Python if...elif...else and Nested if

Decision making is required when we want to execute a code only if a certain condition is satisfied. The `if…elif…else` statement is used in Python for decision making.

Python if Statement Syntax

```
if test expression:
    statement(s)
```

Here, the program evaluates the `test expression` and will execute statement(s) only if the text expression is `True`. If the text expression is `False`, the statement(s) is not executed. In Python, the body of the `if` statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end. Python interprets non-zero values as `True`. `None` and `0` are interpreted as `False`.

Python if Statement Flowchart

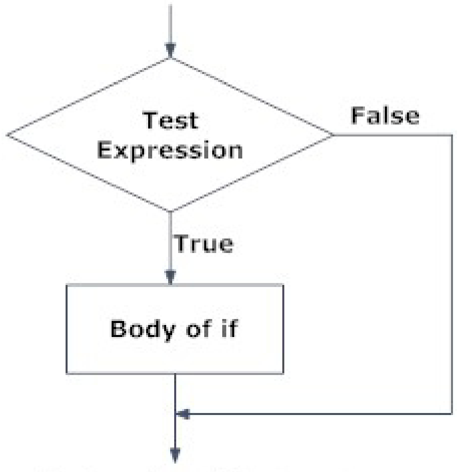


Fig: Operation of if statement

Example: Python if Statement

```

# In this program, user inputs a number.
# If the number is positive, we print an appropriate message
```

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
print("This is always printed")
```

Output 1

```
Enter a number: 3
Positive number
This is always printed
```

Output 2

```
Enter a number: -1
This is always printed
```

In the above example, `num > 0` is the test expression. The body of `if` is executed only if this evaluates to `True`. When user enters 3, test expression is true and body inside body of `if` is executed. When user enters -1, test expression is false and body inside body of `if` is skipped. The `print()` statement falls outside of the `if` block (unindented). Hence, it is executed regardless of the test expression. We can see this in our two outputs above.

Python if...else

Syntax of if...else

```
if test expression:
    Body of if
else:
    Body of else
```

The `if..else` statement evaluates `test expression` and will execute body of `if` only when test condition is `True`. If the condition is `False`, body of `else` is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

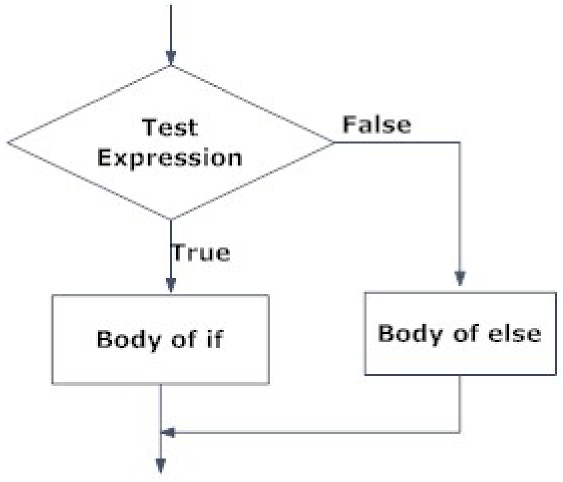


Fig: Operation of if...else statement

Example of if...else

```
# In this program, user input a number
# Program check if the number is positive or negative and display an appropriate message

num = float(input("Enter a number: "))
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 2
Positive or Zero
```

Output 2

