Explain the key features of Python that take it a popular choice for programing

Readability and Simplicity: Python’s syntax is clear and easy to read, making it accessible for beginners and enabling developers to write clean, maintainable code.

High-level Language: As a high-level language, Python abstracts away much of the complexity involved in lower-level programming.

Interpreted Language: Python is an interpreted language, meaning code is executed line-by-line. This allows for quick testing and debugging, making the development process more efficient.

ersatility and General-Purpose Nature: Python can be used for a wide range of applications, from web development (with frameworks like Django and Flask) to scientific computing, automation, and scripting.

obust Frameworks: Python offers powerful frameworks for web development (Django, Flask), data analysis (pandas), machine learning (scikit-learn, TensorFlow), and more, which help streamline development processes and enhance productivity

Extensive Standard Library: Python comes with a comprehensive standard library that supports many common programming tasks such as file I/O, system calls, and data manipulation

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

Predefined keywords in Python are reserved words that have special meanings and purposes within the language. These keywords are used to define the structure and control the flow of a Python program. They cannot be used as identifiers (variable names, function names, etc.) because they are part of the syntax of the language.

```
Example
x = 10
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")
```

```
x is greater than 5
```

Compare and contrast mutable and immutable objects in Python with examples

Immutable Objects

Characteristics: Cannot be Changed: Once created, the value of an immutable object cannot be altered. Hashable: Because their value doesn't change, immutable objects can be used as keys in dictionaries and elements in sets. Memory Efficiency: Since they cannot change, multiple references can point to the same object, saving memory.

```
Example:-  string , Tuples
Integers: x = 5
x = 5
y = x
x += 1  # x now points to a new object with value 6, y still points to
```

```
5
print(x, y)  # Output: 6 5
```

```
6 5
```

Mutable Objects Characteristics: Can be Changed: The state or content of a mutable object can be modified after it is created. Not Always Hashable: Because their state can change, mutable objects generally cannot be used as dictionary keys or set elements. Memory Overhead: Since their content can change, care must be taken when multiple references point to the same object.

Examples: Lists: lst = [1, 2, 3] Dictionaries: d = {'a': 1, 'b': 2}

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

1.Arithmetic Operators

2. Comparison Operators

3. Logical Operators

4. Assignment Operators

5. Bitwise Operators

6. Membership Operators

##7. Identity Operators

1.Arithmetic Operators Arithmetic operators are used to perform mathematical operations. Addition (+): Adds two operands.

a = 5 b = 3 result = a + b # result is 8

Subtraction (-): Subtracts the second operand from the first. result = a - b # result is 2

Multiplication (*): Multiplies two operands. result = a b # result is 15

Division (/): Divides the first operand by the second.

result = a / b # result is 1.6666666666666667 Floor Division (//): Divides the first operand by the second and returns the largest integer less than or equal to the result. result = a // b # result is 1

Modulus (%): Returns the remainder of the division of the first operand by the second. result = a % b # result is 2

1. Comparison Operators Comparison operators compare two values and return a Boolean result (True or False). Equal (==): Checks if two operands are equal. result = (a == b) # result is False Not Equal (!=): Checks if two operands are not equal. result = (a != b) # result is True Greater Than (>): Checks if the first operand is greater than the second. result = (a > b) # result is True Less Than (<): Checks if the first operand is less than the second. result = (a < b) # result is False Greater Than or Equal To (>=): Checks if the first operand is greater than or equal to the second. result = (a >= b) # result is True Less Than or Equal To (<=): Checks if the first operand is less than or equal to the second. result = (a <= b) # result is False
2. Logical Operators Logical operators are used to combine conditional statements. AND (and): Returns True if both operands are true. result = (a > 2 and b < 5) # result is True OR (or): Returns True if at least one of the operands is true. result = (a > 5 or b < 5) # result is True NOT (not): Returns True if the operand is false. result = not(a > 5) # result is True

Explain the concept of type casting in Python with examples

Type Casting Functions

int(): Converts a value to an integer. float(): Converts a value to a floating-point number. str(): Converts a value to a string. bool(): Converts a value to a boolean. list(): Converts a value to a list. tuple(): Converts a value to a tuple. set(): Converts a value to a set. dict(): Converts a value to a dictionary. Examples

1. Integer to Float x = 5 y = float(x) print(y) # Output: 5.0
2. Float to Integer x = 5.8 y = int(x) print(y) # Output: 5
3. Integer to String x = 123 y = str(x) print(y) # Output: "123" print(type(y)) # Output: <class 'str'>
4. String to Integer x = "123" y = int(x) print(y) # Output: 123 print(type(y)) # Output: <class 'int'>
5. String to Float x = "123.45" y = float(x) print(y) # Output: 123.45 print(type(y)) # Output: <class 'float'>
6. Float to String x = 123.45 y = str(x) print(y) # Output: "123.45" print(type(y)) # Output: <class 'str'>
7. Integer to Boolean x = 0 y = bool(x) print(y) # Output: False x = 1 y = bool(x) print(y) # Output: True
8. String to Boolean x = "" y = bool(x) print(y) # Output: False x = "hello" y = bool(x) print(y) # Output: True
9. List to Tuple x = [1, 2, 3] y = tuple(x) print(y) # Output: (1, 2, 3) print(type(y)) # Output: <class 'tuple'>
10. Tuple to List x = (1, 2, 3) y = list(x) print(y) # Output: [1, 2, 3] print(type(y)) # Output: <class 'list'>

