DSP - Practical - QB - Solutions - By - Th3_

Experiment No. 2: Operators

(a) Arithmetic Operators

```
a, b = 10, 5
print("Addition:", a + b)  # Output: 15
print("Subtraction:", a - b)  # Output: 5
print("Multiplication:", a * b)  # Output: 50
print("Division:", a / b)  # Output: 2.0
```

(b) Logical Operators

```
x, y = True, False
print("Logical AND:", x and y)  # Output: False
print("Logical OR:", x or y)  # Output: True
```

(c) Relational Operators

```
a, b = 10, 5
print("Is a equal to b?", a == b)  # Output: False
print("Is a greater than b?", a > b)  # Output: True
```

(d) Conditional Operators

```
num = 15
if num > 10 and num < 20:
    print(f"{num} is between 10 and 20")  # Output: 15 is between 10 and 20
```

(e) Bitwise Operators

```
a, b = 6, 3  # Binary: 6 = 110, 3 = 011
print("Bitwise AND:", a & b)  # Output: 2 (Binary: 010)
print("Bitwise OR:", a | b)  # Output: 7 (Binary: 111)
```

(f) Ternary Operator

```
a = 10
is_even = "Even" if a % 2 == 0 else "Odd"
print(f"{a} is {is_even}")  # Output: 10 is Even
```

DSP - Practical - QB - Solutions - By - Th3_

Experiment No. 3: Conditional Statements

(i) if

```
num = 5
if num > 0:
    print("Number is positive.")  # Output: Number is positive.
```

(ii) if...else

```
num = -3
if num > 0:
    print("Number is positive.")
else:
    print("Number is non-positive.")  # Output: Number is non-positive.
```

(iii) Nested if

```
num = 50
if num > 0:
    if num < 100:
        print("Number is positive and less than 100.")  # Output: Number is positive and
less than 100.
```

(iv) Switch case

```
def switch_case(option):
    match option:
        case 1: print("Option 1 selected.")  # Output: Option 1 selected.
        case 2: print("Option 2 selected.")  # For option 2
        case _: print("Invalid option.")

switch_case(1)
```

Experiment No. 6: Operations on Lists

(a) Create

```
my_list = [1, 2, 3, 4, 5]
print("Created List:", my_list)  # Output: [1, 2, 3, 4, 5]
```

(b) Access

```
my_list = [1, 2, 3, 4, 5]
print("First element:", my_list[0])  # Output: 1
print("Last element:", my_list[-1])  # Output: 5
```

(c) Update

```
my_list = [1, 2, 3, 4, 5]
my_list[1] = 20
print("Updated List:", my_list)  # Output: [1, 20, 3, 4, 5]
```

(d) Delete

```
my_list = [1, 20, 3, 4, 5]
del my_list[2]
print("List after deletion:", my_list)  # Output: [1, 20, 4, 5]
```

Experiment No. 8: Operations on Sets

(a) Create

```
my_set = {1, 2, 3, 4}
print("Created Set:", my_set)  # Output: {1, 2, 3, 4}
```

(b) Access

```
my_set = {1, 2, 3, 4}
for elem in my_set:
    print(elem)
# Output: 1 2 3 4 (order may vary)
```

(c) Update

```
my_set = {1, 2, 3, 4}
my_set.add(5)
print("Set after addition:", my_set)  # Output: {1, 2, 3, 4, 5}
```

(d) Delete

```
my_set = {1, 2, 3, 4, 5}
my_set.remove(3)
print("Set after deletion:", my_set)  # Output: {1, 2, 4, 5}
```

Experiment No. 9: Operations on Dictionaries

(a) Create

```
my_dict = {"a": 1, "b": 2, "c": 3}
print("Created Dictionary:", my_dict)  # Output: {'a': 1, 'b': 2, 'c': 3}
```

(b) Access

```
my_dict = {"a": 1, "b": 2, "c": 3}
print("Value for key 'b':", my_dict["b"])  # Output: 2
```

(c) Update

```
my_dict = {"a": 1, "b": 2, "c": 3}
my_dict["b"] = 20
print("Updated Dictionary:", my_dict)  # Output: {'a': 1, 'b': 20, 'c': 3}
```

