# **SE-Assignment-6**

Assignment: Introduction to Python Instructions: Answer the following questions based on your understanding of Python programming. Provide detailed explanations and examples where appropriate.

Questions:

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.**

- Python is an open-source, object-oriented, high-level, general purpose programming language.

Key features:

* It has an intuitive syntax that resembles a natural English language and hence is **easy to learn**, especially for people who are just entering the world of programming
* Because of its human-friendly syntax, it's **easy to write, read, and debug**
* It provides an **extensive standard library** and a wide choice of well-documented and comprehensive **additional libraries and modules**
* It's **free** both for individuals and businesses
* Thanks to its**huge supporting community**, Python is constantly developed, improved, and expanded
* It can be integrated into any project and used for **solving advanced problems**
* Being a general-purpose language, it has various **applications in many spheres**

**Examples of use cases where Python is particularly effective:**

* Web development: Frameworks like Django and Flask make it easy to build robust web applications.
* Data analysis and visualization: Libraries such as NumPy, Pandas, and Matplotlib are powerful tools for data manipulation and visualization.
* Machine learning and AI: Frameworks like TensorFlow and PyTorch have made Python a go-to language for ML and AI development.
* Scientific computing: Python is widely used in scientific research due to libraries like SciPy and its integration with other scientific tools.
* Automation and scripting: Python's simplicity makes it ideal for writing scripts to automate tasks and workflows.
* Game development: Libraries like Pygame enable developers to create 2D games quickly.
* Network programming: Python's socket programming capabilities make it suitable for building network applications.

1. **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.**

* Install Python in Linux: Most Linux distributions come with Python pre-installed. To install the latest version:

a) Update package list:

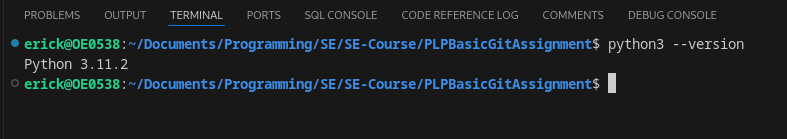
sudo apt update

b) Install Python:

sudo apt install python3

* Verify the installation:

python3 –version

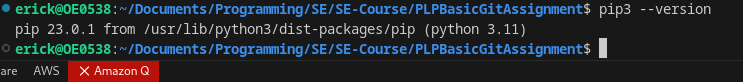


* Install pip (Python package manager)

sudo apt install python3-pip

* Verify pip installation:

pip3 –version



* Install venv for creating virtual environments:

sudo apt install python3-venv

* Create a virtual environment:

python3 -m venv myenv

* Activate the virtual environment:

source myenv/bin/activate

* To deactivate the virtual environment when done:

deactivate

1. 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.

print("Hello, World!")

* Function call: print() is a built-in Python function that outputs text to the console.
* Parentheses: The parentheses () are used to enclose the arguments passed to a function.
* String literal: "Hello, World!" is a string literal, enclosed in double quotes. Python also allows single quotes for strings.
* Statement: This single line is a complete Python statement. In Python, most simple statements are just written on a single line.
* No semicolon: Unlike some other programming languages, Python doesn't require semicolons at the end of statements.
* Indentation: While not visible in this one-line program, Python uses indentation to define code blocks in more complex structures like loops and functions.

1. **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.**

## Scalar Types

* **int:** Positive or negative whole numbers (without a fractional part) e.g. -10, 10, 456, 4654654.
* **float:** Any real number with a floating-point representation in which a fractional component is denoted by a decimal symbol or scientific notation e.g. 1.23, 3.4556789e2.
* complex: A number with a real and imaginary component represented as x + 2y.
* **bool**: Data with one of two built-in values True or False. Notice that 'T' and 'F' are capital. true and false are not valid booleans and Python will throw an error for them.
* **None**: The None represents the null object in Python. A None is returned by functions that don't explicitly return a value.

## Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

* **String:** A string value is a collection of one or more characters put in single, double or triple quotes.
* **List:** A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
* **Tuple:** A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

## Mapping Type

Dictionary: A dictionary Dict() object is an unordered collection of data in a key:value pair form. A collection of such pairs is enclosed in curly brackets. For example: {1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}

## Set Types

* **set:** Set is mutable, unordered collection of distinct hashable objects. The set is a Python implementation of the set in Mathematics. A set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc.
* **frozenset:** Frozenset is immutable version of set whose elements are added from other iterables.

