



GOVERNMENT POLYTECHNIC AMRAVATI

DIPLOMA PROGRAMME IN COMPUTER ENGINEERING

COURSE : PROGRAMMING WITH PYTHON

COURSE CODE :CM5461

Unit 2

Types, Operators and Expression

UNIT 2 :- CONTENTS

1

Standard Datatypes:

2

Operators:

3

Control flow:

4

Programs

COURSE OUTCOMES (COs)

At the end of this course, student will be able to: -

- Write and execute simple 'Python' programs.
- Write 'Python' programs using arithmetic expressions and control structure.
- Develop 'Python' programs using List, Tuples and Dictionary.
- Develop/Use functions in Python programs for modular programming approach.
- Develop 'Python' programs using File Input/output operations.
- Write 'Python' code using Classes and Objects.

Python Data Types

- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:
 - Text Type : str
 - Numeric Types : int, float, complex
 - Sequence Types : list, tuple, range

Python Data Types

- Mapping Type : dict
- Set Types : set, frozenset
- Boolean Type : bool
- Binary Types : bytes, bytearray, memoryview
- **Number data types store numeric values.** They are immutable data types, means that changing the value of a number data type results in a newly allocated object. (supports int, long, float, complex data types)

Python Data Types

- `var1 = 1 var2 = 10`
- You can also delete the reference to a number object by using the `del` statement. The syntax of the `del` statement is –
 - `del var1[,var2[,var3[....,varN]]]`
- You can delete a single object or multiple objects by using the `del` statement. For example –
 - `del var`
 - `del var_a, var_b`

Python Data Types

Number Type Conversion

- Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.
- Type `int(x)` to convert `x` to a plain integer.
- Type `long(x)` to convert `x` to a long integer.
- Type `float(x)` to convert `x` to a floating-point number.

Python Data Types

- Type `complex(x)` to convert `x` to a complex number with real part `x` and imaginary part zero.
- Type `complex(x, y)` to convert `x` and `y` to a complex number with real part `x` and imaginary part `y`. `x` and `y` are numeric expressions
- **Mathematical Functions**
- Python includes following functions that perform mathematical calculations.
`abs(x)`, `ceil(x)`, `cmp(x)`, `exp(x)`, `fabs(x)`, `floor(x)`, `pow(x,y)` etc

Python Data Types

Random Number Functions

- Random numbers are used for games, simulations, testing, security, and privacy applications. Python includes following functions that are commonly used.

- `import random`

```
print(random.randrange(10, 20))
```

```
x = ['a', 'b', 'c', 'd', 'e']
```

- `# Get random choice`

```
print(random.choice(x))
```

Python Data Types

- # Shuffle x

```
random.shuffle(x)
```

- # Print the shuffled x

```
print(x)
```

- # Print random element

```
print(random.random())
```

- 18

- e

- ['c', 'e', 'd', 'b', 'a']

- 0.5682821194654443

Python Data Types

- Python offers a range of compound data types often referred to as sequences. List is one of the most frequently used and very versatile data types used in Python.
- In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

