

Module-2

Conditionals & Loops

Python

1. **Conditionals**

In Python, "conditionals" refer to statements that perform different actions depending on whether a certain condition is true or false.

* **If Statement**

The if statement is used to check a condition, and execute a specific code block if the condition is True.

**Example:**

val2=int(input('Enter any value -> '))

if val2<50:

print('Value is less than 50 and =',val2)

**Output:**

Enter any value -> 45

Value is less than 50 and = 45

* **If-else**

The if-else statement is used to check a condition, and execute a specific code block depending on whether the condition is True or False.

**Example:**

val3 = int(input('Enter any value -> '))

if(val3<=100):

print(f'{val3} is less than or equal to 100.')

else:

print(f'{val3} is greater than 100.')

**Output:**

Enter any value -> 4124

4124 is greater than 100.

* **If-elif-else**

The if-elif-else statement is used to check multiple conditions and execute different code blocks based on the outcome of those conditions.

**Example:**

val4 = int(input('Enter an integer -> '))

if val4<50:

print(f'{val4} is less than 50')

elif val4>50:

print(f'{val4} is greater than 50')

else:

print(f'{val4} is equal to 50')

**Output:**

Enter an integer -> 456

456 is greater than 50

* **Direction of execution of code**

Direction of execution of code in Python is from up to down.

**Example:**

salary=int(input('Enter your salary ->'))

if salary<1000:

print('you are sooo rich! buy a CAR.')

elif salary<500:

print('you are rich! you can buy BIKE.')

elif salary<100:

print('your salary is average! tou can buy BICYCLE.')

else:

print('you are poor! you should do SAVING.')

**Output:**

Enter your salary ->63600

you are poor! you should do SAVING.

**Explanation:**

Enter 90, looks like output should be 'saving', but actually it is 'car'.

Because, since execution is from up to down so, first if condition will be checked and 90<1000 is True, so if block will get executed, and we will get 'car' as output.

**Let's build a proper logic for above problem:**

**Example:**

salary=int(input('Enter your salary '))

if salary<100:

print('you are poor! you should do SAVING.')

elif salary<500:

print('your salary is average! tou can buy BICYCLE.')

elif salary<1000:

print('you are rich! you can buy BIKE.')

else:

print('you are sooo rich! buy a CAR.')

**Output:**

Enter your salary 56000

you are sooo rich! buy a CAR.

**Explanation:**

Enter 90, and output is 'saving'.

Because, execution is from up to down so, first if condition will be checked and 90<100 is True, so if block will get executed, and we will get 'saving' as output.

* **Nested if-else**

In Python, it is possible to use an if-else statement within another if-else statement to create a nested if-else structure. This allows for more complex conditions to be checked and different actions to be taken based on multiple conditions.

**Example:**

salary=int(input('Enter your salary '))

if(salary>1000):

print('good salary')

if(salary>50000):

print('Go for BMW!')

if(True):

print('Congratulations on buying, BMW!')

else:

print('Fake')

elif(salary>10000):

print('You can buy, Maruti!')

elif(salary>5000):

print('Oh! you can go for Nano!')

else:

print('Oops, Sorry!')

else:

print('Your salary is less than or equal to 1000.')

**Output:**

Enter your salary 576576

good salary

Go for BMW!

Congratulations on buying, BMW!

* **Multiple Conditions**

In Python, the **AND** and **OR** operators are used to combine multiple Boolean expressions and create more complex conditions.

* **AND:**

The **AND** operator is used to combine two or more conditions, and returns True only if all the conditions are True.

**Example:**

if val1==10 and val2>20:

print('Valid')

else:

print('not valid')

**Output:**

not valid

* **OR:**

The **OR** operator is used to combine two or more conditions, and returns True if at least one of the conditions is True.

**Example:**

a, b = 10, 20

if a==20 or b>20:

print('Valid')

else:

print('not valid')

**Output:**

not valid

1. **Loops**

In Python, a loop is a control flow statement that allows a block of code to be executed repeatedly.

There are two main types of loops in Python: for loops and while loops.

* **While loop**

A while loop is a control flow statement that repeatedly executes a block of code as long as a given condition is true.

**Example:**

initial\_speed, final\_spped=0,5

#print at every stage your speed increases

while(initial\_speed<=final\_spped): #untill its True

print(initial\_speed)

initial\_speed=initial\_speed+1

**Output:**

0

1

2

3

4

5

* **While-else**

In Python, for and while loops can be used with an optional else clause. The else clause is executed after the loop completes normally, i.e., when the loop condition becomes false and the loop is not terminated by a break statement.

**Example:**

x = 0

while x < 5:

print(x)

x += 1

else:

print("The while loop completed successfully.")

**Output:**

0

1

2

3

4

The while loop completed successfully.

**Explanation:**

In this example, the while loop uses the variable x as the loop condition and it will execute 5 times.

The loop will print the value of x on each iteration and increment x by 1.

Since no break statement is used, the loop will complete execution normally and the else block will be executed, so the message "The while loop completed successfully." will be printed.

