

# 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.

## Estudiante: Diego Cáceres

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
print('3 a la potencia de 5 es: ' + str(3**5))
```

3 a la potencia de 5 es: 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.split()
```

```
Out[2]: ['¡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 = "Monte Everest"
height = 8848
print("La altura del {} es de {} metros".format(mountain, height))
```

La altura del Monte Everest es de 8848 metros

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

```
In [4]: # Code cell 4      #Tomar la palabra "this"
lst = ['a', 'b', [4, 10, 11], ['c', [1, 66, ['this']], 2, 111], 'e', 7]
lst[3][1][2][0]
```

```
Out[4]: 'this'
```

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

```
In [5]: # Code cell 5      #Tomar la palabra "that"
d = {'k1':['val1','val2','val3',{'we':['need','to','go',{'deeper':[1,2,3,'that']}]}]}
d['k1'][3]['we'][3]['deeper'][3]
```

```
Out[5]: 'that'
```

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

La diferencia principal es que el contenido de la tupla es inmutable (no se puede cambiar su contenido), mientras que en una lista sí se puede modificar su contenido después de su creación

## 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" daría: domain.com

```
In [6]: # Code cell 6
def GetDomain(correo):
    return correo.split('@')[-1] # Conteo de reversa inicia desde -1
GetDomain('user@domain.com')
```

```
Out[6]: 'domain.com'
```

```
In [7]: # Code cell 6
#Forma 2

def GetDomain(correo):
    x = False
    dominio=''
    for i in range(len(correo)):
        if x:
            dominio = dominio + correo[i]
        if correo[i] == '@':
            x = True
    return dominio
GetDomain('user@domain.com')
```

```
Out[7]: '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>)

```
In [8]: # Code cell 7
def findInternet(frase):
    return 'internet' in frase.lower().split()
findInternet('The Internet Engineering Task Force was created in 1986')
```

```
Out[8]: True
```

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

```
In [9]: # Code cell 8
```

```
def countIoT(st):
    count = 0
    st = st.lower().split()
    for i in range(len(st)):
        if 'iot' in st[i]:
            count+=1
    return count
countIoT('I don\'t know how to spell IoT ! Is it IoT or iot ? What does iot mean anyway?')
```

Out[9]: 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 [10]: # Code cell 9
#lambda argumentos: expresión
sec = ["datos", "sal", "diario", "gato", "perro"]
list(filter(lambda d: d[0] == 'd', sec))
```

Out[10]: ['datos', 'diario']

**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 [11]: # Code cell 10
sec = ["datos", "sal", "diario", "gato", "perro"]
list(map(lambda palabra: palabra.upper(), sec))
```

Out[11]: ['DATOS', 'SAL', 'DIARIO', 'GATO', 'PERRO']

**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 °C 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 [12]: # Code cell 11
def smart_thermostat(temp, people_in):
    if temp < 20 and people_in == True:
        command = "calefacción encendida"
    elif temp >= 23 or people_in == False :
        command = "calefacción apagada"
```

```
else:
    command = "No hacer nada"
return command
```

```
In [13]: # Code cell 12
# Verify smart_thermostat()
smart_thermostat(21, True)
```

```
Out[13]: 'No hacer nada'
```

```
In [14]: # Code cell 13
# Verify smart_thermostat()
smart_thermostat(21, False)
```

```
Out[14]: '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 [15]: # 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 [16]: # Code cell 15
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
floors_numbers = range(-1, 3)
elevator_dict = dict(zip(floors_numbers, floor_types))
print(elevator_dict)

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

```
In [17]: # Code cell 16
# Verify elevator_dict[-1]
elevator_dict[-1]
```

```
Out[17]: 'Parking'
```

**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 [18]: # Code cell 17
floor_types = ['Estacionamiento', 'Negocios', 'Área de restaurantes', 'Oficinas']
floors_numbers = range(-1,4)

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('El piso de {} es el número: {}'.format(floor_type, self._type_to_num
        else:
            print('No hay ningún piso con {} en este edificio.'.format(floor_type))

    def go_to_floor(self, floor_number):
        try:
            print('¡Vaya al piso del {}!'.format(self._number_to_type_dict[floor_number]
        except Exception:
            print("No existe un piso con el número {} en este edificio." .format(floor_n
```

```
In [19]: # Verify code cell 18
el = Elevator(floor_numbers, floor_types)
el.go_to_floor(1)
```

¡Vaya al piso del Área de restaurantes!

```
In [20]: # Verify code cell 19
##Cambié el mensaje para que tenga un poco más de sentido (desde mi punto de vista)
el.go_to_floor(-2)
```

No existe un piso con el número -2 en este edificio.

```
In [21]: # Verify code cell 20
el.ask_which_floor('Oficinas')
```

El piso de Oficinas es el número: 2.

```
In [22]: # Verify code cell 21
el.ask_which_floor('Piscina')
```

No hay ningún piso con Piscina en este edificio.

# ¡Excelente trabajo!

© 2017 Cisco y/o sus filiales. Todos los derechos reservados. Este documento es información pública de Cisco.