Python Data Types

- # empty list

```
list1 = []
```

- # list of integers

```
list2 = [1, 2, 3]
```

- # list with mixed data types

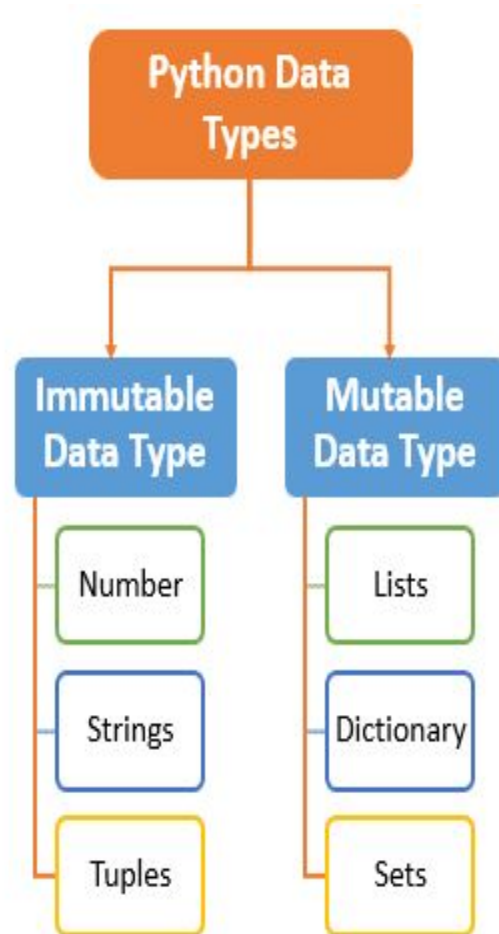
```
list3 = [1, "Hello", 3.4]
```

- A list can also have another list as an item. This is called a nested list.

```
# nested list
```

```
list4 = ["mouse", [8, 4, 6], ['a']]
```

Python Data Types



Python Data Types

Python data types are categorized into two as follows:

Mutable Data Types: Data types in python where the value assigned to a variable can be changed. Some mutable data types in Python include set, list, user-defined classes and dictionary.

Immutable Data Types: Data types in python where the value assigned to a variable cannot be changed. Some immutable data types in Python are int, decimal, float, tuple, bool, range and string.

Python Data Types

- **Numbers:** The number data type in Python is used to store numerical values. It is used to carry out normal mathematical operations.
- **Strings:** Strings in Python are used to store textual information. They are used to carry out operations that perform positional ordering among items.
- **Lists:** The list data type is the most generic Python data type. Lists can consist of a collection of mixed data types, stored by relative positions.

Python Data Types

- **Tuples: Python Tuples** are one among the immutable Python data types that can store values of mixed data types. They are basically a list that cannot be changed.
- **Sets:** Sets in Python are a data type that can be considered as an unordered collection of data without any duplicate items.
- **Dictionaries:** Dictionaries in Python can store multiple objects, but unlike lists, in dictionaries, the objects are stored by keys and not by positions.

Python Data Types

- **Built-in Data Types in Python**
- The categories of built-in data types in Python are:
- **Binary Types** – bytes, memory view, bytearray
- **Mapping Type** – dict
- **Numeric Type** – int, float, complex
- **Text Type** – str
- **Boolean Type** – bool
- **Set Types** – set, frozenset
- **Sequence Types** – list, range, tuple

Operators in Python

- Python language supports the following types of operators.
- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Operators in Python :Comparison operators

Comparison operators are used to compare values. It returns either True or False according to the condition.

Op.	Meaning	Ex.
>	Greater than - True if left operand is greater than the right	$x > y$
<	Less than - True if left operand is less than the right	$x < y$
==	Equal to - True if both operands are equal	$x == y$
!=	Not equal to - True if operands are not equal	$x != y$
>=	Greater than or equal to - True if left operand is greater than or equal to the right	$x >= y$
<=	Less than or equal to - True if left operand is less than or equal to the right	$x <= y$

Operators in Python :Logical operators

Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Operators in Python :Bitwise operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

Operator	Meaning	Example
&	Bitwise AND	$x \& y = 0$ (0000 0000)
	Bitwise OR	$x y = 14$ (0000 1110)
~	Bitwise NOT	$\sim x = -11$ (1111 0101)
^	Bitwise XOR	$x \wedge y = 14$ (0000 1110)
>>	Bitwise right shift	$x >> 2 = 2$ (0000 0010)
<<	Bitwise left shift	$x << 2 = 40$ (0010 1000)

Operators in Python :Assignment operators

Assignment operators are used in Python to assign values to variables. `a = 5` is a simple assignment operator that assigns the value 5 on the right to the variable `a` on the left. There are various compound operators in Python like `a += 5` that adds to the variable and later assigns the same. It is equivalent to `a = a + 5`.

Operator	Example	Equivalent to
<code>=</code>	<code>x = 5</code>	<code>x = 5</code>
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 5</code>	<code>x = x - 5</code>
<code>*=</code>	<code>x *= 5</code>	<code>x = x * 5</code>
<code>/=</code>	<code>x /= 5</code>	<code>x = x / 5</code>
<code>%=</code>	<code>x %= 5</code>	<code>x = x % 5</code>
<code>//=</code>	<code>x //= 5</code>	<code>x = x // 5</code>

Operators in Python :Identity operators

Special operators

Python language offers some special types of operators like the identity operator or the membership operator. They are described below with examples.

