DSA - Problem Set 2

Hanna Getachew - 238179

Hertie School, Spring 2025

Github Repo Link: https://github.com/Hanna-Getachew/dsa-hw2.git

Question 1: Flask App Extension – Circle Calculator

The main task in this problem set is to create an extension for the calculator Flask app from Lab 3. In addition to the basic calculator that we created in the lab, this extension should have a page on which a user can calculate the perimeter of a circle by inputting the radius.

For the calculation of the radius, you should create a `Circle` class, with one method to calculate the perimeter and one method to calculate the area of the circle. You should write one unit test for each of these methods. To make this task easier, start by cloning this repository to begin with a working version of the calculator app from the lab.

You will submit your code as a Pull Request to this repository. We will then check whether your code runs correctly and whether your tests do what they are supposed to. Please provide a link to the Pull Request in the document you submit on Moodle (along with your answers to questions 2 and 3). For full marks, you should:

(a) (3 points): Write the `Circle` class (with the perimeter and area methods) in a separate Python module (like `helper.py`, in the repository) called `circle.py` and import it into the main Flask module.

(b) (3 points): Create a template similar to the `calculator.html` template, with an HTML form in which the user can input the radius, and render this template from the main module.

(c) (3 points): Create a `test_circle.py` file in the root directory, which tests the two methods in the `circle.py` module.

Answer

Folder Structure for Calculator App

To keep the app organized and easy to manage and extend.

```
dsahw2/
├── dsahw2.ipynb         # Jupyter Notebook with answers
to the problem set
├── dsahw2.html          # HTML export of the notebook
(optional)
├── dsahw2.pdf           # PDF export of the notebook (for
```

```
submission)
├── test_circle.py       # Unit tests for the Circle class
└── calc_app/
    ├── __init__.py          # Empty file to make calc_app
a package
    ├── app.py               # Main Flask application file
    ├── circle.py            # Circle class with perimeter
and area methods
    ├── helper.py            # Extra functions (e.g.
convert_to_float, perform_calculation)
    ├── static/
    │   └── main.css         # All CSS styling for the
entire app
    └── templates/
        ├── layout.html      # Base HTML layout for all
templates
        ├── home.html        # Home page with links to
both calculators
        ├── circle.html      # Form and result page for
Circle calculator
        └── calculator.html  # Form and result page for
Basic calculator
```

Contents of app.py

```python
from flask import Flask, render_template, request
from .circle import Circle
from .helper import convert_to_float, perform_calculation  #
using a helper.py file

app = Flask(__name__)

@app.route('/')
@app.route('/home')  # now '/' and '/home' show the same page
def home():
   return render_template('home.html')

@app.route('/calculate', methods=['GET', 'POST'])
def calculate():
   if request.method == 'POST':
       value1 = request.form['value1']
       value2 = request.form['value2']
       operation = str(request.form['operation'])

       if operation not in ['add', 'subtract', 'divide',
'multiply']:
           return render_template('calculator.html',
                                  printed_result='Operation must
be one of "add", "subtract", "divide", or "multiply".')

       try:
           value1 = convert_to_float(value=value1)
           value2 = convert_to_float(value=value2)
       except ValueError:
           return render_template('calculator.html',
printed_result="Cannot perform operation with this input")
```

```
        try:
            result = perform_calculation(value1=value1,
value2=value2, operation=operation)
            return render_template('calculator.html',
printed_result=str(result))

        except ZeroDivisionError:
            return render_template('calculator.html',
printed_result="You cannot divide by zero")

    return render_template('calculator.html')

@app.route('/circle', methods=['GET', 'POST'])
def circle():
    result = None
    if request.method == 'POST':
        radius = float(request.form['radius'])
        c = Circle(radius)
        result = {
            'perimeter': round(c.perimeter(), 2),
            'area': round(c.area(), 2)
        }
    return render_template('circle.html', result=result)

if __name__ == '__main__':
    app.run(debug=True)
```

Contents of helper.py

```
# create helper functions for calculations

def perform_calculation(value1: float, value2: float, operation:
str) -> float:
    """
    Perform a mathematical operation on two values.

    Parameters:
        value1 (float): The first value.
        value2 (float): The second value.
        operation (str): The operation to perform. Can be 'add',
'subtract', 'divide', or 'multiply'.

    Returns:
        float: The result of the operation.

    Raises:
        ZeroDivisionError: If attempting to divide by zero.
    """
    if operation == 'add':
        result = value1 + value2
    elif operation == 'subtract':
        result = value1 - value2
    elif operation == 'divide':
        result = value1 / value2
    else:
```

```
        result = value1 * value2

    return result


def convert_to_float(value: str) -> float:
    """
    Convert string to floating point number.

    Parameters:
        value (str): The value to convert.

    Returns:
        float: The converted float value of input value.

    Raises:
        ValueError: If value cannot be converted to a float.
    """

    float_value = float(value)

    return float_value
```