11. List to Set x = [1, 2, 3, 3, 2, 1] y = set(x) print(y) # Output: {1, 2, 3} print(type(y)) # Output: <class 'set'>
12. Set to List x = {1, 2, 3} y = list(x) print(y) # Output: [1, 2, 3] print(type(y)) # Output: <class 'list'>
13. List to Dictionary When converting a list to a dictionary, the list should contain key-value pairs. x = [("a", 1), ("b", 2), ("c", 3)] y = dict(x) print(y) # Output: {'a': 1, 'b': 2, 'c': 3} print(type(y)) # Output: <class 'dict'>
14. Dictionary to List When converting a dictionary to a list, only the keys are converted. x = {'a': 1, 'b': 2, 'c': 3} y = list(x) print(y) # Output: ['a', 'b', 'c'] print(type(y)) # Output: <class 'list'> Type Casting Considerations Loss of Precision: Converting from float to int may lead to a loss of precision. Invalid Conversion: Not all conversions are valid. For example, converting a non-numeric string to an int will raise a ValueError. x = "abc" try: y = int(x) except ValueError as e: print(e) # Output: invalid literal for int() with base 10: 'abc

How do conditional statements work in Python? Illustrate with examples

Conditional statements in Python are used to execute code based on certain conditions. They allow the program to make decisions and follow different paths based on those decisions. The main types of conditional statements in Python are if, elif, and else.

Basic Syntax if statement: Executes a block of code if the condition is true. elif statement: Stands for "else if" and checks another condition if the previous if condition was false. else statement: Executes a block of code if all the previous conditions are false. Examples

1. Simple if Statement The if statement evaluates a condition and executes the block of code within it if the condition is true. x = 10 if x > 5: print("x is greater than 5") Output: x is greater than 5
2. if-else Statement The if-else statement evaluates a condition and executes one block of code if the condition is true and another block if the condition is false. x = 3 if x > 5: print("x is greater than 5") else: print("x is not greater than 5") Output: x is not greater than 5
3. if-elif-else Statement The if-elif-else statement checks multiple conditions. The first block of code with a true condition will be executed. x = 7 if x > 10: print("x is greater than 10") elif x > 5: print("x is greater than 5 but not greater than 10") else: print("x is 5 or less") Output: x is greater than 5 but not greater than 10
4. Nested if Statement You can nest if statements within other if statements to check multiple conditions. x = 8 if x > 5: print("x is greater than 5") if x < 10: print("x is also less than 10") Output: x is greater than 5 x is also less than 10
5. Multiple Conditions You can combine multiple conditions using logical operators and, or, and not. x = 7 if x > 5 and x < 10: print("x is between 5 and 10") y = 3 if y < 5 or y > 10:

print("y is either less than 5 or greater than 10") z = 0 if not z: print("z is zero") Output: x is between 5 and 10 y is either less than 5 or greater than 10 z is zero Using Conditional Statements with User Input age = int(input("Enter your age: ")) if age < 18: print("You are a minor.") elif 18 <= age < 65: print("You are an adult.") else: print("You are a senior.") If the user inputs 20: You are an adult. Example with Functions and Conditional Statements def check_number(number): if number > 0: return "Positive" elif number < 0: return "Negative" else: return "Zero" num = int(input("Enter a number: ")) result = check_number(num) print(result) If the user inputs -3: mathematica Negative Example with Lists and Conditional Statements numbers = [1, 2, 3, 4, 5] for num in numbers: if num % 2 == 0: print(f"{num} is even") else: print(f"{num} is odd") Output: 1 is odd 2 is even 3 is odd 4 is even 5 is odd

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

Python provides several types of loops that allow you to execute a block of code multiple times. The primary loops in Python are for and while loops. Each has its specific use cases and characteristics.

1. for Loop The for loop is used for iterating over a sequence (such as a list, tuple, dictionary, set, or string) or any other iterable object.

Use Cases: Iterating over a list of items Iterating over a range of numbers Iterating over characters in a string Iterating over keys and values in a dictionary

1. while Loop The while loop repeatedly executes a block of code as long as a given condition is true. Use Cases: Repeating an action until a condition changes Creating loops that need to be terminated with a break statement Waiting for an event to occur Example: Basic while Loop i = 0 while i < 5: print(i) i += 1 Output: 0 1 2 3 4

2. Nested Loops You can nest one loop inside another loop. This is useful for working with multi-dimensional data structures, like lists of lists. Example: Nested for Loops matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] for row in matrix: for num in row: print(num, end=" ") print() Output: 1 2 3 4 5 6 7 8 9

3. Loop Control Statements Python provides loop control statements to change the execution from its normal sequence. They include break, continue, and pass. break The break statement terminates the loop entirely. for i in range(10): if i == 5: break print(i) Output: 0 1 2 3 4

Example: Iterating Over a Range of Numbers for i in range(5): print(i) Output: 0 1 2 3 4 Example: Iterating Over a String string = "hello" for char in string: print(char) Output: h e l l o Example: Iterating Over a Dictionary person = {"name": "Alice", "age": 25, "city": "New York"} for key, value in person.items(): print(f"{key}: {value}") Output: vbnet name: Alice age: 25 city: New York

```
total = 0
num = 1
while num <= 10:
    total += num
    num += 1
print(total)  # Output: 55
```

```
55
```

Example: while Loop with break

```
i = 0
while True:
    print(i)
    i += 1
    if i >= 5:
        break
```

```
0
1
2
3
4
```

```
0
1
2
3
4

4
```