### Mutable and Immutable Types

Data objects of the above types are stored in a computer's memory for processing. Some of these values can be modified during processing, but contents of others can't be altered once they are created in the memory.

Numbers, strings, and Tuples are immutable, which means their contents can't be altered after creation.

On the other hand, items in a List or Dictionary object can be modified. It is possible to add, delete, insert, and rearrange items in a list or dictionary. Hence, they are mutable objects.

**Script demonstrating these data types:**

# Integer

age = 30

print(f"Age (int): {age}")

# Float

height = 1.75

print(f"Height (float): {height}")

# String

name = "Alice"

print(f"Name (str): {name}")

# Boolean

is\_student = True

print(f"Is student (bool): {is\_student}")

# List

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

print(f"Fruits (list): {fruits}")

# Tuple

coordinates = (10, 20)

print(f"Coordinates (tuple): {coordinates}")

# Dictionary

person = {"name": "Bob", "age": 25}

print(f"Person (dict): {person}")

# Set

unique\_numbers = {1, 2, 3, 3, 4, 4, 5}

print(f"Unique numbers (set): {unique\_numbers}")

# Demonstrating type checking

print(f"Type of age: {type(age)}")

print(f"Type of height: {type(height)}")

print(f"Type of name: {type(name)}")

print(f"Type of is\_student: {type(is\_student)}")

print(f"Type of fruits: {type(fruits)}")

print(f"Type of coordinates: {type(coordinates)}")

print(f"Type of person: {type(person)}")

print(f"Type of unique\_numbers: {type(unique\_numbers)}")

This script creates variables of different data types and demonstrates how to use them. It also shows how to check the type of a variable using the type() function.

Key points:

* Variables are created by assigning a value to a name.
* Python uses dynamic typing, meaning you don't need to declare the type of a variable explicitly.
* The f before the strings in the print() functions creates an f-string, allowing easy embedding of variables in strings.
* Lists use square brackets [], tuples use parentheses (), dictionaries use curly braces {} with colons : between keys and values, and sets use curly braces {}.

1. **Control Structures:**
   * **Explain the use of conditional statements and loops in Python. Provide examples of an if-else statement and a for loop.**

**Conditional statements**, sometimes called branching statements, let you run different parts of your code depending on whether certain conditions are true or false. For instance, you can use an ***if statement*** to figure out if a number is even or odd and then display a different message based on the answer.

On the other hand, **loops** allow you to repeat a block of code until a particular condition is satisfied. This is handy when you need to go through a list of items or process data until everything has been handled.

In Python, there are two main types of loops: ***for loops*** and ***while loops***. For loops are great for going through a sequence of items like a list or a string. Meanwhile, ***while loops***are useful when you want to keep running a block of code until a specific condition is fulfilled.

***if-else statement:*** This statement allows you to execute different blocks of code based on whether a condition is *true* or *false*. The syntax is:

if condition:

# Code to execute if condition is true

else:

# Code to execute if condition is false

**Example of a if else:**

age = 18

if age >= 18:

print("You are eligible to vote.")

else:

print("You are too young to vote.")

for loop: This loop iterates over a sequence of elements and executes a block of code for each element. The syntax

for item in sequence:

# Code to execute for each item

**Example of a for loop:**

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

for fruit in fruits:

print(f"I like {fruit}")

# Using range() function

for i in range(5):

print(f"Count: {i}")

1. **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.**

A function in programming is a block of code organized by a set of rules to accomplish a specific task. They can be reused any number of times at any point during software development.

Why are they useful?

* Code organization: Functions help break down complex programs into smaller, manageable parts.
* Reusability: You can call a function multiple times without rewriting the same code.
* Abstraction: Functions allow you to hide complex implementations behind a simple interface.
* Modularity: You can easily modify or update a function without affecting the rest of the code.

Example;

def add\_numbers(a, b):

return a + b

# Example of calling the function

result = add\_numbers(5, 3)

print(result) # Output: 8

his function, add\_numbers, takes two parameters a and b, and returns their sum using the + operator.

In the example, we call the function with arguments 5 and 3, storing the result in the result variable. When we print result, it outputs 8.

1. **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.**

Lists in Python are ordered collections of items and each item is assigned to an index value starting from 0. Lists are mutable, hence the elements can be modified after their creation.

To create a list in Python, you have to use square brackets []. Items in the lists can be of any data type, including numbers, strings, or even lists.

A dictionary is an unordered collection of data in the key: value pair form. The keys are unique, and the value can be of any data type. Like the lists, dictionaries are also mutable, i.e., you can change the key-value pair after creating the dictionary.

* To create a dictionary in Python, you have to use {}. : is used to separate the key and value.
* Elements in the dictionary can be accessed by their corresponding keys.

**Differences:**

|  |  |  |
| --- | --- | --- |
| ****Parameter**** | ****List**** | ****Dictionary**** |
| ****Definition**** | An ordered collection of items. | An unordered collection of data in a key: value pair form. |
| ****Syntax**** | Uses square brackets []. | Uses curly braces {}. |
| ****Ordering**** | Ordered: Items have a defined order, which will not change. | Unordered: Items do not have a defined order. |
| ****Indexing**** | Accessed by index, starting from 0. | Values are accessed using keys. |
| ****Mutability**** | Mutable: Items can be modified after creation. | Mutable: Values can be updated, and key: value pairs can be added or removed. |
| ****Uniqueness**** | Allows duplicate items. | Does not allow duplicate keys. However, values can be duplicated. |
| ****Data Types**** | It can store any data type. | Keys can be of any immutable data type (e.g., strings, numbers, tuples). Values can be of any type. |
| ****Use Case**** | When order matters or when you need to store multiple values for an item. | When you need a unique key for each piece of data. |
| ****Example**** | fruits = [“apple”, “banana”, “cherry”] | person = {“name”: “John”, “age”: 30} |
| ****Built-in Functions**** | append(), remove(), pop(), sort(), etc. | keys(), values(), items(), get(), pop(), etc. |
| ****Performance**** | Faster for ordered operations like sorting. | Faster for lookup operations due to the hash mapping of keys. |

# Creating a list of numbers

numbers = [1, 2, 3, 4, 5]

# Creating a dictionary with key-value pairs

person = {

"name": "Alice",

"age": 30,

"city": "New York"

}

# List operations

print("List operations:")

print(f"Original list: {numbers}")

numbers.append(6) # Add an element to the end

print(f"After append: {numbers}")

numbers.insert(0, 0) # Insert an element at a specific index

print(f"After insert: {numbers}")

popped\_value = numbers.pop() # Remove and return the last element

print(f"Popped value: {popped\_value}")

print(f"After pop: {numbers}")

print(f"Length of list: {len(numbers)}")

print(f"Sum of list: {sum(numbers)}")

# Dictionary operations

print("\nDictionary operations:")

print(f"Original dictionary: {person}")

person["job"] = "Engineer" # Add a new key-value pair

print(f"After adding 'job': {person}")

age = person.get("age") # Get a value by key

print(f"Age: {age}")

person["age"] = 31 # Modify an existing value

print(f"After modifying age: {person}")

removed\_city = person.pop("city") # Remove a key-value pair

print(f"Removed city: {removed\_city}")

print(f"After removing 'city': {person}")

print(f"Keys: {list(person.keys())}")

print(f"Values: {list(person.values())}")

print(f"Items: {list(person.items())}")

1. **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.**

An Exception is an Unexpected Event, which occurs during the execution of the program. It is also known as a run time error. When that error occurs, Python generates an exception during the execution and that can be handled, which prevents your program from interrupting.

* Try: This block will test the excepted error to occur
* Except: Here you can handle the error
* Else: If there is no exception then this block will be executed
* Finally: Finally block always gets executed either exception is generated or not

def divide\_numbers(a, b):

try:

result = a / b

print(f"The result of {a} divided by {b} is: {result}")

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

except TypeError:

print("Error: Please provide numeric values!")

else:

print("Division operation completed successfully.")

finally:

print("This block always executes, regardless of exceptions.")

# Test cases

print("Case 1:")

divide\_numbers(10, 2)

print("\nCase 2:")

divide\_numbers(10, 0)

print("\nCase 3:")

divide\_numbers("10", 2)

print("\nCase 4:")

divide\_numbers(10, "2")

**When you run this script, it will output:**

Case 1:

The result of 10 divided by 2 is: 5.0

Division operation completed successfully.

This block always executes, regardless of exceptions.

Case 2:

Error: Cannot divide by zero!

This block always executes, regardless of exceptions.

Case 3:

Error: Please provide numeric values!

This block always executes, regardless of exceptions.

Case 4:

Error: Please provide numeric values!

This block always executes, regardless of exceptions.

