Double-click (or enter) to edit

Ques.1. Explain the key features of Python that make it a popular choice of programming. Ans.1 Python is a multipurpose and powerful language which offers a wide range of features and capabilities. For the features of simplicity, flexibility, and ease of use Python becomes popular and also widely used for developing software applications and web applications. Some of the important features which make the Python as popular programming language in many domains are as follows:

1. Free and Open Source It is a free open-source platform; which means that anyone can use it, modify it, and distribute it without any restrictions. This makes it an ideal choice for developers and programmers who want to build software applications without bearing high costs.
2. Easy to learn and code Python is a high-level programming language. It is very easy to learn and code with a minimal effort as compared to other languages like C, C++, Java, etc. Anybody can learn Python basics in a few hours or days.

```
#For example, a simple Python program to add two numbers is as follows:


a = 10
b = 20
print(a+b)
```

```
30
```

We write this program within three lines. But, in Java, C++ and C, we have to write more lines. That is why Python is known as an easy and precise language.

3. Object-Oriented Language Python is an object-oriented language. That means Python supports concepts of classes, inheritance, encapsulation, etc. This makes it easy to write code that is reusable and easy to maintain.
4. High-Level Language Python is a high-level general purpose language. This means Python is closer to human language than machine language. For which this is easy to write the code and maintain.
5. Python is a Cross Platform Language Python is also a cross-platform language. This means if we have Python code for Windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.
6. Interpreted Language One of the important features of Python is that it is an interpreted language, which means that the code is executed line by line. This makes the debugging process easier, as errors can be caught and fixed in real time.
7. GUI Programming Support Python has a wide range of GUI libraries such as PyQt5, PyQt4, wxPython, or Tk , which makes it easy to develop graphical user interfaces for applications.
8. Databases Support Python has built-in support for databases such as MySQL, PostgreSQL, and SQLite. This makes it easy to build applications that require data storage and retrieval.
9. Large Standard Library Python has a large standard library that includes modules and functions like NumPy, Pandas, Tensorflow, etc. for different tasks like web development, networking, data processing and many more. So you do not need to write your own code for every single thing.
10. Dynamically Typed Language Python is a dynamically-typed language. That means declaring the type of a variable is not needed. It is decided at run time not in advance.

Double-click (or enter) to edit

```
#For example, let us declare an integer number 10 for a variable a
a = 10
type(a) #Where as in any other language like C, we declare it as int a = 10
```

```
int
```

Ques2. Describe the role of predefined kewords in Python and provide examples how the are used in a program.

Ans.2. The reserved words that hold some predefined meaning in Python is known as Python Keywords. These keywords can not to be used as any variable names, any function names or any kind of identifier. The purpose of this predefined keyword is to define the structure and syntax of the code. Some of the most commonly used Python keywords are as follows:

and: A logical operator that returns True if both of its operands are True, and False otherwise. as: Used to create an alias for a variable or module. assert: Used to raise an assertion error if the given expression is False. break: Used to exit a loop or switch statement. class: Used to define a new class. continue: Used to skip the rest of the current iteration of a loop and continue to the next iteration. def: Used to define a new

function. del: Used to delete a variable or attribute. elif: Used to add an additional condition to an if statement. else: Used to specify code that should be executed if none of the conditions in an if statement are met. except: Used to handle exceptions. finally: Used to specify code that should be executed regardless of whether an exception is raised or not. for: Used to iterate over a sequence of values. from: Used to import specific parts of a module. global: Used to declare a global variable. if: Used to create conditional statements. import: Used to import modules. in: Used to check if a value is present in a sequence. is: Used to test for object identity. lambda: Used to create anonymous functions. not: A logical operator that returns True if its operand is False, and False otherwise. or: A logical operator that returns True if either of its operands is True, and False otherwise. pass: Used as a placeholder for a statement that will be added later. raise: Used to raise an exception. return: Used to return a value from a function. try: Used to handle exceptions. while: Used to create loops.

