SAPTHAGIRI NPS
UNIVERSITY

No.14/5, Chikkasandra, Hesaraghatta Main Road
Bangalore – 560057

## Department of Computer Science and Engineering

# Python Programming Laboratory Manual

**Subject Code –24BTPHY204/24BTELY205**

**Semester – II**

**Academic Year**

**2024-2025**

# SAPTHAGIRI NPS UNIVERSITY, BENGALURU-57

## SCHOOL OF COMUTER SCIENCE AND ENGINEERING

## LABORATORY CERTIFICATE

This is to certify that Mr. / Ms ________________________________________

has satisfactorily completed the course of Experiments in practical __________________ prescribed by the Department during the year __________________ .

Name of the Candidate : _____________________________

SRN No. : _______________________ Semester : _____________________

*Signature of the Staff in-charge :*

| Marks | |
|---|---|
| Max. Marks | Obtained |
| | |

Date :

# Python Programming Laboratory

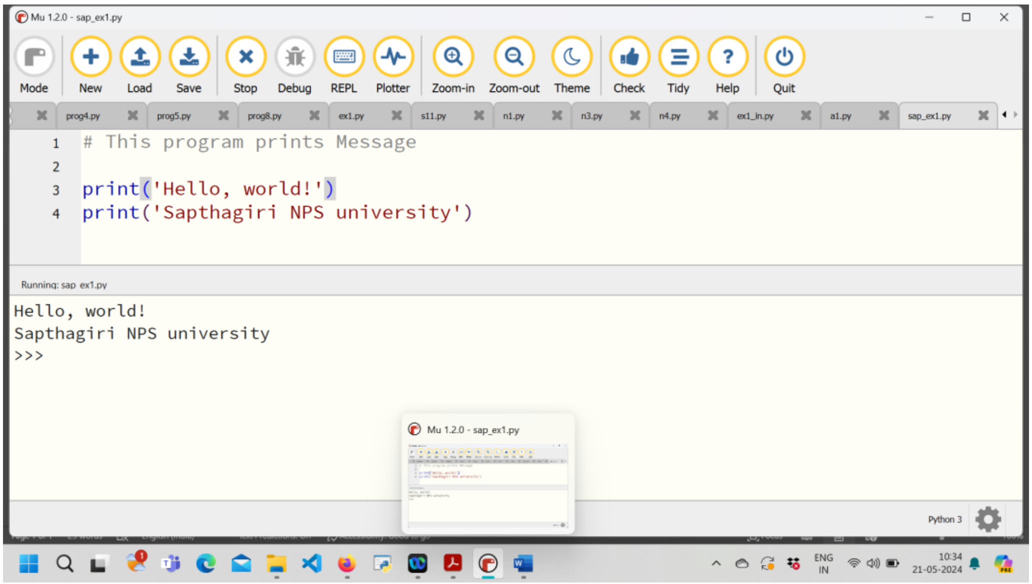
| Exp. No. | Date | Experiment Title |
|---|---|---|
| 1. | | Write simple Python program to display message on screen. |
| 2. | | Write simple Python program using operators:<br>a) Arithmetic Operators b) Logical Operators c) Bitwise Operators |
| 3. | | Write Python program to perform following operations on Tuples:<br>a) Create Set b) Access Set elements c) Update Set d) Delete Set |
| 4. | | Develop user defined Python function for given problem:<br>a) Function with minimum 2 arguments<br>b) Function returning values |
| 5. | | Write a program to double a given number and add two numbers using lambda()? |
| 6. | | Write a program for map() function to double all the items in the list? |
| 7. | | Demonstrate a python code to implement abnormal termination? |
| 8. | | Write a python program to write the content “hi python programming” for the existing file. |
| 9. | | Write a python program to display a particular month of a year using calendar module. |
| 10. | | Write a Regular Expression to represent all 10-digit mobile numbers. Rules: 1. Every number should contain exactly 10 digits. 2. The first digit should be 7 or 8 or 9 Write a Python Program to check whether the given number is valid mobile no. or not? |
| 11. | | Write a Regular Expression to represent all RGM language (Your own language) identifiers.<br>Rules: 1. The allowed characters are a-z, A-Z, 0-9, #. 2. The first character should be a lowercase alphabet symbol from a to k. 3. The second character should be a digit divisible by 3. 4. The length of identifier should be at least 2.<br>Write a python program to check whether the given string is RGM language identifier or not? |
| 12. | | Implement the following Searching and Sorting techniques in Python by using functions.<br>i) Linear Search ii) Binary Search iii) Selection Sort iv) Bubble Sort v) Insertion vi) Merge Sort viii) Quick Sort |
| 13. | | Write a Python program to return multiple values at a time using a return statement. |
| 14. | | Write a program in Python to demonstrate following operations:<br>a) Method overloading b) Method overriding |
| 15. | | Write a program in Python to handle user defined exception for given problem. Problem: You want to check the strength of a password. If the password is too short (less than 6 characters), raise a WeakPasswordError. If the password does not meet the necessary criteria (e.g., missing numbers or special characters), raise another user-defined exception. |

## 1. Write simple Python program to display message on screen.

```
# This program prints Message
print('Hello, world!')
print('Sapthagiri NPS university')
```

**Output:**

```
Hello, world!
Sapthagiri NPS university
```



Mu 1.2.0 - sap_ex1.py

```
# This program prints Message

print('Hello, world!')
print('Sapthagiri NPS university')
```

