Discussion 9

DSC 20, Spring 2023

Final Exam Practice

We're going to run through practice questions in preparation

Final Exam Logistics

- Make sure to bring pen/pencil/eraser and your **student ID**
- Exam takes place Thursday June 15th, 11:30 am - 2:30 pm

Theme

You’ve been hired by a generic online retailer named after a rainforest named after a river as a software engineer intern. You're tasked to fix some issues about their shipping network. Due to budget cuts, you are the only person left and they will accept any solution you give.

Question 1

Even though there've been many layoffs, the company's customer base continues to grow. In order to support this growth, more items in the inventory need to be ordered according to customer preferences. Given a list of warehouse items and items customers are interested in, return a new order list that multiplies every item that customers like by 3.

In [3]:

```
def increase_supply(inventory, interests):
    """
    >>> inv = [('games', 4), ('clothes',12), ('drinks', 3)]
    >>> targets = ['games','clothes']
    >>> increase_supply(inv, targets)
    [('games', 12), ('clothes',36), ('drinks', 3)]
    """
    # Write your implementation here
```

Question 1 Solution

In [22]:
```
def increase_supply(inventory, interests):
    return [(item[0], item[1]*3) if item[0] \
            in interests else item for item in inventory]
```

In [23]:
```
inv = [('games', 4), ('clothes',12), ('drinks', 3)]
targets = ['games','clothes']
increase_supply(inv, targets)
```

Out[23]:
```
[('games', 12), ('clothes', 36), ('drinks', 3)]
```

Question 2

Some items are now considered "dead stock" and need to be removed from warehouses in order to make room for other items. Given a list representing the current inventory and items that are to be kept, use map/filter/lambda to remove dead stock. Return the new inventory and the new space that was made by removing dead stock.

In [61]:
```
def remove_deadstock(inventory, interests):
    """
    >>> inv = [('games', 4), ('clothes',12), ('drinks', 3)]
    >>> targets = ['games','clothes']
    >>> remove_deadstock(inv, targets)
    ([('games', 4), ('clothes', 12)], 3)
    """
    # Write your implementation here
```

Question 2 Solution

In [62]:
```
def remove_deadstock(inventory, interests):
    remove_stock = filter(lambda item: item[0] in interests, inventory)
    new_space = filter(lambda item: item[0] not in interests, inventory
    return list(remove_stock), sum([x[1] for x in new_space])
```

In [63]:
```
inv = [('games', 4), ('clothes',12), ('drinks', 3)]
targets = ['games','clothes']
remove_deadstock(inv, targets)
```

Out[63]:
```
([('games', 4), ('clothes', 12)], 3)
```

Question 3

Now that we've prepared our warehouses, it's time to process some orders from our loyal customers. Given our inventory and a filepath representing orders from customers, update the inventory accordingly. Maintain and return an order history. You may assume our inventory can always facilitate the given orders.

Since you're the only tech saavy person working at the company now, the data pipeline is not well maintained and sometimes feeds erroneous files. If such a case were encountered, throw an appropriate error saying "invalid order bill - unable to process".

In [64]:
```
def process_orders(inventory, order_path):
    """
    >>> inventory = {'Coffee':10, 'Primo Gems': 9999, 'Beads':100}
    >>> process_orders(inventory, 'files/orders.txt')
    ["Charisse's order for Coffee completed.",\
     "Nicole's order for Primo Gems completed.",\
     "Ben's order for Beads completed."]
    >>> process_orders(inventory, 'nope')
    invalid order bill - unable to process
    """
    # Write your implementation here
```