```
Enter a number: -3
Negative number
```

In the above example, when user enters 2, the test epression is true and body of `if` is executed and `body` of else is skipped. When user enters -3, the test expression is false and body of `else` is executed and body of `if` is skipped.

Python if...elif...else

Syntax of if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The `elif` is short for else if. It allows us to check for multiple expressions. If the condition for `if` is `False`, it checks the condition of the next `elif` block and so on. If all the conditions are `False`, body of else is executed. Only one block among the several `if...elif...else` blocks is executed according to the condition. A `if` block can have only one `else` block. But it can have multiple `elif` blocks.

Flowchart of if...elif...else

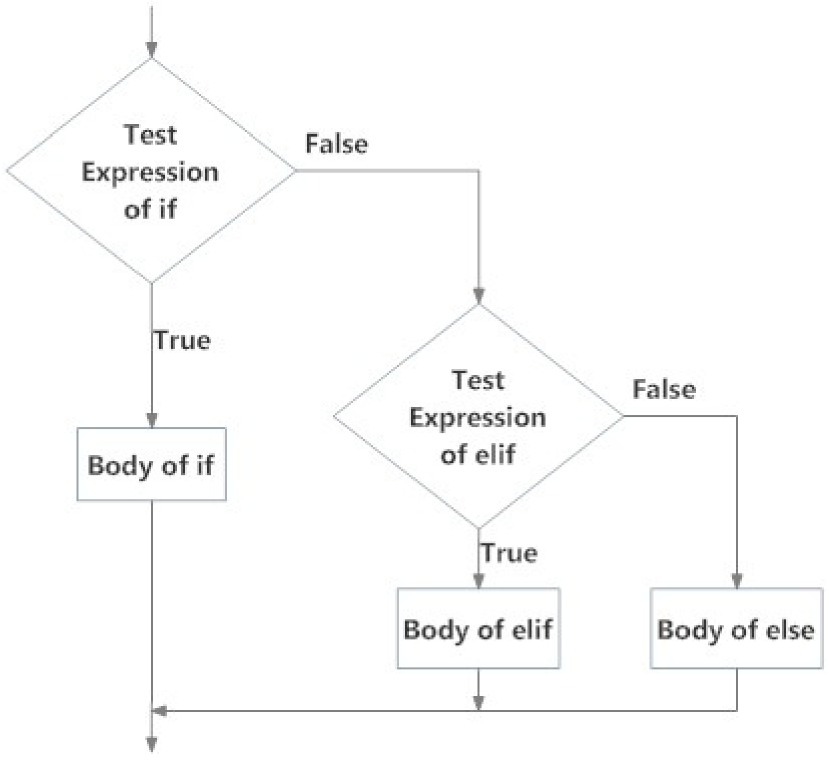


Fig: Operation of if...elif...else statement

Example of if...elif...else

```

# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message

num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 2
Positive number
```

Output 2

```
Enter a number: 0
Zero
```

Output 3

```
Enter a number: -2
Negative number
```

Python Nested if statements

We can have a `if...elif...else` statement inside another `if...elif...else` statement. This is called nesting in computer programming. In fact, any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

Python for Loop

The `for` loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
    Body of for
```

Here, `val` is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of `for` loop is separated from the rest of the code using indentation.

Flowchart of for Loop

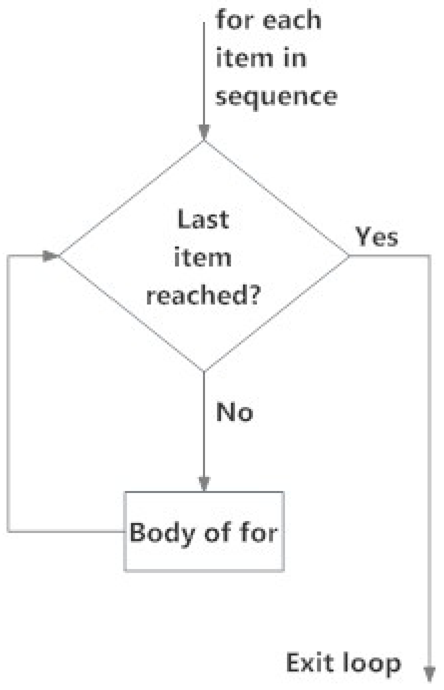


Fig: operation of for loop

Example: Python for Loop

```
# Program to find
# the sum of all numbers
# stored in a list

# List of numbers
numbers = [6,5,3,8,4,2,5,4,11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

# print the sum
print("The sum is",sum)
```

Output

```
The sum is 48
```

The range() function

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers). We can also define the `start`, `stop` and `step size` as `range(start,stop,step size)`. `step size` defaults to 1 if not provided. This function does not store all the values in memory, it would be inefficient. So it remembers the `start`, `stop`, `step size` and generates the next number on the go. To force this function to output all the items, we can use the function `list()`.

The following example will clarify this.

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,8))
[2, 3, 4, 5, 6, 7]
>>> list(range(2,20,3))
```

```
[2, 5, 8, 11, 14, 17]
```

We can use the `range()` function in `for` loops to iterate through a sequence of numbers. It can be combined with the `len()` function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate
# through a list
# using indexing

# List of genre
genre = ['pop','rock','jazz']

# iterate over the list using index
for i in range(len(genre)):
    print("I like",genre[i])
```

Output

```
I like pop
I like rock
I like jazz
```

for loop with else

A `for` loop can have an optional `else` block as well. The `else` part is executed if the items in the sequence used in `for` loop exhausts. `break` statement can be used to stop a `for` loop. In such case, the `else` part is ignored. Hence, a `for` loop's `else` part runs if no break occurs.