Identity operators

is and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operators in Python :Identity operators

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```


Operators in Python :Identity operators

operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Operators in Python :Membership operators

Membership operators

`in` and `not in` are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
<code>in</code>	True if value/variable is found in the sequence	<code>5 in x</code>
<code>not in</code>	True if value/variable is not found in the sequence	<code>5 not in x</code>

Operators in Python :Membership operators

```
x = 'Hello world'
```

```
y = {1:'a',2:'b'}
```

```
# Output: True
```

```
print('H' in x)
```

```
# Output: True
```

```
print('hello' not in x)
```

```
# Output: True
```

```
print(1 in y)
```

```
# Output: False
```

```
print('a' in y)
```

Membership Operator

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Membership Operator

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
    print "Line 1 - a is available in the given list"
else:
    print "Line 1 - a is not available in the given list"
if ( b not in list ):
    print "Line 2 - b is not available in the given list"
else:
    print "Line 2 - b is available in the given list"
a = 2
if ( a in list ):
    print "Line 3 - a is available in the given list"
else:
    print "Line 3 - a is not available in the given list"
```

Identity Operator

Identity operators compare the memory locations of two objects. There are two Identity operators as explained below –

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Identity Operator

a = 20

b = 20

if (a is b):

print "Line 1 - a and b have same identity"

else:

print "Line 1 - a and b do not have same identity"

if (id(a) == id(b)):

print "Line 2 - a and b have same identity"

else:

print "Line 2 - a and b do not have same identity"

b = 30

if (a is b):

print "Line 3 - a and b have same identity"

else:

print "Line 3 - a and b do not have same identity"

GPA if (a is not b):

print "Line 4 - a and b do not have same identity"

Python Operator Precedence.

- The following table lists all operators from highest precedence to lowest.

Sr.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'
7	^ Bitwise exclusive 'OR' and regular 'OR'

- The following table lists all operators from highest precedence to lowest.

8	<code><= < > >=</code> Comparison operators
9	<code><> == !=</code> Equality operators
10	<code>= %= /= //= -= += *= **=</code> Assignment operators
11	<code>is is not</code> Identity operators
12	<code>in not in</code> Membership operators
13	<code>not or and</code> Logical operators

Operators in Python :Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	$x + y + 2$
-	Subtract right operand from the left or unary minus	$x - y - 2$
*	Multiply two operands	$x * y$
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	$x \% y$ (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	$x // y$
**	Exponent - left operand raised to the power of right	$x ** y$ (x to the power y)

Control Flow

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.

if test expression:

statement(s)

Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True. If the test expression is False, the statement(s) is not executed.

Control Flow : If statement

In Python, the body of the if statement is indicated by the indentation. The body starts with an indentation and the first unintended line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.

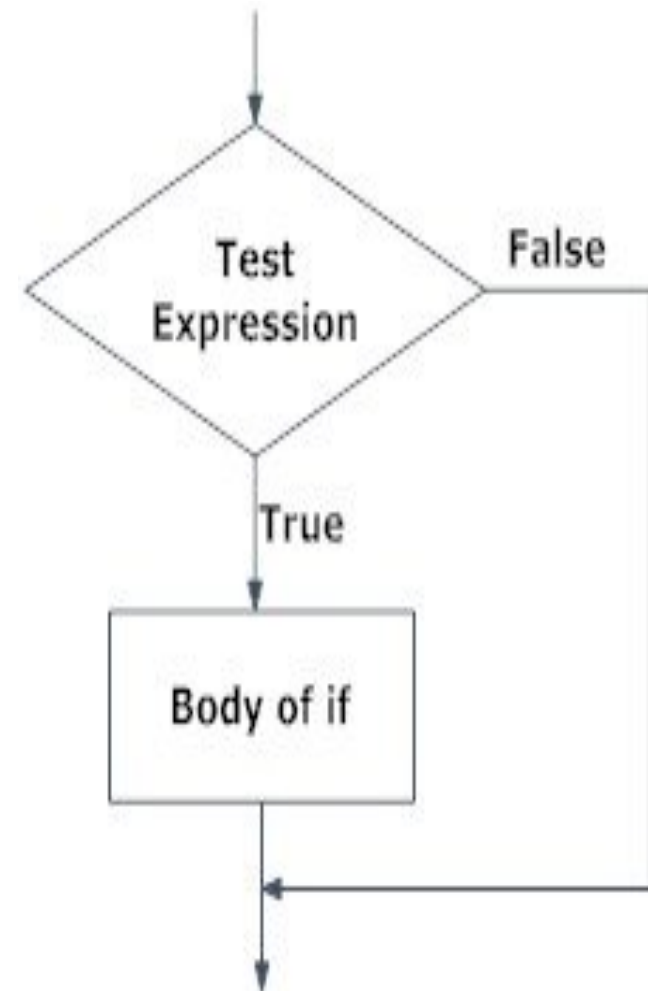


Fig: Operation of if statement

Control Flow

If the number is positive, we print an appropriate message

