Decision Making and Loops

Decision Making Statements

Sometimes, in a program, we may want to make a decision based on a condition. We know that an expression's value can be True or False.

Decision Making statements includes:-

1. If Statements.
2. If-else Statements.
3. Chained Conditionals(the elif ladder).
4. Nested if Statements.

if Statements

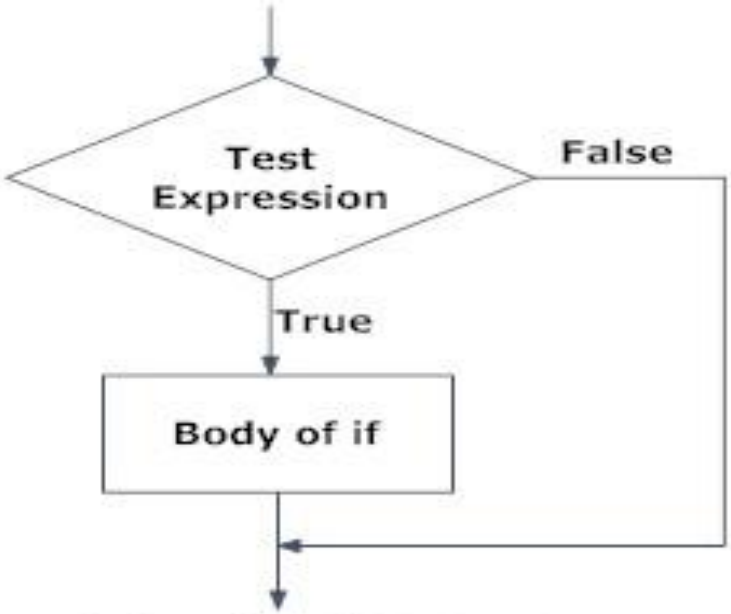


Fig: Operation of if statement

if Statements

- An if statement in python takes an expression with it

Syntax: `if test expression:`

```
statement(s)
```

- If the expression amounts to True, then the block of statements under it is executed.
- If it amounts to False, then the block is skipped and control transfers to the statements after the block.
- Also, use a colon(:) after the condition

```
a=7
if a>6:
    print(f"{a} is good")
```

if-else Statements

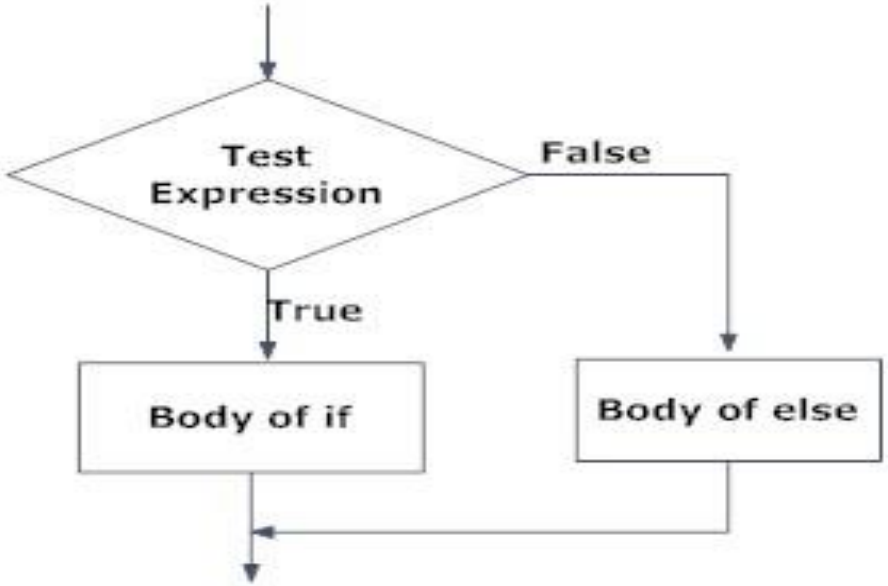


Fig: Operation of if...else statement

if-else Statements

- If the statement in the if blocks turn out false then else statement can be used.

