Bielefeld University
Faculty of Technology

Genome Data Science
Dr. Daniel Dörr

Programming

Summer 2020

Exercises

Number 09, Submission Deadline: July 5, 2020

1. Functional Programming (3P)

1. Implement a function `remove(collection, pos)` as `lambda` function which removes an element at position `pos` from an (ordered) input collection. Use list comprehension in your implementation. (2P)
2. Construct a `reduce` function that concatenates a collection of lists into a single list. Test your implementation with the following input: `[[1, 2, 3], ['string', 'one', 'two'], [0.1, 0.2]]`. Hint: `functools.reduce` provides a third parameter to pass on an initial element. (1P)

2. Lazy Evaluation (3P)

Implement the (infinite) Look-and-Say sequence (https://en.wikipedia.org/wiki/Look-and-say_sequence) through lazy evaluation and compute its first 20 numbers. (3P)

3. Object-oriented Programming (9P)

Model a store with articles and a shopping cart. `Store` and `ShoppingCart` class are already provided below. You may ignore the implementation, but you must consider the information in the docstrings (the comments that are enclosed in `"""`).

A `Store` has a name and no articles at the beginning. You can can add articles and their quantity with the function `add_article`. A `ShoppingCart` belongs to a `Store`. You can add articles by referencing their id with the function `add_article(article_id)`

[2]:
```
class Store:
    """
    Store which holds Articles.

    Parameters
    ----------
    name : string
        The name of the store
    """
    class Item:
        """
        Container for an article and the available number of articles in the␣
 ↪warehouse.
        """
        def __init__(self, article, count):
            self.article = article
            self.count = count

    def __init__(self, name):
        self.name = name
        self.__inventory = {}

    def add_article(self, article, count = 1):
        """
        Add an Article to the stock.

        Parameters
        ----------
        article : Article
            Article which is added to the warehouse.

        count : int, optional, default 1
            Number of articles to be added to the warehouse
        """
        self.__inventory[article.id] = Store.Item(article, count)

    def request_available_item_count(self, article_id):
        """
        Get the available number of items in the warehouse

        Parameter
        ---------
        article_id : string or int
            ID of the article

        Returns
        -------
        count : int
            Returns the available number of items
        """
        if article_id not in self.__inventory:
            return -1
        else:
            return self.__inventory[article_id].count

    def get_article(self, article_id):
        """
        Get the requested article

        Parameter
        ---------
        article_id : string or int
```

```
            ID of the article

        Returns
        -------
        article : Article
            Returns the article with the given id
        """
        return self.__inventory[article_id].article

    def remove_items_from_stock(self, article_id, count = 1):
        """
        Remove articles from the stock

        Parameters
        ----------
        article_id : string or int
            ID of article

        count : int , optional, default 1
            Number of items which are removed from the warehouse
        """
        if self.request_available_item_count(article_id) < count:
            raise ValueError()
        else:
            self.__inventory[article_id].count -= count

    def print_price_list(self):
        """
        Print the pricelist and the available items
        """
        print(self.name, 'price list\n')
        for key in sorted(self.__inventory.keys()):
            item = self.__inventory[key]
            print("""Article id: {}
Number of articles in stock: {}
Price: {:.2f}€
Description: {}
""".format(item.article.id, item.count, item.article.price ,item.article))


class ShoppingCart:
    """
    Shopping Cart in a Store

    Parameter
    ---------
    store : Store
```

```
        Store in which you shop
    """
    def __init__(self, store):
        self.cart = {}
        self.__store = store

    def add_article(self, article_id, count = 1):
        """
        Add an Article to the shopping cart

        Parameters
        ----------
        article : Article
            Article which is added to the cart.

        count : int, optional, default 1
            Number of articles to be added to the cart
        """
        if count <= 0:
            print('The number of article needs to be larger than 0')
            return
        available_count = self.__store.request_available_item_count(article_id)
        if available_count == -1:
            print('An article with id {} does not exist in {}'.
↪format(article_id, store.name))
            return
        elif available_count == 0:
            print('Sorry. The article with the id {} is sold out'.
↪format(article_id))
            return
        elif available_count < count:
            print('Sorry. Only {} items are available instead of the requested␣
↪{} articles'.format(available_count, count))
        added_count = min(count, available_count)
        if article_id in self.cart.keys():
            self.cart[article_id] += added_count
        else:
            self.cart[article_id] = added_count
        store.remove_items_from_stock(article_id, added_count)
        print('Added {} articles with id {}'.format(added_count, article_id))

    def print_content(self):
        """
        Print the content of the shopping card
        """
        total_price = 0
        total_items = 0
        print('Shopping Cart:')
        for article_id in self.cart:
            count = self.cart[article_id]
            article = store.get_article(article_id)
            total_price += count * article.price
            total_items += count
            if count > 0:
                print('{:0>2d}x {}'.format(count, article))
        if total_items == 0:
            print('Empty')
        else:
            print('Total: {:0>2d} items for {:.2f}€'.format(total_items,␣
↪total_price))
```

[3]:
```
store = Store("Smith & Smith Store")
my_cart = ShoppingCart(store)
```

1. Create 7 classes to model the following situation. (7P)
   - An article has an id and a price.
   - A book is an article and has a title and an author. Both, e-book and paper-book, are books and have a file size and a number of pages, respectively.
   - An audio album is an article and has an interpret, title and some number of songs. The specialized audio albums, CD-album and digital album, have a number of CDs and a total file size, respectively.
   - For each fully specialized class, implement the "magic" function `__str__()` which should return a nicely formatted description of the article.

   Create least one more subclass of `Article`
   Hint: `super().__init__()` calls the constructor of the parent class
2. Add at least 6 articles to the `store`. (1P)
   Possible items are
   Article id: 1
   Number of articles in stock: 20
   Price: 9.99€
   Description: PaperBook, 1984, George Orwell, 328 Pages
   Article id: 2
   Number of articles in stock: 7
   Price: 7.99€
   Description: Digital Album, Michael Jackson, Thriller, 9 Songs, 2MB
   Article id: 3
   Number of articles in stock: 10
   Price: 5.99€
   Description: EBook, Lord of the rings, J.R.R. Tolkien, 500KB
   Article id: 4

Number of articles in stock: 5
Price: 12.99€
Description: CD Album, The Beatles, Abbey Road, 17 Songs, 2 CD's

3. Add some articles to the shopping cart `my_cart` and print the content of the cart. (1P)

Try out to put more articles into the cart than there are in the store