Contents of circle.py

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def perimeter(self):
        return 2 * math.pi * self.radius

    def area(self):
        return math.pi * (self.radius ** 2)
```

Contents of home.html

```
{% extends 'layout.html' %}

{% block content %}
    <h1>Welcome to the Calculator App</h1>
    <p>Choose a calculator below:</p>

    <div style="margin-top: 20px;">
        <ul>
            <li><a href="{{ url_for('calculate') }}">Basic
Calculator</a></li>
            <li><a href="{{ url_for('circle') }}">Circle
Calculator</a></li>
        </ul>
    </div>
{% endblock %}
```

Contents of calculator.html

```
{% extends 'layout.html' %}

{% block content %}
<div class="calculator-container">
   <h1>Basic Calculator</h1>
   <p>Enter two numbers and choose an operation:</p>

   <form method="post">
       <input type="number" name="value1" step="any"
placeholder="First number" required>

       <select name="operation" required>
           <option value="">Choose operation</option>
           <option value="add">Add (+)</option>
           <option value="subtract">Subtract (−)</option>
           <option value="multiply">Multiply (×)</option>
           <option value="divide">Divide (÷)</option>
       </select>

       <input type="number" name="value2" step="any"
placeholder="Second number" required>
       <button type="submit">Calculate</button>
   </form>

   {% if printed_result %}
       <div class="result-box">
           <h2>Result: {{ printed_result }}</h2>
       </div>
   {% endif %}
</div>
{% endblock %}
```

Contents of circle.html

```
{% extends 'layout.html' %}
{% block content %}
   <div class="circle-container">
       <h1>Circle Calculator</h1>
       <p>Enter the radius below to calculate the area and
perimeter of a circle.</p>

       <form method="post">
           <input type="number" name="radius" step="any"
placeholder="Enter radius" required />
           <button type="submit">Calculate</button>
       </form>

       {% if result %}
           <div class="results">
               <h2>Results</h2>
               <p><strong>Perimeter:</strong> {{ result.perimeter
}}</p>
               <p><strong>Area:</strong> {{ result.area }}</p>
           </div>
       {% endif %}
```

```
    </div>
{% endblock %}
```

Contents of layout.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Calculator App</title>
    <link rel="stylesheet" type="text/css" href="{{
url_for('static', filename='main.css') }}">
    <style>
        /* simple layout styling */
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f9f9f9;
        }

        .container {
            max-width: 800px;
            margin: 0 auto;
        }

        ul.menu {
            list-style: none;
            padding: 0;
            margin-bottom: 30px;
            display: flex;
            gap: 20px;
        }

        ul.menu li {
            display: inline;
        }

        ul.menu a {
            text-decoration: none;
            color: #007bff;
            font-weight: bold;
        }

        ul.menu a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <div class="container">
        <ul class="menu">
            <li><a href="{{ url_for('home') }}">Home</a></li>
            <li><a href="{{ url_for('calculate') }}">Basic
Calculator</a></li>
            <li><a href="{{ url_for('circle') }}">Circle
```

```
Calculator</a></li>
       </ul>

       {% block content %}
       {% endblock %}
   </div>
</body>
</html>
```

Contents of main.css

```
/* ---------- Global Container ---------- */
.container {
   max-width: 800px;
   margin: 40px auto;
   padding: 20px;
   background-color: #ffffff;
   border-radius: 12px;
   box-shadow: 0 0 12px rgba(0, 0, 0, 0.05);
   font-family: 'Segoe UI', sans-serif;
}

/* ---------- Navigation Menu ---------- */
ul.menu {
   list-style: none;
   padding: 0;
   display: flex;
   gap: 20px;
   margin-bottom: 30px;
   justify-content: center;
}

ul.menu li {
   display: inline;
}

ul.menu a {
   text-decoration: none;
   color: #007bff;
   font-weight: 600;
}

ul.menu a:hover {
   text-decoration: underline;
}

/* ---------- Circle Calculator ---------- */
.circle-container {
   max-width: 500px;
   margin: 0 auto;
   text-align: center;
}

.circle-container h1 {
   color: #343a40;
}
```

```css
.circle-container p {
    font-size: 16px;
}

.results {
    margin-top: 20px;
    padding: 15px;
    background-color: #f1f1f1;
    border-radius: 8px;
}

/* ---------- Basic Calculator ---------- */
.calculator-container {
    max-width: 500px;
    margin: 0 auto;
    text-align: center;
}

.calculator-container h1 {
    color: #343a40;
}

input[type="number"], select, button {
    padding: 10px;
    margin: 10px;
    width: 80%;
    max-width: 300px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 6px;
}

button {
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

.result-box {
    margin-top: 20px;
    background: #f1f1f1;
    padding: 15px;
    border-radius: 8px;
}

/* ---------- Home Page ---------- */
h1, h2 {
    text-align: center;
    color: #222;
}
```

```
ul {
   text-align: center;
   list-style: none;
   padding: 0;
}

ul li {
   margin: 10px 0;
}

ul li a {
   color: #007bff;
   text-decoration: none;
   font-weight: 500;
}

ul li a:hover {
   text-decoration: underline;
}


/* ---------- code for main.css sourced from  copilot.ai---------
- */
```

