Ans 1:
FOLLOWING are the key features which makes python as a choice for all user
a) READABILITY AND SIMPLICITY- its syntax is clear and easy to read
b) Versability- it supports multiple programming including object oriented and functional programming
c) EXTENSIVE LIBRARIES AND FRAMEWORKS- IT FACILITATE TASK LIKE WEB DEVELOPment , data analysis and machine learning
d) COMMUNITY SUPPORT- it allow developers to write bcode that runs on different platform without modification.
e) RAPID DEVELOPMENT
f) Integration capabilities.

Ans 2:
keywords are reserved and have special meaning used to define syntaxand structure of python language.
and also used to define logic and flow of a python program
and cannot be used as a identifiers.
Predefine keywords are listed below
if, elif,else

```
x = 20
if x > 2:
    print("positive")
elif x == 0:
        print("zero")
else:
    print("negative")
```

```
positive
```

```
# for, while, break, continue
for i in range(8):
    if i == 6:
        break
    print(i)
```

```
0
1
2
3
4
5
```

```
i = 0
while i < 5:
    i = i+1
    if i == 3:
        continue
        print(i)
```

```
        # keywords are the backbone of python which helps user bto
write strucure in readable fromate.
```

Ans 3- Mutable objects are those object which can altered or changed after creation
example
a) list
b) dictionary
c)sets
d) array
d) byte
Immutable list
are those list which cant be changed after creation
example- string, tuples, integers, float

Ans-5
type casting in python refers to conversion of one data type to another.
implicit type casting- automatic conversion of data type by python from a smaller to larger data type to prevent data loss.

```
#example
num_int = 10
num_float = 3.12
result = num_int + num_float
print("result (int + float):", result)
print("type of result:", type (result))
```

```
result (int + float): 13.120000000000001
type of result: <class 'float'>
```

explicit
it means manual conversion of onr data type to another using built in function.

```
a = 8.89
type(a)
#conversion of float to int
int(a)
```

```
8
```

```
# float() covert to a float
#str() convert to string
#list() convert to list
```

Ans-4
In Python, operators are symbols or keywords used to perform operations on variables and values. Here's an overview of the different types of operators in Python along with examples:

```
1. Arithmetic Operators
Arithmetic operators are used for mathematical calculations.

Addition (+): Adds two operands.
a = 10
b = 5
result = a + b  # result = 15

Subtraction (-): Subtracts the right operand from the left operand.
a = 10
b = 5
result = a - b  # result = 5

Multiplication (*): Multiplies two operand
a = 10
b = 5
result = a * b  # result = 50
Division (/): Divides the left operand by the right operand (always
results in a float).
a = 10
b = 5
result = a / b  # result = 2.0
Floor Division (//): Divides and returns the integer part of the
quotien
a = 10
b = 3
result = a // b  # result = 3
Modulus (%): Returns the remainder of the division.
a = 10
b = 3
result = a % b  # result = 1
Exponentiation (**): Raises the left operand to the power of the right
operand
a = 2
b = 3
result = a ** b  # result = 8
2. Comparison Operators
Comparison operators are used to compare values. They return either
True or False.

Equal to (==): Checks if two operands are equal.
a = 10
b = 5
result = (a == b)  # result = False
Not equal to (!=): Checks if two operands are not equal.
a = 10
b = 5
result = (a != b)  # result = True
Greater than (>): Checks if the left operand is greater than the right
operand.
```

```
a = 10
b = 5
result = (a > b)  # result = True
Less than (<): Checks if the left operand is less than the right
operand
a = 10
b = 5
result = (a < b)  # result = False
Greater than or equal to (>=): Checks if the left operand is greater
than or equal to the right operand.
a = 10
b = 5
result = (a >= b)  # result = True
Less than or equal to (<=): Checks if the left operand is less than or
equal to the right operand.
a = 10
b = 5
result = (a <= b)  # result = False
3. Logical Operators
Logical operators are used to combine conditional statements.

Logical AND (and): Returns True if both operands are
a = True
b = False
result = (a and b)  # result = False
Logical OR (or): Returns True if at least one operand is True.
a = True
b = False
result = (a or b)  # result = True
Logical NOT (not): Returns True if the operand is False, and vice
versa.
a = True
result = not a  # result = False
4. Assignment Operators
Assignment operators are used to assign values to variables.

Assignment (=): Assigns the value on the right to the variable on the
left.
a = 10
Increment (+=): Adds the right operand to the left operand and assigns
the result to the left operand.
a = 10
a += 5  # equivalent to a = a + 5
Decrement (-=): Subtracts the right operand from the left operand and
assigns the result to the left oper.
a = 10
a -= 5  # equivalent to a = a - 5
5. Bitwise Operators
Bitwise operators perform operations on binary representations of
```

```
integers.

Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^): Perform bitwise AND,
OR, and XOR operations respectively.
a = 10  # binary 1010
b = 5   # binary 0101
result_and = a & b   # result_and = 0 (binary 0000)
result_or = a | b    # result_or = 15 (binary 1111)
result_xor = a ^ b   # result_xor = 15 (binary 1111)
Bitwise NOT (~): Inverts all bits of the operand.
a = 10  # binary 1010
result_not = ~a   # result_not = -11
Left Shift (<<), Right Shift (>>): Shift bits to the left or right by
the specified number of positions.
a = 10  # binary 1010
result_left_shift = a << 1   # result_left_shift = 20 (binary 10100)
result_right_shift = a >> 1  # result_right_shift = 5 (binary 101)
6. Membership Operators
Membership operators are used to test if a sequence is present in an
object.

in: Returns True if a value is found in the specified sequence.
list1 = [1, 2, 3, 4, 5]
result = (3 in list1)  # result = True
not in: Returns True if a value is not found in the specified
sequence.
list1 = [1, 2, 3, 4, 5]
result = (6 not in list1)  # result = True
7. Identity Operators
Identity operators compare the memory locations of two objects.

is: Returns True if both variables point to the same object.
a = [1, 2, 3]
b = a
result = (a is b)  # result = True
is not: Returns True if both variables do not point to the same
object.
a = [1, 2, 3]
b = [1, 2, 3]
result = (a is not b)  # result = True
```

