InterviewQs

Question 1 - Python function to count tuple elements

Suppose you are given a tuple containing ints and strings. Write a Python function to return the # of times a given element, `n`, appears in the tuple.

For example, given the tuple below, where `n=3`, your function should return `4`, since 3 appears 4 times in the tuple.

```
input:
        n = 3
        my_tuple = 2, 'my_string', 4, 3, 3, 3, 2, 3

output:
        4
```

Solution:

Click here to view this solution in an interactive Colab (Jupyter) notebook.

A tuple is an immutable Python list with heterogeneous elements. Although this is different than a traditional Python list, we can use many of the same built-in Python functions.

For this question, we use Python's count to count the occurrences of the element `n`, and feed that count into a variable.

```
def countElement(my_tuple, n):
    # count occurrences of element n
    count = my_tuple.count(n)

    # save that count to our variable, return it
    return count
```

InterviewQs

Question 2 - Counting capital letters

Using Python, write code that will read a file and return the number of capital letters.

Once you have your initial piece of code, see if you can condense into a one-liner.

Solution:

Our first piece of code is just a simple for loop that uses Python's built-in function, isupper(). We loop through each line in the file, then each character, adding to our count variable each time we encounter an uppercase letter.

```
count = 0
for line in fh:
    for character in txt:
        if character.isupper():
            count += 1
```

Now, if we want to get this into one line, one way to do that is to use sum(), which takes any iterable. We can rewrite our latest solution containing nested for loops and a single if-else statement as a generator expression:

```
count = sum(1 if character.isupper() else 0 for line in file for character in line)
```

InterviewQs

Question 3 - Naive Bayes

What is Naive Bayes' theorem, and why is it considered 'naive'? Why is it often used in practical applications rather than trying to implement an algorithm based on Bayes' Theorem (non-naive)?

Solution:

Naive Bayes is a machine learning implementation of Bayes' Theorem. In short, Bayes' Theorem is an algorithm that describes the probability of an event, based on prior knowledge of conditions that might be related to the event. In other words, it describes how to update the probability of an event given new evidence.

Naive Bayes is 'naive' because while it uses conditional probability to make classifications, the algorithm assumes that all features of a class are independent. This is considered naive because, in reality, it is not often the case. Naive Bayes is often used over bayesian classifiers since the latter is very difficult to compute. Naive Bayes offers a good approximation that runs much quicker, is easier to follow, and works just fine for many problems.

If this is a new concept to you, further, in-depth reading can be found here and here with some of the math behind the two concepts.

Question 4 - Planning for a new office location, using SQL

Suppose you work for a retail company and have access to two tables:

Customers:

customer_id	city
0	New York
1	New York
2	Los Angeles
3	Jacksonville

Suppliers:

supplier_id	city
0	Omaha
1	New York
2	San Francisco

supplier_id	city
3	Los Angeles

You've been tasked to find which cities have a strong overlap with both customers and suppliers, as your company explores opening an additional office. Using SQL, write a query to stack rank the frequency of the cities shown across both databases.

Your query should return the following elements:

- City
- # of times city appeared across both tables

Solution:

To solve this one, we can cut our query into two parts.

First, we'll want to build a complete list of all of the cities shown in the two tables. Since the two tables have the common field city that we're after, and we simply want to stack them into a long list, the SQL UNION operator will work well for this. This is what we're seeing in the nested query.

The next step is to just query that new stacked list to obtain a unique list of cities along with their counts, which we can do with a simple group by.

```
SELECT
    stg.city as city,
    count(stg.city) as city_frequency
        #build a full table containing a list of all cities by
        #using UNION statement to stack the two queries
        (SELECT
            #aggregate total revenue by channel
            a.city as city
            FROM customers a
        #Note: UNION ALL here will grab *all rows*, so we can count
```

```
        #the frequency above
        # -- UNION statement only grabs distinct rows
        UNION ALL
        SELECT
            b.city as city
            FROM suppliers b) stg
#Group by city, since we're aggregating the count(similar to a pivot table))
GROUP BY 1
```

Our output will look like this:

	city	city_frequency
0	New York	3
1	Los Angeles	2
2	San Francisco	1
3	Jacksonville	1
4	Omaha	1

So, from a pure lens of supplier/customer overlap, New York would be the top city for an office location. There are of course plenty of other factors to consider (such as cost, potential customers in the market, normalizing list of current customers/suppliers for spend), which would be good to acknowledge in an interview so folks know you understand the business problem (of finding a new office location).