Contents of test_circle.py

```
import unittest
from calc_app.circle import Circle
import math

class TestCircle(unittest.TestCase):

   def test_perimeter(self):
       c = Circle(5)
       expected = 2 * math.pi * 5
       self.assertAlmostEqual(c.perimeter(), expected, places=5)

   def test_area(self):
       c = Circle(5)
       expected = math.pi * (5 ** 2)
       self.assertAlmostEqual(c.area(), expected, places=5)

if __name__ == '__main__':
   unittest.main()
```

In [ ]:
```
from IPython.display import Image
Image(filename='s1.png')
```

In [ ]:
```
from IPython.display import Image
Image(filename='s2.png')
```

In [ ]:
```
from IPython.display import Image
Image(filename='s3.png')
```

In [ ]:
```
from IPython.display import Image
Image(filename='s4.png')
```

Question 2: Object-Oriented Design & Gale-Shapley Algorithm

Consider the following Python code snippet:

```
from abc import ABC, abstractmethod

class Agent(ABC):
    def __init__(self, name: str, preferences: list[str]):
        self.name = name
        self.preferences = preferences
        self.match = None

    @abstractmethod
    def propose(self):
        pass

    @abstractmethod
    def respond(self, proposer: 'Agent') -> bool:
        # Remark: 'Agent' is the dtype. Since we are still in the
class Agent and
        # it is not fully defined yet, we create a placeholder
using ''.
        pass

class Proposer(Agent):
    def propose(self):
        """
        Propose to the next preferred proposee.
        """
        if not self.preferences:
            raise ValueError(f"{self.name} has no more
preferences to propose to.")
        return self.preferences.pop(0)

    def respond(self, proposer: 'Agent') -> bool:
        """
        Proposers do not respond to proposals; they only propose.
        """
        raise NotImplementedError("Proposers cannot respond to
proposals.")

class Proposee(Agent):
    def propose(self):
        """
        Proposees do not propose; they only respond.
        """
        raise NotImplementedError("Proposees cannot propose.")

    def respond(self, proposer: 'Agent') -> bool:
        """
        Respond to a proposal based on preferences.
        """
```

```
        if self.match is None:
            return True
        else:
            # If proposee is matched, compare preferences
            current_match_index =
self.preferences.index(self.match.name)
            proposer_index =
self.preferences.index(proposer.name)
            if proposer_index < current_match_index:
                return True
            else:
                return False
```

(a) (1 point):

In a maximum of 3 lines, explain the relationship between the classes.
Hint: The code allows for constructing objects that would represent the individuals in groups in a matching problem.

(b) (2 points):

Write a class that implements the Gale-Shapely algorithm. It should receive a list of proposers and a list of proposees, constructed using the classes given above. Test the correctness of the implementation against the example we discussed in the class (slide #13). Run it for both cases where a group is a proposer and a proposee. Hint: You could consider completing this code:

```
# Gale-Shapley Algorithm
class GaleShapley:
    def __init__(self, proposers: list[Proposer], proposees:
list[Proposee]):
        self.proposers = proposers
        self.proposees = proposees
        self.matches = {}  # Stores the final matches

    def match(self) -> dict:
        """
        Run the Gale-Shapley algorithm to find a stable matching.
        """
        free_proposers = list(self.proposers)  # Initialize all
proposers as free

        while free_proposers:
            proposer = free_proposers.pop(0)  # Pick a free
proposer
            proposee_name = proposer.propose()  # Propose to the
next preferred proposee
            proposee = next(p for p in self.proposees if p.name
== proposee_name)

            ### YOUR CODE ###
```

Answer

(a) The relationship between the classes

- Agent is a general template (abstract class) for any person in the matching problem.
- Proposer is someone who proposes to others, based on their list of preferences.
- Proposee is someone who receives proposals and decides whether to accept or reject them.

This setup allows us to model how two groups can be matched based on preferences.

