Part I
Software testing basis

1. What is software testing, and why is it a critical part of the software development life cycle.

   Software testing involves assessing a software application to uncover any errors, deficiencies, or unmet requirements compared to the actual specifications. This process ensures that the software functions as intended and adheres to quality standards prior to its release to users.

   This are the important of SDLC,

   Quality Assurance: Guarantees that the software maintains high standards and fulfills user needs.
   Bug Detection: Spots and resolves bugs prior to the software launch, minimizing the chances of failures.
   Cost-Effective: Detecting issues early can help save money on fixing bugs later in the development process.
   Security: Aids in uncovering vulnerabilities and ensures the software remains secure.
   User Satisfaction: Confirms that the software is dependable and functions effectively, resulting in greater user satisfaction.

   Software testing plays a vital role in the software development life cycle. It ensures that the software remains functional and reliable, delivering a positive experience for users. By catching and resolving problems early in the development phase, software testing helps avoid expensive and lengthy fixes down the line, which ultimately supports the overall success of the software product.

a. Explain the importance of regression testing. How does automation help in regression testing?

Regression Testing checks that recent changes, like bug fixes or new features, haven't broken anything that was already working. It's critical for maintaining the reliability of the software as it evolves.

Regression testing is essential because it ensures that new updates or changes in the software do not disrupt existing functionality. This helps maintain consistent performance over time, ensuring that the software continues to meet user expectations and deliver reliable results. Automation plays a significant role in enhancing regression testing. It saves time by running tests much faster than manual methods, allowing developers to focus on other tasks. Automated tests are also highly consistent, reducing the risk of human errors and ensuring accurate results. While the initial setup of automation can be costly, it becomes cost-effective in the long run, especially for projects requiring frequent and repetitive testing.

2. Compare manual testing and automated testing in terms of cost, time, and accuracy.

	Manual	Automated
Cost	Cheaper to start but gets expensive over time because it needs people to do the testing	Expensive to set up initially but saves money in the long run for repetitive tests.
Time	Slower, since humans need time to perform the tests.	Much faster, especially for repetitive tasks.
Accuracy	May have errors because humans can make mistakes.	Very accurate because tools run the tests consistently.

Best For	Good for one-time tests, exploring new features or checking usability.	Ideal for repetitive tests, like regression testing.

3. Write a function multiply_numbers that multiplies two numbers. Include doctest to validate the function.

```python
def multiply_numbers(a, b):
    """
    Multiplies two numbers and returns the result.

    Args:
        a (float): The first number.
        b (float): The second number.

    Returns:
        float: The product of the two numbers.

    Examples:
        >>> multiply_numbers(2, 3)
        6
        >>> multiply_numbers(-4, 5)
        -20
        >>> multiply_numbers(0, 10)
        0
        >>> multiply_numbers(1.5, 2)
        3.0
    """
    return a * b

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Part II

Analyze the differences between Java and Python in terms of syntax, behavior, and functionality.
Convert the following Java code into Python and explain the differences:

```python
class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id
        self.grades = []

    def add_grade(self, grade):
        self.grades.append(grade)

    def get_average_grade(self):
        if not self.grades:
            return 0  # Avoid division by zero
        total = sum(self.grades)
        return total / len(self.grades)


# Example usage
if __name__ == "__main__":
    student = Student(name="John", student_id="1111")
    student.add_grade(90)
    student.add_grade(80)
    print(f"Average Grade: {student.get_average_grade()}")
```

= RESTART: C:/Users/sahan/Desktop/Projects/python/advance se/test1.py

Average Grade: 85.0

Java and Python differ greatly in syntax. Java requires type declarations (e.g., `String`, `int`), semicolons to end statements, and curly braces for code blocks. Python is dynamically typed, has no semicolons, and uses indentation for code blocks, making it simpler and more concise.

Java has a strict type system, requiring you to define the types of variables and collections, like using `ArrayList` for dynamic lists, which must be explicitly imported. In contrast, Python has built-in lists that can store elements of any type, making it more concise and flexible for handling collections.

Java requires explicit iteration to perform tasks like summing or processing a list, which can make the code longer and more complex. Python, on the other

hand, offers built-in functions like `sum()` and `len()` for such operations, simplifying the code and reducing the effort needed.