**Ans 1)** Python's popularity stems from a variety of features that make it a favorite among both beginners and experienced programmers. Here are some of the key features:

1. Easy to Read and Write: Python's syntax is clear and readable, making it an excellent choice for beginners.

2. Interpreted Language: Python is an interpreted language, which means you can run the code directly without having to compile it first.

3. Extensive Standard Library: Python comes with a rich standard library that covers many common programming tasks, from file I/O to web development, which reduces the need to write code from scratch.

4. Large Community and Ecosystem: Python boasts a large and active community, which means plenty of resources, tutorials, and third-party libraries are available.

5. Platform Independent: Python is cross-platform, meaning you can run Python programs on various operating systems like Windows, macOS, and Linux without making significant changes to the code.

6. Versatility: Python is used in a wide range of domains such as web development, data science, artificial intelligence, scientific computing, and automation. Its versatility makes it a go-to language for many applications.

**Ans2)** Predefined keywords in Python are reserved words with special meanings that define the language's syntax and structure. They can’t be used as variable names, function names, or identifiers because Python interprets them for specific operations. These keywords help in controlling program flow, defining functions, handling exceptions, and performing logical operations.

Examples of Python Keywords and Their Usage

1. `if`, `else`, `elif` – Control program flow based on conditions.

if age >= 18:

print("Adult")

else:

print("Minor")

2. `for`, `while` – Create loops for repetitive tasks.

for num in range(5):

print(num) # Prints 0 to 4

while x < 10:

x += 1

3. `def`’ – Defines a function.

def greet(name):

print(f"Hello, {name}!")

4. `return` – Exits a function and sends back a value.

def add(a, b):

return a + b

5. try`, `except`, `finally’ – Handle exceptions (errors).

try:

result = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero!")

6. ‘import` – Loads external modules.

import math

print(math.sqrt(16)) # Output: 4.0

7. `True`, `False`, `None` – Represent boolean values and null.

is\_valid = True

value = None

8. `and`, `or`, `not` – Perform logical operations.

if x > 0 and y < 10:

print("Valid range")

**Ans3)** In Python, objects can be classified into two categories: mutable and immutable. The difference lies in whether the object can be changed after its creation.

Mutable objects are those that can be modified after they are created. This means you can change their content without creating a new object.

Eg- Lists:

my\_list = [1, 2, 3]

my\_list[0] = 10

print(my\_list)

Immutable objects, on the other hand, cannot be changed after they are created. Any attempt to modify an immutable object will result in the creation of a new object.

Eg- Strings:

my\_string = "Hello"

new\_string = my\_string.replace("H", "J") # Creating a new string

print(my\_string)

print(new\_string)

**Ans 4)** Python offers a diverse set of operators, which are tools used to perform operations on variables and values. Here is a categorized overview with examples:

1. Arithmetic Operators

These operators are used to perform mathematical calculations.

- `+` (Addition): Adds two values.

Example: `result = 10 + 5` → `result` becomes `15`.

- `-` (Subtraction): Subtracts one value from another.

Example: `result = 10 - 5` → `result` becomes `5`.

- `\*` (Multiplication): Multiplies two values.

Example: `result = 10 \* 5` → `result` becomes `50`.

- `/` (Division): Divides one value by another.

Example: `result = 10 / 5` → `result` becomes `2.0`.

- `%` (Modulus): Returns the remainder of division.

Example: `result = 10 % 3` → `result` becomes `1`.

2. Comparison Operators

Used to compare two values, returning a Boolean result.

- `==` (Equal to): Checks if values are equal.

Example: `10 == 5` returns `False`.

- `!=` (Not equal to): Checks if values are not equal.

Example: `10 != 5` returns `True`.

- `>` (Greater than): Checks if left value is greater.

Example: `10 > 5` returns `True`.

3. Logical Operators

Used in logical conditions.

- `and`: Returns `True` if both conditions are `True`.

Example: `(5 > 3) and (8 > 6)` → `True`.

- `or`: Returns `True` if at least one condition is `True`.

Example: `(5 > 3) or (8 < 6)` → `True`.

- `not`: Reverses the Boolean value.

Example: `not(5 > 3)` → `False`.

4. Assignment Operators

Used to assign and modify values.

- `=`: Assigns value.

Example: `x = 5`.

- `+=`: Adds and assigns.

Example: `x += 2` → `x` becomes `7`.

5. Bitwise Operators

Used for binary computations.

- `&` (AND): Performs bitwise AND operation.

Example: `5 & 3` → `1`.

6. Membership Operators

Used to check if a value is part of a sequence like a list or string.

- `in`: Returns `True` if value exists.

Example: `'a' in 'apple'` → `True`.

- `not in`: Returns `True` if value doesn't exist.

Example: `'b' not in 'apple'` → `True`.