InterviewQs

Question 5 - Sell, sell, sell!

Suppose we are given an array of `n` integers which represent the value of some stock over time. Assuming you are allowed to buy the stock exactly once and sell the stock once, what is the maximum profit you can make? Can you write an algorithm that takes in an array of values and returns the maximum profit?

For example, if you are given the following array:

```
[2, 7, 1, 8, 2, 8, 14, 25, 14, 0, 4, 5]
```

The maximum profit you can make is 24 because you would buy the stock when its price is 1 and sell when it's 25. Note that we cannot make 25, because the stock is priced at 0 after it is priced at 25 (e.g you can't sell before you buy).

Solution:

Click here to view this solution in an interactive Colab (Jupyter) notebook.

As with any programming problem, there are several ways we can go about solving it. We're going to start with the most straightforward way, then we're going to try to optimize it.

Version 1 - O(n^2) time:

The first version of this solution, we're going to consider all pairs of values, and pick the one with the highest net profit.

- Line 1: We define the function.
- Line 2: We're going to keep track of the maximum profit, and to do this we need to declare a variable. We'll start out by setting max_profit = 0.
- Line 3-5: We're going to iterate across all pairs of values. We're going to start at the first number in the array, and create pair across all other numbers in the array (e.g. given the above array, we'll start with 2 (i), and create pairs with 7 (j), 1 (j+1), 8 (j+2)...ect). We will compare each pair to the current max_profit value. If the current pair is more than the max profit, we will set max_profit to the new pair's delta.

```
1 def version1_SellProfit(n):
2    max_profit = 0;
3    for i in range(0, len(n)):
4        for j in range (i + 1, len(n)):
5            max_profit = max(max_profit, n[j] - n[i])
6
7    return max_profit
```

Version 2 - O(n log n) time:

We can try to make the first version more elegant by utilizing recursion. On a high level, we'll divide the array into two halves, compute the maximum profit in both halves, find the minimum value in the first half and find the maximum value in the second half and compute the maximum profit the min and max of both halves. Lastly, we'll compare all 3 computed values and take the maximum value.

- Line 1: We define the function.
- Line 3-4: If the array has 0 or 1 elements, the best case maximum profit is 0, so we will return 0 if this is true.
- Line 6-7: Next we're going to divide the array into two equal pieces.
- Line 9-10: We're going to call the function we're working on for both parts of the divided array from line 6-7.
- Line 12: We'll find the maximum value on the right array, and the minimum value from the left array and calculate the profit.
- Line 14: We will return the best value between the right max, left max, and cross max.

```
1: def version2_SellProfit(n):
2:
3:    if len(n) <= 1:
4:        return 0;
5:
6:    left  = n[ : len(n) // 2]
7:    right = n[len(n) // 2 : ]
```

```
8:
9:     leftMax  = version2_SellProfit(left)
10:    rightMax = version2_SellProfit(right)
11:
12:    crossMax = max(right) - min(left)
13:
14:    return max(leftMax, rightMax, crossMax)
```

InterviewQs

Question 6 - Categorizing foods

You are given the following dataframe and are asked to categorize each food into 1 of 3 categories: **meat, fruit, or other**.

	food	pounds
0	bacon	4.0
1	STRAWBERRIES	3.5
2	Bacon	7.0
3	STRAWBERRIES	3.0
4	BACON	6.0
5	strawberries	9.0
6	Strawberries	1.0
7	pecans	3.0

Given this, write code to add a new column categorizing each row. Solution will be written in python for premium users.

Solution:

Click here to view this solution in an interactive Colab (Jupyter) notebook.

We're going to assume the dataframe above is called data. Let's break this problem into steps:

1. The first thing we'll notice is that the table has a lot of the same foods, just with different cases. To reduce redundancy with labeling, we're going to convert all of the foods to lower case. (Line 1-2 in code below)
2. Next, we'll realize that the table of foods only has 4 different foods, so we can write a simple function that takes in a series and returns the corresponding category. (Line 4-12 in code below)
3. Lastly, we're going to apply the function we created in step 2 into a newly created column in the data frame. (Line 14 in code below)

```
1:  lower = lambda x: x.lower()
2:  data['food'] = data['food'].apply(lower)
3:
4:  def food_category(series):
5:      if series['food'] == 'bacon':
6:          return 'meat'
7:      elif series['food'] == 'strawberries':
8:          return 'fruit'
9:      elif series['food'] == 'turkey':
10:          return 'meat'
11:      else:
12:          return 'other'
13:
14:  data['category'] = data.apply(food_category, axis='columns')
15:  data
```

InterviewQs

Question 7 - Twitch content creators

You are working for a company like Twitch.tv. Twitch.tv is a live streaming platform, where content creators (e.g. the people creating content on the live streams) can get donations from viewers for producing content they support.

