción apagada". Cuando no se cumpla ninguna de estas condiciones, la función es "No hacer nada".

In [1]:

```
# Code cell 11
#def smart_thermostat(temp, people_in):
#    ...
#    return command

def smart_thermostat(temp, people_in):
    if temp < 20 and people_in:
        return "calefacción encendida"
    elif temp >= 23 or not people_in:
        return "calefacción apagada"
    else:
        return "No hacer nada"
```

In [2]:

```
# Code cell 12
# Verify smart_thermostat()
smart_thermostat(21,True)
```

Out[2]:

```
'No hacer nada'
```

In [3]:

```
# Code cell 13
# Verify smart_thermostat()
smart_thermostat(21,False)
```

Out[3]:

```
'calefacción apagada'
```

13) La función zip(list1, list2) devuelve una lista de tuplas, donde la tupla i-th contiene el elemento i-th de cada una de las listas de argumento. Utilice la función zip para crear la siguiente lista de tuplas:

'comprimido = [(“Estacionamiento”, -1), (“Negocios”, 0), (“Área de restaurantes”, 1), (“oficinas”, 2)]'

In [87]:

```
# Code cell 14
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
floor_numbers = range(-1,3)
zipped = list(zip(floor_types, floor_numbers))
print(zipped)
```

```
[('Parking', -1), ('Shops', 0), ('Food Court', 1), ('Offices', 2)]
```

14) Utilice la función zip y dict() para crear un diccionario, elevator_dict, donde las teclas sean los tipos de piso y los valores sean el número correspondiente del piso, de modo que:

elevator_dict[- 1] = “Estacionamiento”

In [7]:

```
# Code cell 15
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
floor_numbers = range(-1, 3)

elevator_dict = dict(zip(floor_numbers, floor_types))
elevator_dict[-1] = "Estacionamiento"

print(elevator_dict)
```

```
{-1: 'Estacionamiento', 0: 'Shops', 1: 'Food Court', 2: 'Offices'}
```

In [8]:

```
# Code cell 16
# Verify elevator_dict[-1]
elevator_dict[-1]
```

Out[8]:

```
'Estacionamiento'
```

15) Cree una clase de 'Ascensor'. El constructor acepta la lista de cadenas 'floor_types' y la lista de números enteros 'floor_numbers'. La clase implementa los métodos 'ask_which_floor' y 'go_to_floor'. La salida de estos métodos debe verse de la siguiente manera: `floor_types = ['Estacionamiento', 'Negocios', 'Área de restaurantes', 'Oficinas'] floors_numbers = rango(-1,4)

el = Elevador(floor_numbers, floor_types)

el.go_to_floor(1)`

```
¡Vaya al piso del área de restaurantes!

el.go_to_floor(-2)

En este edificio está el piso número -2.

El.ask_which_floor('Oficinas')

El piso de oficinas es el número: 2

El.ask_which_floor('Piscina')

No hay ningún piso con piscina en este edificio.
```

In [14]:

```
# Code cell 17
class Elevator:

    def __init__(self, floor_numbers, floor_types):

        self._floor_numbers = floor_numbers
        self._floor_types = floor_types
        self._number_to_type_dict = dict(zip(floor_numbers, floor_types))
        self._type_to_number_dict = dict(zip(floor_types, floor_numbers))

    def ask_which_floor(self, floor_type):

        if floor_type in self._floor_types:
            print('The {} floor is the number: {}.'.format(floor_type, self._type_to_num
        else:
            print('There is no {} floor in this building.'.format(floor_type))

    def go_to_floor(self, floor_number):

        if floor_number in self._floor_numbers:
            if floor_number == -1:
                print('Going to Parking Floor!')
            else:
                print('Going to {} Floor!'.format(self._number_to_type_dict[floor_number
        else:
            print('There is no floor number {} in this building.'.format(floor_number))
```

In [15]:

```
# Verify code cell 18
el = Elevator(floor_numbers, floor_types)
el.go_to_floor(1)
```

```
Going to Food Court Floor!
```

In [16]:

```
# Verify code cell 19
el.go_to_floor(-2)
```

```
There is no floor number -2 in this building.
```

In [17]:

```
# Verify code cell 20
el.ask_which_floor('Offices')
```

```
The Offices floor is the number: 2.
```

In [18]:

```
# Verify code cell 21
el.ask_which_floor('Swimming Pool')
```

```
There is no Swimming Pool floor in this building.
```

¡Excelente trabajo!