SE-Assignment 6: Introduction to Python Instructions.

1. Python Basics:
   What is Python, and what are some of its key features that make it popular among developers? Provide examples of use cases where Python is particularly effective.
   ANSWER
   Python is a high-level, interpreted programming language known for its simplicity, readability, and flexibility. It was created by Guido van Rossum and first released in 1991. Its key features include:
   - Readability - Python's syntax is simple and easy to learn, making it ideal for beginners and lowering the cost of program maintenance.
   - Simplicity - Python values simplicity and minimalism, allowing developers to focus on problem solution rather than syntax complexity.
   - Versatility - Python offers numerous programming paradigms (procedural, object-oriented, and functional), making it suitable to a variety of development requirements.
   - Extensive standard library - Python includes a huge standard library that contains modules and methods for many common tasks, encouraging speedy development.
   - Interpreted language and dynamically typed - Python is interpreted and dynamically typed, which allows for faster prototyping and better debugging.
   - Large Community and Ecosystem - Python has a thriving community and a vast ecosystem of third-party libraries and frameworks that enable a variety of applications.

   Example of use cases;
   - Web Development - Python frameworks such as Django and Flask are well-known for creating scalable web applications with clean code.
   - Data Science and Machine Learning - Python libraries (such as Pandas, NumPy, and SciPy) are commonly used for data manipulation, analysis, and machine learning.
   - Scientific computing - Python's libraries (such as Matplotlib and SciPy) enable scientific computation, simulations, and data visualization.
   - Artificial intelligence and robotics - Artificial intelligence and robotics rely heavily on Python libraries such as TensorFlow, PyTorch, and OpenCV.

2. Installing Python:
   Describe the steps to install Python on your operating system (Windows, macOS, or Linux). Include how to verify the installation and set up a virtual environment.
   ANSWER
   Visit python.org and download the latest python installer for windows
   Run the installer and ensure you check the “Add Python to PATH” box during installation.
   Follow the on-screen prompts to complete installation.
   To verify whether Python has been successfully installed, open the command prompt and type “python - -version” then press Enter and the installed python version will be displayed.
   creating a virtual environment:

Open git bash and navigate to your project directory(**cd desktop then mkdir your_folder_name then cd your_folder**)
Run the command "**python -m pip install virtualenv**" to install a virtual environment.
Run "**python -m virtualenv your_project_name**" to create a virtual environment.
Activate the virtual environment "**source your_project_name\scripts\activate**"
You can proceed to install packages "**python -m pip install package_name"**

3. Python Syntax and Semantics:
   Write a simple Python program that prints "Hello, World!" to the console. Explain the basic syntax elements used in the program.
   ANSWER

   print("Hello,World!")
   - Function call (print("Hello,World!")) - The built-in Python function print() sends text or other data to the console (or standard output). Inside the parenthesis (), you supply the data you want to print, in this case the string "Hello, World!"
   - String literal ("Hello, World!") - "Hello, World!" is a string literal, which is a collection of characters surrounded in double quotations (""). Strings in Python can also be contained in single quotes ('), but double quotes are more commonly used in Python codebases for uniformity.

   When you run, the output will be **Hello,World!**

4. Data Types and Variables:
   List and describe the basic data types in Python. Write a short script that demonstrates how to create and use variables of different data types.
   ANSWER
   - Integer ('int') - represents whole numbers e.g age = 20
   - Boolean ('bool') - Represents truth values 'True' or 'False' e.g is_old = True
   - String ('str') - represents characters enclosed in quotes e.g name = "Alice"
   - Floating-point ('float') - represents decimal numbers e.g pi = 3.14159
   - NonType ('None') - represents null or absence of a value e.g result = None

```
# Integer variable
age = 28

# Floating-point variable
height = 1.55

# Boolean variable
is_student = True

# String variable
name = "Faridah Kaberia"

# NoneType variable
result = None
```

```
# Printing variables and their types
print(f"age: {age}, type: {type(age)}")
print(f"height: {height}, type: {type(height)}")
print(f"is_student: {is_student}, type: {type(is_student)}")
print(f"name: {name}, type: {type(name)}")
print(f"result: {result}, type: {type(result)}")
```