Question 3 Solution

In [68]:
```
def process_orders(inventory, order_path):
    """
    >>> inventory = {'Coffee':10, 'Primo Gems': 9999, 'Beads':100}
    >>> process_orders(inventory, 'files/orders.txt')
    ["Charisse's order for Coffee completed.",\
     "Nicole's order for Primo Gems completed.",\
     "Ben's order for Beads completed."]
    >>> process_orders(inventory, 'nope')
    invalid order bill - unable to process
    """
    history = []
    try:
        with open(order_path, 'r') as f:
            orders = f.readlines()
            for order in orders:
                name, item, amount = order.strip().split(', ')
                inventory[item] -= int(amount)
                history.append("%s's order for %s completed."%(name,ite
        return history

    except FileNotFoundError as e:
        print('invalid order bill - unable to process')
```

In [69]:
```
inventory = {'Coffee':10, 'Primo Gems': 9999, 'Beads':100}
process_orders(inventory, 'files/orders.txt')
```

Out[69]:
```
["Charisse's order for Coffee completed.",
 "Nicole's order for Primo Gems completed.",
 "Ben's order for Beads completed."]
```

In [70]:
```
process_orders(inventory, 'nope')
```

```
invalid order bill - unable to process
```

Question 4

Some code on the code base needs to be updated - some teenagers are breaking the website by passing in arguments that don't make sense. Given the following code, write assert statements to prevent nonsensical inputs.

In [79]:
```
def collect_review(item, name, quantity, rating, review):
    """
    Collects a review and stores output in a dictionary.
    """
    rating_prop = rating / 5
    return {'name': name, 'item':item, \
    'quantity': quantity, 'rating': rating_prop, 'review': review}
collect_review('genshin impact', 'nikki',1,4.5,'xiao nyan')
```

Out[79]:
```
{'name': 'nikki',
 'item': 'genshin impact',
 'quantity': 1,
 'rating': 0.9,
 'review': 'xiao nyan'}
```

Question 4 Solution

In [82]:
```
def collect_review(item, name, quantity, rating, review):
    """
    Collects a review and stores output in a dictionary.
    """
    assert isinstance(item, str)
    assert isinstance(name, str)
    assert isinstance(quantity, int)
    assert quantity > 0
    assert rating >= 0 and rating <= 5
    assert isinstance(review, str)
    rating_prop = rating / 5
    return {'name': name, 'item':item, \
    'quantity': quantity, 'rating': rating_prop, 'review': review}
collect_review('genshin impact', 'nikki',1,4.5,'xiao nyan')
```

Out[82]:
```
{'name': 'nikki',
 'item': 'genshin impact',
 'quantity': 1,
 'rating': 0.9,
 'review': 'xiao nyan'}
```

Question 5

Our company codebase somehow forgot to have a representation for delivery drivers until now, so as the only person who knows how to code, write a class to represent drivers. Each driver will be assigned an ID number on creation. Each driver will have a name, salary, location, delivery count. The driver class will have 2 methods: perform_delivery() and query_promotion(). perform_delivery will increment delivery count by some value and query promotion will check if the driver can be promoted (the driver must have done 100 deliveries to be promoted). If the driver is promoted, their salary increases by 2 dollars. Return whether the operation was successful or not.

In [ ]:
```
class driver:
    # Write your implementation here
```

Question 5 Solution

In [97]:

```
class driver:
    id = 1
    def __init__(self, name, salary, location):
        self.id = driver.id
        self.name = name
        self.salary = salary
        self.location = location
        self.deliver_count = 0
        driver.id+=1

    def perform_delivery(self, num_d):
        self.deliver_count += num_d

    def query_promotion(self):
        if self.deliver_count >= 100:
            self.salary += 2
            return True
        return False
```

Question 6

Write at least 3 doctests to test the previous class and its methods.