Syntax:

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

- An else statement comes right after the block after 'if'.

```
if 2<1:
  print("2")
else:
  print("1")
```

Chained Conditionals (elif ladder)

- 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

Syntax:-

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

if...elif...else

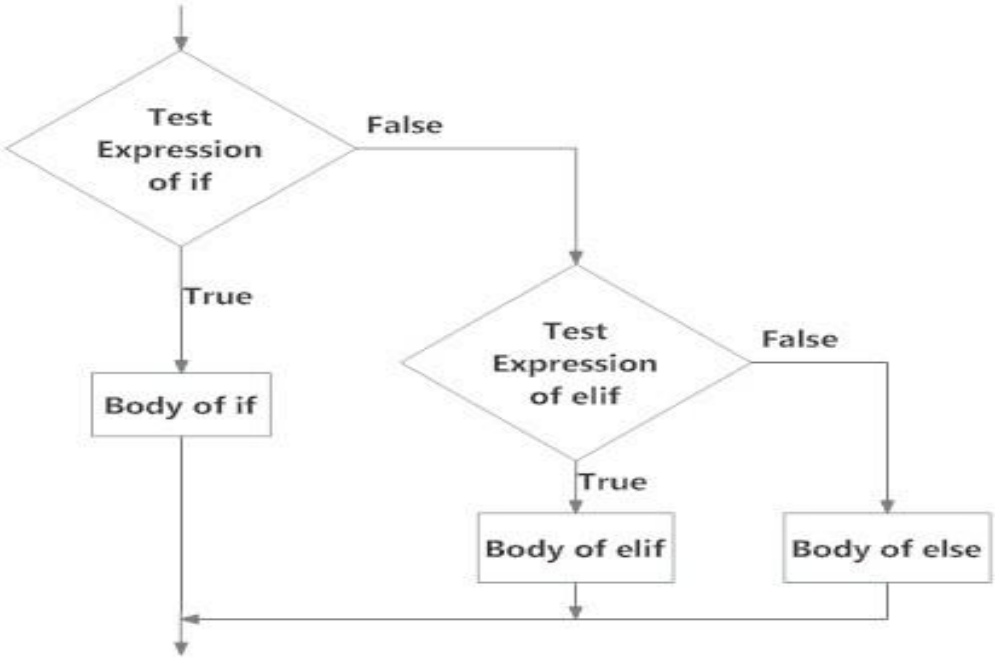


Fig: Operation of if...elif...else statement

Nested if statements

- We can have a if...elif...else statement inside another if...elif...else statement.
- This is called nesting in computer programming.
- Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting.

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 : -

```
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.

for Loop

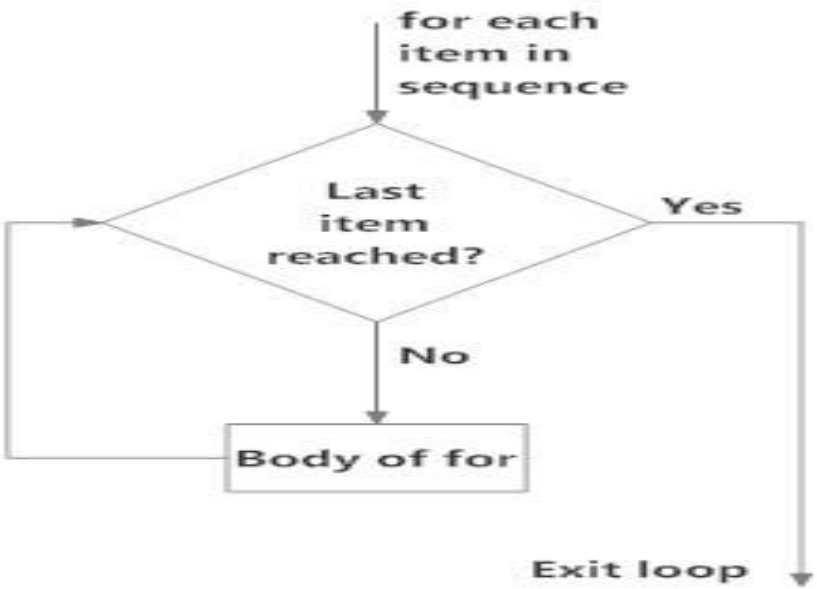


Fig: operation of for loop

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

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.

for loop with else