Giving an example of some keywords which were taught yet

```
#Example of some keywords used in Python
#printing of pwskills if the variable a is greater than 5 and b is less 10
#use of and
a = 7
b = 5
if a > 5 and b < 10:
    print("pwskills")
```

```
pwskills
```

```
#printing of the number from 1 to 5
#use of while
n=5
i=1
while i<n:
  print(i)
  i=i+1
```

```
1
2
3
4
```

```
#printing the number from 1 to 5
#use of if, else, break
n=5
i=1
while i<n:
  print(i)
  i=i+1
  if i==4:
    print("loop is broken")
    break
else:
  print("Statement will be executed when the while loop will run")
```

```
1
2
3
loop is broken
```

```
#printing of even numbers from 1 to 10
#use of for
for i in range(1,10):
  if i%2 == 0:
    print(i)
```

```
2
4
6
8
```

```
#printing all of the numbers from 1 to 10 skipping number 7
#use of continue
for i in range(1, 10):
    if i == 7:
        continue
    print(i)
```

```
1
2
3
4
5
6
8
9
```

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

Ans3. Mutable obejects are those objects which can be changed or modified after creation where as immutable objects can not be changed once it is created in Python programming. Immutable objects shift their memory address whenever they are updated.

The Mutable and immutable data types in Python are as follows:

Mutable Lists Dictionaries Sets

Immutable Numbers (Integer, Float, Complex, Decimal, Rational & Booleans) Tuples Strings

```
# A program to show that a list is a mutable data type
# Creating a list

list1 = ['Python', 'PWSkills', 10, False, 2.2]
print("The original list: ", list1)

# After Changing the value at index 3 of the list
list1[3]='True'
print("The modified list: ", list1)
```

```
The original list:  ['Python', 'PWSkills', 10, False, 2.2]
The modified list:  ['Python', 'PWSkills', 10, 'True', 2.2]
```

```
#A program to show that integer is a immutable data type
number = 1234
number[2]=5
print(number)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-6d5ab5fdef1f> in <cell line: 3>()
      1 #A program to show that integer is a immutable data type
      2 number = 1234
----> 3 number[2]=5
      4 print(number)

TypeError: 'int' object does not support item assignment
```

Double-click (or enter) to edit

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

Ans4. Operators are differnt symbol which are used to perform operations on values and variables. In Python there are 7 types of operator such as 1.Arithmetic Operators 2.Comparison (Relational) Operators 3.Assignment Operators 4.Logical Operators 5.Bitwise Operators 6.Membership Operators 7.Identity Operators Lets explain one by one as follows

Arithmetic Operators Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, division, modulus, exponentiation, and floor division. Operator Name Example

- Addition 10 + 20 = 30
- Subtraction 20 – 10 = 10
- Multiplication 10 * 20 = 200 / Division 20 / 10 = 2 % Modulus 22 % 10 = 2
- - Exponent 4**2 = 16 // Floor Division 9//2 = 4

```
#Example of Arithmetic operators
a = 31
b = 20
# Addition
print ("a + b : ", a + b)
# Subtraction
print ("a - b : ", a - b)
# Multiplication
print ("a * b : ", a * b)
# Division
print ("a / b : ", a / b)
# Modulus
print ("a % b : ", a % b)
# Exponent
print ("a ** b : ", a ** b)
# Floor Division
print ("a // b : ", a // b)
```

```
a + b :  51
a - b :  11
a * b :  620
a / b :  1.55
a % b :  11
a ** b :  671790528819082282036142601601
a // b :  1
```

Assignment Operators

Assignment operators are used to assign values to the variables.

Operator Name Example = Assignment operator a = 5
+= Addition Assignment a += 5 (Same as a = a + 5) -= Subtraction Assignment a -= 5 (Same as a = a - 5) *= Multiplication Assignmeta *= 5 (Same as a = a * 5) /= Division Assignment a /= 5 (Same as a = a / 5) %= Remainder Assignment a %= 5 (Same as a = a % 5) **= Exponent Assignment a **= 2 (Same as a = a * 2) //= Floor Division Assignment a //= 3 (Same as a = a // 3)

```
#Example of Assignment Opertaors in Python
a = 40
b = 5
print('a=b:', a==b)
print('a+=b:', a+b)
print('a-=b:', a-b)
print('a*=b:', a*b)
print('a%=b:', a%b)
print('a**=b:', a**b)
print('a//=b:', a//b)
```

```
a=b: False
a+=b: 45
a-=b: 35
a*=b: 200
a%=b: 0
a**=b: 102400000
a//=b: 8
```

Comparison operators Comparison operators are used when we need to compare two values/variables and it returns a boolean value, either True or False

Operator Name Example == Equal 4 == 5 is not true. != Not Equal 4 != 5 is true.

