1. **Write a Python program to calculate the area of a rectangle given its length and width.**

```
Def calculate_rectangle_area(length, width):
    Area = length * width
    Return area

Def main():
    Length = float(input("Enter the length of the rectangle: "))
    Width = float(input("Enter the width of the rectangle: "))

    Area = calculate_rectangle_area(length, width)
    Print("The area of the rectangle is:", area)

If __name__ == "__main__":
    Main()
```

2. **Write a program to convert miles to kilometers**

```
Def miles_to_kilometers(miles):
  # 1 mile is equal to 1.60934 kilometers
  Kilometers = miles * 1.60934
  Return kilometers

Def main():
  Miles = float(input("Enter the distance in miles: "))
  Kilometers = miles_to_kilometers(miles)
  Print(f"{miles} miles is equal to {kilometers:.2f} kilometers.")

If __name__ == "__main__":
  Main()
```

3. **Write a function to check if a given string is a palindrome**

```
Def is_palindrome(s):
  # Convert the string to lowercase and remove non-alphanumeric characters
  S = ''.join(char.lower() for char in s if char.isalnum())

  # Compare the string with its reverse
  Return s == s[::-1]
```

```
# Test the function
Def main():
  Test_strings = [“racecar”, “hello”, “A man, a plan, a canal, Panama!”, “Was it a car or a cat I saw?”]
  For test_string in test_strings:
    If is_palindrome(test_string):
      Print(f’”{test_string}” is a palindrome.’)
    Else:
      Print(f’”{test_string}” is not a palindrome.’)

If __name__ == “__main__”:
  Main()
```

4. **Write a Python program to find the second largest element in a list.**

```
Def find_second_largest(nums):
  If len(nums) < 2:
    Return “List should have at least two elements.”

  # Initialize the largest and second largest variables
  Largest = float(‘-inf’)
  Second_largest = float(‘-inf’)

  # Iterate through the list to find the largest and second largest elements
  For num in nums:
    If num > largest:
      Second_largest = largest
      Largest = num
    Elif num > second_largest and num != largest:
      Second_largest = num

  Return second_largest

# Test the function
Def main():
  Nums = [10, 5, 8, 20, 15]
  Second_largest = find_second_largest(nums)
  Print(“The second largest element in the list is:”, second_largest)
```

```
If __name__ == “__main__”:
   Main()
```

5. **Explain what indentation means in Python.**

In Python, indentation is used to define the structure and hierarchy of code blocks, such as loops, conditional statements, function definitions, and class definitions. Unlike many other programming languages that use braces {} or keywords like begin and end to denote code blocks, Python uses indentation to indicate the beginning and end of blocks of code.

Here are some key points about indentation in Python:

Indentation Level: Python code blocks are defined by increasing or decreasing the level of indentation relative to the surrounding code. Each level of indentation typically corresponds to four spaces, although tabs can also be used (although mixing tabs and spaces is discouraged). Consistency in indentation style is crucial for readability and to avoid syntax errors.

Code Blocks: Code blocks in Python, such as those within loops (for, while), conditional statements (if, elif, else), function definitions (def), and class definitions (class), are indicated by indentation. The lines of code within a block must have the same level of indentation.

Colon (☺ Usage: In Python, a colon (☺ is used to indicate the start of a code block, and the following lines with increased indentation are considered part of that block. For example, after an if statement, the code block that follows is executed only if the condition is true.

Whitespace Sensitivity: Python is whitespace-sensitive, meaning that the interpreter uses indentation to determine the structure of the code. Incorrect indentation can lead to syntax errors or unexpected behavior. It is essential to ensure consistent and proper indentation for the code to execute as intended.

No Curly Braces: Unlike many other programming languages like C, Java, or JavaScript, Python does not use curly braces {} to define code blocks. Instead, indentation serves this purpose. This makes Python code visually cleaner and more readable, but it also requires careful attention to indentation levels.

Best Practices: It is recommended to use a consistent indentation style throughout the codebase to enhance readability and maintainability. Most Python developers use four

spaces for each level of indentation, following the Python Enhancement Proposal (PEP) 8 style guide.

In summary, indentation in Python is a critical aspect of the language syntax, defining the structure and hierarchy of code blocks. It is used to denote the beginning and end of blocks of code, and consistent and proper indentation is essential for writing clear, readable, and error-free Python code.

6. **Write a program to perform set difference operation**

```
Def set_difference_using_operator(set1, set2):
  Return set1 – set2

Def set_difference_using_method(set1, set2):
  Return set1.difference(set2)

Def main():
  Set1 = {1, 2, 3, 4, 5}
  Set2 = {3, 4, 5, 6, 7}

  # Using the – operator
  Difference_operator = set_difference_using_operator(set1, set2)
  Print("Set difference using operator:", difference_operator)

  # Using the difference() method
  Difference_method = set_difference_using_method(set1, set2)
  Print("Set difference using method:", difference_method)

If __name__ == "__main__":
  Main()
```

7. **Write a Python program to print numbers from 1 to 10 using a while loop.**

```
  Def main():
# Initialize the counter
Num = 1
```

```
# Print numbers from 1 to 10 using a while loop
While num <= 10:
    Print(num)
    Num += 1

If __name__ == “__main__”:
    Main()
```