```
digits = [0, 1, 5]
for i in digits:
    print(i)
else:
    print("No items left.")
```

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.

Syntax:-

```
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 until the test_expression evaluates to False.

while Loop

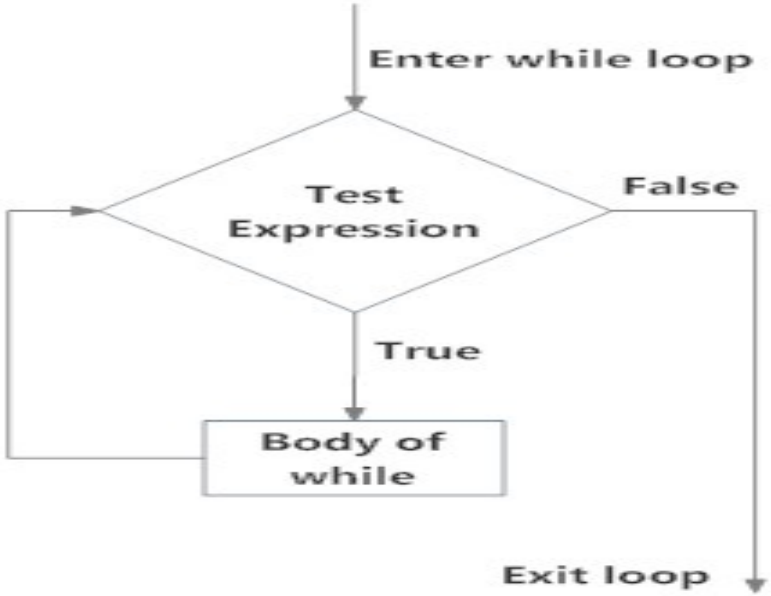


Fig: operation of while loop

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.
- The while loop can be terminated with a break statement. In such case, the else part is ignored.

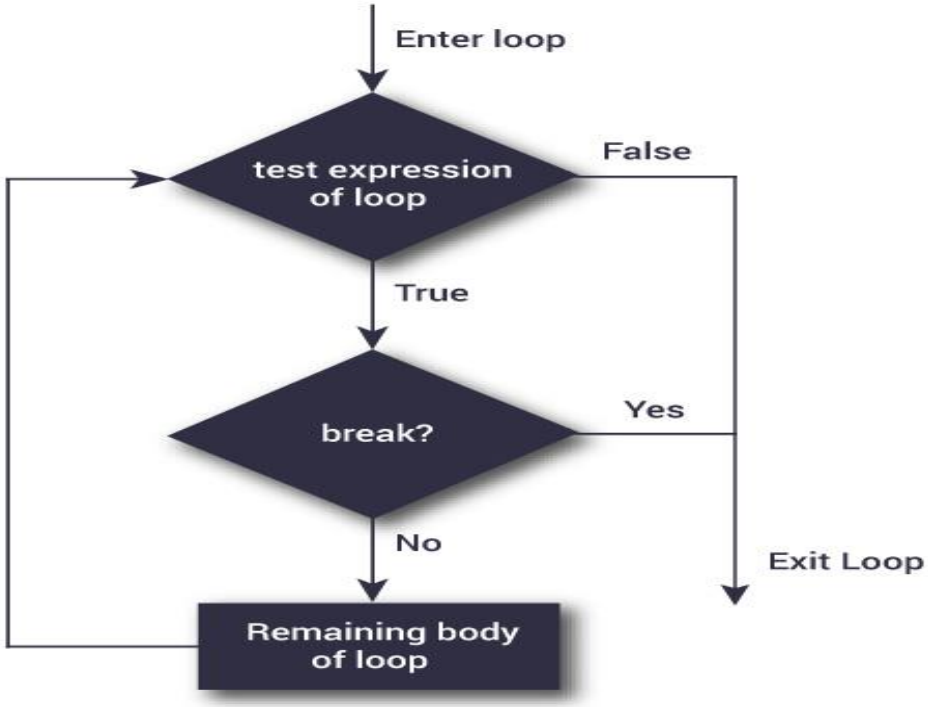
break and continue

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 break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax: - `break`

