CMPSC132 – Fall 2024 Examination Session Quiz #1

INSTRUCTIONS

- You have 1 hour and 15 minutes to complete the exam. 6 questions, 75 points total
- Your final answers should be written in the ANSWER SHEETS provided. Those are the pages we will be grading.
- Write your name in your exam.
- The exam is closed book, closed notes, and closed computer, except for the two sheets of notes that were created by you.
- The use of cellphones, earbuds, headphones, tablets, or other communication devices during the exam is prohibited. CELLPHONES MUST BE MUTED AND AWAY FROM YOUR WORKSPACE.
- Write your answers and show all your work on the exam itself. We will not grade answers written on scratch paper.
- If you need to use the restroom during the exam, bring your phone and exam materials to the front of the room to a TA.
- Some questions forbid the use of certain libraries/functions. If you use them in your answer, you will not receive any credit for that question.
- Violations of academic integrity are serious and will be handled as defined in the syllabus, resulting in a zero score for this exam. Violations include but are not limited to:
  - Verbal or written communication between students (e.g., sending questions to other students).
  - Give or receive unauthorized help (including, but not limited to cell phones, smart devices, other students).
- You must not discuss this exam with any student who has already taken the exam prior to you taking it, post any information about this exam anywhere, or discuss the exam with any student in this course who has not yet completed the exam.

First Name	Jack
Last Name	Mussoline
9-digit PSU id	9 5908 1224
Email id (**@psu.edu)	jvm7210@psu.edu

Methods and operators that may be useful for your implementations

List methods/functions

len(x)	Returns the number of elements in list x
y in x	Returns True if y is in list x; False otherwise
x.append(y)	Adds y at the end of list x
[1, 2, 3] + [2, 3, 4]	Returns a new list with that contains all elements in both lists: [1,2,3,2,3,4]

String methods/functions

len(x)	Returns the number of elements in string x
y in x	Returns True if y is in string x; False otherwise
k * x	Returns the string x repeated k times
'hello' + 'world'	Concatenates the 2 strings: 'helloworld'
f'{variable1} has {variable2} coins'	To use formatted string literals: variable1='Sam', variable2= 5 -> 'Sam has 5 coins'
''.join(sequence)	Takes all items in an sequence of strings and joins them into one string: '-'.join(['1', '2', '3']) returns '1-2-3'

Dictionary methods/functions

len(x)	Returns the number of elements in dictionary x
y in x	Returns True if y is in the keys of dictionary x; False otherwise
x[y] = z	Adds key y with value z to the dictionary x
x[y]	Returns the value with key y in dictionary x if exists, raises a KeyError if y is not in the dictionary
x.get(y)	Returns the value with key y in dictionary x if exists, returns None if y is not in the dictionary

Built-in methods/operators

isinstance(object, class)	Returns True if the specified object is of the specified class, False otherwise
%	Returns the remainder of dividing the left-hand operand by right-hand operand: 3%3 = 0, 4%3 = 1, 5%3 = 2, 6%3 = 0, 7%3 = 1, 8%3 = 2, 9%3 = 0
//	Performs integer division 10 // 3 = 3
[start:end]	Returns a subsequence from start to end-1: lst = [3, 4, 5, 6], lst[1:3] returns [4, 5]

Throughout these questions you will need to make use of strings, lists or dictionaries. You can find syntax for common methods and operators in lists, strings and dictionaries on Page 2. Read the description of each question carefully. Coding questions are not graded on efficiency, but basic syntax is expected and is part of the grade (proper use of loops, control structures, indexing, using methods in the correct data types, etc.).

*** Verify your implementation (regardless of completion or correctness) follows the guidelines of the question, otherwise no credit will be given

~ All the Best !

QUESTION #1 [10 pts.]: Write the Python code to implement the function `multiply_values` that takes two dictionaries d1 and d2 and combines them into a new dictionary. The new dictionary should have the same keys as the original dictionaries, and the values should be the product of the values for the keys that both dictionaries have. If a key is only in one of the dictionaries, it should still be in the new dictionary with the same value as in the original dictionary (keeps unique keys from both dictionaries).