8. **Write a program to calculate the factorial of a number using a while loop.**

```
def calculate_factorial(num):
    factorial = 1
    while num > 0:
        factorial *= num
        num -= 1
    return factorial

def main():
    num = int(input("Enter a number: "))
    factorial = calculate_factorial(num)
    print(f"The factorial of {num} is: {factorial}")

if __name__ == "__main__":
    main()
```

9. **Write a Python program to check if a number is positive, negative, or zero using if-elif-else statements.**

```
Def check_number(num):
    “””
    Check if a number is positive, negative, or zero using if-elif-else statements.
    :param num: The number to be checked
    “””
    If num > 0:
```

```
    Print(num, “is positive.”)
  Elif num < 0:
    Print(num, “is negative.”)
  Else:
    Print(num, “is zero.”)

# Example usage:
If __name__ == “__main__”:
  Number = float(input(“Enter a number: “))
  Check_number(number)
```

10. Write a program to determine the largest among three numbers using conditional statements.

```
Def find_largest(num1, num2, num3):
  “””
  Determine the largest among three numbers using conditional statements.
  :param num1: First number
  :param num2: Second number
  :param num3: Third number
  :return: The largest number among the three
  “””
  If num1 >= num2 and num1 >= num3:
    Return num1
  Elif num2 >= num1 and num2 >= num3:
    Return num2
  Else:
    Return num3

# Example usage:
If __name__ == “__main__”:
  Number1 = float(input(“Enter the first number: “))
  Number2 = float(input(“Enter the second number: “))
  Number3 = float(input(“Enter the third number: “))

  Largest_number = find_largest(number1, number2, number3)
  Print(“The largest number among”, number1, “,”, number2, “, and”, number3, “is:”,
largest_number)
```

11. **Write a Python program to create a numpy array filled with ones of given shape**
Import numpy as np

```
Def create_ones_array(shape):
  “””
  Create a NumPy array filled with ones of given shape.
  :param shape: Tuple specifying the shape of the array
  :return: NumPy array filled with ones
  “””
  Return np.ones(shape)

# Example usage:
If __name__ == “__main__”:
  Shape = tuple(map(int, input(“Enter the shape of the array (separate dimensions by space): “).split()))
  Ones_array = create_ones_array(shape)
  Print(“Array filled with ones of shape”, shape, “:\n”, ones_array)
```

12. **Write a program to create a 2D numpy array initialized with random integers**
Import numpy as np

```
Def create_random_array(rows, cols, min_val, max_val):
  “””
  Create a 2D NumPy array initialized with random integers.
  :param rows: Number of rows in the array
  :param cols: Number of columns in the array
  :param min_val: Minimum value for the random integers
  :param max_val: Maximum value for the random integers
  :return: 2D NumPy array initialized with random integers
  “””
  Return np.random.randint(min_val, max_val+1, size=(rows, cols))

# Example usage:
If __name__ == “__main__”:
  Rows = int(input(“Enter the number of rows: “))
  Cols = int(input(“Enter the number of columns: “))
  Min_val = int(input(“Enter the minimum value for random integers: “))
  Max_val = int(input(“Enter the maximum value for random integers: “))
```

```
    Random_array = create_random_array(rows, cols, min_val, max_val)
    Print("2D NumPy array initialized with random integers:\n", random_array)
```

13. Write a Python program to generate an array of evenly spaced numbers over a specified range using linspace.

```
Import numpy as np

Def generate_linspace(start, stop, num):
    """
    Generate an array of evenly spaced numbers over a specified range using linspace.
    :param start: Start of the range
    :param stop: End of the range
    :param num: Number of samples to generate
    :return: NumPy array of evenly spaced numbers
    """
    Return np.linspace(start, stop, num)

# Example usage:
If __name__ == "__main__":
    Start = float(input("Enter the start of the range: "))
    Stop = float(input("Enter the end of the range: "))
    Num = int(input("Enter the number of samples to generate: "))

    Linspace_array = generate_linspace(start, stop, num)
    Print("Array of evenly spaced numbers over the range [{}, {}] with {} samples:\n{}".format(start, stop, num, linspace_array))
```

14. Write a program to generate an array of 10 equally spaced values between 1 and 100 using linspace.

```
Import numpy as np

# Generate an array of 10 equally spaced values between 1 and 100 using linspace
Result_array = np.linspace(1, 100, 10)
```

```
# Print the generated array
Print("Array of 10 equally spaced values between 1 and 100:")
Print(result_array)
```

15. **Write a Python program to create an array containing even numbers from 2 to 20 using arrange**

```
Import numpy as np

# Create an array containing even numbers from 2 to 20 using arange
Even_array = np.arange(2, 21, 2)

# Print the generated array
Print("Array containing even numbers from 2 to 20:")
Print(even_array)
```

16. **Write a program to create an array containing numbers from 1 to 10 with a step size of 0.5 using arange.**

```
Import numpy as np

# Create an array containing numbers from 1 to 10 with a step size of 0.5 using
arange
Array_with_step = np.arange(1, 10.5, 0.5)

# Print the generated array
Print("Array containing numbers from 1 to 10 with a step size of 0.5:")
Print(array_with_step)
```