



# 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
resultado = 3**5
print(resultado)
```

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 = ?
#?.split()
s = "¡Este curso es increíble!"
lista_s = s.split()
print(lista_s)
```

[ '¡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 [5]: # Code cell 3
mountain = "Mt. Everest"
height = 8848
#print("....".format( ? , ? ))

cadena = "La altura del {} es de {} metros.". format(mountain, height)
print(cadena)
```

La altura del Mt. Everest es de 8848 metros.

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

```
In [6]: # Code cell 4
lst = ['a','b',[4,10,11],['c',[1,66,['this']],2,111],'e',7]
#lst[?][?][?][?]
palabra = lst[3][1][2][0]
print(palabra)
```

this

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

```
In [8]: # Code cell 5
d = {'k1':['val1','val2','val3',{'we':['need','to','go',{'deeper':[1,2,3,'that']}]}]
#d[?][?][?][?][?][?]
palabra = d['k1'][3]['we'][3]['deeper'][3]
print(palabra)
```

that

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

*La principal diferencia es que la tupla no se puede modificar su contenido después de crearla y la lista si permite modificar después de crearla.*

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

#Ejemplo de uso
email = "user@domain.com"
dominio = GetDomain(email)
print(dominio)
```

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 [11]: # Code cell 7
```

```
#def findInternet(?):
    #return 'internet' in ?.lower().split()
#findInternet('The Internet Engineering Task Force was created in 1986')
def findInternet(text):
    return 'internet' in text.lower()

#Ejemplo de uso
result = findInternet('The Internet Engineering Task Force was created in 1986')
print(result)
```

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 [14]: # Code cell 8
#def countIoT(st):
#
#
#
#
#    return count
#countIoT('I don\'t know how to spell IoT ! Is it IoT or iot ? What does iot mean a
def countIoT(st):
    return st.count('IoT')

#Ejemplo de uso
result = countIoT('I don\'t know how to spell IoT ! Is it IoT or iot ? What does io
print(result)
```

2

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 [15]: # Code cell 9
seq = ['data', 'salt', 'dairy', 'cat', 'dog']
resultado = list(filter(lambda x: x.startswith('d'), seq))
print(resultado)
```

```
[ '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 [16]: # Code cell 10
#list(map())
sec = ["datos", "sal", "diario", "gato", "perro"]
resultado = list(map(lambda x: x.upper(), sec))
print(resultado)
```

```
['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 [18]: # 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 hace nada"
resultado_1 = smart_thermostat(21, True) # Temperatura 21 y personas en casa
resultado_2 = smart_thermostat(21, False) # Temperatura 21 y no hay personas en casa

print(resultado_1)
print(resultado_2)
```

```
No hace nada
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 [19]: # Code cell 14  
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']  
floor_numbers = range(-1,3)  
#Usamos zip para crear la lista de tuplas  
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 [22]: # Code cell 15  
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']  
floors_numbers = range(-1,3)  
#Usamos zip y dict para crear el diccionario  
  
elevator_dict = dict(zip(floor_numbers, floor_types))  
print(elevator_dict)  
  
{-1: 'Parking', 0: 'Shops', 1: 'Food Court', 2: 'Offices'}
```

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 [23]:

```
# 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_t  
        else:  
            print('There is no {} floor in this building.'.format(floor_type))  
  
    def go_to_floor(self, floor_number):  
        if floor_number in self._number_to_type_dict:  
            print(f"¡Vaya al piso del {self._number_to_type_dict[floor_number]}!")  
        else:  
            print(f"En este edificio esta el piso número {floor_number}.")  
  
#Datos de entrada  
floor_types = ['Estacionamiento', 'Negocio', 'Área de restaurantes', 'Oficinas']  
floor_numbers = range (-1, 4)  
  
#Creamos el objeto de la clase Elevador  
el = Elevator(floor_numbers, floor_types)
```

```
#Verificamos el método go_to_floor  
el.go_to_floor(1)      #Va al piso de Área de restaurantes  
  
#Verificamos el método ask_which_floor  
el.ask_which_floor('Oficinas') # Muestra el número de piso de Oficinas  
el.ask_which_floor('Piscina') # Deberia decir que no existe el piso Piscina
```

¡Vaya al piso del Área de restaurantes!  
The Oficinas floor is the number: 2.  
There is no Piscina floor in this building.

## ¡Excelente trabajo!

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

In [ ]: