**Name: Ann Kairuthi**

**Email:** [**kairuthiann2@gmail.com**](mailto:kairuthiann2@gmail.com)

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

**Questions:**

**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.

***What is python***

Python is an interpreted programming language that is object-oriented, and considered to be high-level too.

***Key features that make python popular***

* Python is one of the easiest yet most useful programming languages which is why it is popular among developers.
* Due to its easiest syntax, it is recommended for beginners who are new to the software engineering field
* It has hundreds of libraries and frameworks to work with
* Python has a huge and active community which contributes to its ecosystem.

***Use cases of python***

People use Python for Competitive Programming, Web Development, and creating software.

**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.

***Steps to install python on Windows***

Download Python:

Go to the official Python website at https://www.python.org/. On the homepage, you will see a “Downloads” section. Click on the “Download Python” button.

Choose the Version:

You will be directed to a page where you can choose the version of Python you want to download. Python usually has two main versions available; select Python 3 for windows operating system

Add Python to PATH (Optional):

On Windows, you may be given the option to add Python to your system’s PATH environment variable. This makes it easier to run Python from the command line.

Install Python:

Click the “Install Now” button to begin the installation. The installer will copy the necessary files to your computer.

Verify the Installation:

After the installation is complete, you can verify that Python was installed correctly by opening a command prompt (on Windows) or a terminal (on macOS or Linux) and typing python --version. This should display the version of Python you installed.

***Setting up a virtual environment.***

Open the command prompt or git bush on windows

Type `python –version` and press enter to verify the installed python version

Install `pip` a python’s package installer by running `python get-pip.py`.

Navigate to your project directory using `cd` command.

Install virtual environment by running `python -m pip install virtualenv`.

Create the virtual environment to start using it `python -m virtualenv myproject`. Replace `myproject` with your project name.

Activate the virtual environment with `source myproject/scripts/activate. You should see the environment name in the git bush.

To deactivate, run `deactivate` from anywhere in the directory.

**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!")

#Then run

Print: this is a built-in python function used to output data to the console. In this case it has been used to display “hello world” to the user.

Parentheses (): they enclose the text in parentheses indicating that this is a function call.

Quotation marks “”: these equitation marks are used to define a string in python.

Hello world: This is the actual text that will be printed to the console.

**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.

***Basic data types in python:***

**Numeric Data**: Represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int, Python float, and Python complex classes in Python.

#numeric data

a = 7

print("Type of a: ", type(a))

b = 2.0

print("\nType of b: ", type(b))

c = 3 + 4j

print("\nType of c: ", type(c))

**Sequence Type:** This is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python: Python String, Python List and Python Tuple.

#list

grades = [90, 65, 76, 92]

grades.append(88)

print("Grades:", grades)

print("Updated grades:", grades)

#tuple

coordinates = (11.3, 21.5)

print("Coordinates:", type(coordinates))

#String

name = "Ann"

print("Her name is:", type(name))

**Boolean:** This is a Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). True and False with capital ‘T’ and ‘F’ are valid Booleans otherwise python will throw an error.

#Boolean

student = True

print("Is student:", type(student))

**Set:** A Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

unique\_numbers = {2, 4, 6, 8}

print("Unique Numbers:", unique\_numbers)

**Dictionary:** A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a ‘comma’.

#Dictionary

student\_info = {

"name": "Bob",

"age": 22,

"major": "Software Development"

}

student\_name = student\_info["name"]

student\_major = student\_info["major"]

print(f"Student {student\_name} is majoring in {student\_major}.")

**Control Structures:**

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

***Use of conditional statements***

Conditional statements in python are used to execute code based on certain conditions. One common conditional statement is the `if-else` statement which allows you to execute one block of code if a condition is false.

#if else

a = 20

b =10

if a > b:

print("You are an adult.")

else:

print("You are a minor.")

***Loops***

Loops are used to execute a block of code repeatedly as long as a certain condition is met. The two main types of loops in python are `for` loops and `while` loops.