Running: sap_ex1.py

```
Hello, world!
Sapthagiri NPS university
>>>
```

## 1. Write simple Python program using operators:

**a) Arithmetic Operators b) Logical Operators c) Bitwise Operators**

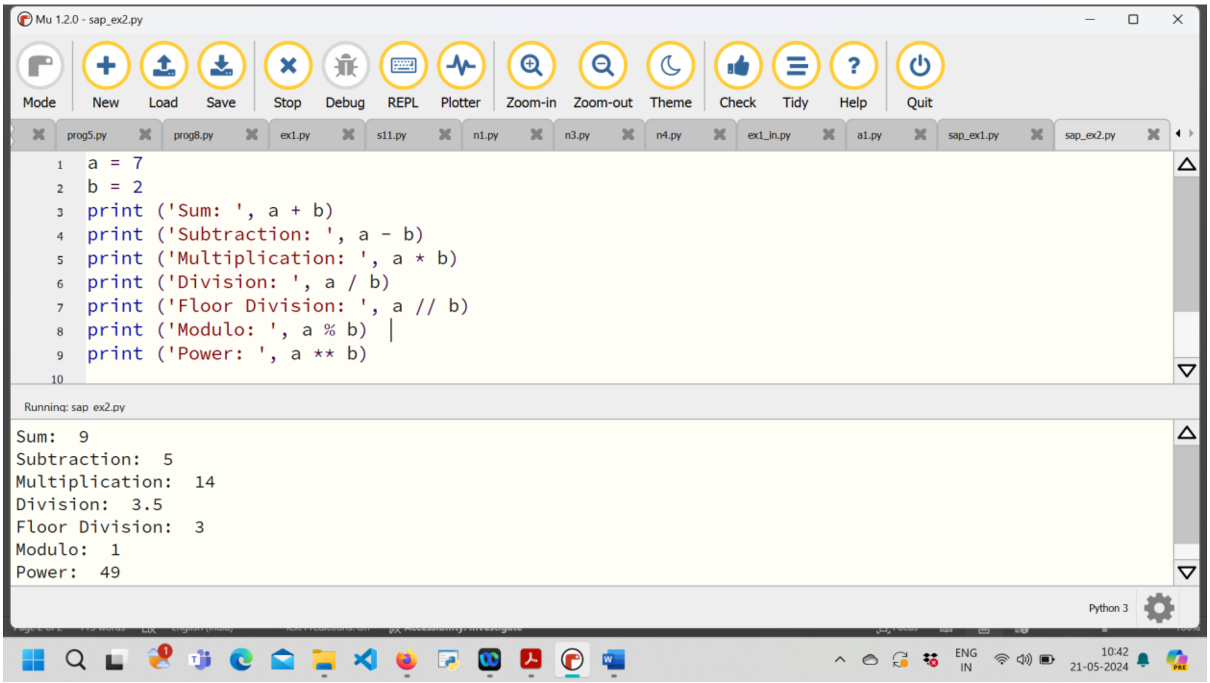
### a) Arithmetic Operators

```
a = 7
b = 2
print ('Sum: ', a + b)
print ('Subtraction: ', a - b)
print ('Multiplication: ', a * b)
print ('Division: ', a / b)
print ('Floor Division: ', a // b)
print ('Modulo: ', a % b)
print ('Power: ', a ** b)
```

**OUTPUT:**

```
Sum: 9
Subtraction: 5
```

```
Multiplication: 14
Division: 3.5
Floor Division: 3
Modulo: 1
Power: 49
```



```python
a = 7
b = 2
print ('Sum: ', a + b)
print ('Subtraction: ', a - b)
print ('Multiplication: ', a * b)
print ('Division: ', a / b)
print ('Floor Division: ', a // b)
print ('Modulo: ', a % b)
print ('Power: ', a ** b)
```

Running: sap_ex2.py

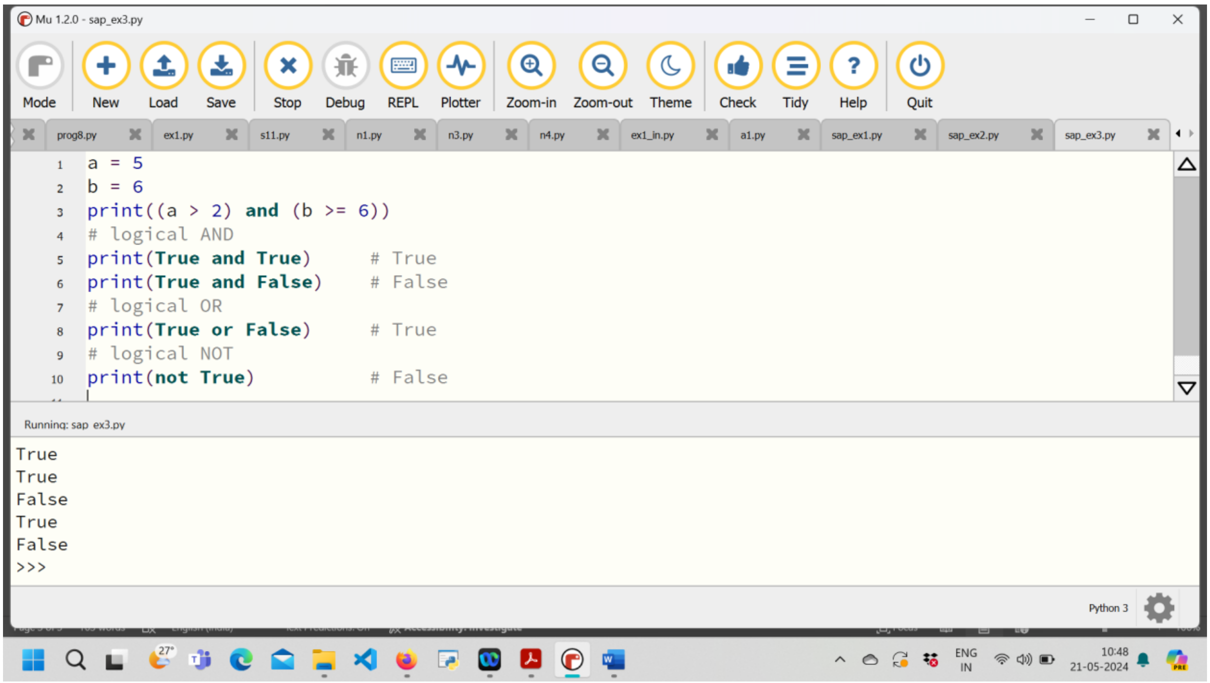
```
Sum:  9
Subtraction:  5
Multiplication:  14
Division:  3.5
Floor Division:  3
Modulo:  1
Power:  49
```

## b) Logical Operators

```python
a = 5
b = 6
print((a > 2) and (b >= 6))
# logical AND
print(True and True)     # True
print(True and False)    # False
# logical OR
print(True or False)     # True
# logical NOT
print(not True)          # False
```

**OUTPUT:**

```
True
True
False
True
False
```



```
a = 5
b = 6
print((a > 2) and (b >= 6))
# logical AND
print(True and True)      # True
print(True and False)     # False
# logical OR
print(True or False)      # True
# logical NOT
print(not True)           # False
```

