Assignment 1

- Due 1/31 by 5pm
- Topics: containers, udfs, comprehensions, error handling

Instructions

- Rename this notebook with your NAME_LASTNAME_ASSIGN_1
- Finish the test on this notebook
- Download your notebook as a PDF
- Turn in both the notebook and PDF (in my case I would get two files RAFAEL_VESCOVI_ASSIGN_1.pdf and RAFAEL_VESCOVI_ASSIGN_1.ipynb)
- Show the output of your code working on the sample data, like below.
- If techniques are specified, use them. Otherwise, feel free to solve the problem how you want.
- Try to keep your code clean, concise and avoid loops if necessary.
- You have to show your work (code).
- You will be graded:
    - The functionality of your code (Does it do what it was meant to)
    - Showing the output on the sample data provided
    - How concise it is (did you use a for loop when you could have used a comprehension for instance). Simply put, try and write clean, concise and readable code (don't use 10 lines for what could have been done in 4).
    - Even if the answer isn't perfect, make an honest attempt as partial credit will be given.
- you should not have to explicitly use a for loop except for question 12, meaning it is possible to complete the entire assignment without using the keyword "for".
- the only third party functions/modules you are allowed to use are permutations and combinations from itertools. yes, there is a built in median function in numpy, but the idea is to program yourself and solve the problems on your own using basic python techniques

Sample Question

- Create a UDF that adds 2 to a number

In [1]:
```
a = 5
```

In [2]:
```
def add_func(x):
    return(x+2)

number = add_func(a)
print(number)
```

```
7
```

Question 1

- Create a function that takes 2 lists as params, of equal length and does the following:
  - returns a dictionary where the keys are the first list and the values are the second list
  - make use of zip and comprehension to make the dictionary
  - if the lists are not of equal length, use error handling to indicate the mismatch

In [3]:
```
# a = [5,7]
# b = ["x", "z"]
# return {5:"x", 7:"z"}
```

In [49]:
```
a = [1,2,3,4,5,6,7,8]
b = ["a","b","c","d","e","f","g","h"]
```

In [50]:
```
def make_dict(x,y):
    assert len(x) == len(y), "The two lists as params have to be of equa
l length."

    output = {key:value for key, value in zip(x,y)}

    return(output)

dictionary = make_dict(a,b)
print(dictionary)
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h'}
```

Question 2

- Create a UDF that takes a list and returns the permutations and combinations (length 2) as nested lists, along with the number of permutations and combinations.
- The function should return 4 things

In [52]:
```
a = [1,2,3,4,5]
```

```
In [53]: from itertools import permutations, combinations

         def perm_comb(x):

             perm_output = list(permutations(x, 2))
             perm_output_len = len(list(permutations(x, 2)))
             comb_output = list(combinations(x, 2))
             comb_output_len = len(list(combinations(x, 2)))

             return perm_output, perm_output_len, comb_output, comb_output_len

         perm_output, perm_output_len, comb_output, comb_output_len = perm_comb(a
         )
         print("Permutations are", perm_output)
         print("There are", perm_output_len, "permutations.")
         print("Combinations are", comb_output)
         print("There are", comb_output_len, "combinations.")
```

```
Permutations are [(1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 3), (2,
4), (2, 5), (3, 1), (3, 2), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4,
5), (5, 1), (5, 2), (5, 3), (5, 4)]
There are 20 permutations.
Combinations are [(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2,
5), (3, 4), (3, 5), (4, 5)]
There are 10 combinations.
```

Question 3

- Map a function to the below list that
  - returns the remainder of each number divided by 3 (52/3 = 17 with a remainder of 1, we want the 1 to be returned)
  - use lambdas for the mapping
  - hint, look at modulus (%) in python

```
In [13]: lst = list(range(100))
In [14]: list(map(lambda x: x%3, lst))
```

```
Out[14]: [0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
          0,
          1,
          2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0,
1,
2,
0]
```

Question 4

- Using comprehension, replace the keys in the list with their value in the dictionary
- example below

In [ ]:
```
#lst = [1,2]
#dict = {1:"a", 2:"b"}
#new_lst = ["a","b"]
```

In [15]:
```
a = list(range(5))
dct = {
    0:"a",
    1:"b",
    2:"c",
    3:"d",
    4:"e"
}
```

In [16]:
```
new_lst = list(dct[i] for i in a)
print(new_lst)
```

```
['a', 'b', 'c', 'd', 'e']
```

Question 5

- Make a UDF that takes a variable amount of numbers and returns a list with the numbers having been squared
- use comprehension to perform the squaring of each item
- hint, this should be doable in one line using args

