ASSIGNMENT 6 ON INTRODUCTION TO PYTHON
MODULE: SOFTWARE ENGINEERING

Question 1 : Python basics
Python is a high-level, interpreted, and general-purpose programming language. Python's design philosophy emphasizes code readability and simplicity, which makes it an excellent choice for both beginners and experienced developers.
Key python features include;

Readability: Python's syntax is clear and concise, making it easy to read and write code.

- **Interpreted Language**: Python code is executed line-by-line, which makes debugging easier.
- **Dynamically Typed**: Variable types are determined at runtime, allowing for more flexibility in coding.
- **High-Level Language**: Python abstracts complex details of the computer from the programmer, focusing on ease of development.
- **Extensive Standard Library**: Python comes with a vast standard library that supports many common programming tasks, such as file I/O, system calls, and internet protocols.
- **Support for Multiple Paradigms**: Python supports procedural, object-oriented, and functional programming styles.

Python are applicable in a number of use cases including the following;

Web development; frameworks like Django and Flask make web development with python very easy.

Data Science and machine learning; python libraries such as Pandas and Numpy are very effective in the field of machine learning with python.

Game development; pygame is used in creating 3D games using python as primary programming language.

Question 2: Python Installation

I updated my package list

sudo apt update

Install Python

sudo apt install python3

Verify Python installation

python3 --version

Install pip

```
sudo apt install python3-pip

# Verify pip installation
pip3 --version

# Install venv module
sudo apt install python3-venv

# Create project directory and navigate into it
mkdir myproject1
cd myproject1

# Create a virtual environment
python3 -m venv venv

# Activate the virtual environment
source venv/bin/activate

# Install packages within the virtual environment
pip install requests

# Deactivate the virtual environment
deactivate
```

Question 3: Python syntax and semantics

```
#to print Hello World in python I used the following syntax
print(“Hello , World!”)
```

In this line of code; print is a function used in python to show text in the console window

“” are used to define string in python

() shows the body of the function printer

Question4: Data Types and Variables in python

I) Integer- used to represent whole numbers in python

age = 56

ii) String – used to represent a group of characters in python

first_name = “John”

iii) Floating point – to represent numbers with decimal points

salary = 58900.30

iv) Boolean – used in python to represent values with two values(True,False)

has_reached_menopause = False

Below is a python script to demonstrate usage of the above explained data Types

```
# Integer
age = 56
print("Age:", age, type(age))

# Float
salary = 58900.30
print("Salary: ", salary, float(salary))

# String
first_name = "John"
print("Name:", name, type(name))

# Boolean
has_reached_menopause = True
print("Has reached Menopause? :", has_reached_menopause, type(has_reached_menopause)

# Using the variables
print(f"{name} is {age} years old .")
print(f"{name} has reached menopause: {has_reached_menopause}")
```

Question 5: Python Control Structures

Conditional statements allow the execution of certain code blocks based on whether a condition is `True` or `False`. The most common conditional statements are `if`, `elif`, and `else`.

Loops are used to repeat a block of code multiple times. Python supports two types of loops: `for` and `while`.

Examples of their implementations are shown below:

```
# Define a list of fruits

fruits = ["apple", "banana", "cherry"]

# Use a for loop to iterate over the list

for fruit in fruits:

    print(fruit)
```

Example of else if statements

```
# Define a variable

age = 45

# Use if-else to check the condition

if age < 18:

    print("You are a minor.")

elif age >= 18 and age < 50:

    print("You are an adult and can still give birth.")

elif age >= 50 and age < 65:

    print("You are an adult and have likely reached menopause.")

else:

    print("You are a senior citizen.")
```

Question 6: Functions in python

Functions in Python are reusable blocks of code designed to perform a specific task. They allow for modular, organized, and efficient programming by enabling code reuse and reducing redundancy.

The following are benefits of functions in python programming

Code Reuse: Write code once and use it multiple times without rewriting it.

Readability: Improve the readability of the code by abstracting complex operations.

Maintainability: Easier to maintain and update code since changes need to be made in one place.

Testing: Simplify testing by isolating individual functions.

Example of a function , how to define and call functions

```
# Define the function to calculate the sum of two numbers
def add(x,y):
    return x + y

# Example of how to call the function
sum= add(10, 7)
print("The sum is:", sum)  # Output: The sum is: 17
```

Question 7: Lists and Dictionaries in python

Lists and **dictionaries** are two fundamental data structures in Python that are used to store collections of data. They have different characteristics and use cases.

