LIST OF EXPERIMENTS

**Experiment 1: Basic Tuple Assignment**

**Theory:**

Tuple assignment allows assigning multiple variables at once.

Code:

python

x, y, z = 1, 2, 3

print(x, y, z)

# Output: 1 2 3

**Experiment 2: Swapping Values with Tuple Assignment**

**Theory:**

Tuple assignment enables the swapping of values between variables without using a temporary variable.

Code:

python

x = 5

y = 10

x, y = y, x

print(x, y)

# Output: 10 5

**Experiment 3: Unpacking a Tuple**

**Theory:**

Unpacking a tuple assigns its elements to variables directly.

Code:

python

coordinates = (3, 4)

x, y = coordinates

print(x, y)

# Output: 3 4

**Experiment 4: Unpacking Elements with the Asterisk (\*) Operator**

**Theory:**

The asterisk (\*) operator unpacks elements, handling multiple or remaining elements.

Code:

python

numbers = (1, 2, 3, 4, 5)

first, \*rest, last = numbers

print(first, rest, last)

# Output: 1 [2, 3, 4] 5

**Experiment 5: Ignoring Unwanted Values in Tuple Assignment**

**Theory:**

Using \_ allows ignoring unwanted values during unpacking.

Code:

python

data = (10, 20, 30)

x, \_, z = data

print(x, z)

# Output: 10 30

**Experiment 6: Unpacking Nested Tuples**

**Theory:**

Nested tuples can be unpacked, assigning their elements to variables.

Code:

python

point = (3, (4, 5))

x, (y, z) = point

print(x, y, z)

# Output: 3 4 5

**Experiment 7: Using Tuple Assignment in Loops**

**Theory:**

Tuple assignments can simplify iteration over sequences.

Code:

python

coordinates = [(1, 2), (3, 4), (5, 6)]

for x, y in coordinates:

print(f"Point: ({x}, {y})")

**Experiment 8: Returning Multiple Values from Functions**

**Theory:**

Functions can return multiple values as a tuple.

Code:

python

return x + y, x - y

sum\_result, diff\_result = calculate(10, 5)

print(sum\_result, diff\_result)

# Output: 15 5

**Experiment 9: Tuple Assignment with Functions**

**Theory:**

Tuple assignment can be used to simplify function calls.

Code:

python

def get\_coordinates():

return 7, 8

x, y = get\_coordinates()

print(x, y)

# Output: 7 8

**Experiment 10: Handling Function Return Values Using Tuple Assignment**

**Theory:**

Handling returned values allows for selective processing.

Code:

python

def func():

return 1, 2, 3

first, \*rest = func()

print(first, rest)

# Output: 1 [2, 3]

**Experiment 11: Assigning Single-Element Tuple with Comma**

**Theory:**

A single-element tuple requires a trailing comma.

Code:

python

single\_tuple = (5,)

print(single\_tuple)

# Output: (5,)

**Experiment 12: Unpacking Variable-Length Tuples**

**Theory:**

Tuple assignment accommodates variable-length tuples.

Code:

python

numbers = (1, 2, 3, 4, 5)

first, \*middle, last = numbers

print(first, middle, last)

# Output: 1 [2, 3, 4] 5

**Experiment 13: Tuple Assignment with Default Values**

**Theory:**

Default values can be assigned during tuple unpacking.

Code:

python

x, y, z = (1, 2)

print(x, y, z)

# Output: 1 2

**Experiment 14: Unpacking Specific Elements**

**Theory:**

Selectively unpacking specific elements from a tuple.

Code:

python

data = (10, 20, 30, 40)

first, \_, third, \_ = data

print(first, third)

# Output: 10 30

**Experiment 15: Simultaneous Assignment and Reassignment**

**Theory:**

Tuple assignment allows simultaneous assignment and reassignment.

Code:

python

x, y = 5, 10

x, y = y, x + y

print(x, y)

# Output: 10 15

**Experiment 16: Unpacking Tuple from Function Return**

**Theory:**

Functions returning tuples can directly be unpacked.

Code:

python

def get\_data():

return "hello", 42

message, number = get\_data()

print(message, number)

# Output: hello 42

**Experiment 17: Tuple Assignment in List Comprehension**

**Theory:**

List comprehension can utilize tuple assignments.

Code:

python

data = [(1, 2), (3, 4), (5, 6)]

sums = [x + y for x, y in data]

print(sums)

# Output: [3, 7, 11]

**Experiment 18: Assigning Unpacked Tuple to a Variable**

**Theory:**

Assigning an unpacked tuple to a variable.

Code:

python

data = (7, 8)

coordinates = x, y = data

print(coordinates) # Output: (7, 8)