> Greater Than 4 > 5 is not true

< Less Than 4 < 5 is true = Greater than or Equal to 4 >= 5 is not true. <= Less than or Equal to 4 <= 5 is true.

```
#Program of comprarision operators
a = 10

b = 5

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

```
a == b = False
a != b = True
a > b = True
a < b = False
a >= b = True
a <= b = False
```

Start coding or generate with AI.

Logical Operators Logical opertaors are used when we want to make some decision under certain conditions. There are three logical operators in Python. They are "and", "or" and "not". All must be in written in lowercase.

"and" operator In and operator both the condition of the expression must be true. If any one condition is false then the expression returns false.

The truth table of and operator :

a b a and b F F F F T F T F F T T T

"or" operator This operator is the contrast of and operator. That means any one of the condition of an expression is true, then the expresiion returns true.

The truth table of or operator :

a b a or b F F F F T T T F T T T T

"not" operator This is a unary operator, means that it takes only one operand for the logical operation and returns the complementary of the Boolean value of the operand. For example, if we give false as an operand to the not keyword we get true as the return value.

Logical "not" Operator Truth Table

a not (a) F T T F

```
#Use of logical operators
x = 20
y = 40
print("x > 0 and x < 10:",x > 0 and x < 10)
print("x > 0 and y > 10:",x > 0 and y > 10)
print("x > 10 or y > 10:",x > 10 or y > 10)
print("x%2 == 0 and y%2 == 0:",x%2 == 0 and y%2 == 0)
print ("not (x+y<25):", not (x+y)>25)
```

```
x > 0 and x < 10: False
x > 0 and y > 10: True
x > 10 or y > 10: True
x%2 == 0 and y%2 == 0: True
not (x+y<25): False
```

Bitwise Operators

Python bitwise operators are used to perform bitwise calculations only on integers. The integers are first converted into binary and then operations are performed on each bit. The result returned in decimal format. There are 6 major bitwise operator in Python such as

AND represented as & - operator takes two equal-length bit patterns as parameters. The two-bit integers are compared. If the bits in the compared positions of the bit patterns are 1, then the resulting bit is 1. If not, it is 0.

OR represented as | - It takes two equivalent length bit designs as boundaries; if the two bits in the looked-at position are 0, the next bit is zero. If not, it is 1.

NOT represented as ~ - The bitwise NOT operator only requires one parameter. It flips all of the bits of a number given to implement logical negation upon it.

XOR represented as ^ - In this operator, when one of the bits is 1, another is 0, it returns true; otherwise, it returns false.

Left Shift represented as<< - Here the individual bits of the number are shifted to the right, and the gaps on the left are filled with 0 (or 1 if the number is negative). The result is similar to dividing a value by a power of 2.

Right Shift represented as>> - In this operation, the individual bits of the integer to the left are filled with 0 on the voids to the right. The result is similar to multiplying a value by a power of 2.

```
#Program using Bitwise operators
x = 10
y = 20
# Bitwise AND operation
print("x & y =", x & y)
#Bitwise OR operation
print("x | y =", x | y)
#Bitwise NOT operation  print("~y =", ~ y)
#Bitwise XOR operation
print("x ^ y =", x ^ y)
#Bitwise right shift operator
print("x >> 1 =", x >> 3)
print("y >> 1 =", y >> 3)
#Bitwise left shift operator
print("x << 1 =", x << 1)
print("y << 1 =", y << 1)
```

```
x & y = 0
x | y = 30
x ^ y = 30
x >> 1 = 1
y >> 1 = 2
x << 1 = 20
y << 1 = 40
```

Membership Operators

Membership operators are operators used to check whether a value exists in a sequence, such as a list, tuple, or string. There are two types of membership operators used in Python, in, not in. The “in” membership operator returns True if a specified value is found within a sequence. Whereas the “not in” operator returns True if a specified value is not found within a sequence.