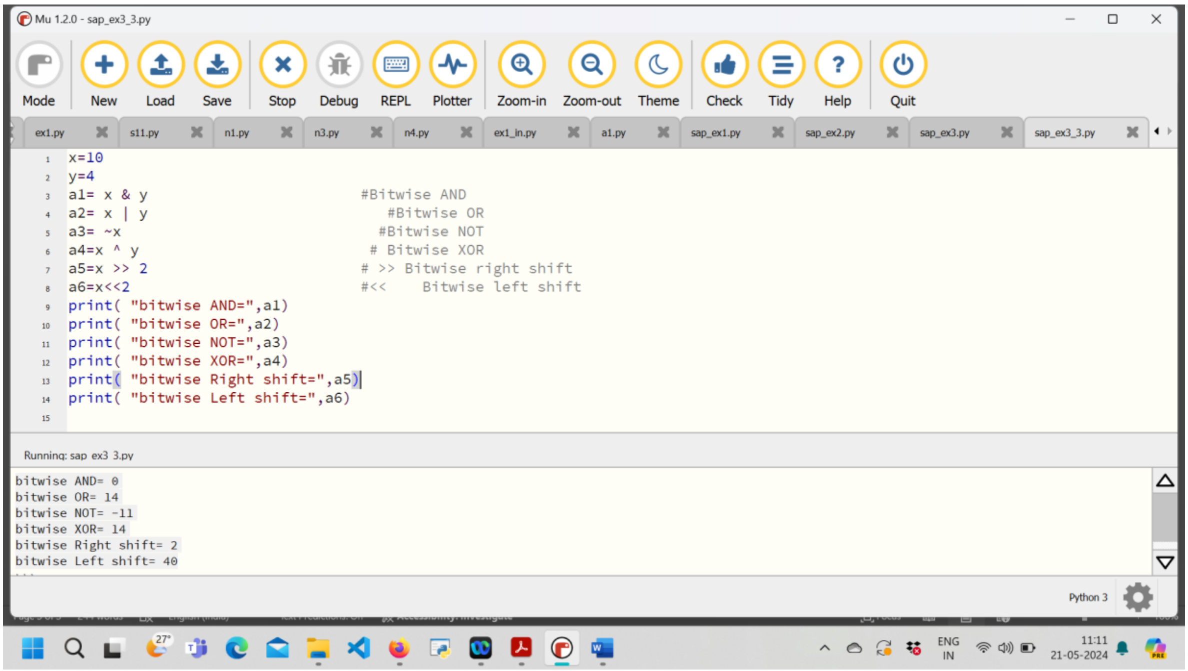
```
True
True
False
True
False
>>>
```

## C) Bitwise operator

```
x=10
y=4
a1= x & y                   #Bitwise AND
a2= x | y                   #Bitwise OR
a3= ~x                      #Bitwise NOT
a4=x ^ y                    # Bitwise XOR
a5=x >> 2                   # >> Bitwise right shift
a6=x<<2                     #<<Bitwise left shift
print( "bitwise AND=",a1)
print( "bitwise OR=",a2)
print( "bitwise NOT=",a3)
print( "bitwise XOR=",a4)
print( "bitwise Right shift=",a5)
print( "bitwise Left shift=",a6)
```

**OUTPUT:**

```
bitwise AND= 0
bitwise OR= 14
bitwise NOT= -11
bitwise XOR= 14
bitwise Right shift= 2
bitwise Left shift= 40
```



```
x=10
y=4
a1= x & y                          #Bitwise AND
a2= x | y                             #Bitwise OR
a3= ~x                               #Bitwise NOT
a4=x ^ y                            # Bitwise XOR
a5=x >> 2                          # >> Bitwise right shift
a6=x<<2                            #<<    Bitwise left shift
print( "bitwise AND=",a1)
print( "bitwise OR=",a2)
print( "bitwise NOT=",a3)
print( "bitwise XOR=",a4)
print( "bitwise Right shift=",a5)
print( "bitwise Left shift=",a6)
```

Running: sap ex3 3.py

```
bitwise AND= 0
bitwise OR= 14
bitwise NOT= -11
bitwise XOR= 14
bitwise Right shift= 2
bitwise Left shift= 40
```

## 2. Write Python program to perform following operations on Tuples: a) Create Set b) Access Set elements c) Update Set d) Delete Set

```
a) Create Set
my_set = {1, 2, 3, 4, 5}
print("Original Set:", my_set)

# b) Access Set elements
# We can iterate over the set or check if an element exists
for element in my_set:
print("Element:", element)

# Check if an element is in the set
print("Is 3 in the set?", 3 in my_set)

# c) Update Set
# Adding an element
my_set.add(6)
print("Set after adding 6:", my_set)

# Adding multiple elements
my_set.update([7, 8, 9])
print("Set after adding [7, 8, 9]:", my_set)

# Removing an element (if exists)
my_set.discard(2)
print("Set after discarding 2:", my_set)

# d) Delete Set
# Clearing all elements from the set
```

```
my_set.clear()
print("Set after clearing all elements:", my_set)

# Deleting the set
del my_set
```

**OUTPUT:**

```
Original Set: {1, 2, 3, 4, 5}
Element: 1
Element: 2
Element: 3
Element: 4
Element: 5
Is 3 in the set? True
Set after adding 6: {1, 2, 3, 4, 5, 6}
Set after adding [7, 8, 9]: {1, 2, 3, 4, 5, 6, 7, 8, 9}
Set after discarding 2: {1, 3, 4, 5, 6, 7, 8, 9}
Set after clearing all elements: set()
```

## 3. Develop user defined Python function for given problem:
## a) Function with minimum 2 arguments b) Function returning values.

```
def calculate_rectangle_properties(length, width):

    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter

# Example usage
length = 5
width = 3
area, perimeter = calculate_rectangle_properties(length, width)
print(f"Area: {area}, Perimeter: {perimeter}")
```

**OUTPUT:**

```
Area: 15, Perimeter: 16
```

**Note:**

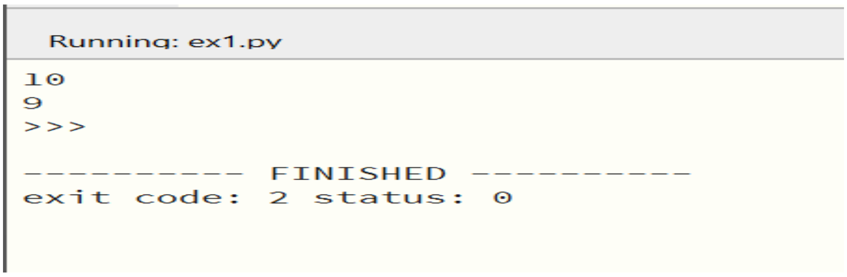
1. **f-string**: The `f` before the string denotes an f-string (formatted string literal). F-strings provide a way to embed expressions inside string literals, using curly braces `{}`.
2. **Expressions inside braces**: Inside the curly braces, Python expressions can be evaluated and their values inserted into the string.