Your company is trying to launch a new product that will benefit content creators that get a large amount of donations per streaming session. You are given the following tables:

Table: all_donations

Column Name	Data Type	Description
creator_id	integer	unique id of content creator
viewer_id	integer	unique id of viewer
session_id	integer	unique session id of stream
date	string	format is "YYYY-MM-DD"
donation_amount	integer	amount donated in USD

Table: sessions_info

Column Name	Data Type	Description

creator_id	integer	unique id of content creator
session_id	integer	unique id of viewer
date	string	format is "YYYY-MM-DD", date of session
length	integer	length of session

Table: session_viewers

Column Name	**Data Type**	**Description**
creator_id	integer	unique id of content creator
viewer_id	integer	unique id of viewer
date	string	format is "YYYY-MM-DD"
session_id	integer	unique session id of stream
mins_viewed	integer	total number of the viewer watched the stream

Given this, write a SQL query to find the top 10 content creators in 2018 that have the highest average donations per viewer.

Solution:

We are going to break this into a few of steps. We are first focus on how to get total donations per session and total viewers per session. Then we're going to join these together to get the average donations per viewer.

1. First, we're going to write a query to calculate the total amount of donations for each creator's session.

```
SELECT
     creator_id,
     session_id,
     SUM(dontaion_amount) AS total_donations
FROM
     all_donations
WHERE
     date >= '2018-01-01'
GROUP BY
     creator_id,
     session_id
```

2. Next, we're going to get the total number of viewers per session.

```
SELECT
     session_id,
     COUNT(DISTINCT viewer_id) AS num_viewers
FROM
     session_viewers
WHERE
     date >= '2018-01-01'
     AND mins_viewed > 0
```

```
GROUP BY
     session_id
```

3. Now we are going to join these two queries above, and we can calculate num_viewers per total_donations with these two subqueries.

```
SELECT
    donations.creator_id,
    donations.sessions_id,
    total_dontaions/num_viewers AS donations_per_viewer
FROM(
    (
    SELECT
        session_id,
        COUNT(DISTINCT viewer_id) AS num_viewers
    FROM
        session_viewers
    WHERE
        date >= '2018-01-01'
        AND mins_viewed > 0
    GROUP BY
        session_id
    ) AS viewers

    JOIN
    (
    SELECT
        creator_id,
        session_id,
        SUM(dontaion_amount) AS total_donations
    FROM
        all_donations
    WHERE
        date >= '2018-01-01'
```

```
    GROUP BY
        creator_id,
        session_id
    ) AS donations

    ON donations.session_id = viewers.session_id
)
```