Here is an example to illustrate this.

```
# Program to show
# the control flow
# when using else block
# in a for loop

# a list of digit
list_of_digits = [0,1,2,3,4,5,6]

# take input from user
input_digit = int(input("Enter a digit: "))
```

```
# search the input digit in our list
for i in list_of_digits:
    if input_digit == i:
        print("Digit is in the list")
        break
else:
    print("Digit not found in list")
```

Output 1

```

Enter a digit: 3
Digit is in the list
```

Output 2

```

Enter a digit: 9
Digit not found in list
```

Here, we have a list of digits from 0 to 6. We ask the user to enter a digit and check if the digit is in our list or not. If the digit is present, `for` loop breaks prematurely. So, the `else` part does not run. But if the items in our list exhausts (digit not found in our list), the program enters the `else` part.

Python while Loop

The `while` loop in Python is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop

```
while test_expression:
    Body of while
```

In `while` loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues untill the `test_expression` evaluates to `False`.

In Python, the body of the `while` loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

Flowchart of while Loop

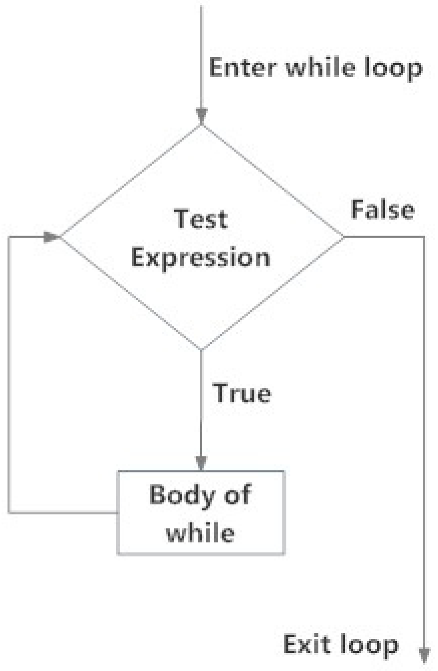


Fig: operation of while loop

Example: Python while Loop

```
# Program to add natural
# numbers upto n where
# n is provided by the user
# sum = 1+2+3+...+n

# take input from the user
n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is",sum)
```

Output

```
Enter n: 10
```

```
The sum is 55
```

In the above program, we asked the user to enter a number, *n*. `while` loop is used to sum from 1 to that number. The condition will be `True` as long as our counter variable *i* is less than or equal to *n*. We need to increase the value of counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop). Finally the result is displayed.

while loop with else

Same as that of `for` loop, we can have an optional `else` block with `while` loop as well. The `else` part is executed if the condition in the `while` loop evaluates to `False`. `while` loop can be terminated with a `break` statement. In such case, the `else` part is ignored. Hence, a `while` loop's `else` part runs if no break occurs and the condition is false.

Here is an example to illustrate this.

```
# Example to illustrate
# the use of else statement
# with the while loop

counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

Output

```
Inside loop
Inside loop
Inside loop
Inside else
```

Here, we use a counter variable to print the string `Inside loop` three times. On the forth iteration, the condition in `while` becomes `False`. Hence, the `else` part is executed.

Python break and continue Statement

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

break statement

The `break` statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If it is inside a nested loop (loop inside another loop), `break` will terminate the innermost loop.

Syntax of break

```
break
```

Flowchart of break

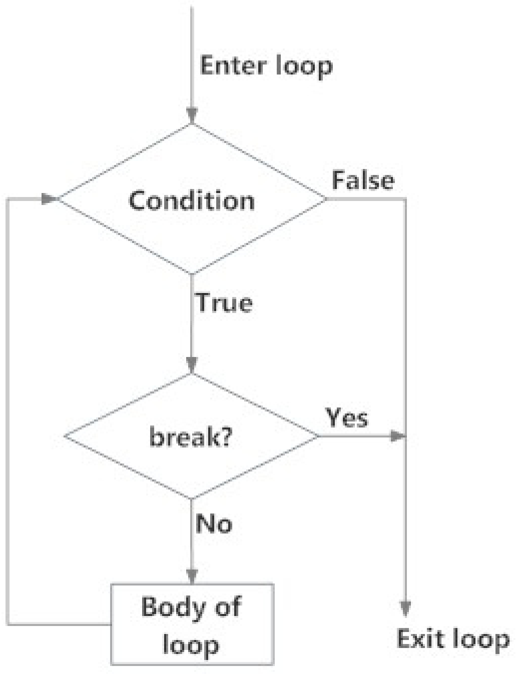


Fig: flowchart of break

The working of `break` statement in for loop and while loop is shown below.

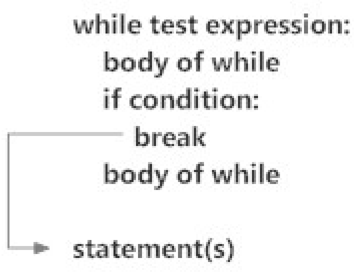


Fig: working of break in while loop

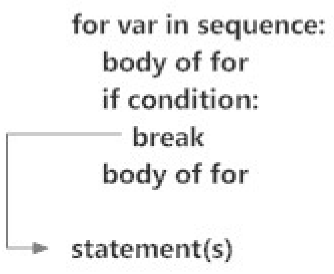


Fig: working of break in for loop

Example: Python break

