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

Python programming language is widely used all over the world because of its key feature and they are as follows:

1. Readability: It is easy to read for beginners as its's syntax is easy and readable and also for experience as well.
2. Large Libraries: It has got large libraries 137000 because of it lots of work is reduced it has got libraries like pandas, scikit-learn, matplotlib, seaborn, NumPy because of this it can also be used for various application purpose as well like data analyst, data science, machine learning and etc.
3. Community Support: It has got large community support which are active because of it learning, solving problem, developing libraries, improvement in language.
4. Versatility: It can be used multiple domain Django for web development, TensorFlow and Keras for ml, backend, automation, image processing and etc.
5. Interaction with Databases: It is also used for interaction with Databases as well in industry like data analyst.

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

Keywords: Keywords are predefined words that holds a special meaning and it has specific purpose. To access keywords lists of keywords we use **help('keywords').** This will return list of all current list of keywords like: False, None, break, class, if, elif, def, from and etc. It can be used anywhere else.

Roles

1. Control flow of the programs.
2. Define functions.
3. Handle errors and etc.

Examples:

```
# Using ' If' keywords
X= 10
 if x > 7:
    print("x is greater than 7")
else: print("x is not greater than 7")
# Using `for` loop
for i in range(5):
    print(i)


# Using `while` keywords
count = 0
while count < 3:
    print(count)
    count = count + 1
```

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

Mutable: Mutable objects values can be changed after they are created. Like list, dictionaries, sets

Immutable: Immutable objects values can not be changed after are created. Like Strings, Tuples, Integers, Floats

Examples:

Mutable

```
a = [10,2,7]

a[2]= 20 # modifying list

print(a) #output [10,2,20]
```

Immutable

```
my_string = "Ayush"

my_string = "Mathur" # Creating a new string

print(my_string) # Output: Mathur
```

Differences

1. **Modification**: Mutable can be modified whereas immutable cannot be modified.
2. Memory Usage: Mutable objects are memory efficient as they can be modified.
3. Useful: Mutable are more useful when need arises to change anything if required.

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

Operators are special keywords/symbols that are used to perform operation on values or symbols. It helps in managing data, do computation and make decision using data.

Different types of operators with examples in python are as follows:

1.Arithematic Operators: It is used to perform mathematical operations like +, -, *, / and etc.

Example

```
#Using '+' operator

a = 2

b=2

add = a + b

add

#Using'– 'operator

a = 2

b=2

Sub = a – b

Sub
```

2.Comparision Operators: It is used to compare two values return Boolean values

Example

```
#Using'=', '<', '>' Operators

3==3

True

10<3

False
```

3>1

True

3.Logical Operators: They are used for combining conditional statements

Examples:

#Using ‘and’, ‘or’, ‘not’

1. and : Returns True if both statements are true
   example
   (6 > 3 and 7 < 10)
   True
2. or : Returns True if one of the statements is true
   example
   (6> 3 or 1> 10)
   True
3. not : Reverses the result
   example
   not(4 > 1)
   False

4.Bitwise Operators: It works on binary numbers

Like & : AND,| : OR, ^ : XOR, ~ : NOT, << : Left shift, >> : Right shift
Examples:

```
#Using ‘&’
a = 5        # 5 in binary is 0101
b = 3        # 3 in binary is 0011
result = a & b  # 0101 & 0011 = 0001 (1 in decimal)
print(result)   # Output: 1
#Using ‘|’
a = 5        # 0101
b = 3        # 0011
result = a | b  # 0101 | 0011 = 0111 (7 in decimal)
print(result)   # Output: 7

#Using’ Left Shift (<<)’
```

```
a = 5        # 0101
result = a << 1  # 0101 << 1 = 1010 (10 in decimal)
print(result)    # Output: 10
```

5.Assignment Operators: Used to assign value to variable and can include operations like =, +=, -=, *= and etc.

Examples:

```
#Using '(=)'

x = 53

print(x)  # Output: 53

#Using '+='

x = 6

x += 5   # Equivalent to x = x + 5

print(x)  # Output: 11

#Using '*='

x = 11

x *= 2   # Equivalent to x = x * 2

print(x)  # Output: 22
```

6.Membership Operators: It is used to test if a sequence (e.g., list, tuple) contains an item. The two membership operators are in and not in.

Example:

```
#Using'in','not'

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

print("banana" in fruits)    # Output: True

print("orange" in fruits)    # Output: False
```

Example with a string

text = "Hello, world!"

print("Hello" in text)   # Output: True

print("Python" in text)   # Output: False

7.Identity Operators: Identity operators in Python, is and is not, are used to compare the memory locations of two objects.

Example:

#Using 'is','is not'

x = 6

y = 6

print(x is y) # Output: True (small integers are cached, so they point to the same memory location)

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

Type casting is used to convert one data type into another make data compatible for certain operations, as different types cannot always interact directly. For example, one can't add a string and an integer without converting one to match the other's type.