**Ans 5)** Type casting in Python is the process of converting one data type to another. It allows programmers to perform operations or comparisons on values that might initially be incompatible due to differing data types. Type casting can be done either implicitly or explicitly:

Implicit Type Casting

This occurs automatically during certain operations where Python converts one data type to another to prevent data loss or errors.

Example:

```python

a = 5 # Integer

b = 2.5 # Float

result = a + b # Python implicitly converts 'a' to a float

print(result) # Output: 7.5

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

```

Here, the integer `a` is implicitly converted to a float to match `b`.

Explicit Type Casting

In this case, the programmer manually converts a data type using casting functions like `int()`, `float()`, `str()`, and more.

Examples:

1. Integer to String:

num = 42

converted\_num = str(num) # Explicitly converts the integer to a string

print(converted\_num) # Output: "42"

print(type(converted\_num)) # Output: <class 'str'>

2. Float to Integer:

pi = 3.14

converted\_pi = int(pi) # Explicitly converts the float to an integer

print(converted\_pi) # Output: 3

3. String to Float:

text = "100.5"

converted\_text = float(text) # Converts string to float

print(converted\_text) # Output: 100.5

print(type(converted\_text)) # Output: <class 'float'>

4. List to Tuple:

list\_data = [1, 2, 3]

tuple\_data = tuple(list\_data) # Converts list to tuple

print(tuple\_data) # Output: (1, 2, 3)

**Ans 6)** Conditional Statements in Python (With Examples)

Conditional statements allow a program to execute different blocks of code based on whether a condition is `True` or `False`. Python uses `if`, `elif` (else-if), and `else` for decision-making.

1. Basic `if` Statement

- Executes a block only if the condition is `True`.

Example:

age = 18

if age >= 18:

print("You are eligible to vote.")

Output:

You are eligible to vote.

2. `if-else` Statement

- Executes one block if the condition is `True`, otherwise runs the `else` block.

Example:

num = 10

if num % 2 == 0:

print("Even number")

else:

print("Odd number")

Output:

Even number

3. `if-elif-else` Ladder

- Checks multiple conditions in sequence. The first `True` condition runs its block, and the rest are skipped.

Example:

score = 85

if score >= 90:

print("Grade: A")

elif score >= 80:

print("Grade: B")

elif score >= 70:

print("Grade: C")

else:

print("Grade: D")

Output:

Grade: B

4. Nested `if` Statements

- An `if` block inside another `if` block for complex conditions.

Example:

num = 15

if num > 0:

if num % 5 == 0:

print("Positive and divisible by 5")

else:

print("Positive but not divisible by 5")

else:

print("Non-positive number")

Output:

Positive and divisible by 5

5. Ternary Operator (Short `if-else`)

- A concise way to write `if-else` in a single line.

Syntax:

value\_if\_true if condition else value\_if\_false

Example:

age = 20

status = "Adult" if age >= 18 else "Minor"

print(status)

Output:

Adult

**Ans 7)** Loops in Python allow you to execute a block of code repeatedly based on a condition. Python supports two main types of loops:

1. `for` loop – Iterates over a sequence (list, tuple, string, etc.)

2. `while` loop – Repeats code while a condition is `True`

1. `for` Loop

Use Case: Best when you know the number of iterations in advance (e.g., looping through a list).

Syntax:

for item in sequence:

Code to execute

Examples:

a) Looping Through a List

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

Output:

apple

banana

cherry

b) Looping with `range()`

for i in range(5): # 0 to 4

print(i)

Output:

0

1

2

3

4

c) Looping Through a String

word = "Python"

for letter in word:

print(letter)

Output:

P

y

t

h

o

n

2. `while` Loop

Use Case: Best when you don’t know how many times the loop should run (e.g., user input validation, game loops).

Syntax:

while condition:

Code to execute

Examples:

a) Basic Counter

count = 0

while count < 5:

print(count)

count += 1

Output:

0

1

2

3

4

b) User Input Validation

password = ""

while password != "admin123":

password = input("Enter password: ")

print("Access granted!")

Output:

Enter password: hello

Enter password: test

Enter password: admin123

Access granted!

c) Infinite Loop with `break`

while True:

user\_input = input("Type 'quit' to exit: ")

if user\_input == "quit":

break # Exit the loop

print("You typed:", user\_input)

Output:

Type 'quit' to exit: hello

You typed: hello

Type 'quit' to exit: quit

3. Loop Control Statements

- `break`→ Exits the loop immediately.

- `continue` → Skips the current iteration and moves to the next.

- `else` in loops → Executes when the loop completes normally (without a `break`).

Example:

for num in range(10):

if num == 5:

break # Stops at 5

print(num)

else:

print("Loop completed") # Won't run due to 'break'

Output:

0

1

2

3

4