Example of usage

```
    # Working with Lists
numbers = [1, 2, 3, 4, 5]
print("List:", numbers)
print("First element:", numbers[0])
print("Last element:", numbers[-1])
numbers.append(6)
print("List after appending 6:", numbers)
numbers.remove(3)
print("List after removing 3:", numbers)
numbers[2] = 10
print("List after modifying the third element:", numbers)

print("\n")
```

```
# Working with Dictionaries
person = {
    "name": "John",
    "age": 25,
    "city": "Nairobi"
}
print("Dictionary:", person)
print("Name:", person["name"])
print("Age:", person["age"])
person["email"] = "john@gmail.com"
print("Dictionary after adding email:", person)
del person["city"]
print("Dictionary after removing city:", person)
person["age"] = 26
print("Dictionary after modifying age:", person)
```

Question 8: Exception handling

Exception handling in Python allows you to manage and respond to errors (exceptions) that occur during the execution of a program. This ensures that the program can continue running or fail gracefully rather than crashing unexpectedly.

Example in usage

```
        def divide_numbers(a, b):
    try:
        # Code that might raise an exception
        result = a / b
    except ZeroDivisionError as e:
        # Code that runs if a ZeroDivisionError occurs
        print("Error: Cannot divide by zero.")
        print("Exception message:", e)
```

```
        result = None
    except TypeError as e:
        # Code that runs if a TypeError occurs
        print("Error: Both arguments must be numbers.")
        print("Exception message:", e)
        result = None
    else:
        # Code that runs if no exception occurs
        print("Division successful. The result is:", result)
    finally:
        # Code that runs no matter what
        print("Execution of the try-except block is complete.")
    return result

# Test cases
print(divide_numbers(10, 2))  # print the result and success message
print(divide_numbers(10, 0))  #  handle division by zero error
print(divide_numbers(10, 'a'))  # handle type error
```

Question 9: python modules and packages

A module is a single file containing Python code that can define functions, classes, and variables. It can also include runnable code. Modules allow you to break your code into manageable sections.

A package is a collection of modules organized in directories. It typically contains a special file `__init__.py` that indicates the directory is a package. Packages allow you to organize modules hierarchically.

Example using math packages

```
        import math
# Using a function from the math module
result = math.sqrt(16)
print(f"The square root of 16 is: {result}")
```

Question 10: Files input/output in python

To read from a file in Python, follow these steps:

1. **Open the File**: Use the `open()` function with the file path and mode (`'r'` for reading) to open the file.
2. **Read the Content**: Use methods like `read()`, `readline()`, or `readlines()` to read the content of the file.
3. **Close the File**: Always close the file using the `close()` method to free up system resources.

Example: Reading from a file

def read_file(file_path):

    try:

        # Open the file in read mode

        with open(f /home/reel/Documents/test.txt, 'r') as file:

            # Read the entire content of the file

            content = file.read()

            # Print the content

            print("File Content:")

            print(content)

    except FileNotFoundError:

        print(f"Error: The file '{/home/reel/Documents/test.txt}' was not found.")

    except Exception as e:

        print(f"Error: {e}")

Usage example

read_file('test.txt')

Reading from and writing to files is a fundamental operation in Python. Python provides built-in functions and methods to handle file operations efficiently.

Reading from a File

To read from a file in Python, follow these steps:

1. **Open the File**: Use the `open()` function with the file path and mode (`'r'` for reading) to open the file.
2. **Read the Content**: Use methods like `read()`, `readline()`, or `readlines()` to read the content of the file.

3. **Close the File**: Always close the file using the `close()` method to free up system resources.

Here's an example script that reads the content of a file and prints it to the console:

```
python
Copy code
# Example: Reading from a file
def read_file(/home/reel/Documents/test.txt):
    try:
        # Open the file in read mode
        with open(/home/reel/Documents/test.txt, 'r') as file:
            # Read the entire content of the file
            content = file.read()
            # Print the content
            print("File Content:")
            print(content)
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except Exception as e:
        print(f"Error: {e}")

# Usage example
read_file('test.txt')
```

To write to a file in Python, follow these steps:

1. **Open the File**: Use the `open()` function with the file path and mode (`'w'` for writing) to open the file. If the file does not exist, it will be created.
2. **Write Content**: Use methods like `write()` to write data to the file.
3. **Close the File**: Always close the file using the `close()` method after writing to ensure all data is flushed to disk.

Example: Writing to a file

def write_to_file(/home/reel/Documents/test.txt, 7):

try:

Open the file in write mode

with open(/home/reel/Documents/test.txt, 'w') as file:

Write each line from the list to the file

for line in lines:

file.write(line + "\n")

print(f"Successfully wrote {len(lines)} lines to '{file_path}'.")

except Exception as e:

print(f"Error: {e}")

```
# Usage example
lines_to_write = ["First line", "Second line", "Third line"]
write_to_file('output.txt', lines_to_write)
```