Types of Type Casting in Python

1. Implicit Type Casting:In this python automatically convert one data type two another without any intervention from outside

Example:

num_int = 8   # Integer

num_float = 3.14 # Float

```
# Python automatically converts 'num_int' to float before adding

result = num_int + num_float

print(result)      # Output: 11.14

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

2. Explicit Type Casting: In explicit casting, also known as type conversion, you manually convert a value from one type to another using built-in functions like int(), float(), and str().

   ```
   # Converting string to integer

   age_str = "25"

   age_int = int(age_str)

   print(age_int)      # Output: 25

   print(type(age_int)) # Output: <class 'int'>


   # Converting integer to float

   num = 8

   num_float = float(num)

   print(num_float)    # Output: 8.0


   # Converting integer to string

   number = 50

   number_str = str(number)
   ```

```
print(number_str)   # Output: "50"

print(type(number_str)) # Output: <class 'str'>



# Converting string to list

name = "Ayush"

name_list = list(name)

print(name_list)    # Output: ['A', 'y', 'u', 's', 'h']
```

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

Conditional statements in Python allow us to execute certain blocks of code based on whether a given condition is **True** or **False**. These statements help control the flow of execution depending on logical conditions, enabling decision-making in programs.

Types of Conditional Statements in Python

1. **if statement**: Executes a block of code if the condition is true.
2. **else statement**: Executes a block of code if the condition in the if statement is false.
3. **elif statement**: Stands for "else if" and allows checking multiple conditions, executing a block of code if the specific condition is true.

Syntax:

```
if condition:
   # code block if condition is True
elif another_condition:
   # code block if another_condition is True
else:
   # code block if no conditions are True
```

Examples:

1. Using *if* statement

The if statement checks a single condition. If the condition evaluates to True, the code block inside the if statement runs.

```
age = 20
if age >= 18:
   print("You are an adult.")
```

Output:You are an adult.

Here, the condition age >= 18 is True, so the message "You are an adult." is printed.

2. Using *if* and *else* statement

You can use the else statement to specify a block of code that will execute when the condition in the if statement is False.

```
age = 16
if age >= 18:
   print("You are an adult.")
else:
   print("You are a minor.")
```

Output:You are a minor.

Since age is less than 18, the condition evaluates to False, and the code inside else executes.

3. Using if, elif, and else statements

The elif statement allows checking multiple conditions. If the first condition is False, it moves on to the next condition in elif.

```
score = 85

if score >= 90:
    print("Grade: A")
elif score >= 75:
    print("Grade: B")
else:
    print("Grade: C")
```

Output:

```
Grade: B
```

In this case, the score is 85, which is greater than 75 but less than 90, so the block under elif executes.

Nested Conditional Statements

You can also nest conditional statements inside each other to check for more complex conditions.

```
age = 20
has_permission = True

if age >= 18:
    if has_permission:
        print("You can enter.")
    else:
        print("You cannot enter without permission.")
else:
    print("You are not allowed to enter.")
```

Output:

```
You can enter.
```

In this example, the program checks if the person is an adult first. If they are, it then checks if they have permission.

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

In Python, loops are used to repeat a block of code multiple times based on a condition. Loops allow you to automate repetitive tasks efficiently. There are mainly **two types of loops** in Python:

for loop

while loop

Both loops allow you to iterate over a sequence (like a list, string, or range) or repeat a block of code until a condition is met.

1. for Loop

The for loop is used to iterate over a sequence (like a list, tuple, string, or range). It executes the block of code for each item in the sequence.

Syntax:

```
for item in sequence:
    # code block
```

Example:

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

Output:

```
apple
banana
cherry
```

Here, the for loop iterates over each element in the fruits list and prints each item.

Use Case:

Iterating over lists or tuples to perform operations on each element.

Iterating over a range of numbers to repeat an operation multiple times.

Example with range():

```
for i in range(5):
    print(i)
```

Output:

```
0
1
2
3
4
```

The range(5) generates numbers from 0 to 4, and the for loop prints each number.

2. while Loop

The while loop is used to repeatedly execute a block of code as long as a given condition is True. The condition is evaluated before each iteration.

Syntax:

```
while condition:
    # code block
```

Example:

```
counter = 0
while counter < 5:
    print(counter)
    counter += 1
```

Output:

```
0
1
```

```
2
3
4
```

In this example, the loop continues until counter becomes 5. The value of counter is incremented by 1 on each iteration.

Use Case:

When the number of iterations is not known and depends on a condition (e.g., waiting for user input or until a specific condition is met).

Infinite loops, where the condition is intentionally set to always be true until manually stopped.

Example of an infinite loop:

```
while True:
    user_input = input("Enter 'exit' to quit: ")
    if user_input == 'exit':
        break
    print("You entered:", user_input)
```

This loop continues until the user types 'exit', at which point the break statement exits the loop.

Key Differences:

for loop is ideal for iterating over a known sequence of items, such as a list or range, or when the number of iterations is known beforehand.

while loop is best used when the condition is more dynamic or depends on an external factor, and you don’t know the exact number of iterations upfront.

Nested Loops

Python also supports nested loops, which means you can have a loop inside another loop.

Example:

```
for i in range(3):   # Outer loop
    for j in range(2):  # Inner loop
        print(f"i = {i}, j = {j}")
```

Output:

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

This example demonstrates a nested for loop, where the outer loop runs 3 times, and for each iteration of the outer loop, the inner loop runs 2 times.