```
#example of using membership operator

list1 = ["apple", "orange", "banana"]

print("banana" in list1)

# returns True because a sequence with the value "banana" is in the list

print("grapes" not in list1)

# returns True because a sequence with the value "grapes" is not in the list
```

```
True
True
```

Identity operators

In Python identity operators are used to compare the memory addresses of two objects, determining whether two variables or objects reference the same memory location. There are two primary identity operators in Python:

is operator: Returns True if two variables refer to the same memory location. is not operator: Returns True if two variables do not refer to the same memory location.

```
#Example of identity operator
x = [1, 2, 3]
y = x
result = x is y
print(result) # result is true because both x and y reference the same memory location

a = "hello"
b = "world"
result = a is not b #is not operator
print(result) # print the result true because a and b reference different memory locations
```

```
True
True
```

Ques. 5. Ans.5 Type casting is a process that converts the data type of a variable to another data type. In Python Type casting can be two types, namely implicit and explicit.

Implicit data type is done when Python interpreter automatically converts one data type to another. For example, when we add one integer to a float number, Python interpreter automatically converts it to a float.

```
# Implicit type casting
x = 2
y = 3.5
z=x+y
print(z)
print(type(x))
```

```
5.5
<class 'int'>
```

In explicit type casting we need to convert manually one data type to another by using int(), float() and str() built-in functions. For example, if we want to add a string to a number, we will use explicit conversion.

```
# Explicit type casting
# Converting the string into an integer number.
string1 = "44"
num1 = 44
strnum = int(string1)
sum = num1 + strnum
print("The Sum of both the numbers is: ", sum)
```

```
The Sum of both the numbers is:  88
```

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

Ans6. Conditional statements are used to control the flow of a block of code based on certain given condition. This means, condition allow us to make decisions based on the values of variables or the result of comparisons. if, else and elif conditional statements are used in Python.

if statement : The if statement allow to execute a block of code if a certain condition is true. Else statement: The else statement is executed when the block of code in the if condition is false. Elif statement: The elif statement in Python enables you to check multiple conditions and execute specific blocks of statements if the previous conditions were false.

For example

```
#Use of if,elif and else condition
num = int(input("Enter a number:"))
if num > 0:
  print("The number is positive.")
elif num==0:
    print("The number is equal to 0." )
else:
    print("The number is negetive")
```

```
Enter a number:0
The number is equal to 0.
```

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

Ans.7. A loop is a control flow statement that repeats multiple times as long as some condition is met. Loops are necessary for operations, when we need repetitive execution of some statements, such as iterating through a Python list of items or doing calculations many times. There are 3 types of loop in Python, those are for loop, while loop and nested loop. for loop: A for loop is a control flow statement in Python that allows you to iterate over a sequence of elements such as a list, tuple, or string. Each time of iteration, the loop variable in Python will take on the value of the next item in the sequence.

```
#iterate through the list using for loop
fruits = ["Orange","Mango", "Litchi"]
for i in fruits:
  print(i)
```

```
Orange
Mango
Litchi
```

While Loop While loop is used to execute a set of statements in Python code as long as a condition is true.

```
#use of while loop
num = 1
while num<5:
  print(num)
  num=num+1
```

```
1
2
3
4
```

Nested Loop If a loop exists inside the body of another loop, it is called a nested loop. This indicates that the inner loop is executed within each outer loop iteration.

```
i=2
print ("Prime number between 1 to 100")
while(i < 100):
 j = 2
 while(j <= (i/j)):
  if not(i%j): break
  j = j + 1
 if (j > i/j) : print (i, " is prime")
 i = i + 1
```

```
Prime number between 1 to 100
2  is prime
3  is prime
5  is prime
7  is prime
11  is prime
13  is prime
17  is prime
19  is prime
23  is prime
29  is prime
31  is prime
37  is prime
41  is prime
43  is prime
47  is prime
53  is prime
59  is prime
61  is prime
67  is prime
71  is prime
73  is prime
79  is prime
83  is prime
89  is prime
97  is prime
```