Spring 2024

[EC2202] Data Structures

Midterm: 1 pm, Tuesday, Apr. 16

INSTRUCTIONS

- **Do not open your exam sheet** until directed to do so, otherwise you would be considered cheating.
- You have **1 hour and 50 minutes** to complete the exam (8 problems; maximum of 100 points).
- The exam is closed book, closed notes, closed computer, and closed calculator.
- Mark your answers on the exam sheet itself and be sure to **write your answers in the space (boxes) provided**. We will not consider the answers given outside the boxes (use other spaces for brainstorming).

POLICIES & CLARIFICATIONS

- If you need to use the **restroom**, bring your exam sheet to the back of the room. Only one person is allowed at a time.
- You may use built-in Python functions that do not require import, such as min, max, pow, len, and abs.
- For **What Would Python Print (WWPP) problems**, write the results that would show up in the Colab environment.
- For **coding problems**, we will ignore minor grammar mistakes for evaluation (focus on the logic and flow). Moreover, your code must be indented correctly.
- Use **English** for your answers and name.

Student ID Number	Name

Q1. (16 points) What Would Python Print (WWPP)

For the code blocks given on the left, write what Python would print in the box on the right. For your answers, include ***all the resulting print outputs*** in ***the correct sequential order***. Assume that you are running the code blocks in the Colab environment.

(a) **(4 points)**

```
func = lambda : print("I love GIST")
print(func())
func
```

```
I love GIST
None
<function __main__.<lambda>()>
```

(b) **(4 points)**

```
multi = lambda f: lambda x: f(f(f(x)))
multi(lambda y: y + 1)(0)
```

```
3
```

(c) **(4 points)**

```
print(True and -1)
print(print(3) or "")
print(max or 0)
```

```
-1
3

<built-in function max>
```

(d) **(4 points)**

```
nums = {7: 'G', 5: 'I', 10: 'S', 2: 'T'}
print(nums.get('X', 0))
print(sum(nums))
del nums['X']
```

```
0
24
KeyError
```

Q2. (10 points) Functions

(a) (**5 points**) Implement `swipe`, which prints the digits of argument `n`, one per line, first backward then forward. The leftmost digit is printed only once. Do not use while or for or str; but use recursion.

```
def swipe(n):
    """
    >>> swipe(2837)
    7
    3
    8
    2
    8
    3
    7
    """
```

```
    if n < 10:
        print(n)
    else:
        print(n % 10)
        swipe(n // 10)
        print(n % 10)
```

(b) (**5 points**) Implement the below function which returns the product of every other positive integer, starting with `n`.

```
def skip_factorial(n):
    """Return the product of positive integers n * (n - 2) * (n - 4) * ...

    >>> skip_factorial(5) # 5 * 3 * 1
    15
    >>> skip_factorial(8) # 8 * 6 * 4 * 2
    384
    """
```

```
    if n <= 2:
        return n
    else:
        return n * skip_factorial(n - 2)
```

