CS146 Fall 2024 – Midterm 2 Solution

Name: _____ **Section: A**

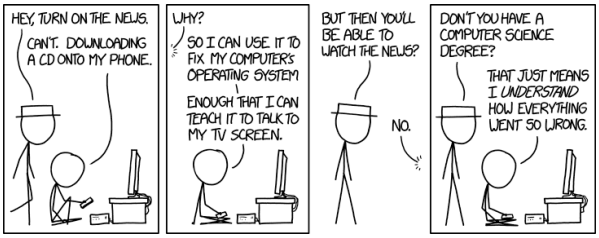
Date: _____ **Start time:** _____ **End time:** _____

Honor Code:

Signature: _____

This exam is closed book, closed notes, closed computer, closed calculator, etc. You may only use (1) the midterm 2 "cheat sheet" from the course page, and (2) a single double-sided letter-sized sheet of notes of your own creation. **You have 2.5 hours.** Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. Circle or otherwise indicate your answer if it might not be easily identified. You may use extra sheets of paper, stapled to your exam, if you need more room, as long as the problem number is clearly labeled and your name is on the paper. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.

You <u>do</u> need to include module imports (if relevant for your code), but <u>do not</u> need to include comments, docstrings or constants in your code.



Question 9: Connect Python to the outside world with file I/O and command line arguments

Write an entire program (not just function) that when run with the green arrow in Thonny reads a file containing e-mail addresses provided as a command line argument and prints them out with a new domain also provided as a command line argument. Example usage is shown below. Your program should work for any filename and domain provided. You can assume the inputs are correctly formatted, i.e., file contents are always correctly formatted e-mail addresses with one "@" between the username and domain and the new domain name is valid. If your program is imported, nothing should be printed. You can assume a correct number of command line arguments are provided (i.e., you don't need to print a "usage" string). Your solution will be exclusively evaluated on correctness.

```
>>> %Run new_domain.py emails.txt middlebury.edu
panther@middlebury.edu
gamelial@middlebury.edu
twilight@middlebury.edu
```

Example contents of `emails.txt` file:

```
panther@example.com
gamelial@example.com
twilight@example.com
```

Solution:

```
import sys

def new_domain(filename, domain):
    with open(filename, "r") as f:
        for line in f:
            at = line.index("@")
            print(line[:at+1] + domain)

if __name__ == "__main__":
    new_domain(sys.argv[1], sys.argv[2])
```

Question 10: Implications of the Python memory model

Add to the body the `mystery` function (writing relevant code on the lines) such that after the code below executes, `y` is not equal to its initial value. If no such body is possible, check the box below. Briefly explain your answer. If you do provide code, you do not need to use all of the lines.

√ **No such function body exists**

```
def mystery(x):
    x = x[:]
    for i in range(len(x)):
        x[i] = x[i][:]
```

_____

_____

```
y = [[1, 2], [3, 4], [5, 6]]
mystery(y)
# y is no longer [[1, 2], [3, 4], [5, 6]]
```

Solution: Since the `mystery` copies both outer and nested lists, no parts of `x` and `y` remain aliased. Therefore, there is no way to change the value of `y` on those two lines.

Question 11: Applications of data structures

You are writing a program that receives timestamped log messages (i.e., a timestamp in seconds since the epoch followed by a message) from many different senders via the Internet (including some sent at the same time). Due to the nature of the Internet, messages may arrive in any order, but you need to analyze them ordered by timestamp. Which of the following data structures would be an appropriate choice for this task? *Select all that apply.* Briefly explain your choices, including why data structures you did not select would not be appropriate.

- √ **List**
- ◯ Set
- ◯ Dictionary
- ◯ Tuple

Solution: A list is an appropriate choice because it enables to maintain the order the messages and can be sorted by timestamp so the messages can be analyzed in order. Further it can allow messages with identical timestamps. A set would not be appropriate because it does not maintain order. A dictionary (e.g., with the timestamps as keys) would not be appropriate because it is would require additional data to handle duplicate timestamps and is maintained in insertion order, instead of our desired order by timestamp. A tuple would not be appropriate because it is immutable and so cannot be sorted.

Question 12: Writing functions with sets

I typically teach two classes per semester. I want to know how many students I am teaching in the fall are also enrolled in one of my spring classes. Write a function named `enroll` that takes four lists of names, two for students enrolled in the fall, two for students enrolled in the spring, and returns the number of students overlapping between fall and spring. You can assume all student names are unique and each unique student should only be counted once. Your solution will be evaluated based on the correctness, efficiency and conciseness of your approach. For example:

```
>>> fall1 = ["Hopper", "Church"]
>>> fall2 =  ["Babbage", "Turing"]
>>> spring1 = ["Turing", "Hopper"]
>>> spring2 = ["Liskov", "Hopper"]
>>> enroll(fall1, fall2, spring1, spring2)
2
```

Note that I taught "Hopper" and "Turing" in both the fall and the spring, but "Hopper" is only counted once despite being in both spring classes.

Solution:

```
def enroll(fall1, fall2, spring1, spring2):
    set1 = set(fall1 + fall2)
    set2 = set(spring1 + spring2)
    return len(set1 & set2)
```

Question 13: Writing functions with dictionaries

I maintain a record of all the students I have ever taught as a list of lists. Each sublist contains the names of the students in a particular class, e.g., CSCI146 Fall 2024. Write a function named `num_taught` that returns a list of students I have taught exactly `num` times. Your function should take two parameters, `classes`, which is a list of lists of strings, and `num`, an integer. Your solution will be exclusively evaluated on correctness. You can assume all student names are unique. Example usage is shown below.

