LIST OF EXPERIMENTS

**Experiment 1: Understanding Basic Data Types**

python

Copy code

# Integer data type

integer\_variable = 42

# Float data type

float\_variable = 3.14159

# String data type

string\_variable = "Hello, World!"

# Boolean data type

boolean\_variable = True

# Printing the data types of variables

print(type(integer\_variable))

print(type(float\_variable))

print(type(string\_variable))

print(type(boolean\_variable))

**Explanation:** In this experiment, we explore the basic data types in Python: integers, floats, strings, and booleans. We assign values to variables of these data types and use the type() function to print the data type of each variable.

**Experiment 2: Type Conversion**

python

Copy code

# Convert float to integer

float\_number = 3.99

int\_number = int(float\_number)

# Convert integer to string

integer\_value = 42

string\_value = str(integer\_value)

# Convert string to float

string\_number = "3.14"

float\_value = float(string\_number)

# Print the converted values and their types

print(int\_number, type(int\_number))

print(string\_value, type(string\_value))

print(float\_value, type(float\_value))

**Explanation:** This experiment demonstrates type conversion in Python. We convert a float to an integer, an integer to a string, and a string to a float.

**Experiment 3: Lists (Mutable Sequences)**

python

Copy code

# Creating a list

my\_list = [1, 2, 3, 4, 5]

# Accessing elements

first\_element = my\_list[0]

last\_element = my\_list[-1]

# Modifying elements

my\_list[2] = 99

# Appending elements

my\_list.append(6)

# Length of the list

list\_length = len(my\_list)

# Printing the list and its properties

print(my\_list)

print(first\_element, last\_element)

print(list\_length)

**Explanation:** This experiment introduces lists, which are mutable sequences in Python. We create a list, access and modify its elements, append an element, and determine its length.

**Experiment 4: Tuples (Immutable Sequences)**

python

Copy code

# Creating a tuple

my\_tuple = (1, 2, 3, 4, 5)

# Accessing elements

first\_element = my\_tuple[0]

last\_element = my\_tuple[-1]

# Trying to modify a tuple (will result in an error)

# Length of the tuple

tuple\_length = len(my\_tuple)

# Printing the tuple and its properties

print(my\_tuple)

print(first\_element, last\_element)

print(tuple\_length)

**Explanation:** This experiment introduces tuples, which are immutable sequences in Python. We create a tuple, access its elements, attempt to modify it (which will raise an error), and determine its length.

**Experiment 5: Strings (Immutable Sequences)**

python

Copy code

# Creating a string

my\_string = "Python"

# Accessing characters

first\_char = my\_string[0]

substring = my\_string[1:4]

# Concatenating strings

new\_string = my\_string + " is awesome!"

# Length of the string

string\_length = len(my\_string)

# Printing the string and its properties

print(my\_string)

print(first\_char)

print(substring)

print(new\_string)

print(string\_length)

**Explanation:** This experiment explores strings, which are immutable sequences of characters in Python. We access characters, concatenate strings, and determine the length of a string.

**Experiment 6: Sets (Unordered, Unique Elements)**

python

Copy code

# Creating a set

my\_set = {1, 2, 3, 4, 4, 5}

# Adding elements

my\_set.add(6)

# Removing elements

my\_set.remove(3)

# Checking membership

is\_present = 2 in my\_set

# Length of the set

set\_length = len(my\_set)

# Printing the set and its properties

print(my\_set)

print(is\_present)

print(set\_length)

**Explanation:** This experiment introduces sets, which are unordered collections of unique elements in Python. We add and remove elements, check for membership, and determine the length of a set.

**Experiment 7: Dictionaries (Key-Value Pairs)**

python

Copy code

# Creating a dictionary

my\_dict = {

'name': 'Alice',

'age': 30,

'city': 'Wonderland'

}

# Accessing values

name = my\_dict['name']

age = my\_dict.get('age')

# Modifying values

my\_dict['age'] = 31

# Adding new key-value pairs

my\_dict['country'] = 'Nowhere'

# Checking if a key exists

is\_present = 'city' in my\_dict

# Printing the dictionary and its properties

print(my\_dict)

print(name)

print(age)

print(is\_present)

**Explanation:** This experiment introduces dictionaries, which are collections of key-value pairs in Python. We access values, modify values, add new key-value pairs, and check for the existence of keys.

**Experiment 8: Type Coercion and Arithmetic Operations**

python

Copy code

# Integer division

result1 = 7 // 2

# Float division

result2 = 7 / 2

# Exponentiation

result3 = 2 \*\* 3

# Modulus

result4 = 10 % 3

# Type coercion in arithmetic

result5 = 7 + 2.0

# Printing the results

print(result1)

print(result2)

print(result3)

print(result4)

print(result5)

**Explanation:** In this experiment, we explore type coercion and common arithmetic operations in Python, including integer division, float division, exponentiation, modulus, and type coercion in arithmetic.

**Experiment 9: None Type**

python

Copy code

# Creating a None value

empty\_value = None

# Checking if a variable is None

is\_empty = empty\_value is None

# Printing the result

print(is\_empty)

**Explanation:** The None type represents the absence of a value. In this experiment, we create a variable with a None value and check if it is None.

**Experiment 10: Checking Data Types**

python

Copy code

# Function to check data types

def check\_data\_type(variable):

if isinstance(variable, int):

return "Integer"

elif isinstance(variable, float):

return "Float"

elif isinstance(variable, str):

return "String"

elif isinstance(variable, bool):

return "Boolean"

else:

return "Unknown"

# Test the function

value1 = 42

value2 = 3.14

value3 = "Hello, World!"

value4