4. **Write a program to double a given number and add two numbers using lambda()?**

**Program:**

```
double = lambda x:2*x
print(double(5))
add = lambda x,y:x+y
print(add(5,4))
```

**OUTPUT:**



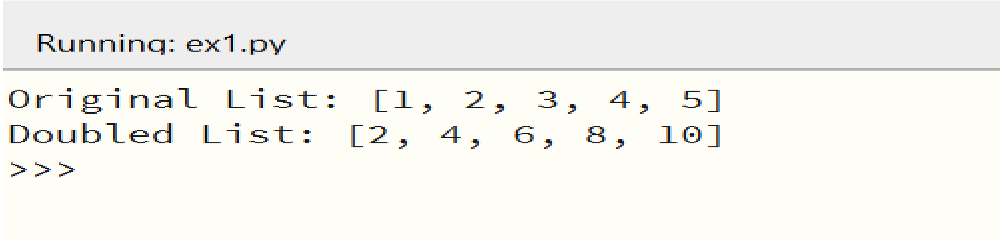
Running: ex1.py

```
10
9
>>>

----------- FINISHED -----------
exit code: 2 status: 0
```

**6. Write a program for map() function to double all the items in the list?**

```
# Function to double a number
def double(num):
return num * 2

# List of numbers
numbers = [1, 2, 3, 4, 5]

# Using map() to apply the double function to each element
doubled_numbers = list(map(double, numbers))

# Print the result
print("Original List:", numbers)
print ("Doubled List:", doubled_numbers)
```

**OUTPUT:**



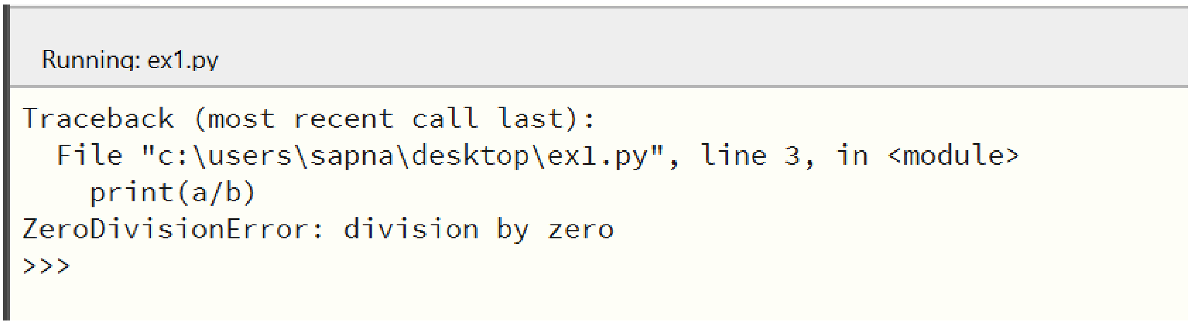
Running: ex1.py

```
Original List: [1, 2, 3, 4, 5]
Doubled List: [2, 4, 6, 8, 10]
>>>
```

7. **Demonstrate a python code to implement abnormal termination?**

```
a=5
b=0
print(a/b)
print("bye")
```

**OUTPUT:**



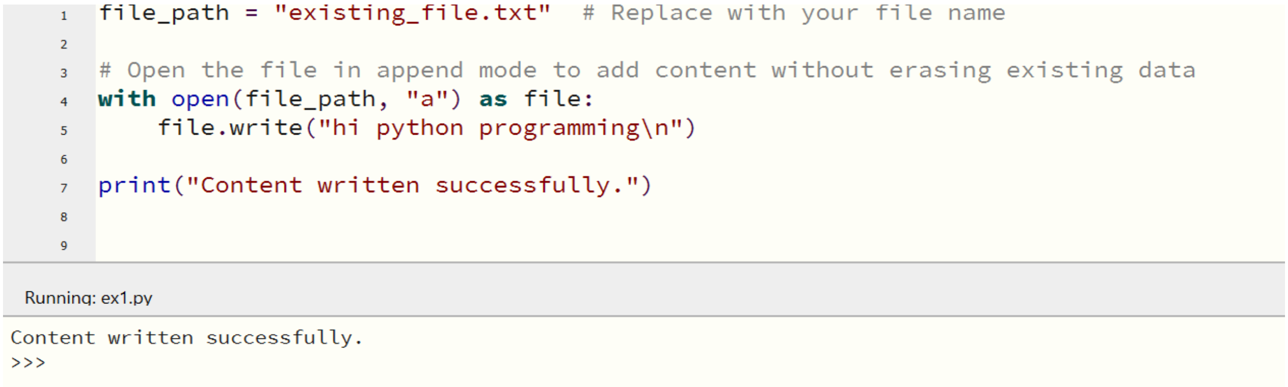
Running: ex1.py

```
Traceback (most recent call last):
  File "c:\users\sapna\desktop\ex1.py", line 3, in <module>
    print(a/b)
ZeroDivisionError: division by zero
>>>
```

## 8. Write a python program to write the content “hi python programming” for the existing file.

```
file_path = "existing_file.txt"  # Replace with your file name

# Open the file in append mode to add content without erasing existing data
with open(file_path, "a") as file:
    file.write("hi python programming\n")

 print ("Content written successfully.")
```

**OUTPUT:**



```
file_path = "existing_file.txt"  # Replace with your file name

# Open the file in append mode to add content without erasing existing data
with open(file_path, "a") as file:
    file.write("hi python programming\n")

print("Content written successfully.")


```

Running: ex1.py

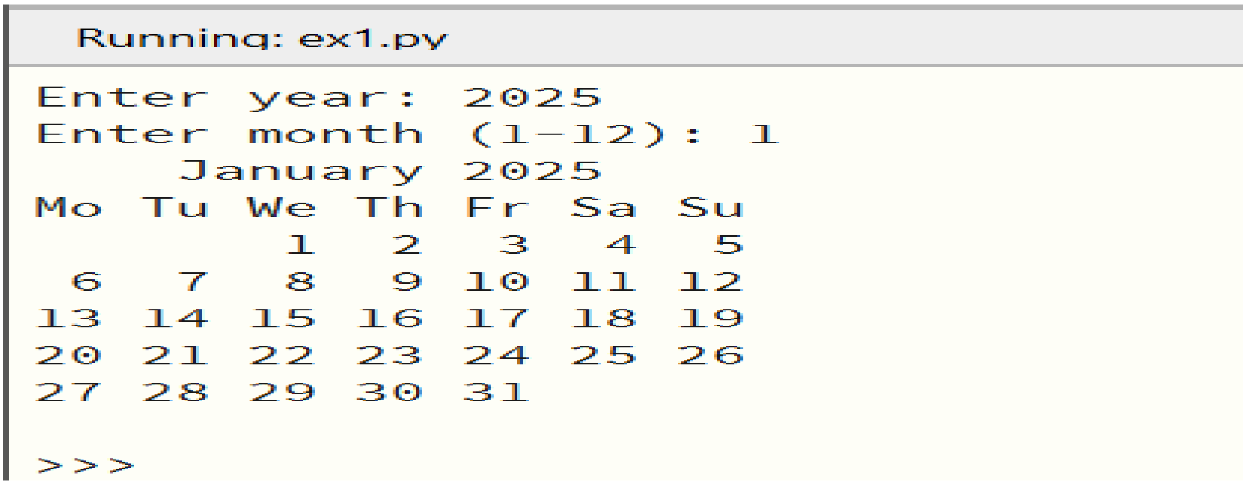
```
Content written successfully.
>>>
```