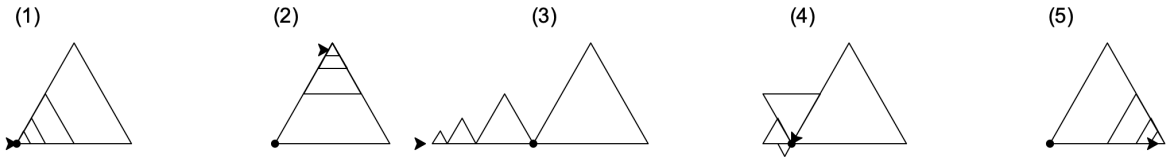
```
>>> my_classes = [
    ["Hopper", "Church", "Babbage", "Turing"],
    ["Wolfram", "Simon", "Hopper", "Babbage", "Knuth"],
    ["Liskov", "Knuth", "Turing"]
]
>>> num_taught(my_classes, 1)
['Church', 'Wolfram', 'Simon', 'Liskov']
>>> num_taught(my_classes, 2)
['Hopper', 'Babbage', 'Turing', 'Knuth']
>>> num_taught(my_classes, 3)
[]
```

Solution:

```
def num_taught(classes, num):
    counts = {}
    for students in classes:
        for student in students:
            if student in counts:
                counts[student] += 1
            else:
                counts[student] = 1
    result = []
    for student, count in counts.items():
        if count == num:
            result.append(student)
    return result
```

Question 14: Understanding and using recursive functions

(a) For each of the 4 functions `funA - funD` defined below, identify the resulting picture by number among the 5 options below. Assume in each case the function was called like "`funX(100,4)`", e.g., `funA(100,4)`. The turtle starts at the origin (marked with a dot) facing to the right and ends at the position and heading shown by the arrow.




```
import turtle as t

def triangle(s):
    for i in range(3):
        t.forward(s)
        t.left(120)
```

4

```
def funA(s, n):
    if n > 0:
        triangle(s)
        t.left(60)
        funA(s/2, n-1)
```

1

```
def funC(s, n):
    if n > 0:
        triangle(s)
        funC(s/2, n-1)
```

2

```
def funB(s, n):
    if n > 0:
        triangle(s)
        t.penup()
        t.left(60)
        t.forward(s/2)
        t.right(60)
        t.pendown()
        funB(s/2, n-1)
```

3

```
def funD(s, n):
    if n > 0:
        triangle(s)
        t.penup()
        t.backward(s/2)
        t.pendown()
        funD(s/2, n-1)
```

(b) For the remaining 5th picture, provide the missing function definition below. Your function should be named `funE`. Write the corresponding picture number on the line below.

5

Solution:

```
def funE(s, n):
    if n > 0:
        triangle(s)
        t.forward(s/2)
        funE(s/2, n-1)
```

Question 15: Finding errors (in recursive functions)

The following recursive function is supposed to translate a string to its corresponding integer, e.g., "1000" to 1000. Unfortunately, there are several different problems with this function. Identify (by line number), explain, and fix all of the problems, and rewrite the corrected function, *using a recursive strategy without loops that only invokes* `int` *on one digit at a time*. The errors should not be variations of the same issue and should impact correctness, not just style. Examples of the intended behavior are shown below.

```
>>> string_to_int("1000")
1000
>>> string_to_int("0")
0
>>> string_to_int("1234")
1234
```

```
1 def string_to_int(s):
2     if len(s) == 1:
3         return 0
4     else:
5         index = len(s) - 1
6         return int(s) * (10 ** index) + string_to_int(s[1:])
```

Solution:

1. The base case on line 2 should be `if len(s) == 0`, not `if len(s) == 1`. Alternatively the implementation of the base case on line 3 could be changed to

   ```
   if len(s) == 1:
       return int(s)
   ```

2. The first part of the recursive case on line 6 should be `int(s[0])`, as is it produces a number that is too large.

```
def string_to_int(s):
    if len(s) == 0:
        return 0
    else:
        index = len(s) - 1
        return int(s[0]) * (10 ** index) + string_to_int(s[1:])
```

Question 16: Using Object-Oriented Programming

Consider the following classes for representing a bank account:

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    def get_balance(self):
        return self.balance

    def deposit(self, amount):
        self.balance += amount
```

(a) Write a method for `BankAccount` named `withdraw` that takes a single parameter, `amount`, and withdraws that amount from the account, adjusting the balance accordingly. If a withdrawal would result in a negative balance (i.e., insufficient funds) the method should not make any changes and return `False`. If there are sufficient funds, the method should perform the withdrawal and return `True`. You do not need to provide the whole class, just the method implementation. Your solution will be evaluated based on the correctness, efficiency and conciseness of your approach.

Solution:

```
def withdraw(self, amount):
    if self.balance < amount:
        return False
    else:
        self.balance -= amount
        return True
```

(b) A subset of the bank's customers have another type account with overdraft protection, which allows the account to have a negative balance up to a certain limit (i.e., withdraw more than the current balance). You will implement this type of account with the class `OverdraftAccount` that inherits from `BankAccount`. Which of the following methods should be overridden in the `OverdraftAccount` class? *Select all that apply.* Briefly explain your answer.

√ `__init__`
◯ `get_balance`
◯ `deposit`
√ `withdraw`

Solution: The assumption when writing the problem was that each account had a different overdraft limit and so we would need to override `__init__` to set an instance variable with that limit. But that was not stated in the problem. Answers that did or did not override `__init__` were accepted. The `get_balance` and `deposit` methods do not need to be overridden as they are the same for both types of accounts. The `withdraw` method should be overridden to allow withdrawals that exceed the current balance up to the overdraft limit.

Page intentionally left blank.