

QUESTION ONE

a

In the provided code, there are three classes: **Calculations1**, **Calculations2**, and **Derived**. **Calculations1** has a method **Sum** that prints the values of **x1** and **x2** and returns their sum. **Calculations2** has methods for Subtraction, Multiplication, and Division. The **Derived** class inherits from both **Calculations1** and **Calculations2**.

Input:

x1 = 5

x2 = 8

Output:

5 8

13

b

Detecting Regressions: Regression testing is designed to identify unintended changes or "regressions" in the software. As existing code is modified during updates and enhancements, there is a risk of introducing bugs or breaking existing functionality. Regression tests help catch these regressions early in the development cycle.

Continuous Integration and Deployment (CI/CD): With continuous updates and enhancements, development teams often employ CI/CD practices to automate the deployment process. Regression tests are a fundamental part of the CI/CD pipeline, providing confidence that new changes do not break existing functionality, this enables a smooth and reliable deployment process.

Maintaining Data Integrity: In a data processing module, maintaining the integrity of data is paramount. Regression testing ensures that data processing updates do not introduce errors that compromise the accuracy or consistency of processed data.

Verifying Existing Functionality: A data processing module is likely to have various complex algorithms and data manipulation processes. Regression tests ensure that the core functionality of the module remains intact after each update. This is vital for maintaining the reliability of the module and ensuring that it continues to produce accurate results.

c

```
import matplotlib.pyplot as plt
```

```
class Calculations1:
```

```
    def __init__(self, x, y):
```

```
        self.x1 = x
```

```
        self.x2 = y
```

```
    def Sum(self):
```

```
        print (self.x1, self.x2)
```

```
        return self.x1 + self.x2
```

```
class Calculations2:
```

```
    def Subtraction(self):
```

```
        return self.x1 - self.x2
```

```
    def Multiplication(self):
```

```
        return self.x1 * self.x2
```

```
    def Division(self):
```

```
        return self.x1 / self.x2
```

```
class Derived(Calculations1, Calculations2):
```

```
    def __init__(self, x, y):
```

```
        super().__init__(x, y)
```

```
# Input
y1 = int(input("x1="))
y2 = int(input("x2="))

# Create an instance of the Derived class
derived1 = Derived(y1, y2)

# Perform calculations
result_sum = derived1.Sum()

# Output
print("Sum:", result_sum)

# Visualization
labels = ['x1', 'x2']
values = [derived1.x1, derived1.x2]

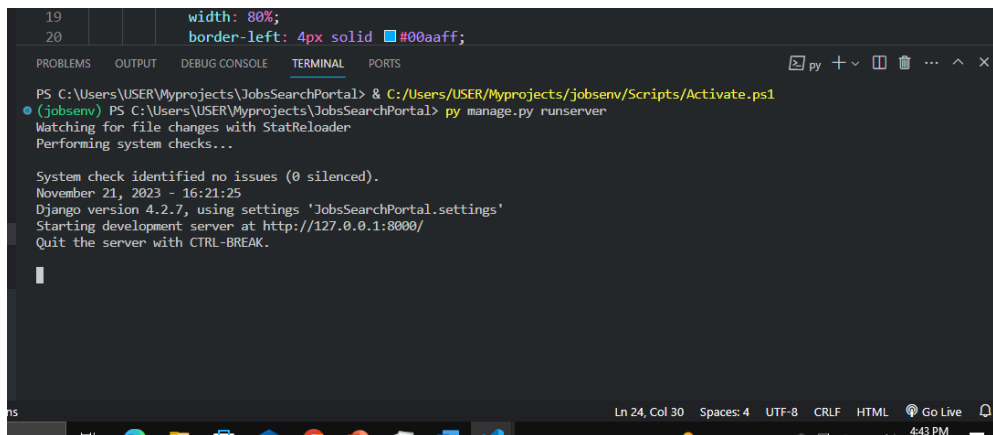
# Plotting input data
plt.bar(labels, values, color=['blue', 'green'])
plt.title('Input Data')
plt.show()

# Plotting output data
plt.bar(['Sum'], [result_sum], color='orange')
plt.title('Output Data')
plt.show()
```

QUESTION TWO

Part a

1. OS Command injection might occur If you use background tasks or scheduled jobs that involve shell commands, and to safeguard this, make sure any inputs used in constructing these commands are validated and sanitized to prevent injection vulnerabilities. Below is our snippet.

