Creating Python Programs : Input and Output Statements, Control statements(Branching, Looping, Conditional Statement, Exit function, Difference between break, continue), Defining Functions, default arguments, Errors and Exceptions.

Taking Input from the user

Sometimes a developer might want to take input from the user at some point in the program. To do this Python provides an input() function.

Syntax:

```
input('prompt')
```

where, prompt is a string that is displayed on the string at the time of taking input.

Example 1: Taking input from the user with a message.

# Taking input from the user 	
name = input("Enter your name: ")
 	
# Output 	
print("Hello, " + name)
	Enter your name: Amit
	
	Hello, Amit

Example 2: By default **input()** function takes the user's input in a string. So, to take the input in the form of int, you need to use **int()** along with input function.

# Taking input from the user as integer 	
num = int(input("Enter a number: "))
 	
add = num + 1
 	
# Output 	
print(add)
	Enter a number: 25
	
	26

Displaying Output

Python provides the print() function to display output to the console.

# Python program to demonstrate 	
# print() method 	
print("Amit")
	Amit

Control statements:

If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation.

if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

Program find even number

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

```
enter the number?10
Number is even
```

Program to print the largest of the three numbers.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

Output

```
Enter a? 100
Enter b? 120
Enter c? 130
c is largest
```

if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
```

```
Enter your age? 90
You are eligible to vote !!
```

print("You are eligible to vote !!"); **else**: **print**("Sorry! you have to wait !!");	

Program to check whether a number is even or not.

num = int(input("enter the number?")) **if** num%2 == 0: **print**("Number is even...") **else**: **print**("Number is odd...")	enter the number?10 Number is even

elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

number = int(input("Enter the number?")) **if** number==10: **print**("number is equals to 10") **elif** number==50: **print**("number is equal to 50"); **elif** number==100: **print**("number is equal to 100"); **else**: **print**("number is not equal to 10, 50 or 100");	Enter the number?15 number is not equal to 10, 50 or 100
marks = int(input("Enter the marks? "))	

```
if marks > 85 and marks <= 100:
   print("Congrats ! you scored grade A ...")
lif marks > 60 and marks <= 85:
   print("You scored grade B + ...")
lif marks > 40 and marks <= 60:
   print("You scored grade B ...")
lif (marks > 30 and marks <= 40):
   print("You scored grade C ...")
else:
   print("Sorry you are fail ?")
```

Loops

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.

Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).
4. There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the

	code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

for loop

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

Iterating string using for loop

```
str = "WEBSOL"
for i in str:
    print(i)
```

```
WEBSOL
```

Program to print the table of the given number .

```
list = [1,2,3,4,5,6,7,8,9,10]
n = 5
for i in list:
    c = n*i
    print(c)
```

```
5
10
15
20
25
30
35
40
45
50
```

Program to print the sum of the given list.

