1. Explain the key features of Python that make it a popular choice for programming.

```
- Easy to Learn: Simple and readable syntax, easy to learn .
- Dynamically Typed: No need to declare variable types.
- Versatile: Supports procedural, object-oriented, and functional
programming.
- Extensive Standard Library: Built-in modules for various tasks.
- Large Ecosystem: Many third-party libraries and frameworks.
- Cross-platform: Runs on Windows, macOS, and Linux.
- Automation and Scripting: Ideal for automating tasks.
These are some basic features make Python a flexible, powerful, and
user-friendly language.
```

2. Describe the role of predefined keywords in Python and provide examples of how they are used in a program.

```
- In Python, predefined keywords are special reserved words that have
specific meanings within the language. You cannot use them as variable
names, function names, or any other custom identifier. These keywords
define the syntax and structure of your Python programs.
Some predefinied keywords; False, True, if, elif, else, while, def,
lambda etc.

- Some examples how this predefined keyword use in to write a program,

  a: Time_identifying program by using "if_elif_else" predefined
keyword;

      Time = 5

        if (Time <= 6):
            print("It's still early morning")
        elif (6<Time<=12):
            print(" Good Morning")
        elif (12<Time<=17):
            print("Good Afternoon")
        elif (17<Time<=22):
            print(" Good Evening")
        else:
            print(" Good Night")

    b: Create a square of stars by using "While" predefined keyword;

        n = 4  # Size of the square
        i = 0
```

```
            while i < n:
                j = 0
                while j < n:
                    print('*', end=' ')
                    j += 1
            print()  # Move to the next line after printing a row
            i += 1
```

3. Compare and contrast mutable and immutable objects in Python with examples.

```
- Mutable objects: In python which objects / containers can be changed
after their cretion are called mutable objects.
    Examples;
             - Lists: [1, 2, 3] - You can add, remove, or change
elements within the list.
             - Dictionaries: {'name': 'Alice', 'age': 25} - You can
add, remove, or change key-value pairs.
             - Sets: {1, 2, 3} - You can add or remove elements, but
the order is not guaranteed.

- Immutable objects: In python which objects / containers can not be
changed after their cretion are called immutable objects. Any attempt
to change the object results in a new object being created.
    Examples;
             - Tuples: (1, 2, 3) - Similar to lists, but elements
cannot be changed.
             - Numbers (Integers, Floats): 10, 3.14 - Their values
cannot be changed.
             - Strings: "hello" - You cannot change individual
characters within the string.
```

Key Differences:

```
   a)  State Change;
         - Mutable Objects: Their internal state can be changed after
they are created.
         - Immutable Objects: Their internal state cannot be changed
once they are created. Any modification results in a new object.

  b)   Memory Efficiency:
         - Mutable: More memory efficient for frequent updates as they
do not create new objects for each change.
         - Immutable: May lead to increased memory usage if many
modifications are needed, as each modification results in a new
object.
```

```
  c)   Thread Safety:
         - Mutable: Not inherently thread-safe as their state can be
changed by multiple threads.
         - Immutable: Thread-safe since their state cannot be changed
once created.
```

4. Discuss the different types of operators in Python and provide examples of how they are used.

```
- Python provides a rich set of operators that allow you to perform
various operations on variables and values. These operators can be
categorized into several types:

        - a)  Arithmetic Operators:

                These operators perform basic mathematical
calculations on numeric data (integers and floats).

                Examples;
                + (addition): x = 5 + 3 (assigns 8 to x)
                - (subtraction): y = 10 - 2 (assigns 8 to y)
                * (multiplication): z = 4 * 6 (assigns 24 to z)
                / (division): result = 12 / 3 (assigns 4.0 to result)
                % (modulo): remainder = 11 % 4 (assigns 3 to
remainder)
                ** (exponentiation): power = 2 ** 3 (assigns 8 to
power).

        - b) Comparison Operators:

                These operators compare values and return boolean
results (True or False).

                Examples;
                == (equal to): a == 5 (checks if a is equal to 5)
                != (not equal to): b != 10 (checks if b is not equal
to 10)
                > (greater than): c > 20 (checks if c is greater than
20)
                < (less than): d < 15 (checks if d is less than 15)
                >= (greater than or equal to): e >= 0 (checks if e is
greater than or equal to 0)
                <= (less than or equal to): f <= 100 (checks if f is
less than or equal to 100)

        - c) Assignment Operators:

                These operators assign values to variables and can be
combined with arithmetic operators.
```

```
                Examples;
                = (simple assignment): x = 10 (assigns 10 to x)
                += (add and assign): x += 5 (equivalent to x = x + 5)
                -= (subtract and assign): y -= 3 (equivalent to y = y
- 3)
                *= (multiply and assign): z *= 2 (equivalent to z = z
* 2)
                /= (divide and assign): result /= 4 (equivalent to
result = result / 4)

        - d) Logical Operators:

                These operators combine conditional statements using
and, or, and not.

                Examples;
                and: if (x > 0) and (y < 10): (checks if both
conditions are True)
                or: if (a == 5) or (b == 8): (checks if at least one
condition is True)
                not: if not (name is None): (checks if name is not
None)

        - e) Bitwise Operators:

                These operators perform bit-level operations on
integers (not commonly used).

                Examples;
                & (bitwise AND): x & y (performs AND operation on each
bit of x and y)
                | (bitwise OR): x | y (performs OR operation on each
bit of x and y)
                ^ (bitwise XOR): x ^ y (performs XOR operation on each
bit of x and y)
                ~ (bitwise NOT/ Tilda): ~x (inverts the bits of x)
               >> (bitwise RIGHT shift):
               << (bitwise LEFT shift): 35 << 3 (put zeros on right
side)
        - f) Membership Operators: 280 >> 3 (remove number of
elements)

                These operators check if a value is present in a
sequence (list, tuple, string).

                Examples;
                in: if "apple" in fruits: (checks if "apple" is in the
fruits list)
                not in: if number not in range(1, 11): (checks if
```

```
number is not in the range)

        - g)  Identity Operators:

                These operators check or compare the memory location
of two objects.

                Examples:
                is: if x is y: (checks if x and y refer to the same
object)
```