## 9. Write a python program to display a particular month of a year using calendar module.

```
import calendar

year = int(input("Enter year: "))
```

```
month = int(input("Enter month (1-12): "))

# Display the calendar for the specified month and year
print(calendar.month(year, month))
```

**OUTPUT:**



```
  Running: ex1.py
Enter year: 2025
Enter month (1-12): 1
    January 2025
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

>>>
```

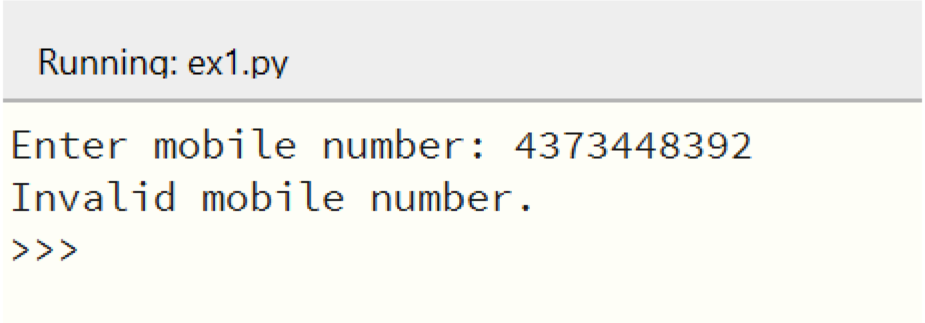
## 10. Write a Regular Expression to represent all 10-digit mobile numbers.

**Rules: 1. Every number should contain exactly 10 digits.**
**2. The first digit should be 7 or 8 or 9**

**Write a Python Program to check whether the given number is valid mobile no. or not?**

```
import re

def is_valid_mobile_number(number):
    # Regular expression to check the mobile number
    pattern = r"^[789]\d{9}$"

    # If the number matches the pattern
    if re.match(pattern, number):
        return True
    else:
        return False

# Input from the user
mobile_number = input("Enter mobile number: ")

# Check if the mobile number is valid
if is_valid_mobile_number(mobile_number):
     print("Valid mobile number.")
else:
    print("Invalid mobile number.")
```

**OUTPUT:**



Running: ex1.py

```
Enter mobile number: 4373448392
Invalid mobile number.
>>>
```

**11. Write a Regular Expression to represent all RGM language (Your own language) identifiers.**

**Rules: 1. The allowed characters are a-z, A-Z, 0-9, #.**  
**2. The first character should be a lowercase alphabet symbol from a to k.**  
**3. The second character should be a digit divisible by 3.**  
**4. The length of identifier should be at least 2.**

**Write a python program to check whether the given string is RGM language identifier or not?**

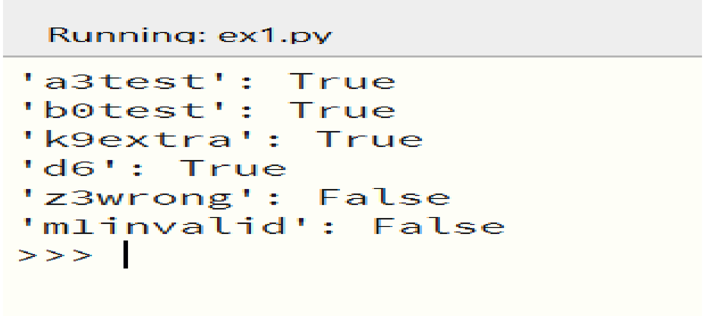
**Explanation:**

1. **Regular Expression `^[a-k][0369].*`:**
   - `^[a-k]` ensures the first character is a lowercase letter from 'a' to 'k'.
   - `[0369]` ensures the second character is a digit divisible by 3 (0, 3, 6, or 9).
   - `.*` allows any additional characters after the first two characters.
2. **Function `is_rgm_identifier(identifier)`:**
   - It checks if the input string matches the regular expression pattern.
   - If it matches, it returns `True`, meaning the string is a valid identifier; otherwise, it returns `False`.

```
import re

def is_rgm_identifier(identifier):
    # Regular Expression for the RGM language identifier
    pattern = r'^[a-k][0369].*'

    # Check if the given identifier matches the pattern
    if re.match(pattern, identifier):
        return True
    else:
        return False

# Test the function
test_strings = ["a3test", "b0test", "k9extra", "d6", "z3wrong", "m1invalid"]
```

```
for test in test_strings:
    print(f"'{test}': {is_rgm_identifier(test)}")
```

OUTPUT:



```
Running: ex1.py
'a3test': True
'b0test': True
'k9extra': True
'd6': True
'z3wrong': False
'm1invalid': False
>>> |
```

**12. Write a Regular Expression to represent all RGM language (Your own language) identifiers.**

**Rules: 1. The allowed characters are a-z, A-Z, 0-9,**
**2. The first character should be a lowercase alphabet symbol from a to k.**
**3. The second character should be a digit divisible by 3.**
**4. The length of identifier should be at least 2.**

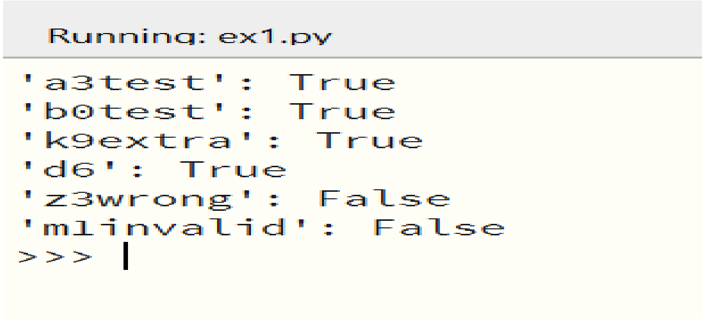
**Write a python program to check whether the given string is RGM language identifier or not?**