Ans-6
Conditional statements in Python are used to make decisions based on whether a certain condition evaluates to True or False. The primary conditional statements in Python are if, elif (short for else if), and else. Here’s how they work, illustrated with examples:

if statement
The if statement executes a block of code only if a specified condition is True.

```
example
x = 10

if x > 5:
    print("x is greater than 5")  # This line will be printed because
x > 5 is True
In this example:

x > 5 is the condition being checked.
Since x is 10, which is indeed greater than 5, the statement inside
the if block (print("x is greater than 5")) is executed.

2- if else statement:-
The else statement follows an if statement and executes a block of
code if the if condition is False.
Example 2: if-else statement
x = 2

if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")  # This line will be printed
because x > 5 is False
In this example:

x > 5 is False because x is 2.
Therefore, the else block (print("x is not greater than 5")) is
executed.
3. if ... elif ... else Statement
The elif statement allows you to check multiple conditions. It follows
an if statement and is followed by an optional else statement.

 Example 3: if-elif-else statement
x = 3

if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")  # This line will be printed because x >
5 and x == 5 are False
In this example:

x > 5 is False because x is 3.
x == 5 is also False.
Therefore, the else block (print("x is less than 5")) is executed.
Nested if Statements
You can also nest if statements within one another to create more
```

complex decision-making processes.

Example 4: Nested if statements

```
x = 10
y = 5

if x > y:
    if x % 2 == 0:
        print("x is greater than y and x is even")
    else:
        print("x is greater than y but x is odd")
else:
    print("x is not greater than y")
```

In this example:

The outer if checks if x is greater than y.
If x is indeed greater than y, it then checks whether x is even or odd using another if statement nested inside.
Depending on the conditions, different messages will be printed.
Using Conditional Expressions (Ternary Operator)
Python also supports a ternary conditional expression which provides a compact way to evaluate a condition.

Example 5: Ternary conditional expression

```
x = 10
y = 5

message = "x is greater than y" if x > y else "x is not greater than y"
print(message)  # Output: x is greater than y
```

Ans-7
In Python, loops are used to repeatedly execute a block of code until a certain condition is met. There are several types of loops available: for loop, while loop, and nested loops. Each type has its own specific use cases and syntax. Let's explore each one with examples:

1. for Loop
A for loop is used to iterate over a sequence (like a list, tuple, string, or range) or any iterable object.

Example 1: Iterating over a list

```
#Iterating over a list
fruits = [apple, banana cherry]
for fruit in fruits:
    print(fruit)
```

Output:

Copy code

```
apple
banana
cherry
Example 2: Iterating over a range
 Iterating over a range
for i in range(1, 5):
    print(i)
Output:

Copy code
1
2
3
4
Example 3: Iterating over a string
 Iterating over a string
for char in "hello":
    print(char)
Output:
h
e
l
l
o
2. while Loop
A while loop executes a block of code as long as a specified condition
is True.

Example 4: Using a while loop to print numbers from 1 to 5
 Using a while loop
i = 1
while i <= 5:
    print(i)
    i += 1
Output
1
2
3
4
5
```