Q3. (20 points) Vending Machine

Create a vending machine that only outputs a single product and provides change when needed. Implement a class called VendingMachine that represents a vending machine for certain products. A ***VendingMachine*** object returns strings describing its interactions. Remember to ***match exactly the strings in the doctests*** including punctuation and spacing! Fill in the VendingMachine class, adding attributes and methods as appropriate, such that its behavior matches the doctests. Each method is worth ***5 points***.

```
class VendingMachine:
    """A vending machine that vends some product for some price.

    >>> v = VendingMachine('candy', 10)
    >>> v.vend()0
    'Nothing left to vend. Please restock.'
    >>> v.add_funds(15)
    'Nothing left to vend. Please restock. Here is your $15.'
    >>> v.restock(2)
    'Current candy stock: 2'
    >>> v.vend()
    'Please add $10 more funds.'
    >>> v.add_funds(7)
    'Current balance: $7'
    >>> v.vend()
    'Please add $3 more funds.'
    >>> v.add_funds(5)
    'Current balance: $12'
    >>> v.vend()
    'Here is your candy and $2 change.'
    >>> v.add_funds(10)
    'Current balance: $10'
    >>> v.vend()
    'Here is your candy.'
    >>> v.add_funds(15)
    'Nothing left to vend. Please restock. Here is your $15.'

    >>> w = VendingMachine('soda', 2)
    >>> w.restock(3)
    'Current soda stock: 3'
    >>> w.restock(3)
    'Current soda stock: 6'
    >>> w.add_funds(2)
    'Current balance: $2'
    >>> w.vend()
    'Here is your soda.'
    """

    def __init__(self, product, price):
        self.product = product
        self.price = price
        self.stock = 0
```

```python
        self.balance = 0

    def restock(self, n):
        self.stock += n
        return f'Current {self.product} stock: {self.stock}'

    def add_funds(self, n):
        if self.stock == 0:
            return f'Nothing left to vend. Please restock. Here is your ${n}.'
            # Alternatively, we could have:
            # return self.vend() + f' Here is your ${n}.'
        self.balance += n
        return f'Current balance: ${self.balance}'

    def vend(self):
        if self.stock == 0:
            return 'Nothing left to vend. Please restock.'
        difference =self.price - self.balance
        if difference > 0:
            return f'Please add ${difference} more funds.'
        message = f'Here is your {self.product}'
        if difference != 0:
            message += f' and ${-difference} change'
        self.balance = 0
        self.stock -= 1
        return message + '.'
```

Q4. (5 points) Algorithm Analysis

Evaluate the runtime bound in Θ for the function below.

```
def runtime_func(n):
    k = 0
    i = 1
    while i < n * n * n:
        i *= 2
        k += 1
    return k
```

$\Theta(log\, n)$

Q5. (12 points) Binary Search

Given a non-negative integer n, compute and return the square root of n. The decimal digits are truncated, and only the integer part of the result is returned. ***Do not use recursion, but iteration***. The expected runtime complexity is O(log n). You are ***not allowed to use any built-in exponent function or operator***.

```
def sqrt_custom(n):
    '''
    >>> sqrt_custom(4)
    2
    >>> sqrt_custom(8)
    2
    >>> sqrt_custom(16)
    4
    >>> sqrt_custom(24)
    4
    '''
```

```
    if x == 0:  return 0
    if x == 1:  return 1

    low, high = 0,  x

    # binary search
    while low <= high:
        mid = low + (high - low) // 2

        if mid ** 2 > x:  # if mid * mid > x:
            high = mid - 1
        elif mid ** 2 < x:
            low = mid + 1
        else:  # mid ** 2 == x
            return mid
    return high
```

Q6. (10 points) Matrix Rotation

Given an N x N 2D matrix `mat` representing an image, `rotate_matrix` rotates the image by 90 degrees (anti-clockwise). You need to do this in place. Note that you ***should not create an additional array***.

```
# Function to print the matrix
def print_matrix(mat, size):
   for i in range(0, size):
       for j in range(0, size):
           print (mat[i][j], end = ' ')
       print ("")

# You just need to implement this function
def rotate_matrix(mat, size):
    '''
    >>> mat = [[1, 2, 3],
    ...        [4, 5, 6],
    ...        [7, 8, 9]]
    >>> rotate_matrix(mat, 3)
    >>> print_matrix(mat, 3)
    3 6 9
    2 5 8
    1 4 7
    >>> mat = [[1, 2],
    ...        [4, 5]]
    >>> rotate_matrix(mat, 2)
    >>> print_matrix(mat, 2)
    2 5
    1 4
    '''
```

```
    for x in range(0, int(size / 2)):
        # Consider elements in group of 4 in current square
        for y in range(x, size-x-1):
            # store current cell in temp variable
            temp = mat[x][y]

            # move values from right to top
            mat[x][y] = mat[y][size-1-x]

            # move values from bottom to right
            mat[y][size-1-x] = mat[size-1-x][size-1-y]

            # move values from left to bottom
            mat[size-1-x][size-1-y] = mat[size-1-y][x]

            # assign temp to left
            mat[size-1-y][x] = temp
```

