Activity 1.1 Python Challenge

Name: Course and Section: Instructor: Date Performed: Date Submitted

Instructions: Answer the questions or complete the tasks outlined below, use the specific method described if applicable. Insert a code cell after each text cell.

Objectives:

- Test understanding of Python basics
- Review Python basic to intermediate commands

1. What is 3 to the power of 5?

```
3**5
```

```
243
```

1. Create a variable, s, containing the string "This course is amazing!". Using the variable, split the string into a list.

```
s = "This course is amazing!"
s.split()
```

```
['This', 'course', 'is', 'amazing!']
```

1. Given the variables height and mountain, use .format() to print the following string: `The height of Mt. Everest is 8848 meters.`

```
height = 8848
mountain = "Mt. Everest"
print(f"The height of {mountain} is {height} meters.")
```

```
The height of Mt. Everest is 8848 meters.
```

1. Given the following nested list, use indexing to grab the word "this".

```
lst = ['a','b',[4,10,11],['c',[1,66,['this']],2,111],'e',7]
```

```
lst = ['a','b',[4,10,11],['c',[1,66,['this']],2,111],'e',7]
this = lst[3][1][2]
print(this)
```

```
['this']
```

1. Given the following nested dictionary, grab the word "that". This exercise is a little more difficult.

```
d = {'k1':['val1','val2','val3',{'we':['need','to','go',{'deeper':
[1,2,3,'that']}]}]}
```

```
d = {'k1':['val1','val2','val3',{'we':['need','to','go',{'deeper':
[1,2,3,'that']}]}]}
```

```
that = d['k1']
print(that)
```

```
['val1', 'val2', 'val3', {'we': ['need', 'to', 'go', {'deeper': [1, 2, 3, 'that']}]}]
```

1. Create a function, GetDomain(), that grabs the email website domain from a string in the form: user@domain.com. So for example, passing "user@domain.com" would return: domain.com

```
def GetDomain(e):
    d = e.lstrip('@')
    return d
email = "user@domain.com"
domain = GetDomain(email)
print(domain)
```

```
user@domain.com
```

1. Create a basic function, findInternet(), that returns True if the word 'Internet' is contained in the input string. Don't worry about edge cases like punctuation being attached to the word, but account for capitalization. (Hint: Please see https://docs.python.org/2/reference/expressions.html#in)

```
def findInternet(s):
    internet = False
    if s == 'internet':
        return true
input_string = "internet"
if_internet = ""
print(if_internet)
```

1. Create a function, countIoT(), that counts the number of times the word "IoT" occurs in a string. Ignore edge cases but take into account capitalization.
1. Use lambda expressions and the filter() function to filter out words from a list that do not start with the letter 'd'. For example:

```
seq = ['data','salt' ,'dairy','cat', 'dog']
```

should be filtered down to:

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

```
seq = ['data','salt' ,'dairy','cat', 'dog']
new_seq = filter(lambda seq: seq[i][0] != 'd')
print(new_seq)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
```

```
last)
Cell In[30], line 2
      1 seq = ['data','salt' ,'dairy','cat', 'dog']
----> 2 new_seq = filter(lambda seq: seq[i][0] != 'd')
      3 print(new_seq)

TypeError: filter expected 2 arguments, got 1
```

1. Use lambda expressions and the map() function to convert a list of words to upper case. For example:

```
seq = ['data','salt' ,'dairy','cat', 'dog']
```

should become:

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

seq = ['data','salt' ,'dairy','cat', 'dog']
new_seq = lambda seq: seq[i].upper()
print(new_seq)

<function <lambda> at 0x0000022C1BB65A80>
```

1. Imagine a smart thermostat that is connected to the door, so that it can detect, in addition to the temperature, when people enter or leave the house. Write a function that, if the temperature is lower than 20 degrees Celsius, and there are people in the house (encoded as a boolean value to be passed as a parameter to the function), turns on the heating by returning the string "Heating on". When the temperature reaches 23 degrees or there are no people in the house, it returns the string "Heating off". When none of these conditions are met, the function returns "Do nothing".
1. The function zip(list1, list2) returns a list of tuples, where the i-th tuple contains the i-th element from each of the argument lists. Use the zip function to create the following list of tuples:

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

# Code cell 13
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
floor_numbers = range(-1,3)
#zipped = list(...)
print(zipped)
```

1. Use the zip function and dict() to create a dictionary, elevator_dict, where the keys are the floor types and the values are the corresponding floor number so that:

```
elevator_dict[-1] = 'Parking'

# Code cell 14
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
```

```
floors_numbers = range(-1,3)
#elevator_dict = dict(...)
print(elevator_dict)

# Code cell 15
# Verify elevator_dict[-1]
elevator_dict[-1]
```

1. Create an Elevator class. The constructor accepts the list of strings floor_types and the list of integers floor_numbers. The class implements the methods 'ask_which_floor' and 'go_to_floor'. The output of this methods should look as follows:

```
floor_types = ['Parking', 'Shops', 'Food Court', 'Offices']
floors_numbers = range(-1,4)
el = Elevator(floor_numbers, floor_types)
el.go_to_floor(1)
Going to Food Court floor!
el.go_to_floor(-2)
There is floor number -2 in this building.
el.ask_which_floor('Offices')
The floor Offices is the number: 2
el.ask_which_floor('Swimming Pool')
There is no Swimming Pool floor in this building.
```

Good Job!