```
list = [10,30,23,43,65,12]
sum = 0
```

```
The sum is: 183
```

for i in list: sum = sum+i print("The sum is:",sum)	

For loop Using range() function

The range() function

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:

1. range(start,stop,step size)
   - The start represents the beginning of the iteration.
   - The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
   - The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Program to print numbers in sequence.	
for i **in** range(10): **print**(i,end = ' ')	0 1 2 3 4 5 6 7 8 9
Program to print even number using step size in range().	
n = int(input("Enter the number ")) **for** i **in** range(2,n,2): **print**(i)	Enter the number 20 2 4 6 8 10 12 14 16 18

Nested for loop in python

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop.

Program to number pyramid.	

rows = int(input("Enter the rows"))
for i **in** range(0,rows+1):
  for j **in** range(i):
  print(i,end = '')
  print()
	1
	22
	333
	4444
	55555

While loop

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

i = 1
while i < 6:
  print(i)
  i += 1
	1 2 3 4 5

use of break and continue

In Python, `break` and `continue` statements can alter the flow of a normal loop. Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

The `break` and `continue` statements are used in these cases.

break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

continue statement

The continue statement in Python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.The continue statement in Python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

Exit

`exit()` is defined in `site.py` and it works only if the site module is imported so it should be used in the interpreter only. It is like a synonym of `quit()` to make the Python more user-friendly. It too gives a message when printed:

Break	Continue	Exit

str = "python"
for i **in** str:
  if i == 'o':
  break
  print(i);
	i = 0
	while(i < 10):
	  i = i+1
	  if(i == 5):
	  continue
	  print(i)
		i in range(10):
		# If the value of i becomes
		# 5 then the program is forced to exit
		if i == 5:
		  print(exit)
		  exit()
		print(i)

Function

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.

Python provide us various inbuilt functions like **range()** or **print()**. Although, the user can create its functions, which can be called user-defined functions.

There are mainly two types of functions.

- **User-define functions** - The user-defined functions are those define by the **user** to perform the specific task.
- **Built-in functions** - The built-in functions are those functions that are **pre-defined** in Python.

Advantage of Functions in Python

There are the following advantages of Python functions.

- Using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call Python functions multiple times in a program and anywhere in a program.
- We can track a large Python program easily when it is divided into multiple functions.
- Reusability is the main achievement of Python functions.

- However, Function calling is always overhead in a Python program.

Creating a Function

Python provides the **def** keyword to define the function. The syntax of the define function is given below.

Let's understand the syntax of functions definition.

- The **def** keyword, along with the function name is used to define the function.
- The identifier rule must follow the function name.
- A function accepts the parameter (argument), and they can be optional.
- The function block is started with the colon (:), and block statements must be at the same indentation.
- The **return** statement is used to return the value. A function can have only one **return**

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

return statement

The return statement is used at the end of the function and returns the result of the function. It terminates the function execution and transfers the result where the function is called. The return statement cannot be used outside of the function.

```
#function definition
def hello_world():
    print("hello world")
# function calling
hello_world()
```

output :

```
hello world
```

```
# Defining function
def sum():
    a = 10
    b = 20
    c = a+b
    return c
# calling sum() function in print state
ment
print("The sum is:",sum())
```

output :

```
The sum is: 30
```

```
#Python function to calculate the sum of two var
iables
#defining the function
def sum (a,b):
    return a+b;

#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))

. #printing the sum of a and b
. print("Sum = ",sum(a,b))
```

```
# Defining function
def sum():
    a = 10
    b = 20
    c = a+b
# calling sum() function in print state
ment
print(sum())
```

Errors and Exceptions in Python

Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when the some internal events occur which changes the normal flow of the program.

Two types of Error occurs in python.

1. Syntax errors
2. Logical errors (Exceptions)

Syntax errors

When the proper syntax of the language is not followed then syntax error is thrown.

```
# initialize the amount variable
amount = 10000

# check that You are eligible to
# purchase Dsa Self Paced or not
if(amount>2999)
        print("You are eligible to purchase Dsa Self Paced")
```

```
  File "/home/ac35380186f4ca7978956ff46697139b.py", line 4
    if(amount>2999)
                  ^
SyntaxError: invalid syntax
```

It returns a syntax error message because after if statement a colon : is missing. We can fix this by writing the correct syntax.

logical errors(Exception)

When in the runtime an error occurs after passing the syntax test is called exception or logical type. For example, when we divide any number by zero then `ZeroDivisionError` exception is raised, or when we import a module that does not exist then `ImportError` is raised.

```
# initialize the amount variable
marks = 10000

# perform division with 0
a = marks / 0
```

```
print(a)
```

```
Traceback (most recent call last):
  File "/home/f3ad05420ab851d4bd106ffb04229907.py", line 4, in <module>
    a=marks/0
ZeroDivisionError: division by zero
```

Some of the common built-in exceptions are other than above mention exceptions are:

Exception	Description
IndexError	When the wrong index of a list is retrieved.
AssertionError	It occurs when assert statement fails
AttributeError	It occurs when an attribute assignment is failed.
ImportError	It occurs when an imported module is not found.
KeyError	It occurs when the key of the dictionary is not found.
NameError	It occurs when the variable is not defined.
MemoryError	It occurs when a program run out of memory.
TypeError	It occurs when a function and operation is applied in an incorrect type.

Error Handling

When an error and an exception is raised then we handle that with the help of Handling method.

- **Handling Exceptions with Try/Except/Finally**
  We can handle error by Try/Except/Finally method.we write unsafe code in the try, fall back code in except and final code in finally block.

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

```
#The try block will generate an error, because x is not defined:

try:
  print(x)
except:
  print("An exception occurred")
```

Output : `An exception occurred`