Q7. (10 points) Reverse Polish Notation with Unary Operators

Implement the function `eval_rpn_unary` that evaluates Reverse Polish notation, also referred to as Polish postfix notation. Notice that the function ***supports unary operators*** such as -(1+2), i.e., [“1”, “2”, “+”, “-”]. Moreover, remember that there are only two types of unary operators: + and -. You also need to ***handle exceptional cases*** properly.

```python
def eval_rpn_unary(tokens):
    '''
    >>> eval_rpn_unary(["3", "1", "+", "4", "*"])  # (3 + 1) * 4 = 16
    16
    >>> eval_rpn_unary(["2", "1", "+", "3", "*"])  # ((2 + 1) * 3) = 9
    9
    >>> eval_rpn_unary(["4", "13", "5", "/", "+"]) # (4 + (13 / 5)) = 6
    6
    >>> eval_rpn_unary(["1", "2", "+", "-"])       # -(1 + 2) = -3
    -3
    '''
```

```python
    operations = {
        "+": lambda a, b: a + b,
        "-": lambda a, b: a - b,
        "/": lambda a, b: int(a/b),
        "*": lambda a, b: a * b
    }
    stack = []
    for e in tokens:
        if e in operations:
            if len(stack) == 0:
                raise Exception("No Numbers Found")
            elif len(stack) == 1:
                if e in ["+", "-"]:
                    if e == "-": stack.append(-stack.pop())
                    else: stack.append(+stack.pop())
                else:
                    raise Exception("Only Unary +/- Supported")
            else:
                n2 = stack.pop()
                n1 = stack.pop()
                stack.append(operations[e](n1, n2))
        else:
            stack.append(int(e))
    return stack[-1]
```

Q8. (17 points) Queues with a Single Sentinel

Complete the implementation of the `Queue` class using ***a single sentinel***. A sentinel is a dummy node that does not contain meaningful items; the sentinel makes the implementation clean by removing the necessity of checking exceptional cases. The `Queue` starts with a single sentinel and becomes ***circular*** after a few enqueue operations (the last node of the `Queue` points back to the sentinel node). Correct implementation of each method is worth ***5 points*** except `is_empty`.

```python
class Node:
  def __init__(self, item, prev=None, next=None):
    self.item = item
    self.prev = prev
    self.next = next

class Queue:
  '''
  >>> q = Queue()
  >>> q.is_empty()
  True
  >>> q.front()
  'Q is empty!'
  >>> q.enqueue(4)
  >>> q.front()
  4
  >>> q.is_empty()
  False
  >>> q.enqueue(5)
  >>> q.enqueue(6)
  >>> q.enqueue(7)
  >>> q.dequeue()
  4
  >>> q.front()
  5
  >>> print(q)
  5 -> 6 -> 7
  >>> q.dequeue()
  5
  >>> q.dequeue()
  6
  >>> q.dequeue()
  7
  >>> q.dequeue()
  'Q is empty!'
  >>> q.is_empty()
  True
  '''
  def __init__(self):
      self.sentinel = Node(None)
      self.sentinel.next = self.sentinel
      self.sentinel.prev = self.sentinel
```

```python
def __str__(self):
    result = []
    temp = self.sentinel.next
    while temp != self.sentinel:
        result.append(str(temp.item))
        temp = temp.next
    return " -> ".join(result)
```

```python
def is_empty(self):
    return self.sentinel.next is self.sentinel
```

```python
def front(self):
    if self.is_empty():
        return "Q is empty!"
    return self.sentinel.next.item
```

```python
def dequeue(self):
    if self.is_empty():
        return "Q is empty!"
    item = self.sentinel.next.item
    self.sentinel.next = self.sentinel.next.next
    self.sentinel.next.next.prev = self.sentinel
    return item
```

```python
def enqueue(self, x):
    temp = Node(x)
    temp.prev = self.sentinel.prev
    temp.next = self.sentinel

    self.sentinel.prev.next = temp
    self.sentinel.prev = temp
```