## Explanation:

1. **Regular Expression `^[a-k][0369].*`:**
   - `^[a-k]` ensures the first character is a lowercase letter from 'a' to 'k'.
   - `[0369]` ensures the second character is a digit divisible by 3 (0, 3, 6, or 9).
   - `.*` allows any additional characters after the first two characters.
2. **Function `is_rgm_identifier(identifier)`:**
   - It checks if the input string matches the regular expression pattern.
   - If it matches, it returns `True`, meaning the string is a valid identifier; otherwise, it returns `False`.

```
import re

def is_rgm_identifier(identifier):
    # Regular Expression for the RGM language identifier
    pattern = r'^[a-k][0369].*'

    # Check if the given identifier matches the pattern
    if re.match(pattern, identifier):
        return True
    else:
        return False

# Test the function
test_strings = ["a3test", "b0test", "k9extra", "d6", "z3wrong", "m1invalid"]
for test in test_strings:
 print(f"'{test}': {is_rgm_identifier(test)}")
```

**OUTPUT:**



```
Running: ex1.py
'a3test': True
'b0test': True
'k9extra': True
'd6': True
'z3wrong': False
'm1invalid': False
>>> |
```

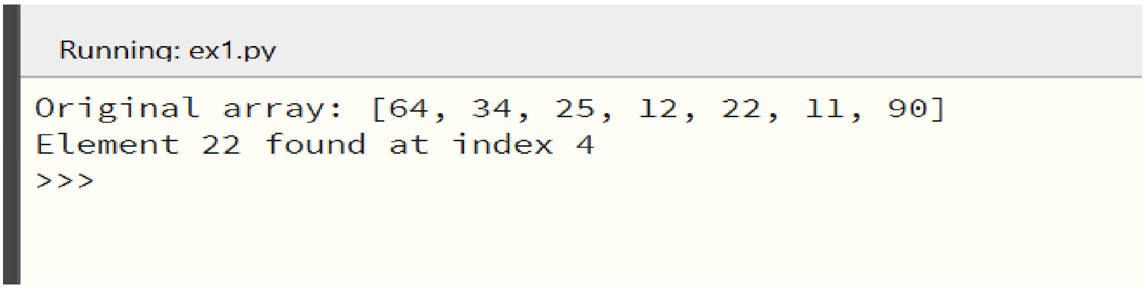
**13. Implement the following Searching and Sorting techniques in Python by using functions.**

**i) Linear Search ii) Binary Search iii) Selection Sort iv) Bubble Sort**
**v) Insertion  vi) Merge Sort   viii) Quick Sort**

**i)Linear Search**

```
def linear_search(arr, target):
    """Performs linear search on an array."""
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Return the index if the element is found
    return -1  # Return -1 if the element is not found
# Example usage:
arr = [64, 34, 25, 12, 22, 11, 90]
print("Original array:", arr)
# Searching for an element
target = 22
index = linear_search(arr, target)
if index != -1:
    print(f"Element {target} found at index {index}")
else:
    print(f"Element {target} not found in the array")
```

## OUTPUT:



```
Running: ex1.py
Original array: [64, 34, 25, 12, 22, 11, 90]
Element 22 found at index 4
>>>
```

## ii) Binary search

## Explanation:

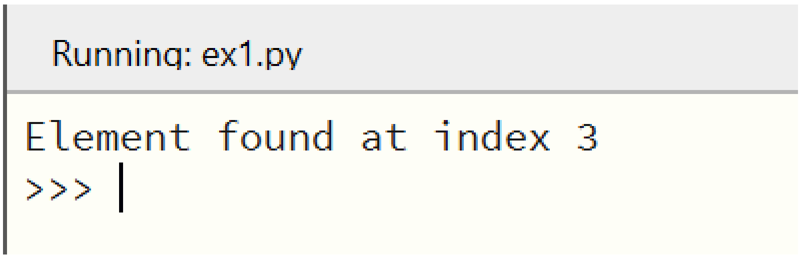
1. **Initialize** `left` to 0 and `right` to the last index of `arr`.
2. **Loop** while `left` is less than or equal to `right`:
   - Calculate the **middle index**.
   - If `arr[mid]` is equal to `target`, return `mid`.
   - If `arr[mid]` is **less than** `target`, move `left` to `mid + 1` (search right half).
   - If `arr[mid]` is **greater than** `target`, move `right` to `mid - 1` (search left half).
3. If the loop exits without finding the target, return `-1`.

## Program:

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2  # Avoids overflow in some languages

        if arr[mid] == target:
            return mid  # Target found
        elif arr[mid] < target:
            left = mid + 1  # Search in the right half
        else:
            right = mid - 1  # Search in the left half

    return -1  # Target not found

# Example usage
arr = [1, 3, 5, 7, 9, 11, 13, 15]
```

```
target = 7
result = binary_search(arr, target)

if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found")
```

**OUTPUT:**



```
Running: ex1.py

Element found at index 3
>>> 
```

### iii) Selection Sort

## Explanation:

Selection Sort is a simple sorting algorithm that works by repeatedly finding the smallest element from the unsorted part of the list and moving it to the beginning.

**Steps:**

1. Start with the first element as the minimum.
2. Scan the rest of the array to find the smallest element.
3. Swap the smallest element with the first element.
4. Move to the next position and repeat until the array is sorted.

**Algorithm Breakdown (Step-by-Step)**

For an array [64, 25, 12, 22, 11]:

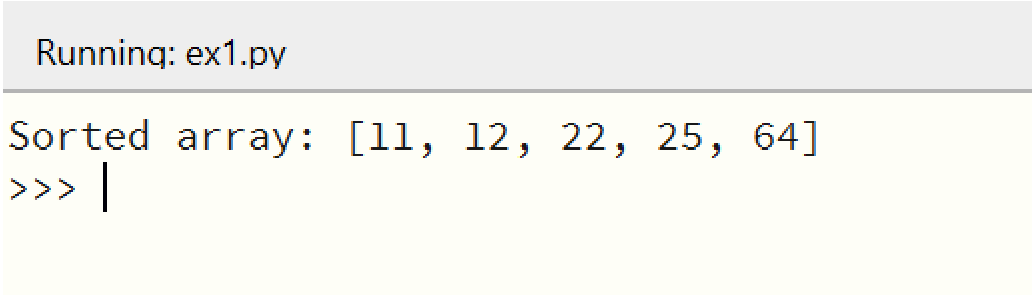
1. **First Pass (i = 0)**:
   - Find the minimum: **11**
   - Swap it with **64** → [11, 25, 12, 22, 64]
2. **Second Pass (i = 1)**:
   - Find the minimum: **12**
   - Swap it with **25** → [11, 12, 25, 22, 64]
3. **Third Pass (i = 2)**:
   - Find the minimum: **22**
   - Swap it with **25** → [11, 12, 22, 25, 64]
4. **Fourth Pass (i = 3)**:
   - Find the minimum: **25**
   - No swap needed → [11, 12, 22, 25, 64]

5. **Array is Sorted!**

**Program:**

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]  # Swap elements

# Example usage
arr = [64, 25, 12, 22, 11]
selection_sort(arr)
print("Sorted array:", arr)
```