```
num = 3
```

```
if num > 0:
```

```
    print(num, "is a positive number.")
```

```
print("This is always printed.")
```

```
num = -1
```

```
if num > 0:
```

```
    print(num, "is a positive number.")
```

```
print("This is also always printed.")
```

Control Flow

If the number is positive, we print an appropriate message

```
num = 3
```

```
if num > 0:
```

```
    print(num, "is a positive number.")
```

```
    print("This is always printed.")
```

```
num = -1
```

```
if num > 0:
```

```
    print(num, "is a positive number.")
```

```
    print("This is also always printed.")
```

Control Flow

if test expression:

 Body of if

else:

 Body of else

The if....else statement evaluates test expression and will execute the body of if only when the test condition is True.

If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

Control Flow :IF Else Statement

Program checks if the number is positive
or negative

And displays an appropriate message

num = 3

Try these two variations as well.

num = -5

num = 0

```
if num >= 0:
```

```
    print("Positive or Zero")
```

```
else:
```

```
    print("Negative number")
```

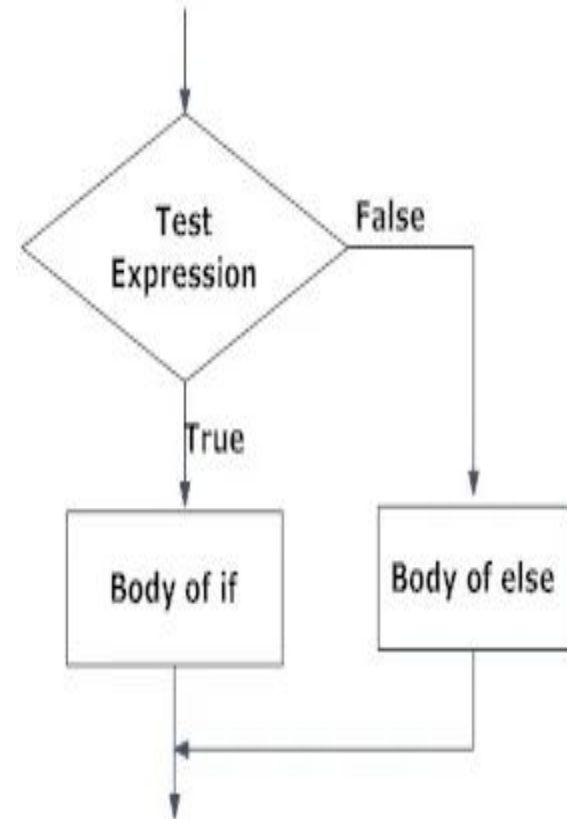


Fig: Operation of if...else statement

Control Flow :IF Else Statement

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, the body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

Control Flow :IF ...ElseIf... Else Statement

if test expression:

Body of if

elif test expression:

Body of elif

else:

Body of else

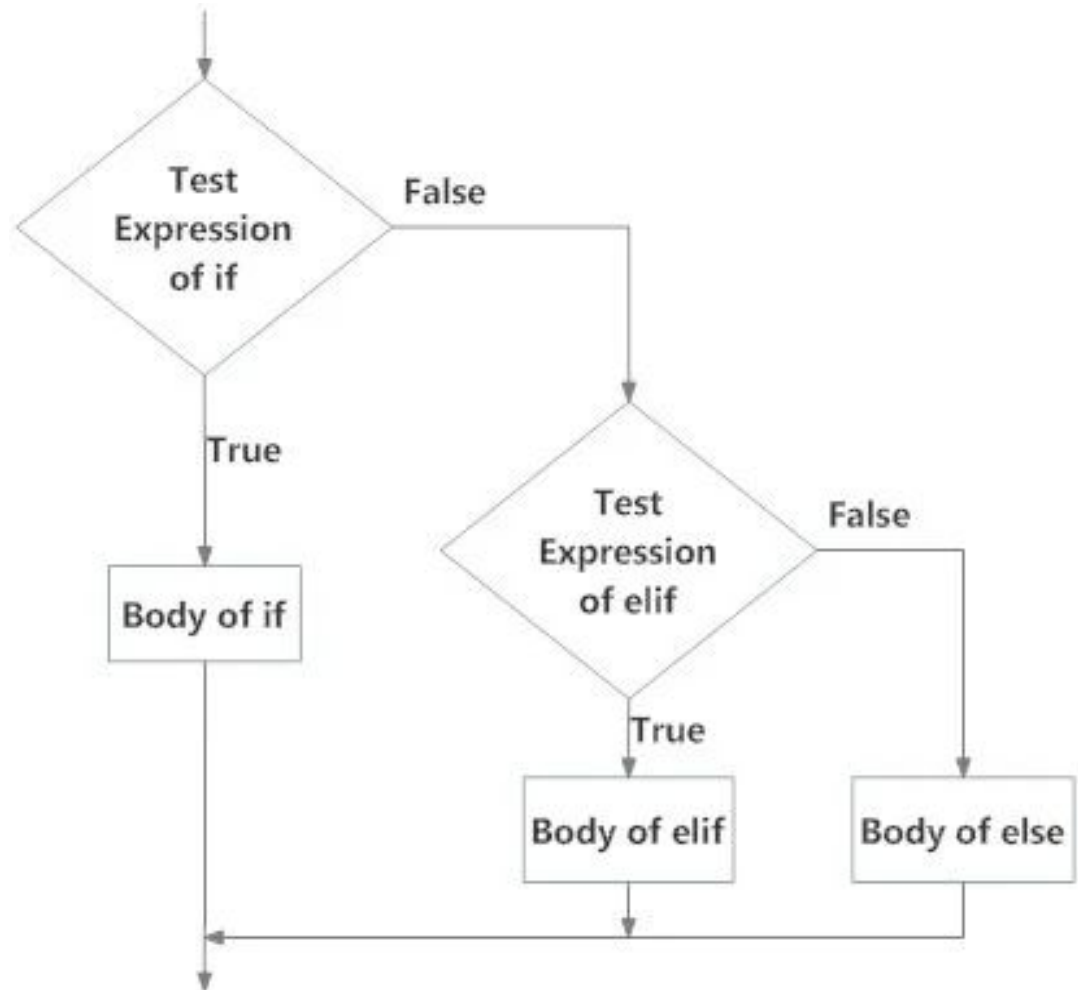


Fig: Operation of if...elif...else statement

Control Flow :IF ...ElseIf... Else Statement

```
num = 3.4
```

```
if num > 0:
```

```
    print("Positive number")
```

```
elif num == 0:
```

```
    print("Zero")
```

```
else:
```

```
    print("Negative number")
```

When variable num is positive,
Positive number is printed.

If num is equal to 0, Zero is
printed.

If num is negative, Negative
number is printed.

Control Flow :IF ...ElseIf... Else Statement

Python Nested if statements

We can have an if...elif...else statement inside another if...elif...else statement. This is called nesting in programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. They can get confusing, so they must be avoided unless necessary.

Control Flow :IF ...ElseIf... Else Statement

```
num = float(input("Enter a number: "))
```

```
if num >= 0:
```

```
    if num == 0:
```

```
        print("Zero")
```

```
    else:
```

```
        print("Positive number")
```

```
else:
```

```
    print("Negative number")
```

Looping :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.

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.

Looping :For loop

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
```

```
sum = 0
```

```
for val in numbers:
```

```
    sum = sum+val
```

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

Looping :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. The break keyword can be used to stop a for loop. In such cases, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

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


Looping :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. The break keyword can be used to stop a for loop. In such cases, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

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

Here, the for loop prints items of the list until the loop exhausts. When the for loop exhausts, it executes the block of code in the else and prints No items left.

