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

Ans. Python is widely popular due to its simple and readable syntax, which makes it beginner-friendly and easy to learn. It's an interpreted language, allowing for immediate feedback and easier debugging. Python is dynamically typed, so developers don't need to declare variable types, making code concise. Its extensive standard library supports various tasks, from web development to file handling. Python is cross-platform, meaning it runs on different operating systems. Additionally, it supports multiple programming paradigms like object-oriented and functional programming, and has a large ecosystem of libraries for tasks like data science, machine learning, and automation.

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

Ans. Predefined keywords in Python are reserved words that have specific meanings and cannot be used as variable names. They form the backbone of the language's syntax and control its structure.

Examples:

- Control flow: if, else, elif, for, while, break, continue
- Data types: int, float, str, bool, list, tuple, dict, set
- Functions: def, return
- Classes: class, self
- Modules: import
- Exceptions: try, except, raise

```
# Control flow

if x > 0:

print("Positive")

else:

print("Negative or zero")

 # Data types

numbers = [1, 2, 3]

name = "Alice"
```

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

Ans. Mutable objects In Python, an object is considered mutable if its value can be changed after it has been created. This means that any operation that modifies a mutable object will modify the original object itself.

Examples:

- **Lists:** my_list = [1, 2, 3]
- **Dictionaries:** my_dict = {"name": "Alice", "age": 30}
- **Sets:** my_set = {1, 2, 3}

Code example:

```
my_list = [1, 2, 3]

my_list.append(4)      # Modifying the list

print(my_list)

#Output: [1, 2, 3, 4]
```

Immutable Objects

In Python, an object is considered immutable if its value cannot be changed after it has been created. This means that any operation that modifies an immutable object returns a new object with the modified value.

Examples:

- **Numbers:** x = 5
- **Strings:** name = "Alice"
- **Tuples:** my_tuple = (1, 2, 3)

Code example:

```
x = 5

y = x + 1                # Creating a new integer object

print(x, y)

#Output: 5 6
```

Key differences:

- **Mutability:** Mutable objects can be changed, while immutable objects cannot.
- **Memory usage:** Modifying a mutable object doesn't create a new object, while modifying an immutable object creates a new one.
- **Efficiency:** Mutable objects can be more efficient for large data structures, but immutable objects can be more efficient for small, frequently used values

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

Ans. Operators in Python are symbols that perform specific operations on values or variables. They can be categorized into several types:

Arithmetic Operators

Used for mathematical calculations.

- **Addition:** +
- **Subtraction:** -
- **Multiplication:** *
- **Division:** /
- **Floor division:** // (returns the quotient as an integer)
- **Modulus:** % (returns the remainder)
- **Exponentiation:** **

```
result = 5 + 3
print(result)  # Output: 8
```

Comparison Operators

Used to compare values and return a boolean result.

- **Equal to:** ==
- **Not equal to:** !=
- **Greater than:** >
- **Less than:** <
- **Greater than or equal to:** >=
- **Less than or equal to:** <=

```
x = 10
y = 5
if x > y:
    print("x is greater than y")
```

Assignment Operators

Used to assign values to variables.

- **Simple assignment:** =
- **Addition assignment:** +=
- **Subtraction assignment:** -=

- **Multiplication assignment:** *=
- **Division assignment:** /=
- **Floor division assignment:** //=
- **Modulus assignment:** %=
- **Exponentiation assignment:** **=

```
x = 5
x += 2  # Equivalent to x = x + 2
print(x)  # Output: 7
```

Logical Operators

Used to combine boolean expressions.

- **And:** and
- **Or:** or
- **Not:** not

Python
```
a = True
b = False
if a and b:
    print("Both a and b are True")
```

Bitwise Operators

Used to perform operations on individual bits of binary numbers.

- **Bitwise AND:** &
- **Bitwise OR:** |
- **Bitwise XOR:** ^
- **Bitwise NOT:** ~
- **Left shift:** <<
- **Right shift:** >>

```
x = 5  # Binary representation: 00000101
y = 3  # Binary representation: 00000011
result = x & y  # Bitwise AND
print(result)  # Output: 1 (Binary representation: 00000001)
```

Membership Operators

Used to check if a value is a member of a sequence (like a list or tuple).

- **In:** in
- **Not in:** not in

```
my_list = [1, 2, 3]
if 2 in my_list:
    print("2 is in the list")
```

Identity Operators

Used to check if two objects are the same object in memory.

- **Is:** is
- **Is not:** is not

```
x = 5
y = 5
if x is y:
   print("x and y are the same object")
```

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

Ans. Type Casting in Python

Type casting, also known as type conversion, is the process of converting one data type to another in Python. This is often necessary when you need to perform operations that require specific data types or when you want to display data in a particular format.

Common type casting operations in Python:

1. **Integer to float:**

   ```
   x = 5
   y = float(x)
   print(y)  # Output: 5.0
   ```

2. **Float to integer:**

   ```
   x = 3.14
   y = int(x)
   print(y)  # Output: 3
   ```

3. **String to integer:**

   ```
   x = "10"
   y = int(x)
   print(y)  # Output: 10
   ```

4. **String to float:**

   ```
   x = "3.14"
   y = float(x)
   print(y)  # Output: 3.14
   ```

5. **Integer or float to string:**

   ```
   x = 5
   y = str(x)
   print(y)  # Output: "5"
   ```

6. **List to tuple:**

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple)  # Output: (1, 2, 3)
```

7. **Tuple to list:**

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)  # Output: [1, 2, 3]
```

Que_6. How do conditional statements work in Python? Illustrate with examples?

Conditional Statements in Python

Conditional statements in Python allow you to execute different code blocks based on certain conditions. They provide a way to make your code more flexible and responsive to different situations.

The `if` statement:

The most basic conditional statement is the `if` statement. It executes a block of code only if a specified condition is true.

if `condition:`

```
    # Code to be executed if the condition is true
```

Example:

```
age = 18
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

The `if-else` statement:

The `if-else` statement allows you to execute one block of code if a condition is true, and another block if it's false.

```
if condition:
    # Code to be executed if the condition is true
else:
    # Code to be executed if the condition is false
```

Example:

Code snippet

```
score = 85
if score >= 90:
    print("Excellent!")
else:
    print("Good job!")
```

The `if-elif-else` statement:

The `if-elif-else` statement allows you to check multiple conditions and execute different code blocks based on the first condition that is true.

```
if condition1:
    # Code to be executed if condition1 is true
elif condition2:
    # Code to be executed if condition2 is true
else:
    # Code to be executed if
 none of the conditions are true
```

Example:

```
grade = 85
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
else:
    print("C")
```

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

Ans. Loops in Python are used to execute a block of code repeatedly until a certain condition is met. There are two main types of loops: for loops and while loops.

for Loops

- **Use cases:** Iterating over sequences (like lists, tuples, strings, or dictionaries).
- **Syntax:**

```
for element in sequence:
    # Code to be executed
```

Example:

```
        fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

while Loops

- **Use cases:** Repeating code as long as a condition is true.
- **Syntax:**

```
while condition:
    # Code to be executed
```

Example:

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

Other Loop Constructs

- **break:** Exits the loop immediately, regardless of the condition.
- **continue:** Skips the current iteration and continues with the next one.
- **else clause:** Executes code if the loop terminates normally (without break).

Example:

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    if number == 3:
        break
    print(number)
else:
    print("The loop completed normally")
```