In [128]:

```
class driver:
    id = 1
    def __init__(self, name, salary, location):
        self.id = driver.id
        self.name = name
        self.salary = salary
        self.location = location
        self.deliver_count = 0
        driver.id+=1

    def perform_delivery(self, num_d):
        self.deliver_count += num_d

    def query_promotion(self):
        if self.deliver_count >= 100:
            self.salary += 2
            return True
        return False
```

Question 6 Solution

In [129]:
```
>>> d1 = driver('nikki', 0, 'LA')
>>> d1.name # == 'nikki'
>>> d1.deliver_count # == 0
>>> d1.query_promotion() # False
>>> d1.perform_delivery(250)
>>> d1.query_promotion() # True
>>> d1.salary # 2
>>> d1.id # 1
```

Out[129]: 1

Question 7

Now we have a broad class for delivery drivers, but there are also different types of drivers. Let's call drivers that are third-party contractors guest_drivers. Write a sub class of driver that has a salary of 10 and will always be rejected for promotion.

In [ ]:
```
class guest_driver(driver):
    # Write your implementation here
```

Question 7 Solution

In [ ]:

```
class guest_driver(driver):
    def __init__(self, name, location):
        super().__init__(name, 16, location)
    def query_promotion(self):
        return False
```

Question 8

Write at least 3 doctests to test the previous class and its methods.

In [ ]:

```
class guest_driver(driver):
    def __init__(self, name, location):
        super().__init__(name, 16, location)
    def query_promotion(self):
        return False
```

Question 8 Solution

In [132]:
```
>>> gd1 = guest_driver('cassidy', 'SJ')
>>> gd1.name # cassidy
>>> gd1.perform_delivery(1000)
>>> gd1.query_promotion() # False
>>> gd1.id # 2
```

```
cassidy
False
2
```

Question 9

Since budget cuts are evereywhere, we also need to find optimal paths for drivers. Given a route for a driver where 1's represent houses, write a function that uses recursion to find the longest sequence of 1's.

In [200]:
```
def longest_seq(nums):
    """
    >>> longest_seq([0,1,0,1,1,0,1,1,1,1,0])
    4
    """
    # Write your implementation here
```

Question 9 Solution

In [201]:
```
def longest_seq(nums):
    """
    >>> longest_seq([0,1,0,1,1,0,1,1,1,1,0])
    4
    """
    if len(nums) == 0:
        return 0

    if nums[0] == 0:
        return longest_seq(nums[1:])

    count = 1
    for i in range(1, len(nums)):
        if nums[i] == 1:
            count += 1
        else:
            break

    remaining_sequence = longest_seq(nums[count:])
    return max(count, remaining_sequence)
```

In [202]:
```
longest_seq([0,1,0,1,0,1,1,1,1])
```

Out[202]: 4

Question 10

Some code on the server seems to be running slow. Seems like it could be coming from a function that the old manager wrote to find overstocked items and remove them. Figure out the time complexity of this function that may potentially be problematic and figure out a way to improve it.

In [57]:
```
def manager_did_it(inventory, threshold):
    data = list(inventory.items())
    data.sort(key = lambda x: x[1])
    overflow = []
    maintained_items = list(inventory.keys())
    for entry in data:
        if entry[1] >= threshold:
            overflow.append(entry[0])
            maintained_items.remove(entry[0])
    return overflow, maintained_items
```

In [56]:
```
inv = {'Coffee':10, 'Primo Gems': 9999, 'Beads':100}
manager_did_it(inv, 50)
```

Out[56]:
```
(['Beads', 'Primo Gems'], ['Coffee'])
```

Question 10 Solution

The previous function written by the manager had a runtime of because of the .remove() inside of a for loop that was dependent on the number of entries entered. It was also unnecessarily complicated by having a sort at the start.

In [34]:
```
def fixed(inventory, threshold):
    overflow = []
    maintained_items = []
    for item, count in inventory.items():
        if count >= threshold:
            overflow.append(item)
        else:
            maintained_items.append(item)
    return overflow, maintained_items
```

In [35]:
```
fixed(inv, 50)
```

Out[35]:
```
(['Primo Gems', 'Beads'], ['Coffee'])
```

The runtime of our fixed function is , a marginal improvement to the manager's old work. It's also way more readable :)

Thanks for coming!

There's a discussion quiz on canvas!