Looping :For loop with else

This for...else statement can be used with the break keyword to run the else block only when the break keyword was not executed. Let's take an example:

```
# program to display student's marks from record
student_name = 'Ram'
marks = {'Pawan': 90, 'Rahul': 55, 'Roshan': 77}
for student in marks:
    if student == student_name:
        print(marks[student])
        break
else:
    print('No entry with that name found.')
```

Looping :For loop with else

Iterating by Sequence Index

An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example –

```
fruits = ['banana', 'apple', 'mango']  
for index in range(len(fruits)):  
    print 'Current fruit :', fruits[index]  
    print "Good bye!"
```

Looping :For loop with else

```
for num in range(10,20):  
    if num % 2==0:  
        print("The even No",num)  
    # else:  
    # print("Odd Number",num)
```

Looping :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 the number of times to iterate beforehand.

while test_expression:

Body of while

In the 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.

Looping :While loop

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

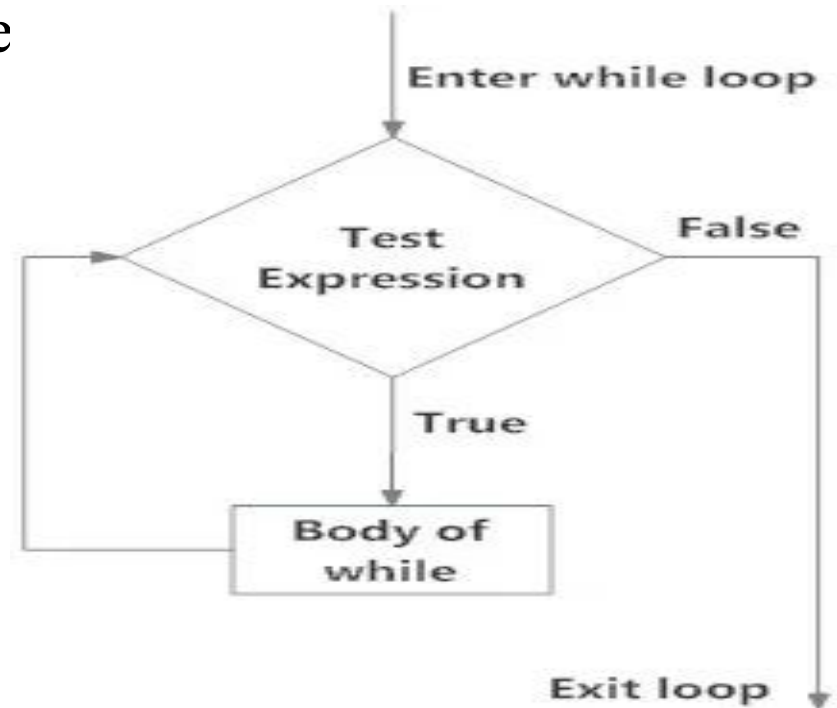


Fig: operation of while loop

Looping :While loop

```
n = 10
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1    # update counter
# print the sum
print("The sum is", sum)
```

In the above program, the test expression will be True as long as our counter variable *i* is less than or equal to *n* (10 in our program).

We need to increase the value of the 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.

