Practice Final Exam: Solutions

Overview

This practice test is covered by Bowdoin's Honor Code.

- You may not use the Internet (outside of the CodeRunner Exam IDE) or use/share any other resources not explicitly allowed.
- You are prohibited from sharing or discussing the contents of this practice exam with other students (including future students).
- This practice exam is Level 3 in terms of the CS collaboration policy. You are only allowed to ask the professor for clarification.

The test consists of 17 questions (ignoring Question 1), and is 175 minutes long (2 hours, 55 minutes).

- Each question (ignoring Q1) is worth 1 point, for a total of 17 points.
- Each multiple-choice or fill-in-the-blank question will have a **25%** penalty per wrong submission.
- Each programming question will have a **0%** penalty for wrong submissions.

You do not have to pass Pylint checks for this practice exam. (Docstrings, good spacing style, variable name lengths, etc. are not graded during programming questions.)

Before you begin, please make sure your cellphone is off and in silent mode.

Question 1

By selecting "True", you agree that you understand and will adhere to the Bowdoin Academic Honor code and the CSCI 1101 policies. Failure to adhere to these rules may result in a Judicial Board hearing.

By selecting "False" or leaving this question blank, you forfeit the opportunity to complete the exam.
Select one:

- ☐ True
- ☐ False

Note that the programming solutions below do not have docstrings, and many of the following are merely examples of possible answers. You may have found an alternative solution that works just as well.

Question 2

What is the difference between ‘‘`True`’’ and `True` in Python?

One is a string while the other is a Boolean.

Question 3

Name at least three different *sequences*. How do they differ?

There are many possible responses here, but you might have listed strings, lists, or tuples.

Question 4

Which of the following is a list method?

1. union
2. **extend**
3. keyerror
4. add
5. concatenate

Question 5

Why is it important to close a file when you are done with it?

Again, there are a number of reasons - most significantly, this can impact performance.

Question 6

What is the difference between a `for-each` and a `while` loop?

Responses vary, but there are generally differences in terms of overall setup and when to use each (a while loop can be used if we don't know when it will stop). Consider what variables need to be specified in each case and where.

Question 7

Write a function `near_and_far()` that accepts three ints `a`, `b`, and `c` as arguments. Return True if one of `b` or `c` is "close" (differing from `a` by at most 1), while the other is "far", differing from both other values by 2 or more.

```
def near_and_far(a, b, c):
    cond1 = abs(a-b) <= 1 and abs(b-c) >= 2 and abs(a-c) >= 2
    cond2 = abs(a-c) <= 1 and abs(a-b) >= 2 and abs(c-b) >= 2
    return cond1 or cond2
```

Question 8

Write a function `sum_without_twenties(a, b, c)` that returns the sum of three int arguments `a`, `b`, and `c`. However, do not include any int as part of the sum if it is within the range [20, 29] (inclusive).

```
def sum_without_twenties(a, b, c):
    nums = [a, b, c]
    sum = 0
    for num in nums:
        if not is_twenty_something(num):
            sum += num
    return sum

def is_twenty_something(num):
    return 20 <= num <= 29
```

Question 9

Write a function `get_substring_positions()` that accepts two strings as arguments. Return the number of the positions where they contain the same substring of length 2. For example, "docatzz" and "dobatz" should yield 3, since the "do", "at", and "tz" substrings appear in the same place in both strings.

```
def get_substring_positions(a, b):
    min_length = min(len(a), len(b))
    count = 0
    for i in range(min_length-1):
        if a[i:i+2] == b[i:i+2]:
            count += 1
    return count
```

Question 10

Write a function `debug()` that takes in two sets, and makes a new set that contains the elements both sets have in common. If "bug" is one of the elements, remove that from the new set. Return the new set.

For example,

```
debug({"bug", "rice", "apple"}, {"rice", "bug", "sugar"})
```

should evaluate to `{'rice'}`.

```
def debug(set1, set2):
    return set1.intersection(set2) - {"bug"}
```

Question 11

Write a class named `Book` that represents each book in a library. Each book has the following attributes:

- `title`: the title of the book.
- `author`: the author of the book.
- `content`: a string representing the text of the book. Each page is separated by the '\n' character.

The `Book` class should have an initializer that accepts the book's `title`, `author`, and `content` as arguments. These values should be assigned to the object's respective attributes.

The class should also have the following methods:

- `read()`: takes in a page number, and prints out the content of each page starting from the beginning until the given page number is reached.
- `__str__()` : return a string describing this book in terms of its name and author (e.g., "Adventures of Huckleberry Finn by Mark Twain").

```
class Book:
    def __init__(self, title, author, content):
        self.title = title
        self.author = author
        self.content = content

    def read(self, page):
        pages = self.content.split("\n")
        for i in range(0, page-1):
            print(pages[i])

    def __str__(self):
        return self.title + " by " + self.author
```

On the exam, you will have example test cases. These questions were meant to get you thinking about what those test cases would be (to help prepare you for hidden tests, and for general practice with computational thinking). You could have included more error-checking here (for example, checking whether the page number is valid), but this is an acceptable solution.