**OUTPUT:**



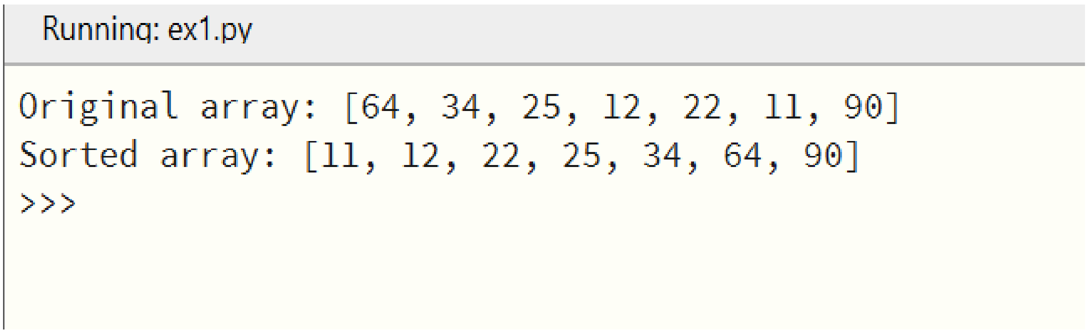
```
Running: ex1.py

Sorted array: [11, 12, 22, 25, 64]
>>> |
```

## iv) Bubble Sort

```
def bubble_sort(arr):
    """Performs bubble sort on an array."""
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  # Swap elements


# Example usage:
arr = [64, 34, 25, 12, 22, 11, 90]
print("Original array:", arr)

# Sorting the array
bubble_sort(arr)
```

```
        print("Sorted array:", arr)
```

**OUTPUT:**



Running: ex1.py

```
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array: [11, 12, 22, 25, 34, 64, 90]
>>>
```

## v) Insertion Sort

### Algorithm Steps

1. Start with the second element (`index = 1`), considering the first element as sorted.
2. Pick the current element (`key`) and compare it with elements in the sorted portion (to its left).
3. Shift elements greater than `key` one position to the right.
4. Insert `key` in its correct position.
5. Repeat this process for all elements until the list is sorted.

### Example Walkthrough

Given an array: `[64, 25, 12, 22, 11]`

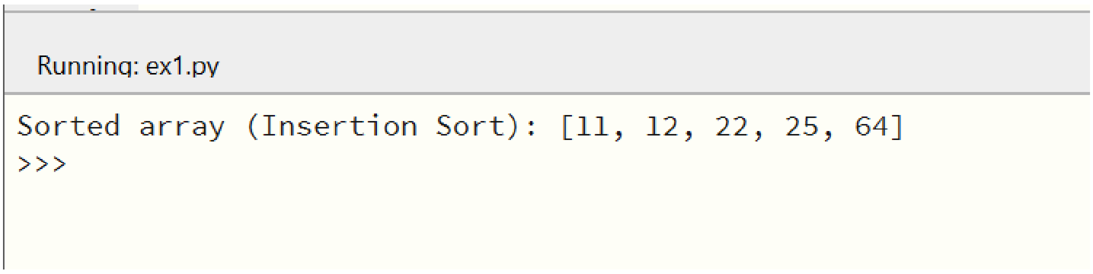
1. **Iteration 1** (`key = 25`):
   - Compare `25` with `64` → Shift `64` → `[64, 64, 12, 22, 11]`
   - Insert `25` → `[25, 64, 12, 22, 11]`
2. **Iteration 2** (`key = 12`):
   - Compare `12` with `64` → Shift `64`
   - Compare `12` with `25` → Shift `25`
   - Insert `12` → `[12, 25, 64, 22, 11]`
3. **Iteration 3** (`key = 22`):
   - Compare `22` with `64` → Shift `64`
   - Compare `22` with `25` → Shift `25`
   - Insert `22` → `[12, 22, 25, 64, 11]`
4. **Iteration 4** (`key = 11`):
   - Compare `11` with `64` → Shift `64`
   - Compare `11` with `25` → Shift `25`
   - Compare `11` with `22` → Shift `22`

  - Compare `11` with `12` → Shift `12`
  - Insert `11` → `[11, 12, 22, 25, 64]`

## Program:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# Example usage
arr = [64, 25, 12, 22, 11]
insertion_sort(arr)
print("Sorted array (Insertion Sort):", arr)
```

## OUTPUT:



```
Running: ex1.py

Sorted array (Insertion Sort): [11, 12, 22, 25, 64]
>>>
```

## vi) Merge Sort

## Explanation of Merge Sort Algorithm

Merge Sort is a **divide-and-conquer** algorithm that works by recursively breaking down the input array into smaller sub-arrays, sorting them, and then merging them back together. Here's a detailed breakdown of how it works:

### 1. Divide Step (Recursive Splitting):

The array is recursively divided into two halves until each sub-array contains only one element (since a single element is inherently sorted).

For example:

- Given the array `[38, 27, 43, 3, 9, 82, 10]`, it is split into two halves:
  - Left half: `[38, 27, 43]`
  - Right half: `[3, 9, 82, 10]`

- This process continues recursively, breaking each half into smaller parts, until we reach sub-arrays of size 1. For instance:
    - `[38, 27, 43]` becomes `[38]`, `[27]`, `[43]`
    - `[3, 9, 82, 10]` becomes `[3]`, `[9]`, `[82]`, `[10]`

## 2. Merge Step (Combining Sorted Arrays):

After the array has been broken down into smaller arrays, the algorithm **merges** them back together, but in sorted order. This is where the "merge" function comes into play.

- The two halves are compared element by element, and the smaller element is placed into the sorted array. This continues until all elements from both halves are merged into a single sorted array.
- If there are leftover elements in either half, they are appended directly to the result since they are already sorted.

## Example of Merging:

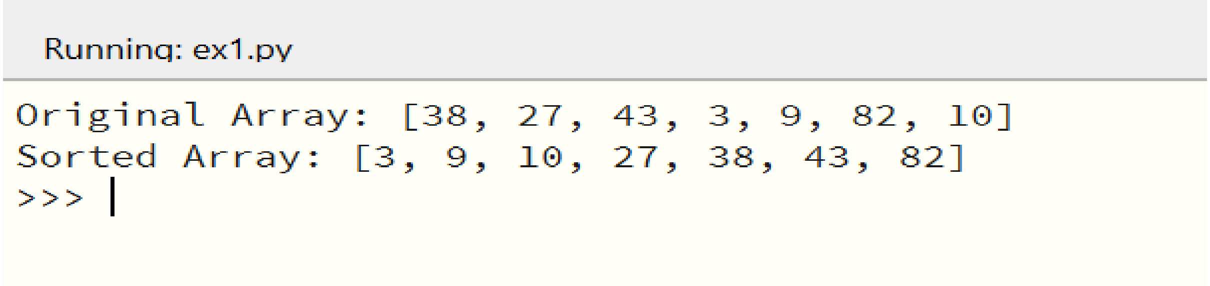
For the sub-arrays `[38]`, `[27]`, and `[43]`, merge them as follows:

- Compare `38` and `27`. Since `27` is smaller, it goes into the sorted array.
- Now compare `38` and `43`. Since `38` is smaller, it goes into the sorted array.
- The remaining element `43` is added to the sorted array, and the result is `[27, 38, 43]`.

Similarly, the merge process occurs for other parts of the array, eventually combining all the sorted sub-arrays into one sorted array.