5. Control Structures:
   Explain the use of conditional statements and loops in Python. Provide examples of an if-else statement and a for loop.
   ANSWER
   Conditional statements enable you to run specific blocks of code depending on whether a condition is True or False.

```
# Example: Checking if a number is even or odd
number = 20

if number % 2 == 0:
    print(f"{number} is even.")
else:
    print(f"{number} is odd.")
```

   Loops in Python allow you to repeat over sequences or execute repetitive activities until a specific condition is met.

```
# Example: Iterating over a list of names
names = ["Faridah", "Kevin", "Derrick", "Charity"]

for name in names:
    print(f"Hello, {name}!")
```

6. Functions in Python:
   What are functions in Python, and why are they useful? Write a Python function that takes two arguments and returns their sum. Include an example of how to call this function.
   ANSWER
   Python functions are reusable code blocks that execute a certain task. They help you structure your code into digestible chunks, making it easier to read, debug, and maintain. In Python, functions are defined with the def keyword.
   Why they are useful;
   - Modularity - Functions wrap logic into reusable units, increasing code reuse and minimizing redundancy.
   - Abstraction - Functions hide implementation details, letting you to focus on what the function does rather than how it accomplishes it.
   - Organization - Functions help to organize big programs into smaller, logical components, making code more structured and understandable.

- Debugging - Functions facilitate debugging by allowing you to isolate and test specific areas of your code separately.

Python function to calculate sum

```
def calculate_sum(a, b):
 """
Function to calculate the sum of two numbers.

Parameters:
a (int or float): First number.
b (int or float): Second number.

Returns:
int or float: Sum of a and b.
 """
return a + b
```

explanation

`def calculate_sum(a, b):`:This line defines a function named `calculate_sum` that takes two parameters `a` and `b`.

Docstring (`"""..."""`): Provides a brief description of the function's purpose, parameters, and return value.

`return a + b`: This statement returns the sum of `a` and `b` to the caller.

Calling the function

```
# Calling the function with arguments 10 and 3
result = calculate_sum(10, 3)
print("Sum:", result)  # Output: Sum: 13
```

explanation

`calculate_sum(10, 3)`: Calls the `calculate_sum` function with arguments `10` and `3`.

`result = calculate_sum(10, 3)`: Stores the returned value (sum of `10` and `3`, which is `13`) in the variable `result`.

`print("Sum:", result)`: Prints the result, which is 13 in this case.

7. Lists and Dictionaries:
   Describe the differences between lists and dictionaries in Python. Write a script that creates a list of numbers and a dictionary with some key-value pairs, then demonstrates basic operations on both.

   ANSWER

   Lists are ordered collections of items, each item has an index starting from zero and lists can contain duplicate items.

   Characteristics

   Elements can be accessed using their index.

Elements can be of any data type (e.g., integers, strings, other lists, etc.).
Lists are mutable, meaning you can change, add, or remove elements after the list is created.
E.g my_list = [1,2,3,4]

Dictionaries are unordered collections of key value pairs.
Characteristics
Elements are accessed by keys instead of indices.
Keys are unique within a dictionary and must be immutable (typically strings or numbers).
Values can be of any data type (similar to lists).
Dictionaries are mutable.
E.g my_dict = {‘a’ : 1, ‘b’ : 2,’c’ : 3}

Script example

```
# Creating a list of numbers
numbers_list = [1, 2, 3, 4, 5]

# Creating a dictionary of key-value pairs
my_dict = {'apple': 3, 'banana': 5, 'cherry': 7}

# Accessing elements in list and dictionary
print("First element in numbers_list:", numbers_list[0])
print("Value for 'banana' in my_dict:", my_dict['banana'])

# Modifying elements in list and dictionary
numbers_list[2] = 10
my_dict['cherry'] = 8

# Adding new elements to list and dictionary
numbers_list.append(6)
my_dict['grape'] = 4

# Removing elements from list and dictionary
del numbers_list[3]
del my_dict['apple']

# Displaying modified lists and dictionaries
print("Modified numbers_list:", numbers_list)
print("Modified my_dict:", my_dict)
```