#While loop

i = 6

while i < 10:

print(i)

i += 1

#For loop

for x in "banana":

print(x)

fruits = ["Apple", "Mango", "kiwi", "Banaa"]

for x in fruits:

print(x)

if x == "Kiwi":

break

**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.

Function is a block of statements that returns the specified tasks. The concept is to group some often or repeatedly performed activities together and create a function so that instead of writing the same code for different inputs, we can use the function calls to reuse the code included in it over and over.

***Why functions are useful***

They help break down complex problems into smaller, manageable pieces.

Functions make the code more organized and easier to maintain.

They allow you to reuse code without rewriting it.

They hide the implementation details and expose only the necessary interface.

***Writing a python function***

def add\_numbers(a, b):

return a + b

result = add\_numbers(10, 2)

print(f"The sum of 10 and 2 is: {result}")

**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** | **Dictionaries** |
| Lists maintain the order of elements | Dictionaries do not maintain the order of elements |
| Elements in a list can be accessed using their index, starting from 0 | Dictionaries store data in key-value pairs |
| Lists can contain duplicate elements. | Dictionary keys must be unique. |
| Lists are defined using square brackets []. | Dictionaries are defined using curly braces { } with a colon : separating keys and values |
|  |  |

# creating a list of numbers

numbers = [10, 12, 14, 16, 18, 20]

# Accessing elements

print("First number in the list:", numbers[4])

# Basic operations in list

# Adding an element to the list

numbers.append(22)

print("List after adding an element:", numbers)

# Removing an element from the list

numbers.remove(14)

print("List after removing an element:", numbers)

# Creating a dictionary with key-value pairs

client\_info ={

"name": "Brenda",

"age": 22,

"city": "New york",

}

# Basic operationson the dictionary

#Accessing values by keys

print("Client's name:", client\_info["name"])

# Adding a new key-value pair

client\_info["lives is"] = "New york"

print("Dictionary after adding a key-value pair:", client\_info)

# Removing a key-value pair

del client\_info["age"]

print("Dictionary after adding a key-value pair:", client\_info)

**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.

Exception handling in Python is a mechanism to handle runtime errors gracefully. When an error occurs, Python stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash. Exception handling allows a program to continue executing or to terminate gracefully, providing informative error messages to the user.

# Exception handling

try:

print(x)

except:

print("Something went wrong")

finally:

print("The 'try except' is finished")

**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.

A module is similar to code library. It is a file containing a set of functions you want to include in your application. It can include functions, classes, and variables as well as runnable code. Modules assist to divide code into different namespaces, making it easier to manage big codebases.

import math

# Calculate the square root of a number

num = 16

sqrt\_num = math.sqrt(num)

print(f"The square root of {num} is {sqrt\_num}")

# Calculate the cosine of an angle (in radians)

angle = math.pi / 4 # 45 degrees

cos\_angle = math.cos(angle)

print(f"The cosine of 45 degrees is {cos\_angle}")

# Calculate the factorial of a number

n = 5

factorial\_n = math.factorial(n)

print(f"The factorial of {n} is {factorial\_n}")

A package is a way of organizing multiple modules into a directory hierarchy. A package is essentially a directory containing a special \_\_init\_\_.py file (which can be empty) and multiple module files.

**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 you can read from and write to files using built-in functions. The open() function is used to open a file and you can specify the module in which to open the file such as read (‘r’), write (‘w’), or append (‘a’),

# read\_file.py

def read\_file(file\_path):

try:

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

content = file.read()

print(content)

except FileNotFoundError:

print(f"The file {file\_path} does not exist.")

file\_path = 'example.txt'

read\_file(file\_path)

# write\_file.py

def write\_to\_file(file\_path, lines):

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

for line in lines:

file. write(line + '\n')

file\_path = 'output.txt'

lines = ["Hello, world!", "This is a test file.", "Python file handling is simple."]

write\_to\_file(file\_path, lines)

**References**

Beazley, D., & Jones, B. K. (2013). *Python Cookbook* (3rd ed.). O'Reilly Media.

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.