In [ ]:
```
# my_func(1,2,3)
    # do some stuff

# return_lst = [1,4,9]
```

In [ ]:
```
# for this pass in the numberse 1,2,3
```

In [17]:
```
def num_squared(*args):
    return(list(i**2 for i in args))


return_lst = num_squared(1,2,3)
print(return_lst)
```

```
[1, 4, 9]
```

Question 6

- Make a UDF that takes two lists as params and returns a list of the intersecting items (items that appear in both lists)

In [18]:
```
# a = [1,2]
# b = [2,3]
# return_lst = [2]
```

```
In [19]: a = [1,2,3,4,5,6,7]
         b = [2,3,4,6,8,10,11,21]
```

```
In [20]: def intersection(list1,list2):
             return list(filter(lambda x: x in list1, list2))

         return_lst = intersection(a,b)
         print(return_lst)

         [2, 3, 4, 6]
```

Question 7

- write a function that takes a list of numbers and filters for odds only
- use the filter method and a lambda as the function

```
In [28]: # my_lst = [1,2]
         # new_lst = [1]
```

```
In [29]: a = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

```
In [30]: def find_odds(list1):
             return(list(filter(lambda x: x%2 == 1, list1)))

         new_lst = find_odds(a)
         return(new_lst)

Out[30]: [1, 3, 5, 7, 9, 11, 13, 15]
```

Question 8

- Write a function (lambda or not) that returns the absolute value of a number and adds 2
- Use the function to map it to a list comprehension

```
In [31]: # my_lst = [1,-2,1]
         # using a comprehension return [3,4,3]
```

```
In [32]: a = [1,-2,1,8,-10,15,-12]
```

```
In [34]: return_lst = list(map(lambda x: abs(x)+2, a))
         print(return_lst)

         [3, 4, 3, 10, 12, 17, 14]
```

Question 9

- write a function that takes 2 lists and finds the euclidean distance between then
- note you may not use any third party modules. use comprehensions

```
In [35]: # a = [1,2]
         # b = [3,1]
         # distance = ~2.23
```

```
In [36]: a = [1,2,3,4,1]
         b = [3,1,2,4,3]
```

```
In [37]: def find_eucli_dist(list1, list2):
             return sum([(x-y)**2 for x,y in zip(list1, list2)])**0.5

         distance = find_eucli_dist(a,b)
         print(distance)

         3.1622776601683795
```

Question 10

- write a function that takes a list, sorts it and finds the middle value.
- use error handling to check the list is odd in length
- if it's even, return an error message
- feel free to use a try/except or assert
- show the output run on both lists

```
In [38]: # a = [4,6,1]
         # sort = [1,4,6]
         # middle val = 4
         # if even number in size, return error
```

```
In [2]: a = [4,6,1,3,12,41,4,1,24,12,17]
        b = [4,6,1,3,12,41,4,1,24,12]
```

In [3]:
```
def sort_median (lst):

    assert len(lst)%2 ==1, "The length of list has to be odd in length."

    lst.sort()
    sort = lst
    middle_val = lst[len(lst)//2]

    return sort, middle_val

sort, middle_val = sort_median(a)
print(sort, middle_val)
```

```
[1, 1, 3, 4, 4, 6, 12, 12, 17, 24, 41] 6
```

In [4]:
```
sort, middle_val = sort_median(b)
print(sort, middle_val)
```

```
---------------------------------------------------------------------------
----
AssertionError                            Traceback (most recent call l
ast)
<ipython-input-4-41f8d44cea54> in <module>
----> 1 sort, middle_val = sort_median(b)
      2 print(sort, middle_val)

<ipython-input-3-79b0625eceec> in sort_median(lst)
      1 def sort_median (lst):
      2
----> 3     assert len(lst)%2 ==1, "The length of list has to be odd in
length."
      4
      5     lst.sort()

AssertionError: The length of list has to be odd in length.
```

Question 11

- Make a generator that parses the list of transaction data
- Put the contents in a dictionary where the key is the user id and the value is the list of items the user has purchased
- note the transactions are pipe (|) delimtied, meaning you'll have to use some string manipulations to get the values from each transaction
- make sure to skip the header somehow

In [42]:
```
# transactions = ['A', 'item_a'], ['A', 'item_b']
# dict = {A:[item_a, item_b]}
```

```
In [41]: transactions = [
             ["user_id|item_id"],
             ["A|item_a"],
             ["B|item_a"],
             ["C|item_a"],
             ["C|item_b"],
             ["C|item_c"],
             ["B|item_c"],
             ["D|item_b"],
             ["D|item_b"]
         ]
```

```
In [43]: records =  (j.rstrip().split("|") for i in transactions for j in i)
         next(records)

         keys = [j.rstrip().split("|")[0] for i in transactions for j in i][1:]

         answer = {key: [] for key in keys}

         [answer[key].append(value) for key, value in records]

         print(answer)
```

```
{'A': ['item_a'], 'B': ['item_a', 'item_c'], 'C': ['item_a', 'item_b',
'item_c'], 'D': ['item_b', 'item_b']}
```

Question 12

- Make a function that takes 2 lists of equal length and returns a True when the values in a given index are the same in both lists and a false if they are not.
- Use error handling of some kind if the lists are not of equal length.