5. Explain the concept of type casting in Python with examples.

```
- Type casting in Python refers to the process of converting one data
type into another. This can be useful when you need to perform
operations that require a certain data type. Python provides several
built-in functions to facilitate type casting.

Here are two types of type casting in Python;
```

- a) Implicit Type Casting:

```
                Python automatically converts one data type to another
without explicit intervention from the user.

                # Implicit type casting;

                        r = 5        # Integer
                        s = 3.5      # Float

                        result = r + s        # Adding integer and
float

                        print(result)         # Output: 7.5
                        print(type(result))   # Output: <class
'float'>
```

- b) Explicit Type Casting:

```
                The user manually converts one data type to another
using built-in functions.

                # Explicit type casting;

              ->  # Integer to float

                        r = 9
                        s = float(r)
```

```
                        print(s)          # Output: 9.0
                        print(type(s))    # Output: <class 'float'>

               ->  # String to integer

                        o = "987"
                        f = int(o)
                        print(f)          # Output: 987
                        print(type(f))    # Output: <class 'int'>

               ->  # String to float

                        k = "3.27"
                        l = float(k)
                        print(l)          # Output: 3.27
                        print(type(l))    # Output: <class 'float'>
```

6. How do conditional statements work in Python? Illustrate with examples.

```
  - Conditional statements in Python allow you to execute specific
blocks of code based on certain conditions. These statements are
essential for making decisions in a program.

      There are three type cinditional statement;
      -> if statement.
      -> if-else statement.
      -> if-elif-else sattement.


  - a) if statement:

        The 'if' statement evaluates a condition, and if it is 'True',
the code block under it is executed.

      # Example;
        p = 10

        if (p > 6):
            print("p is greater than 6")

        Output > x is greater than 6


  - b) if-else statement:

          The 'if-else' statement provides an alternative block of
code that runs if the condition is 'False'.
```

```
        #Example;

        Checking Even or Odd:

        num = 9

        if (num % 2 == 0):
            print("Even")
        else:
            print("Odd")

        Output >> Odd


    - c) if-elif-else statement:

            The 'if-elif-else' statement is used for multiple
conditions. If the first condition is 'False', the next 'elif'
condition is checked, and so on. If none of the conditions are 'True',
the 'else' block is executed.

            #Example;

            Grade classfication:

            Marks = 95

            if (score >= 90):
                print("You got A >> Excellent")
            elif score >= 80:
                print("You got B >> Very Good")
            elif score >= 70:
                print("You got C >> Good")
            elif score >= 60:
                print("You got D >> Need improvent")
            else:
                print("You got F >> Failed")

            Output >>> You got A >> Excellent
```

7. Describe the different types of loops in Python and their use cases with examples.

```
   - In Python, loops are used to repeatedly execute a block of code
as long as a condition is met.

     Python provides two main types of loops: 'for' loop and 'while'
loops.
```

- a) For Loop;

A for loop is used to iterate over a sequence (such as a list, tuple, dictionary, set, or string) and execute a block of code for each element in the sequence.

#Use case;
-> Iterating through elements of a list, tuple, or set.
-> Iterating through keys or values of a dictionary.
-> Iterating through characters in a string.
-> Iterating with a specific range of numbers.

#Example;

Iterating through a List:

```
a = [1, 2, 3, 4, 5, 6, "Rohan", "PWskills"]
for r in a:
    print(r)
```

Output >>
```
1
2
3
4
5
6
Rohan
PWskills
```

Iterating through a string:

```
for char in "Rohan Sil":
    print(char, end = " ")
```

Output >> R o h a n   S i l

- b) While Loop;

A 'while' loop is used to repeatedly execute a block of code as long as a condition is True. The condition is checked before the code block is executed.

#Use case;
-> When the number of iterations is not known beforehand and depends on a condition.
-> When you need to repeatedly execute code until a specific

```
condition changes.

    #Example;

    Reverse order:

       count = 10
       while (count > 0):
           print(count)
           count = count - 1

    Output >> 10
              9
              8
              7
              6
              5
              4
              3
              2
              1

    Break the loop:

       count = 10
       while (count > 0):
           print(count)
           count = count - 1
           if (count == 2):
               break

    Output >> 10
              9
              8
              7
              6
              5
              4
              3

    Draw Right star pattern:
    # Range is 6

       n = 6
       i = 1

       while i <= n:
           print('*' * i)
           i += 1
```

```
    Output >>  *
               **
               ***
               ****
               *****
               ******
```