break statement



break statement in for and while loop

```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

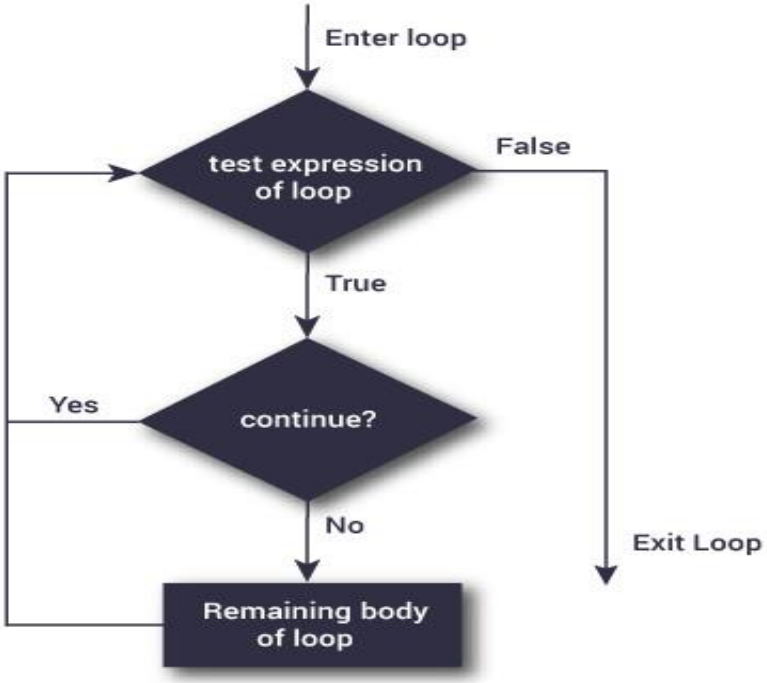
continue statement

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:- `continue`

continue statement



continue statement in for and while loop

```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

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.

  Syntax:-`pass`
- We generally use it as a placeholder.

```
sequence = {'p', 'a', 's', 's'}
for val in sequence:
pass
```

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.

Functions

- Function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes code reusable.

Types of Functions

Basically, we can divide functions into the following two types:

1. **Built-in functions -** Functions that are built into Python.
2. **User-defined functions -** Functions defined by the users themselves.

Functions

Syntax of Function

```
def function_name(parameters):
     """docstring"""
     statement(s)
```

Functions

```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

Functions

- Keyword def marks the start of function header.
- A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.

Functions

- Once we have defined a function, we can call it from another function, program or even the Python prompt.
- To call a function we simply type the function name with appropriate parameters.

Docstring

- The first string after the function header is called the docstring and is short for documentation string.
- It is used to explain in brief, what a function does.
- Although optional, documentation is a good programming practice.
- In the above example, we have a docstring immediately below the function header.
- Generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as __doc__ attribute of the function.

```
print(greet.__doc__)
```

return statement

- The return statement is used to exit a function and go back to the place from where it was called.
- return [expression_list]
- This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.
- Here, None is the returned value.

Anonymous/Lambda Function

- In Python, anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.

How to use lambda Functions in Python?

Syntax of Lambda Function in python

```
lambda arguments: expression
```

- Lambda functions can have any number of arguments but only one expression.
- The expression is evaluated and returned.
- Lambda functions can be used wherever function objects are required.

```
double = lambda x: x * 2
print(double(5))
```

Task

- Write a program to take an input from user a print whether it is positive or negative.
- Program to find the sum of all numbers stored in a list.
- Program to add natural numbers upto sum = 1+2+3+...+n.