Question 12

Write a class named `ComicBook` that inherits the `Book` class from the previous question. A `ComicBook` should have an additional boolean attribute `is_superhero` that indicates whether it is a comic about superheroes or not. The value of this attribute should be able to be set when initialized.

```
class ComicBook(Book):
    def __init__(self, title, author, content, is_superhero):
        super().__init__(title, author, content)
        self.is_superhero = is_superhero
```

Question 13

Write a class named `LibraryEmployee`. Each `LibraryEmployee` has the following attributes:

- `name`: the name of the employee.
- `favorites`: a set of this employee's favorite `Books` (or `ComicBooks`).
- `num_coffees`: an integer representing how many coffees this employee has had today. This value should begin at zero when a `LibraryEmployee` is initialized.

The `LibraryEmployee` class should have an initializer that accepts the employee's `name` and `favorites` as arguments. These values should be assigned to the object's respective attributes.

Additionally, the `LibraryEmployee` class should have the following methods:

- `drink_coffee()`: takes in an integer `num`, and increases this `LibraryEmployee`'s `num_coffees` attribute by that integer.
- `shush_hooligans()`: takes in an integer `num_hooligans`, and prints out "Shhh!" that many times. Also, drinks a coffee for each hooligan shushed.
- `__str__()` : return the `LibraryEmployee`'s name.

```
class LibraryEmployee:
    def __init__(self, name, favorites):
        self.name = name
        self.favorites = favorites
        self.num_coffees = 0

    def drink_coffee(self, num):
        self.num_coffees += num

    def shush_hooligans(self, num_hooligans):
        for i in range(num_hooligans):
            print("Shhh!")
            self.drink_coffee(1)

    def __str__(self):
        return self.name
```

Question 14

Write a class named `Library`. A library has the following attributes:

- `name`: the name of this library.
- `employees`: a list of `LibraryEmployee`s that work at the library.
- `books`: a dictionary that contains `Book`s as keys. The value paired with each key is either an empty string, or a string representing the name of a person who has checked out the `Book`.

The `Library` class should have an initializer that **MAY** accept `employees` and `books` as arguments, but **doesn't have to do so**. If provided, these values should be assigned to the object's respective attributes. If not provided, `employees` and `books` should begin as an empty list and dictionary, respectively. The value for the `name` attribute is always provided.

The class should also have the following methods:

- `hire_employee()`: takes in a new `LibraryEmployee`, and adds it to this library's list of `employees`.
- `add_book()`: takes in a new `Book`, and adds it to this library's `books` (unless the library already has this book). If the library already has the book, print out the name of the library followed by " already has this book!".
- `check_out_book()`: takes in a `Book` and a person's name. If this book is in the dictionary of `books`, change the value at that key in the `books` dictionary to the person's name. Otherwise, print out "Sorry, we don't have " followed by the name of the book and a period.
- `return_book()`: takes in a `Book`. If this book is in the dictionary of `books`, change the value at that key in the `books` dictionary to an empty string. Otherwise, print out "Sorry, wrong library.".
- `__str__()` : return the name of the library.

```python
class Library:
    def __init__(self, name, employees=[], books={}):
        self.name = name
        self.employees = employees
        self.books = books

    def hire_employee(self, employee):
        self.employees.append(employee)

    def add_book(self, book):
        if book in self.books:
            print(self.name + " already has this book!")
        else:
            self.books[book] = ""

    def check_out_book(self, book, person):
        if book in self.books:
            self.books[book] = person
        else:
            print("Sorry, we don't have " + book.title + ".")

    def return_book(self, book):
        if book in self.books:
            self.books[book] = ""
        else:
            print("Sorry, wrong library.")

    def __str__(self):
        return self.name
```

Question 15

Consider the sequence 2, 3, 6, 18, 108, 1944, 209952...

Write a **recursive** function `next_num(n)` that calculates the nth number of the sequence.

```
def next_num(n):
    if n == 1:
        return 2
    elif n == 2:
        return 3
    else:
        return next_num(n-1) * next_num(n-2)
```

Question 16

Write a **recursive** function `get_biggest()` that takes in a list parameter and returns the largest number in the list. Hint: remember that you can use the built-in function `max()`.

```
def get_biggest(my_list):
    if len(my_list) == 1:
        return my_list[0]
    else:
        return max(my_list[0], get_biggest(my_list[1:]))
```

Question 17

Write a **recursive** function `reverse()` that returns the value of its (string) input parameter, except reversed. For example, reverse("summer") should return "remmus".

```
def reverse(text):
    if len(text) == 0:
        return ""
    else:
        return text[-1] + reverse(text[0:-1])
```

Question 18

Write a **recursive** function `separate()` that takes in a single list of integers as a parameter. It should separate the odd and even integers so that the odd numbers are together at the beginning of the list, and the even numbers are together at the end of the list. This "separated" list is what should be returned.

For example,

```
print(separate([1, 2, 3, 7, 2, 4]))
```

...should result in something like:

```
[1, 3, 7, 4, 2, 2]
```

```
def separate(data):
    if len(data) == 0:
        return []
    #odds
    if data[0] % 2 != 0:
        return data[0:1] + separate(data[1:])
    #evens
    else:
        return separate(data[1:]) + data[0:1]
```