```
In [13]: # should return [True, True, True, False, False]
         a = [1,2,3,4,5]
         b = [1,2,3,5,6]
```

```
In [14]: def same_index(arr1, arr2):
             assert len(arr1) == len(arr2), "The two lists are not of equal lengt
         h."

             answer = [x1 == x2 for x1,x2 in zip(arr1, arr2)]

             return answer

         same_index(a,b)
```

```
Out[14]: [True, True, True, False, False]
```

Question 13

Briefly explain the use of the following words in python:

- assert
  Assert tests a condition. If the condition is met, assert will not do anything to the program and the program will continue running. However, when a condition is not met, it will raise an error message.

- raise
  The raise statement can be used to forcefully raise an exception error.

- yield
  We use yield in a funciton to make a generator. We use yield as return, but it won't detroy the local state of variables so that we can resume where was left of.

- def
  We use def to define a function.

- lambda
  Lambda is a smaller function but it is anonymous.

- return
  We use this keyword to end the execution of function and return values we want.

- map
  By using map function, we can apply a function to a given iterable. What map function returns is a map object.

- filter
  Filter() filters a sequence of items based on a user-defined criterion. Filter() always comes with function, which is used to test the items in a sequence, and sequence, which is being filtered.

- try
  We use this keyword to try out a part of the code. It is usually used with except. If no exception error happens happens, only the try part will be run and the except part will be skipped. If except error happend, the try part will be skipped and only the exception part will be run.

- except
  It is used with try. Except handles the error that is tested by the try block.

- continue
  Continues is used to skip the rest of the code for current iteration. Loop will just continue onto the next iteration.

- pass
  Nothing happens when pass executes. It is used more like a placeholder for code.

- break
  Break is used to terminate the loop containing this keyword.

- while

While is a loop where we can execute our code as long as the condition is met.

- do (in the context of a while loop)
  Do-while loop does not exist in python.

- for
  We use for loop when we havec a chunk of codes that we want to repeatedly run for a limited number of times.

Question 14

Briefly describe the below:

- list -
  List is a sequence of items enclosed in [] and separated by comma. A list can contain different types of items.

- tuple -
  Like list, tuple is a collection of items enclosed in (). However, unlike list, tuple is immutable. A tuple can contain different data types.

- dictionary -
  Dictionary is a collection of key-value pairs enclosed in {} and seperated by comma.

- set -
  Set is a collection of different data types enclosed in {}. It can contain one or more objects, but not duplicated ones. The items in set are not ordered because the order in set may not be the same as the order of assignment.

Question 15

- What two datatypes can be declared with { }?
  dictionary, set

- What two datatypes can be declared with ( )?
  tuple, zip() Note: Technically, zip() is a function returning an iterator object that is declared with ().

- Provide examples of each of the 4 above.
  zip: zip(x,y)

  Tuple: ("red", "green")

  Dictionary: {"color": "red", "shape": "triangle"}

  Set: {"apple", "banana", "cherry"}

Question 16

Describe what a generator is and how it differs from a list or tuple.

Generator is a function that returns an iterator object. It maintains its local state so it keeps tracks of where we are in an iterator. Tuple is enclosed in () which is just a literal and all the values in () are hardcoded. However, () in generator is for groupings so that generator returns one object at a time when called. The difference between list and generator is mainly in list comprehension and generator comprehension. Python reserves the memory for the entire list but only one item at a time for generator when in demand.

Question 17

Show three ways of attaching a list to itself (one of then should use the function append) .

```
In [ ]: # Method 1: Use +
        a = [1,2,3]
        b = [1,2,3]

        answer = a+b
        print(answer)
```

```
In [ ]: # Method 2: Use extend()
        a = [1,2,3]
        b = [1,2,3]
        a.extend(b)
        print(a)
```

```
In [ ]: # Method 3: Use chain
        a = [1,2,3]
        b = [1,2,3]
        from itertools import chain
        answer = list(chain(a,b))
        print(answer)
```