- Your function should not modify the input dictionaries
- You can assume that all keys will be lowercase strings
- You can assume all values are numerical
- You are **not allowed** to use Python lists

Example usage:

```
>>>d1 = {'a': 2, 'b': 3, 'c': 4}
>>>d2 = {'a': 5, 'b': 2, 'd': 8}
multiply_values(d1, d2))  # Output: {'a': 10, 'b': 6, 'c': 4, 'd': 8}

>>>d3 = {'x': 7}
>>>d4 = {'y': 10, 'z': 5}
multiply_values(d3, d4))  # Output: {'x': 7, 'y': 10, 'z': 5}
```

QUESTION #1 - Code goes here:

```
def multiply_values(d1,d2):
    for i in d2:
        if( i not in d1):
            d1[i] = d2[i]
        else:
            d1[i] *= d2[i]
    return d1
```

QUESTION #2 [15 pts]: Write the Python code to implement the function `group_by_author` that takes as input a dictionary of books, where the key is a string that represents a book name, and the value is a dictionary with information about the book as shown in the example (spaced for readability). This function creates a new dictionary with keys corresponding to authors, and values are a collection of dictionaries stored in a Python list (see example). In the new dictionary, each author will map to a list of their books with the relevant information like publication date and genre. In essence, the new dictionary should contain keys of `book_title`, `publication_date`, and `genre`. You may assume that no books have a title that is the name of a genre. Note that the author should be removed, and the book title should be added.

- You are not allowed to use 2D lists as a replacement for a dictionary
- You might mutate the original dictionary (it is permitted), but it is not required

```
>>> books = {
...     'Harry Potter and the Goblet of Fire': {
...         'author': 'J.K. Rowling',
...         'publication_date': 2000,
...         'genre': 'fantasy'
...     },
...     'The Hobbit': {
...         'author': 'J.R.R. Tolkien',
...         'publication_date': 1937,
...         'genre': 'fantasy'
...     },
...     'Harry Potter and the Sorcerer's Stone': {
...         'author': 'J.K. Rowling',
...         'publication_date': 1997,
...         'genre': 'fantasy'
...     },
...     'Autobiography of a Yogi': {
...         'author': 'Paramahansa Yogananda',
...         'publication_date': 1934,
...         'genre': 'Philosophy'
...     }
...     'The Lord of the Rings: {
...         'author': 'J.R.R. Tolkien',
...         'publication_date': 1954,
...         'genre': 'fantasy'
...     },
... }

>>> grouped = group_by_authors(books)
>>> grouped
{
    'J.K. Rowling':
          [
                {
                      'book_title': 'Harry Potter and the Goblet of Fire',
                      'publication_date': 2000,
                       'genre': 'fantasy'
                },
                {
                      'book_title': 'Harry Potter and the Sorcerer's Stone',
                      'publication_date': 1997,
```

```
                'genre': 'fantasy'
            },
        ],
    'J.R.R. Tolkien':
        [
            {
                'book_title': 'The Hobbit',
                'publication_date': 1937,
                'genre': 'fantasy'
            }
            {
                'book_title': 'The Lord of the Rings',
                'publication_date': 1954,
                'genre': 'fantasy'
            }

        ],
    'Paramahansa Yogananda':
        [
            {
                'book_title': 'Autobiography of a Yogi',
                'publication_date': 1934,
                'genre': 'Philosophy'
            }
        ]
}
```

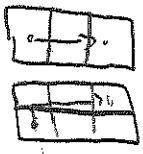
QUESTION #2 - Code goes here:

```
def group_by_author(books):
    solution = {}
    for title, info in books.items():
        solution[info['author']] = [{'book_title': title, 'publication_date':
        -> info['publication_date'], 'genre': info['genre']}]

    return solution
```

QUESTION #3 [10 pts]: Given an n x m grid, write the Python code to implement the recursive function paths(n, m) that returns the number of different possible paths from the bottom left corner to the top right corner. A valid path is any path where at each step you either go one space to the right or one space up.