Looping :While loop with Else

"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")
```

Same as with for loops, while loops can also have an optional else block.

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 cases, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

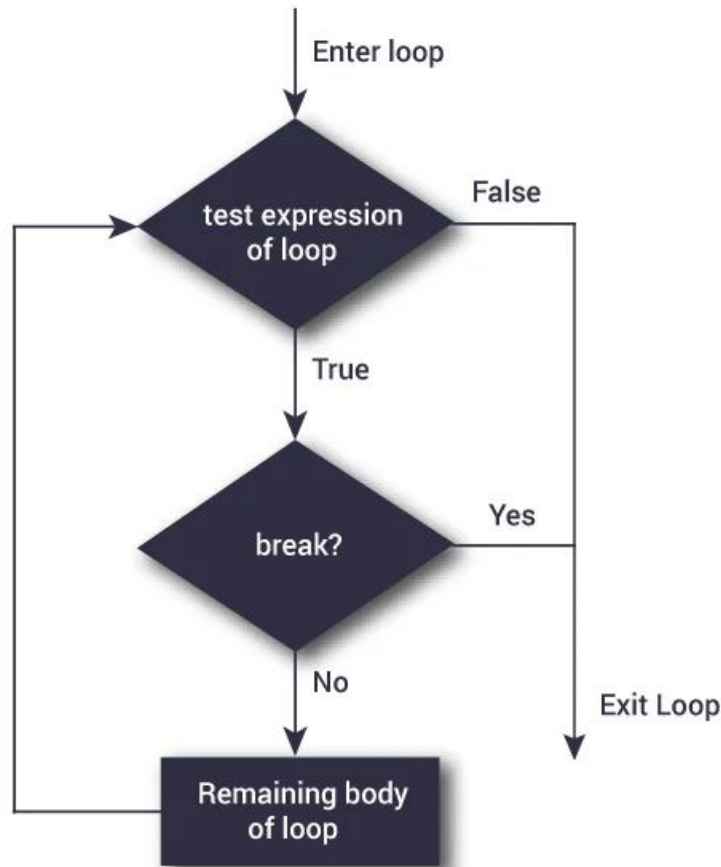
Looping :While loop with Else

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.

Looping : While loop with Else




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

Looping : While loop with Else

```
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
```



Python break statement

```
string = ['C','o','m','p','u','t','e','r']
```

```
for val in string:
```

```
    if val == "p":
```

```
        break
```

```
    print(val)
```

```
print("The end")
```

O/P

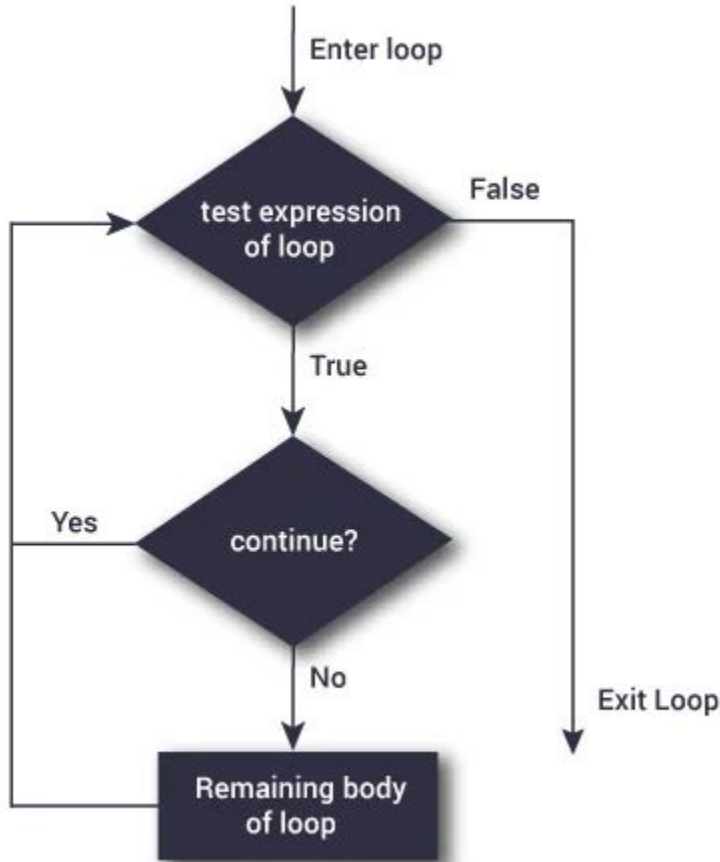
S

t

r

The end

Looping : While loop with Else



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.

Looping :While loop with Else

```
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
```

Program to show the use of
continue statement inside loops

```
string = ['C','o','m','p','u','t','e','r']
for val in string:
```

```
    if val == "i":
        continue
```

```
    print(val)
```

```
print("The end")
```

S

t

r

n

g

The end

