1. Swap Two Numbers

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
temp = a
a = b
b = temp
print("After swapping: a =", a, ", b =", b)
```

2. Add Two Numbers

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
sum = num1 + num2
print("Sum =", sum)
```

3. Factorial of a Number (using recursion)

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)

num = int(input("Enter a number: "))
print("Factorial of", num, "is", factorial(num))
```

4. Check if a String is Palindrome

```
def is_palindrome(s):
    return s == s[::-1]

string = input("Enter a string: ")
if is_palindrome(string):
    print(string, "is a palindrome.")
else:
    print(string, "is not a palindrome.")
```

5. Calculate the Square of a Number

```
num = int(input("Enter a number to find its square: "))
print("Square of", num, "is", num * num)
```

6. Check if a Character is Vowel or Consonant

```
ch = input("Enter a single character: ").lower()
if ch in ['a', 'e', 'i', 'o', 'u']:
    print(ch, "is a vowel.")
else:
    print(ch, "is a consonant.")
```

7. Print all Prime Numbers in a Range

```
lower = int(input("Enter lower range: "))
```

```
upper = int(input("Enter upper range: "))


for num in range(lower, upper + 1):
   if num > 1:
      for i in range(2, num):
         if (num % i) == 0:
            break
      else:
         print(num, "is a prime number")
```

List Operations

1. Creating and Printing a List

python

```
fruits = ["apple", "banana", "cherry"]
print(fruits)  # Output: ['apple', 'banana', 'cherry']
```

2. Accessing List Elements

python

```
print(fruits[0])  # apple (first element)
print(fruits[-1])  # cherry (last element)
```

3. Adding Items to a List

python

```
fruits.append("orange")
print(fruits)  # ['apple', 'banana', 'cherry', 'orange']
```

4. Inserting an Item at a Specific Position

python

```
fruits.insert(1, "blueberry")
print(fruits)  # ['apple', 'blueberry', 'banana', 'cherry', 'orange']
```

5. Removing Items from a List

python

```
fruits.remove("banana")
print(fruits)  # ['apple', 'blueberry', 'cherry', 'orange']
```

6. Slicing a List

python

```
print(fruits[1:3])  # ['blueberry', 'cherry']
```

7. Iterating Over a List

python

```
for fruit in fruits:
    print(fruit)
```

8. List Comprehension to Create a New List

python

```
numbers = [1, 2, 3, 4, 5]
squares = [x**2 for x in numbers]
print(squares)  # [1, 4, 9, 16, 25]
```

9. Sorting a List

python

```
numbers = [5, 2, 9, 1]
numbers.sort()
print(numbers)  # [1, 2, 5, 9]
```

10. Copying a List

python

```
copy_fruits = fruits.copy()
print(copy_fruits)
```

Set Operations

1. Creating a Set and Printing

python

```
fruits = {"apple", "banana", "cherry"}
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

2. Adding Elements to a Set

python

```
fruits.add("orange")
print(fruits)  # Output includes 'orange'
```

3. Removing Elements from a Set

python

```
fruits.remove("banana")
print(fruits)  # 'banana' is removed
```

4. Set Union

python

```
a = {1, 2, 3}
b = {3, 4, 5}
print(a.union(b))  # {1, 2, 3, 4, 5}
# or using |
print(a | b)  # {1, 2, 3, 4, 5}
```

5. Set Intersection

python

```
a = {1, 2, 3, 4}
b = {3, 4, 5}
print(a.intersection(b))  # {3, 4}
# or using &
print(a & b)  # {3, 4}
```

6. Set Difference

python

```
a = {1, 2, 3}
b = {2, 3, 4}
print(a.difference(b))  # {1}
# or using -
print(a - b)  # {1}
```

7. Symmetric Difference

Elements in either a or b but not both.

python

```
a = {1, 2, 3}
b = {2, 3, 4}
print(a.symmetric_difference(b))  # {1, 4}
# or using ^
print(a ^ b)  # {1, 4}
```

8. Set Comprehension Example

python

```
numbers = [1, 2, 2, 3, 4, 4, 5]
unique_squares = {x**2 for x in numbers}
print(unique_squares)  # {1, 4, 9, 16, 25}
```