**Example when else block doesn’t gets executed:**

x = 0

while x < 5:

if x == 3:

# This statement raises an exception

raise ValueError("Invalid value encountered.")

print(x)

x += 1

else:

print("The while loop completed successfully.")

Output:

0

1

2

ValueError: Invalid value encountered.

**Explanation:**

In this example, the while loop uses the variable x as the loop condition and it will execute 5 times.

The loop will print the value of x on each iteration and increment x by 1.

If the value of x is equal to 3, the loop will raise an exception (ValueError) and the else block will not be executed.

* **For loop**

A for loop is a control flow statement that iterates over a sequence of elements, such as a list or a string.

**Example:**

string = 'Hello, Python lovers!'

for i in string:

print(i)

* **For-else**

In Python, for loops can be used with an optional else clause. The else clause is executed after the loop completes normally, i.e., when the loop condition becomes false and the loop is not terminated by a break statement.

**Example:**

numbers = [2, 4, 6, 8, 10]

for number in numbers:

print(number)

else:

print("All numbers have been printed.")

**Output:**

2

4

6

8

10

All numbers have been printed.

**Explanation:**

In this example, the loop will iterate over the list of numbers and print each number.

Since no break or continue statements are used, the loop will iterate over all the items in the sequence and then the else block will be executed, so the message "All numbers have been printed." will be printed'''

**Example when else block doesn’t gets executed:**

numbers = [2, 4, 6, 7, 8, 10]

for number in numbers:

if number == 7:

# This statement raises an exception

raise ValueError("Invalid number encountered.")

print(number)

else:

print("All numbers have been printed.")

**Output:**

2

4

6

ValueError: Invalid number encountered.

**Explanation:**

In this example, the loop will iterate over the list of numbers, and if the number is equal to 7, the loop will raise an exception (ValueError) and the else block will not be executed.'''

1. **Break and continue statements**

Both break and continue statements can be used with for loops and while loops.

* **Break Statement:**

The break statement is used to exit a loop or switch statement early. When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

**Example:**

string = 'Hey all coders here!'

for i in string:

if(i=='a'):

break

print(i)

else:

print('For loop is exhausted completely')

**Output:**

H

e

y

**Example:**

In this example, the loop will iterate over each character in the string 'Hey all coders here!', but as soon as it encounters 'a' character, the break statement will be executed and the loop will exit prematurely, so the string "Hey " will be printed.

* **Continue Statement**

The continue statement is used to skip the current iteration of a loop and continue with the next iteration. When a continue statement is encountered inside a loop, the current iteration of the loop is skipped and the program control resumes at the next iteration.

**Example:**

string = 'Hey all coders here!'

for i in string:

if(i=='a'): #'a'(try with 'u') is not printed, because its written after 'continue'

continue

print(i)

else:

print('For loop is exhausted completely')

**Output:**

H

e

y

l

l

c

o

d

e

r

s

h

e

r

e

!

For loop is exhausted completely

**Explanation:**

In this example, the loop will iterate over each character in the string 'Hey all coders here!', but when it encounters the character "a", the continue statement will be executed and the current iteration will be skipped.

This will cause the loop to print the string "Hey ll coders here!"'''

1. **Range() function**

The range() function is a built-in function that is used to create a sequence of numbers. The range() function takes one or more arguments, and returns an object that generates a sequence of numbers. The numbers generated by the range() function are typically used to control the number of iterations in a for loop.

**Example:**

for i in range(2,21,2):

print(i)

**Output:**

2

4

6

8

10

12

14

16

18

20

1. **Nested while-loop**

Nested while loops are a type of control flow statement in Python where a while loop is used inside another while loop. This allows you to repeatedly execute a block of code multiple times based on the conditions of both loops.

In a nested while loop, the inner loop is executed completely for each iteration of the outer loop. The inner loop's control variable and its termination condition are independent of the outer loop.

**Example:**

i = 0

while i < 5:

j = 0

while j <= i:

print("\*", end=" ")

j += 1

i += 1

print("")

**Output:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Explanation:**

In this example, the outer while loop uses the variable i as the loop condition and it will execute 5 times.

On each iteration of the outer while loop, the inner while loop uses the variable j as the loop condition and it will execute i+1 times.

The inner while loop is responsible for printing one line of \* and the outer while loop is responsible for printing a new line.

The end=" " in the print statement is used to prevent a newline character from being printed after each \*, resulting in the pattern being printed horizontally.

1. **Nested for-loop**

A nested for loop is a loop that is inside another loop. That is, a for loop can be used as part of the statement that creates another for loop. This allows you to iterate over multiple sequences in a single block of code.

**Example:**

for i in range(5):

for j in range(i + 1):

print("\*", end=" ")

print("")

**Output:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Explanation:**

In this example, the outer for loop iterates over the range of 5 and the inner for loop iterates over the range of i+1.

The inner for loop is responsible for printing one line of \* and the outer for loop is responsible for printing a new line.

The end=" " in the print statement is used to prevent a newline character from being printed after each \*, resulting in the pattern being printed horizontally.'''