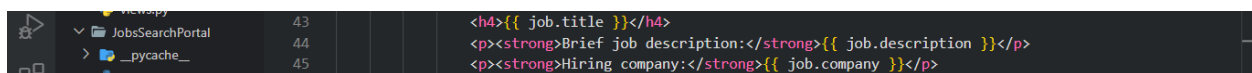


```
19 width: 80%;
20 border-left: 4px solid #00aaff;

PS C:\Users\USER\Myprojects\JobsSearchPortal> & C:\Users\USER\Myprojects\jobsew\Scripts\Activate.ps1
(jobsew) PS C:\Users\USER\Myprojects\JobsSearchPortal> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

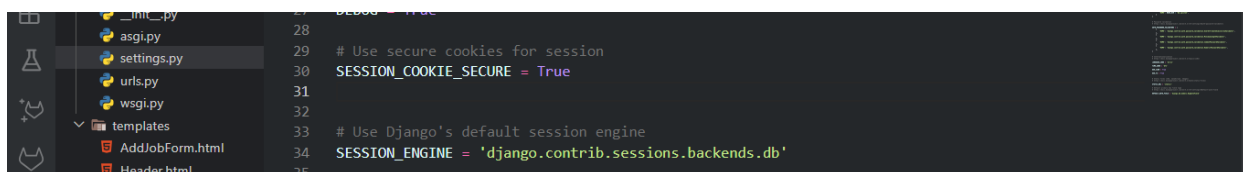
System check identified no issues (0 silenced).
November 21, 2023 - 16:21:25
Django version 4.2.7, using settings 'JobsSearchPortal.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

2.Cross-site scripting might occur in a template when the html file does not seem to escape user inputs when rendering HTML. Below is the snippet.



```
<h4>{{ job.title }}</h4>
<p><strong>Brief job description:</strong>{{ job.description }}</p>
<p><strong>Hiring company:</strong>{{ job.company }}</p>
```

3.Session Hijacking might occur when Django's session settings are not configured securely. Key settings to consider include SESSION_COOKIE_SECURE, SESSION_COOKIE_HTTPONLY, and SESSION_ENGINE. Below is the snippet.



```
28 SESSION_COOKIE_SECURE = True
29 # Use secure cookies for session
30 SESSION_COOKIE_SECURE = True
31
32
33 # Use Django's default session engine
34 SESSION_ENGINE = 'django.contrib.sessions.backends.db'
35
```

4.Cross-Site Request Forgery might occur when the view that has a form does not include the CSRF token explicitly. Below is the snippet.

```
53 <div>
54
55 <form action="{% url 'addjob' %}" method="post">
56     {% csrf_token %}
57     <h3>Add job</h3>
```

5.SQL Injection might occur when Django ORM, which helps prevent SQL injection is not used correctly for example when queries are not parameterized. Below is the snippet.

```
28
29 else:
30     # create a new job object
31     job = Job.objects.create(title=title, description=description, category=category,
32                             company=company, email=email, contact=contact,
33                             location=location, qualifications=qualifications, skills=skills)
34
35     # save the new Job object in the database
36     job.save()
```

Part b

1.Against OS Command injection

Use the subprocess module to execute OS commands.

Always validate and sanitize user input before using it in commands. In this example, the quote function is used to properly escape and quote the user input.

Code

```
import subprocess

from shlex import quote

# Validate and sanitize user input before using it in OS commands

def execute_command(user_input):

    sanitized_input = quote(user_input)

    subprocess.run(['your_command', sanitized_input])
```

2.Against Cross-site scripting

Use the safe filter in Django templates to mark content as safe from HTML escaping.

Ensure that user-generated content is properly escaped to prevent script injection.

Code

```
<!-- Use the "safe" filter to mark content as safe from HTML escaping -->
<p><strong>Brief job description:</strong>{{ job.description|safe }}</p>
```

3.Against Session Hijacking

Configure Django session settings to use secure cookies (SESSION_COOKIE_SECURE), enforce the HttpOnly flag (SESSION_COOKIE_HTTPONLY), and use a secure session engine (SESSION_ENGINE).

Code

```
# settings.py
# Use secure cookies for session
SESSION_COOKIE_SECURE = True
# Use HttpOnly flag for cookies
SESSION_COOKIE_HTTPONLY = True
# Use Django's default session engine
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
```

4.Against Cross-Site Request Forgery

Include the {% csrf_token %} template tag within your HTML forms to include the CSRF token.

This helps protect against Cross-Site Request Forgery attacks.

Code

```
<form method="post" action="{% url 'addjob %}'>
    {% csrf_token %}
    <input type="submit" value="Submit">
</form>
```

5.Against SQL Injection

Use the Django ORM to interact with the database, which automatically handles parameterized queries.

Avoid manually constructing queries using string concatenation with user inputs.

Code

```
from django.db import models

class Job(models.Model):
    # ... your existing code ...

    @classmethod
    def create(cls, **kwargs):
        # Use the Django ORM to create objects with parameterized queries
        job = cls(**kwargs)
        job.save()
        return job
```