(b) A class that implementing the Gale-Shapely algorithm

In [ ]:
```
from abc import ABC, abstractmethod

# base class
class Agent(ABC):
    def __init__(self, name: str, preferences: list[str]):
        self.name = name
        self.preferences = preferences
        self.match = None

    @abstractmethod
    def propose(self):
        pass

    @abstractmethod
    def respond(self, proposer: 'Agent') -> bool:
        pass

# proposer class
class Proposer(Agent):
    def propose(self):
        if not self.preferences:
            raise ValueError(f"{self.name} has no more preferences to pro
        return self.preferences.pop(0)

    def respond(self, proposer: 'Agent') -> bool:
        raise NotImplementedError("Proposers do not respond to proposals.

# proposee class
class Proposee(Agent):
    def propose(self):
        raise NotImplementedError("Proposees cannot propose.")

    def respond(self, proposer: 'Agent') -> bool:
        if self.match is None:
            return True
        else:
            current_match_index = self.preferences.index(self.match.name)
            proposer_index = self.preferences.index(proposer.name)
            return proposer_index < current_match_index

# gale-shapley class
class GaleShapley:
    def __init__(self, proposers: list, proposees: list):
        self.proposers = proposers
        self.proposees = proposees
        self.matches = {}  # Final pairs will be stored here
```

```
    def match(self) -> dict:
        free_proposers = list(self.proposers)  # Start with everyone unma

        while free_proposers:
            proposer = free_proposers.pop(0)
            proposee_name = proposer.propose()
            proposee = next(p for p in self.proposees if p.name == propos

            if proposee.respond(proposer):
                if proposee.match:
                    old_match = proposee.match
                    old_match.match = None
                    free_proposers.append(old_match)

                proposer.match = proposee
                proposee.match = proposer
            else:
                free_proposers.append(proposer)

        for proposer in self.proposers:
            if proposer.match:
                self.matches[proposer.name] = proposer.match.name

        return self.matches

# test matching scenario: students and companies
elena = Proposer("Elena", ["World Bank", "Microsoft", "SoundCloud"])
fanus = Proposer("Fanus", ["Microsoft", "World Bank", "SoundCloud"])
laia = Proposer("Laia", ["SoundCloud", "Microsoft", "World Bank"])

world_bank = Proposee("World Bank", ["Fanus", "Elena", "Laia"])
microsoft = Proposee("Microsoft", ["Laia", "Elena", "Fanus"])
soundcloud = Proposee("SoundCloud", ["Elena", "Fanus", "Laia"])

gs = GaleShapley([elena, fanus, laia], [world_bank, microsoft, soundcloud
result = gs.match()
print(result)

# result: {'Elena': 'World Bank', 'Fanus': 'Microsoft', 'Laia': 'SoundClo
```

```
{'Elena': 'World Bank', 'Fanus': 'Microsoft', 'Laia': 'SoundCloud'}
```

Question 3: Algorithm Analysis – Subarray Sums

Consider the following algorithm:

```
max_sum = None
for i in range(len(array)):
    sum_subarray = 0
    for j in range(i, len(array)):
        sum_subarray += array[j]
        if max_sum is None or max_sum < sum_subarray:
            max_sum = sum_subarray
print(max_sum)
```

This algorithm takes an array (assumed to be non-empty) named array and finds the subarray (contiguous elements of that array of any size) with the largest sum and outputs that sum.

(a) (1 points) Is the algorithm defined above "efficient", that is, is its run-time polynomial in the size of the array? If not, explain why not. If it is, give its runtime in Big-O notation with the smallest d such that O(nd) is true.

(b) (2 points) Now consider the following algorithm:

```
max_sum = float('-inf')
current_sum = 0
for num in array:
    current_sum = max(num, current_sum + num)
    max_sum = max(max_sum, current_sum)
print(max_sum)
```

Answer

(a) Is the first algorithm efficient?

```
max_sum = None
for i in range(len(array)):
    sum_subarray = 0
    for j in range(i, len(array)):
        sum_subarray += array[j]
        if max_sum is None or max_sum < sum_subarray:
            max_sum = sum_subarray
print(max_sum)
```

No, not really.

This algorithm uses two loops:

- The outer loop picks a starting point in the array.
- The inner loop adds up every possible subarray that starts at that point.

So, for every starting point, it checks all the possible endings, meaning it essentially checks **every single subarray**.

If the array has `n` numbers, it will do around `n × n` operations.

It runs in **O(n²)** time. That means the time it takes grows quickly when the array gets large. It's okay for small inputs, but slow for big ones (*in the lab henry mentioned we care about the worst case senario*).

(b) What does the second algorithm do, and how is it better?

```
max_sum = float('-inf')
current_sum = 0
for num in array:
    current_sum = max(num, current_sum + num)
    max_sum = max(max_sum, current_sum)
print(max_sum)
```

The second algorithm is much faster. Instead of checking every subarray, it just goes through the array **once**.

At each number, it decides:

- Should I **keep going** and add this number to the current total?
- Or should I **start over** with just this number?

It always keeps track of:

- **current_sum** → the best sum ending at the current position
- **max_sum** → the best sum found so far

Since it only loops once through the array, it is much faster.

It runs in **O(n)** time: which is more efficient and better suited for large arrays.