```
# Program to show the use of break statement inside loop

for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

Output

```
s
t
r
The end
```

In this program, we iterate through the `"string"` sequence. We check if the letter is `"i"`, upon which we break from the loop. Hence, we see in our output that all the letters up till `"i"` gets printed. After that, the loop terminates.

continue statement

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Syntax of Continue

```
continue
```

Flowchart of continue

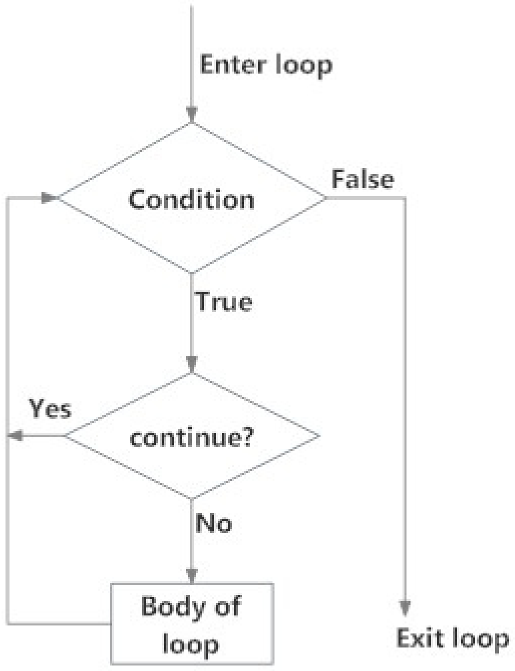
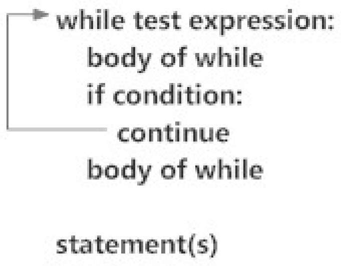


Fig: flowchart of continue

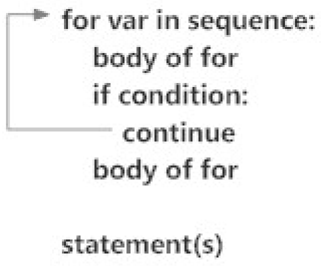
The working of `continue` statement in for and while loop is shown below.



```
while test expression:
    body of while
    if condition:
        continue
    body of while

statement(s)
```

Fig: working of continue in while loop



```
for var in sequence:
    body of for
    if condition:
        continue
    body of for

statement(s)
```

Fig: working of continue in for loop

Example: Python continue

```
# Program to show
# the use of continue
# statement inside loops

for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

Output

```


s
t
r
n
g
The end
```

This program is same as the above example except the `break` statement has been replaced with `continue`. We continue with the loop, if the string is `"i"`, not executing the rest of the block. Hence, we see in our output that all the letters except `"i"` gets printed.

Python pass Statement

In Python programming, `pass` is a null statement. The difference between a comment and `pass` statement in Python is that, while the interpreter ignores a comment entirely, `pass` is not ignored. But nothing happens when it is executed. It results into no operation (NOP).

Syntax of pass

```
pass
```

We generally use it as a placeholder. Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the `pass` statement to construct a body that does nothing.

Example: pass Statement

```
for val in sequence:
    pass
```

We can do the same thing in an empty function or class as well.