1. **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.**

The module is a simple Python file that contains collections of functions and global variables and with having a .py extension file. It is an executable file and to organize all the modules we have the concept called Package in Python. Eg, regex, datetime, random.

The package is a simple directory having collections of modules. This directory contains Python modules and also having \_\_init\_\_.py file by which the interpreter interprets it as a Package. The package is simply a namespace. The package also contains sub-packages inside it. e.g, numpy, pandas.

Example;

import math

from math import sqrt, pi

# Using the full module name

print(f"The value of pi is approximately {math.pi}")

print(f"The sine of 30 degrees is {math.sin(math.radians(30))}")

# Using imported functions directly

print(f"The square root of 16 is {sqrt(16)}")

# Demonstrating some other math functions

numbers = [1, 2, 3, 4, 5]

print(f"The product of {numbers} is {math.prod(numbers)}")

print(f"5 factorial is {math.factorial(5)}")

print(f"The greatest common divisor of 48 and 18 is {math.gcd(48, 18)}")

# Using math.isclose() for float comparisons

a = 0.1 + 0.2

b = 0.3

print(f"Is {a} close to {b}? {math.isclose(a, b)}")

# Calculate the area of a circle with radius 5

radius = 5

area = pi \* radius\*\*2

print(f"The area of a circle with radius {radius} is approximately {area:.2f}")

1. **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.**

In Python, there are six methods or access modes, which are:

* **Read Only ('r’):** This mode opens the text files for reading only. The start of the file is where the handle is located. It raises the I/O error if the file does not exist. This is the default mode for opening files as well.
* **Read and Write ('r+’):** This method opens the file for both reading and writing. The start of the file is where the handle is located. If the file does not exist, an I/O error gets raised.
* **Write Only ('w’):** This mode opens the file for writing only. The data in existing files are modified and overwritten. The start of the file is where the handle is located. If the file does not already exist in the folder, a new one gets created.
* **Write and Read ('w+’):** This mode opens the file for both reading and writing. The text is overwritten and deleted from an existing file. The start of the file is where the handle is located.
* **Append Only ('a’):** This mode allows the file to be opened for writing. If the file doesn't yet exist, a new one gets created. The handle is set at the end of the file. The newly written data will be added at the end, following the previously written data.
* **Append and Read (‘a+’):** Using this method, you can read and write in the file. If the file doesn't already exist, one gets created. The handle is set at the end of the file. The newly written text will be added at the end, following the previously written data.

**# Script to read from a file and print its content to the console**

# Specify the file path

file\_path = "example.txt"

# Open and read the file

try:

with open(file\_path, "r") as file:

content = file.read()

print("File contents:")

print(content)

except FileNotFoundError:

print(f"Error: The file '{file\_path}' was not found.")

except IOError:

print(f"Error: There was an issue reading the file '{file\_path}'.")

**# Script to write a list of strings to a file**

# Specify the file path

file\_path = "output.txt"

# List of strings to write

strings\_to\_write = [

"Hello, world!",

"This is a test file.",

"Python makes file I/O easy.",

"Goodbye!"

]

# Write to the file

try:

with open(file\_path, "w") as file:

for string in strings\_to\_write:

file.write(string + "\n")

print(f"Successfully wrote to {file\_path}")

except IOError:

print(f"Error: There was an issue writing to the file '{file\_path}'.")

**Notes:**

* The "r" mode is for reading (default), "w" for writing (overwrites the file), and "a" for appending (adds to the end of the file).
* Always use exception handling when working with files to gracefully handle potential errors.
* The with statement is recommended as it automatically closes the file, even if an exception occurs.
* For large files, you might want to read line-by-line instead of the whole file at once:

**References:**

https://www.datacamp.com/blog/what-is-python-used-for

https://www.tutorialsteacher.com/python/python-data-types

https://accuweb.cloud/resource/articles/explain-loops-and-conditional-statements-in-python/

https://www.analyticsvidhya.com/blog/2021/08/what-are-functions-in-python-how-to-create-functions-in-python/

https://www.shiksha.com/online-courses/articles/difference-between-list-and-dictionary-in-python/

<https://www.geeksforgeeks.org/try-except-else-and-finally-in-python/>

<https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/>

https://www.freecodecamp.org/news/file-handling-in-python/

# **Submission Guidelines:**

* **Your answers should be well-structured, concise, and to the point.**
* **Provide code snippets or complete scripts where applicable.**
* **Cite any references or sources you use in your answers.**
* **Submit your completed assignment by [due date].**