8. Exception Handling:
   What is exception handling in Python? Provide an example of how to use try, except, and finally blocks to handle errors in a Python script.
   ANSWER

Exception handling in Python enables you to manage and respond to errors that occur during program execution. It ensures that your application does not crash unexpectedly when an error occurs, but instead handles it in a controlled manner. Here's how exceptions are handled with try, except, and finally blocks:

- 'try' block - it is where you write the code that you wish to monitor for exceptions. If an exception occurs within this block, Python searches for the appropriate except block to handle it.
- 'except' block - This block is only executed if the try block throws an exception. You can catch specific types of exceptions (such as ZeroDivisionError, TypeError, FileNotFoundError, and so on) or catch all exceptions with merely except.
- 'finally' block - This block is optional and runs regardless of whether an exception occurred or not. It's commonly used in cleanup code, such shutting files or releasing resources.

```
def divide_numbers(a, b):
    try:
        result = a / b  # Attempt division
    except ZeroDivisionError:
        print("Error: Division by zero!")
        result = None  # Assign None to result if division by zero occurs
    except TypeError as e:
        print(f"Error: {e}")
        result = None  # Assign None to result for other types of errors
    finally:
        print("Executing finally block.")
        # Clean-up code can go here (e.g., closing files, releasing resources)

    return result

# Example usage:
print(divide_numbers(10, 2))    # Output: 5.0
print(divide_numbers(10, 0))    # Output: Error: Division by zero! None
print(divide_numbers(10, '2'))  # Output: Error: unsupported operand type(s) for /: 'int' and 'str' None
```

9. Modules and Packages:
   Explain the concepts of modules and packages in Python. How can you import and use a module in your script? Provide an example using the math module.
   ANSWER
   Modules are files containing Python code (functions, classes) used for code organization and reuse. Imported using '**import module_name**'
   Packages are directories containing modules and an '**_init_.p**y' file. Used in hierarchical organization of modules.

   To import a module in Python, you use the 'import' statement followed by the module name. Once imported, you can access the functions, classes, and variables defined in that module using dot notation ('module_name, item_name').

Example

```
import math

print(math.sqrt(25))    # 5.0
print(math.pi)          # 3.141592653589793
print(math.sin(math.pi / 2))  # 1.0

radius = 5
area = math.pi * radius ** 2
print("Area of a circle with radius 5:", area)  # 78.53981633974483
```

10. File I/O:
    How do you read from and write to files in Python? Write a script that reads the content of a file and prints it to the console, and another script that writes a list of strings to a file.

    ANSWER

    Reading from a file

    Open file using '**open()**' function '**r**' (read mode)
    Use '**red()**', '**redline()**', or '**readlines()**' to read the file contents.
    Close the file using '**close()**' method.

    Example

```
# Reading from a file
def read_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
            print("Content of the file:")
            print(content)
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")

# Example usage:
read_file('sample.txt')  # Replace 'sample.txt' with your file path
```

    Writing a file

    Open file using '**open()**' function '**w**' (read mode)
    Use '**write()**'to write data to the file.
    Close the file using '**close()**' method.

    Example

```
# Writing to a file
def write_to_file(filename, lines):
    try:
        with open(filename, 'w') as file:
            for line in lines:
                file.write(line + '\n')
        print(f"Successfully wrote {len(lines)} lines to '{filename}'.")
```

```
    except IOError:
        print(f"Error: Failed to write to '{filename}'.")

# Example usage:
lines_to_write = ["First line", "Second line", "Third line"]
write_to_file('output.txt', lines_to_write)  # Replace 'output.txt' with your desired file
path
```