```
def function(args):
    pass
```

```
class example:
    pass
```

Python Looping Techniques

Python programming offers two kinds of loop, the for loop and the while loop. Using these loops along with loop control statements like `break` and `continue`, we can create various forms of loop.

The infinite loop

We can create an infinite loop using `while` statement. If the condition of `while` loop is always `True`, we get an infinite loop.

Example of infinite loop

```
# An example of infinite loop
# press Ctrl + c to exit from the loop

while True:
   num = int(input("Enter an integer: "))
   print("The double of",num,"is",2 * num)
```

Output

```
Enter an integer: 3
The double of 3 is 6
Enter an integer: 5
The double of 5 is 10
Enter an integer: 6
The double of 6 is 12
Enter an integer:
Traceback (most recent call last):
```

Loop with condition at the top

This is a normal `while` loop without `break` statements. The condition of the `while` loop is at the top and the loop terminates when this condition is `False`.

Flowchart of Loop With Condition at Top

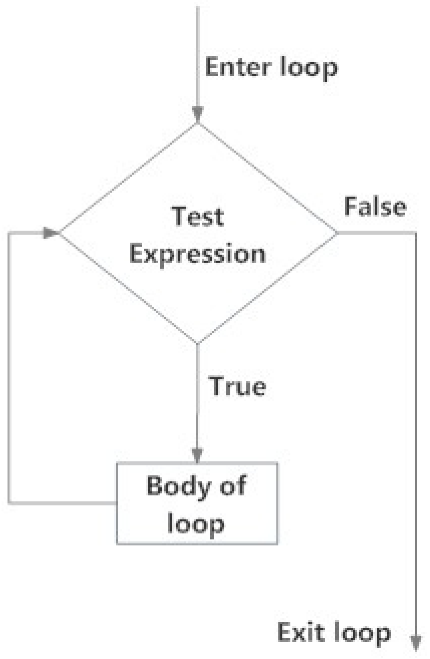


Fig: loop with condition at top

Example

```
# Program to illustrate a loop with condition at the top

n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1

while i <= n:
   sum = sum + i
   i = i+1    # update counter

# print the sum
print("The sum is",sum)
```

Output

```

Enter n: 10
The sum is 55
```

Loop with condition in the middle

This kind of loop can be implemented using an infinite loop along with a conditional break in between the body of the loop.

Flowchart of Loop with Condition in Middle

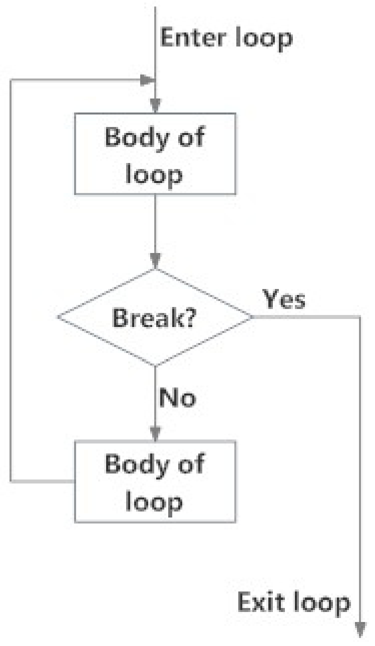


Fig: loop with condition in middle

Example

```
# Program to illustrate a loop with condition in the middle.
# Take input from the user untill a vowel is entered

vowels = "aeiouAEIOU"

# infinite loop
while True:
   v = input("Enter a vowel: ")
   # condition in the middle
   if v in vowels:
       break
   print("That is not a vowel. Try again!")

print("Thank you!")
```

Output

```
Enter a vowel: r
That is not a vowel. Try again!
Enter a vowel: 6
That is not a vowel. Try again!
Enter a vowel: ,
That is not a vowel. Try again!
Enter a vowel: u
Thank you!
```

Loop with condition at the bottom

This kind of loop ensures that the body of the loop is executed at least once. It can be implemented using an infinite loop along with a conditional break at the end. This is similar to the `do...while` loop in C.

Flowchart of Loop with Condition at Bottom

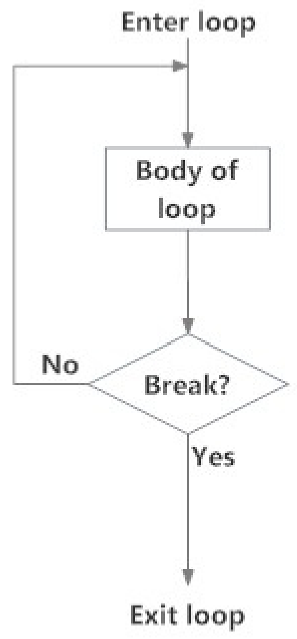


Fig: loop with condition at bottom

Example

```
# Python program to illustrate a loop with condition at the bottom
# Roll a dice untill user chooses to exit

# import random module
import random

while True:
   input("Press enter to roll the dice")
```

```
    # get a number between 1 to 6
    num = random.randint(1,6)
    print("You got",num)
    option = input("Roll again?(y/n) ")

    # contion
    if option == 'n':
        break
```

Output

```


Press enter to roll the dice
You got 1
Roll again?(y/n) y
Press enter to roll the dice
You got 5
Roll again?(y/n) n
```