For example, in a 2 x 3 grid, there are three potential paths: you can go up 1 then right 2, right 1 then up 1 then right 1, or right 2 then up 1.



s→start. Always starts at bottom left

e→ end. Always ends at top right

You can only go UP and RIGHT directions.

To receive ANY credit for this function, you are not allowed to use `for/while` loops, the `in` operator or global variables. No credit will be given if the function is not recursive. You can assume the input follows the described conditions. *Even if you can solve this question mathematically, you must use recursion.

Examples:

```
>>> paths(3, 2)
3
>>> paths(4, 3)
10
```

QUESTION #3 - Code goes here:

```
def Paths ( n , m ):
    if n == m == 2 :
        return 0
    elif n > 2 :
        return n + Paths(n-1, m)
    elif m > 2 :
        return m + Paths(n, m-1)
```

QUESTION #4 [5 pts]: The following code determines if a string matches a pattern. The string will be a string of lowercase alphabetical characters, and the pattern will be a string of lowercase alphabetical characters and potentially question marks. Question marks are wildcards which means they can substitute for any character, or they can substitute as an empty string.

```
1.  def pattern_matching(string, pattern):
2.    """
3.      >>> pattern_matching("abc", "??")
4.      False
5.      >>> pattern_matching("abc", "?b?")
6.      True
7.      >>> pattern_matching("abc", "a?bc")
8.      True
9.      >>> pattern_matching("abc", "abcd")
10.      False
11.    """
12.    if string == pattern == "":
13.          return True
14.    elif string == "" or pattern == "":
15.      return False
16.    elif string[0] == pattern[0]:
17.      return pattern_matching(string[1:], pattern[1:])
18.    elif pattern[0] == "?":
19.      if pattern_matching(string[1:], pattern[1:]) or pattern_matching(string, pattern[1:]):
20.         return True
21.    return False
```

You realize that sometimes you will have a pattern longer than the string and decide that you want a pattern to be considered a match even if it has characters that aren't matched in the string. In other words, if the entire string is matched, you want to consider it a valid match. For example, this would make the final example (abc, abcd) True since having an extra "d" in the pattern would be okay.

- What part of the recursive function should you change to make this work? (line numbers are shown for your convenience)
- Rewrite that section of the code so that it functions as described.

I would change lines 12-15 Since the pattern can have extra letters, we no longer have to have that line to check if they are the same length. Instead, we only have to check that the pattern length is >= the string length. If string is "", then we should return true because pattern has already matched the string

14. elif pattern == "":
15. return false

(Next Page)

QUESTION #4 - Code goes here:

~~14. elif pattern == "":~~
15. return false

12. if string = pattern = "" or string = "":
13. return true
14. elif pattern == "":
15. return false

QUESTION #5 [25 pts]: Write the Python code to implement the `Fraction` class according to the following class explanation. Then, complete the implementation of the `main` function to interact with the `Fraction` class as described in the specification of the function.

Fraction

Attributes

- *can_have_floats*
  - A class attribute that is initially set to False. The use will be explained later.
- *num*
  - An instance attribute that stores the numerator of the fraction.
- *den*
  - An instance attribute that stores the denominator of the fraction.

Methods

- *Initialization function*
  - The initialization function will take in two parameters, the numerator and then the denominator, and initialize the attributes appropriately.
- *reciprocal*
  - A method with no attributes that modifies the current instance so that it becomes the reciprocal of what it currently is. The reciprocal is defined by flipping the numerator and denominator, ie 2/3 -> 3/2. So, you need to swap the numerator and denominator of the fraction instance.
    Do not worry about dividing by zero.
- *Method for multiplication using * operator*
  - A method that enables Fraction * Fraction, Fraction * int, and potentially Fraction* float calculations.
    - Fraction * int returns a new Fraction where both the numerator and denominator are multiplied by the integer.
    - Fraction * Fraction returns a new Fraction with each Fraction's numerator and each Fraction's denominator multiplied together.
    - Fraction * float should function just like Fraction * int if can_have_floats is True for this object, and return "Invalid Operation" otherwise.
  - You may assume the other input will only be either a Fraction, int, or float. This method should not modify the instances.
- *value*
  - A property method that returns a float representation of the Fraction, obtained by dividing the numerator by the denominator.
- While the doctest will assume the presence of string and repr methods, you do not need to implement them.