4. Last step! Now we can calculate the average number of donations per viewer for all of the sessions. Once we calculate the average, we can add in an ORDER BY clause for our AVG aggregation and limit the number of results to 10.

```
 SELECT
     creator_id,
     AVG(donations_per_viewer) AS avg_donations_per_viewer
FROM(
    SELECT
        donations.creator_id,
        donations.sessions_id,
        total_dontaions/num_viewers AS donations_per_viewer
    FROM(
        (
        SELECT
            session_id,
            COUNT(DISTINCT viewer_id) AS num_viewers
        FROM
            session_viewers
        WHERE
            date >= '2018-01-01'
            AND mins_viewed > 0
        GROUP BY
            session_id
        ) AS viewers
```

```
        JOIN
        (
        SELECT
            creator_id,
            session_id,
            SUM(dontaion_amount) AS total_donations
        FROM
            all_donations
        WHERE
            date >= '2018-01-01'
        GROUP BY
            creator_id,
            session_id
        ) AS donations

        ON donations.session_id = viewers.session_id
    )
)
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

InterviewQs

Question 8 - Probability of selecting a wardrobe

Suppose Charlie has 3 pairs of shoes, 4 different coats, and 2 different pants to wear in his wardrobe. Of those items, Charlie has exactly one pair of white shoes, exactly one black coat, and exactly one pair of khaki pants. If Charlie selects each item of his wardrobe at random, what is the probability that he will wear the white shoes, black coat, and khaki pants?

Solution:

Here, we can use probability theory (specifically the rule of product) to solve. The rule of product is used to find the probability of an intersection of events; we can use it to find the intersection of the specified colors in Charlie's wardrobe.

There is a 1/3 probability that Charlie would randomly select the black shoes, a 1/4 probability that he would randomly select the black coat, and a 1/2 probability that he would randomly select the khaki pants. These events are assumed independent (as indicated by 'random' in the question prompt), meaning as the selection of shoes does not affect the selection of the coat, and so on.

Therefore, the probability he will wear the white shoes, black coat, and khaki pants is just: 1/3 * 1/4 * 1/2 = **1/24**

InterviewQs

Question 9 - Longest palindrome substring

A palindrome is "a word, phrase, or sequence that reads the same backward as forward." Below are some example palindromes:

- madam
- racecar
- tat

Can you write a python function that takes in a string and outputs the longest palindrome? If the string itself is a palindrome, the function would output the original string.

Solution:

Click here to view this solution in an interactive Colab (Jupyter) notebook.

We're going to go through the code below step-by-step. Note that within the for loop there are 2 while loops with similar logic that we'll evaluate in parallel.

- Line 1: Define the function longestPalindrome that takes in a string.
- Lines 2-8: Declare variables to keep track of where we are while parsing through the input. maxLength will keep track of our longest palindrome found, and start will keep track of which index of the string the palindrome starts.
- Line 10: Declare a loop that will start a 1 and iterate through the length of the string.
- Lines 11-18/Lines 20-27: Even/odd length palindrome check, respectively.
    - Lines 11-12/Lines 20-21: We're going to set 2 value, high and low, which will represent the center of our palindrome if we find it to be an even length.
    - Line 13/Line 22: While the low and high are within the boundaries of the string, and if the value of the string at the low character equals the value of the string at the high character, we're going to continue to evaluate the potential palindrome.
        - Lines 14-16/Lines 23-25: If the current palindrome string length is greater than the current length of the palindrome, we're going to reset the start of our palindrome to be the low index, and the maxLength to represent the new max length.
        - Lines 17-18/Lines 26-27: We'll reduce the low by 1 and increase the high by one, and if the condition in line 13 is still valid, we'll expand our palindrome to the next set of 2 characters.

- Lines 29: Once all of the characters within the string have been evaluated, we are going to return the string's portion between the start and the end of the palindrome.

```
1: def longestPalindrome(string):
2:     maxLength = 1
3:
4:     start = 0
5:     length = len(string)
6:
7:     low = 0
8:     high = 0
9:
10:     for i in range(1, length):
11:         low = i - 1
12:         high = i
13:         while low >= 0 and high < length and string[low] == string[high]:
14:             if high - low + 1 > maxLength:
15:                 start = low
16:                 maxLength = high - low + 1
17:             low -= 1
18:             high += 1
19:
20:         low = i - 1
21:         high = i + 1
22:         while low >= 0 and high < length and string[low] == string[high]:
23:             if high - low + 1 > maxLength:
24:                 start = low
25:                 maxLength = high - low + 1
26:             low -= 1
27:             high += 1
28:
```

```
29:     return string[start:start + maxLength]
```

InterviewQs

Question 10 - Chipotle item analysis

You are given a data set of Chipotle orders. You're asked to figure out the average order price and the average price per item ordered. Can you describe how you would do this using Python Pandas? The solution will be Python code which walks through the logic and calculation.

Solution:

Click here to view this solution in an interactive Colab (Jupyter) notebook.

```
#importing pandas package
import pandas as pd


# get the data from the url and put it into a dataframe
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'
chipotle_orders = pd.read_csv(url, sep = '\t')


# clean the item_price column and transform it in a float and reassign the column with th
e cleaned prices
chipotle_orders.item_price = [float(value[1 : -1]) for value in chipotle_orders.item_price]


# getting total price per item (which is item_price * quanitity )
chipotle_orders['total_price_per_item'] = chipotle_orders['item_price'] * chipotle_orders['quantity
']


# aggregating the number of items ordered and the total price for each item (which is the
price per order)
```

```
total_price_per_order = chipotle_orders.groupby(['order_id'])[["total_price_per_item", "quantity"]].sum().reset_index()


# calculating the average price per order
average_order_price = total_price_per_order['total_price_per_item'].mean()


# calculating the average price per item ordered
average_price_per_item = total_price_per_order['average_item_price'].mean()
```

The end result is...
average_order_price = **21.39**
average_price_per_item = **7.84**


InterviewQs