(d) Looping

```
my_dict = {"a": 1, "b": 20, "c": 3}
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")
# Output:
# Key: a, Value: 1
# Key: b, Value: 20
# Key: c, Value: 3
```

DSP - Practical - QB - Solutions - By - Th3_

Experiment No. 10: Math Built-in Functions

Code:

```
import math

print("Square root of 16:", math.sqrt(16))  # Output: 4.0
print("Power of 2^3:", math.pow(2, 3))  # Output: 8.0
print("Pi value:", math.pi)  # Output: 3.141592653589793
print("Ceiling of 4.2:", math.ceil(4.2))  # Output: 5
print("Floor of 4.8:", math.floor(4.8))  # Output: 4
```

Experiment No. 11: User-Defined Function

Code:

```
def multiply_and_add(a, b):
    return (a * b) + (a + b)

result = multiply_and_add(3, 4)
print("Result:", result)  # Output: 19
```

Experiment No. 14: NumPy Array Operations

Code:

```
import numpy as np

# Creating arrays
arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)  # Output: [1 2 3 4 5]

# Accessing elements
print("First element:", arr[0])  # Output: 1

# Performing operations
print("Sum of elements:", np.sum(arr))  # Output: 15
print("Reversed array:", arr[::-1])  # Output: [5 4 3 2 1]
```

DSP - Practical - QB - Solutions - By - Th3_

Experiment No. 16: Method Overloading and Overriding

(a) Method Overloading

```
class Overload:
    def add(self, a, b=0):
        return a + b

obj = Overload()
print("Add one argument:", obj.add(5))  # Output: 5
print("Add two arguments:", obj.add(5, 3))  # Output: 8
```

(b) Method Overriding

```
class Parent:
    def show(self):
        print("Parent's method.")

class Child(Parent):
    def show(self):
        print("Child's method.")

child = Child()
child.show()  # Output: Child's method.
```

Experiment No. 17: Types of Inheritance

(a) Single Inheritance

```
class Parent:
    def greet(self):
        print("Hello from Parent!")

class Child(Parent):
    pass

child = Child()
child.greet()  # Output: Hello from Parent!
```

(b) Multilevel Inheritance

```
class Parent:
    def greet(self):
        print("Hello from Parent!")

class Child(Parent):
    pass

class GrandChild(Child):
    pass

grandchild = GrandChild()
grandchild.greet()  # Output: Hello from Parent!
```

(c) Multiple Inheritance

```
class A:
    def method_a(self):
        print("Method from A")

class B:
    def method_b(self):
        print("Method from B")

class C(A, B):
    pass

obj = C()
obj.method_a()  # Output: Method from A
obj.method_b()  # Output: Method from B
```

(d) Hybrid Inheritance

```
class A:
    def method_a(self):
        print("Method from A")

class B(A):
    def method_b(self):
        print("Method from B")

class C(A):
    def method_c(self):
        print("Method from C")

class D(B, C):
    pass

obj = D()
obj.method_a()  # Output: Method from A
obj.method_b()  # Output: Method from B
obj.method_c()  # Output: Method from C
```

(e) Hierarchical Inheritance

```
class Parent:
    def greet(self):
        print("Hello from Parent!")

class Child1(Parent):
    pass

class Child2(Parent):
    pass

child1 = Child1()
child1.greet()  # Output: Hello from Parent!

child2 = Child2()
child2.greet()  # Output: Hello from Parent!
```

DSP - Practical - QB - Solutions - By - Th3_

Experiment No. 18: Array Operations

Code:

```
from array import array

# Array declaration
arr = array('i', [1, 2, 3])

# Insertion
arr.insert(1, 4)
print("Array after insertion:", arr)  # Output: array('i', [1, 4, 2, 3])

# Deletion
arr.remove(2)
print("Array after deletion:", arr)  # Output: array('i', [1, 4, 3])

# Append
arr.append(5)
print("Array after append:", arr)  # Output: array('i', [1, 4, 3, 5])

# Index
print("Index of 5:", arr.index(5))  # Output: 3

# Reverse
arr.reverse()
print("Reversed array:", arr)  # Output: array('i', [5, 3, 4, 1])
```

Experiment No. 23: Evaluate Expression

Code:

```
expression = "3 + 5 * (2 - 8)"
result = eval(expression)
print("Result of the expression:", result)  # Output: -27
```