Tuple Operations

1. Creating a Tuple

python

```
tup = (1, 4.5, 'r', 'p', [3, 4], (5.4, 76), 9.0, 13)
```

2. Accessing Elements in a Tuple

python

```
print(tup[0])      # First element: 1
print(tup[-1])     # Last element: 13
print(tup[2:5])    # Slicing: ('r', 'p', [3, 4])
print(tup[::-1])   # Reversed tuple
```

3. Tuple Concatenation and Multiplication

```
tup1 = (1, 2, 3)
tup2 = ('a', 'b')
print(tup1 + tup2)   # Concatenation: (1, 2, 3, 'a', 'b')
print(tup1 * 3)     # Repetition: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

4. Membership Test

python

```
tup = (1, 2, 'a', 5.9)
print('a' in tup)    # True
print(19 in tup)     # False
```

5. Iterating Over a Tuple

python

```
for item in tup:
   print(item)
```

6. Tuple Unpacking

python

```
my_tuple = ('apple', 'banana', 'cherry')
a, b, c = my_tuple
print(a, b, c)
```

7. Tuple Length and Functions

```
print(len(tup))     # Length of tuple
print(max((3, 1, 2)))   # Max element
print(min((3, 1, 2)))   # Min element
```

8. Converting List to Tuple

```
my_list = ["apple", "banana", "cherry"]
my_tuple = tuple(my_list)
print(my_tuple)
```

Dictionary Operations

1. Creating Dictionaries

Using curly braces {} with key-value pairs:

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Using dict() constructor:

```
my_dict = dict(name='Alice', age=25, city='New York')
```

2. Accessing Values

- Using square brackets:

```
print(my_dict['name'])  # Outputs: Alice
```

- Using get() method (returns None if key not found or default value):

```
print(my_dict.get('age'))  # Outputs: 25
print(my_dict.get('country', 'USA'))  # Outputs 'USA' if 'country' key not present
```

3. Modifying Dictionaries

- Add or update entries:

python

```
my_dict['email'] = 'alice@example.com'  # Adds new key-value
my_dict['age'] = 26                     # Updates existing value
```

- Remove entries:

python

```
my_dict.pop('city')       # Removes key 'city' and returns its value
my_dict.popitem()         # Removes and returns last inserted key-value pair as tuple
```

- Clear all entries:

python

```
my_dict.clear()           # Empties the dictionary
```

4. Common Dictionary Methods

Method	**Description**	**Example Output**
keys()	**Returns all keys**	**dict_keys(['name', 'age'])**
values()	**Returns all values**	**dict_values(['Alice', 25])**
items()	**Returns key-value pairs as tuples**	**dict_items([('name', 'Alice'), ('age', 25)])**
update(other_dict)	**Merges another dictionary into current**	**{'name':'Bob', 'age':30} after updating**

5. Iterating Through Dictionaries

```
for key in my_dict:
    print(key, my_dict[key])


# or using items()
for key, value in my_dict.items():
    print(key, "->", value)
```

6. Example Program

```
person = {
    'name': 'John',
    'age': 30,
    'city': 'Chicago'
}

# Adding a new key-value
person['email'] = 'john@example.com'

# Accessing data
print("Name:", person.get('name'))

# Looping through dictionary
for key, value in person.items():
    print(f"{key}: {value}")
```

```
# Removing 'city'
person.pop('city')
print("Updated dictionary:", person)
```

Functions Example programs

1. Simple Function Without Arguments

```
def greet():
    print("Hello, World!")
greet()
```

2. Function with Parameters and Arguments

```
def greet(name):
    print(f"Hello, {name}!")
greet("Alice")
greet("Bob")
```

3. Function with Return Value

```
def add_numbers(a, b):
    return a + b
result = add_numbers(5, 7)
print("Sum:", result)
```

4. Function with Default Parameter Values

```
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()
greet("Alice")
```

5. Function with Variable Number of Arguments (*args)

```
def multiply_all(*args):
    result = 1
    for num in args:
        result *= num
    return result

print(multiply_all(1, 2, 3))      # Outputs: 6
print(multiply_all(4, 5, 6, 7))   # Outputs: 840
```

6. Recursive Function (Factorial)

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5))   # Outputs: 120
```

7. Function with Keyword Arguments

```
def display_info(name, age):
    print(f"Name: {name}, Age: {age}")


display_info(age=30, name="John")
```