```
# Merge Sort function
def merge_sort(arr):
    # If the array has only one element, it's already sorted
    if len(arr) <= 1:
        return arr

    # Split the array into two halves
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    # Recursively sort each half
    left_sorted = merge_sort(left_half)
    right_sorted = merge_sort(right_half)

    # Merge the two sorted halves
    return merge(left_sorted, right_sorted)

# Merge function to combine two sorted arrays
def merge(left, right):
    sorted_array = []
    i = j = 0
    # Compare elements from both halves and merge them in sorted order
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_array.append(left[i])
            i += 1
        else:
            sorted_array.append(right[j])
            j += 1

    # If there are any remaining elements in left or right, append them
    sorted_array.extend(left[i:])
    sorted_array.extend(right[j:])

    return sorted_array

# Example usage:
arr = [38, 27, 43, 3, 9, 82, 10]
print("Original Array:", arr)
sorted_arr = merge_sort(arr)
print("Sorted Array:", sorted_arr)
```

**OUTPUT:**



```
Running: ex1.py

Original Array: [38, 27, 43, 3, 9, 82, 10]
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
>>> 
```

## vii) Quick Sort

```
# function to find the partition position

def partition(array, low, high):


  # choose the rightmost element as pivot

  pivot = array[high]


  # pointer for greater element

  i = low - 1
  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j] <= pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # return the position from where partition is done
  return i + 1

# function to perform quicksort
def quickSort(array, low, high):
  if low < high:

    # find pivot element such that
    # element smaller than pivot are on the left
    # element greater than pivot are on the right
    pi = partition(array, low, high)

    # recursive call on the left of pivot
    quickSort(array, low, pi - 1)
    # recursive call on the right of pivot
    quickSort(array, pi + 1, high)
data = [8, 7, 2, 1, 0, 9, 6]
print("Unsorted Array")
print(data)
size = len(data)
quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

**OUTPUT:**

```
Unsorted Array
[8, 7, 2, 1, 0, 9, 6]
Sorted Array in Ascending Order:
[0, 1, 2, 6, 7, 8, 9]
```

## 13. Write a Python program to return multiple values at a time using a return statement.

```
    def calculate_values(a, b):
    sum_val = a + b
    product_val = a * b
    return sum_val, product_val

# Call the function
result1, result2 = calculate_values(5, 3)

print("Sum:", result1)
print("Product:", result2)
```

**OUTPUT:**

```
Sum: 8

Product: 15
```

## 14. Write a program in Python to demonstrate following operations:

a) Method overloading

b) Method overriding

### a) Method Overloading

```python
# First product method.
# Takes two argument and print their
# product


def product(a, b):
    p = a * b
    print(p)

# Second product method
# Takes three argument and print their
# product


def product(a, b, c):
    p = a * b*c
    print(p)

# Uncommenting the below line shows an error
# product(4, 5)


# This line will call the second product method
product(4, 5, 5)
```

**OUTPUT:**

```
100
```

### b)Method overriding

```python
# Python program to demonstrate
# calling the parent's class method
# inside the overridden method
class Parent():

    def show(self):
        print("Inside Parent")
```

```
class Child(Parent):

    def show(self):

        # Calling the parent's class
        # method
        Parent.show(self)
        print("Inside Child")

# Driver's code
obj = Child()
obj.show()
```

## OUTPUT:

```
Inside Parent
Inside Child
```

## 15. Write a program in Python to handle user defined exception for given problem.

**Problem: You want to check the strength of a password. If the password is too short (less than 6 characters), raise a WeakPasswordError. If the password does not meet the necessary criteria (e.g., missing numbers or special characters), raise another user-defined exception.**

```
import re

# Step 1: Define the custom exceptions
class WeakPasswordError(Exception):
    """Exception raised for passwords that are too short."""
    def __init__(self, message="Password is too short. Must be at least 6 characters long."):
        self.message = message
        super().__init__(self.message)

class PasswordMissingCriteriaError(Exception):
    """Exception raised for passwords that are missing required criteria (numbers or special characters)."""
    def __init__(self, message="Password must contain at least one number and one special character."):
        self.message = message
        super().__init__(self.message)

# Step 2: Function to check password strength
def check_password_strength(password):
    # Check if the password length is less than 6 characters
    if len(password) < 6:
        raise WeakPasswordError("Password is too short. Must be at least 6 characters long.")

    # Check if the password contains at least one number and one special character
    if not re.search(r'\d', password) or not re.search(r'[\W_]', password):
        raise PasswordMissingCriteriaError("Password must contain at least one number and one special character.")

    print("Password is strong!")

# Step 3: Main function to take user input and handle exceptions
def main():
    try:
        password = input("Enter your password: ")
        check_password_strength(password)
    except WeakPasswordError as e:
        print(f"Error: {e}")
    except PasswordMissingCriteriaError as e:
        print(f"Error: {e}")

# Run the program
if __name__ == "__main__":
    main()
```

## OUTPUT:

**Case 1: Password too short**

```
Enter your password: abc
Error: Password is too short. Must be at least 6 characters long.
```

**Case 2: Password missing  required creteria**

```
Enter your password: abc123
Error: Password must contain at least one number and one special character.
```

**Case 3: Strong Password**

```
Enter your password: abc@123
Password is strong!
```