main() function

This is a standalone function and does not belong to the `Fraction` class. Within a function main, write Python statements to do the following:

- All Fractions created in the future will be able to have numerators with floats.
- Create an instance of Fraction named regular_fraction equivalent to 1/1.
- Modify regular_fraction so that it cannot multiply with a float to create a new Fraction with floats. This should not modify the behavior for any other instances (or future instances).

- Create a Fraction where the numerator is the fraction 1/3, and the denominator is the fraction 3/4. Note that both 1/3 and 3/4 need to be Fraction instances.

Q5. Examples for Fraction class:

```
>>> f1 = Fraction(1, 2)
>>> f2 = Fraction(1, 4)
>>> f1 * f2
Fraction(1, 8)
>>> f1 * 2
Fraction(2, 2)
>>> f1 * 2.5
'Invalid Operation'
>>> f1.value
0.5
>>> f1.reciprocal()#(Returns no value)
>>> f1.value #(division value)
2.0
```

Q6. Examples for Square class:

```
>>> rect = Rectangle(Point(0, 0), 10, 20)
>>> rect.get_area()
200
>>> rect.print_name()
Rectangle
>>> square = Square(Point(0, 0), 10)
>>> square.get_area()
100
>>> square.print_name()
Square
>>> isinstance(square, Rectangle)
True
>>> square.color
'red'
```

QUESTION #5 - Code goes here:

```
class Fraction:
    def __init__(self, num, den):
        self.numerator = num
        self.denominator = den
        can_have_floats = false

    def reciprocal(self):
        temp = self.numerator
        self.numerator = self.denominator
        self.denominator = temp

    def value(self):
        return self.numerator / self.denominator

    def __mul__(self, other):
        if type(other) == int:
            return Fraction(self.numerator * other, self.denominator)
        elif type(other) == Fraction:
            return Fraction(self.numerator * other.numerator, self.denominator * other.denominator)
        elif type(other) == float and self.can_have_floats:
            return Fraction(self.numerator * other, self.denominator)
        else:
            return 'Invalid operation'
```

```
def main():
    Fraction.can_have_floats = True
    regular_fraction = Fraction(1, 1)
    regular_fraction.can_have_floats = False
```

QUESTION #6 [10 pts]: Write the Python code to implement the Square class, a subclass of Rectangle. The following code is given to you.

```
class Point:
	def __init__(self, x, y):
		self.x = x
		self.y = y

class Rectangle:
	def __init__(self, point, width, height):
		self.point = point # This corresponds to the lower left corner of the rectangle
		self.width = width
		self.height = height
		self.color = "yellow"

	def get_area(self):
		return self.width * self.height

	def print_name(self):
		print("Rectangle")
```

Implement Square based on the following:

Use proper inheritance principles when coding this question as that is part of your grade!!!!!

- Initialization function
  - The initialization function takes a point corresponding to the lower left corner of the square, and a side length, and initializes point, width, and height attributes correctly. The color attribute should be initialized as "red".
    - Note: Your initializing function should use the superclass initialization so that you can add additional functionality to Rectangle initialization, without having to add it in both classes.
- print_name
  - Write the minimum necessary code so that print_name for a Square object will just print "Square".
- get_area
  - Write the minimum necessary code in Square so that getArea() on a square object returns the correct area.

QUESTION #6 - Code goes here:

```
class Square(Rectangle):
    def __init__(self, Point, width):
        super().__init__(point, width)
        self.color = "red"
        self.height = self.width
```

(Next Page)

```
def print_name(self):
    return "Square"

def getArea(self):
    return self.width ** 2
```