MARUDHAR KESARI JAIN COLLEGE FOR WOMEN, VANIYAMBADI

PG DEPARTMENT OF COMPUTER APPLICATIONS

Subject Name : PYTHON PROGRAMMING

CLASS : I-MCA

SUBJECT CODE : 23PCA13

UNIT-I

Introduction : Fundamental ideas of Computer Science - Strings, Assignment, and Comments - Numeric Data types and Character sets Expressions Loops and Selection Statements: Definite iteration: the for Loop - selection: if and if-else statements - Conditional iteration: the while Loop.

Introduction to python

These activities rely on something in common: computer technology. Computer technology is almost everywhere, not only in our homes but also in our schools and in the places where we work and play. Computer technology plays an important role in entertainment, education, medicine, manufacturing, communications, government, and commerce. It has been said that we have digital lifestyles and that we live in an information age with an information-based economy. Some people even claim that nature itself performs computations on information structures present in DNA and in the relationships among subatomic particles

Two Fundamental Ideas of Computer Science: Algorithms and Information Processing

Algorithms People computed long before the invention of modern computing devices, and many continue to use computing devices that we might consider primitive. For example, consider how merchants made change for customers in marketplaces before the existence of credit cards, pocket calculators, or cash registers. Making change can be a complex activity. It probably took you some time to learn how to do it, and it takes some mental effort to get it right every time. Let's consider what's involved in this process

Step 1 Write down the two numbers, with the larger number above the smaller number and their digits aligned in columns from the right.

Step 2 Assume that you will start with the rightmost column of digits and work your way left through the various columns.

Step 3 Write down the difference between the two digits in the current column of digits, borrowing a 1 from the top number's next column to the left if necessary.

Step 4 If there is no next column to the left, stop. Otherwise, move to the next column to the left, and go back to step 3.

The algorithm is also one of the fundamental ideas of computer science. An algorithm has the following features: 1. An algorithm consists of a finite number of instructions. 2. Each individual instruction in an algorithm is well defined. This means that the action described by the instruction can be performed effectively or be executed by a computing agent. For example, any computing agent capable of arithmetic can compute the difference between two digits. So an algorithmic step that says "compute the difference between two digits" would be well defined. On the other hand, a step that says "divide a number by 0" is not well defined, because no computing agent could carry it out. 3. An algorithm describes a process that eventually halts after arriving at a solution to a problem. For example, the process of subtraction halts after the computing agent writes down the difference between the two digits in the leftmost column of digits. 4. An algorithm solves a general class of problems.

For example, an algorithm that describes how to make change should work for any two amounts of money whose difference is greater than or equal to $0.00

Information Processing

Like mathematical calculations, information processing can be described with algorithms. In our earlier example of making change, the subtraction steps involved manipulating symbols used to represent numbers and money. In carrying out the instructions of any algorithm, a computing agent manipulates information. The computing agent starts with some given information (known as input), transforms this information according to well-defined rules, and produces new information, known as output

It is important to recognize that the algorithms that describe information processing can also be represented as information. Computer scientists have been able to represent algorithms in a form that can be executed effectively and efficiently by machines. They have also designed real machines, called electronic digital computers, which are capable of executing algorithms.

Strings, Assignment, and Comments

Text processing is by far the most common application of computing. E-mail, text messaging, Web pages, and word processing all rely on and manipulate data consisting of strings of characters.

This section introduces the use of strings for the output of text and the documentation of Python programs.

We begin with an introduction to data types in general. Data Types In the real world, we use data all the time without bothering to consider what kind of data we're using.

For example, consider this sentence: "In 2007, Micaela paid $120,000 for her house at 24 East Maple Street." This sentence includes at least four pieces of data—a name, a date, a price, and an address—but of course you don't have to stop to think about that before you utter the sentence.

You certainly don't have to stop to consider that the name consists only of text characters, the date and house price are numbers, and so on. However, when we use data in a computer program, we do need to keep in mind the type of data we're using. We also need to keep in mind what we can do with (what operations can be performed on) particular data.

Type of Data Python Type Name Example Literals

Integers int –1, 0, 1, 2

Real numbers float –0.55, .3333, 3.14, 6.0

Character strings str "Hi", "", 'A', "66

String Literals

In Python, a string literal is a sequence of characters enclosed in single or double quotation marks.

The following session with the Python shell shows some example strings:

>>> '**Hello there!**' '**Hello there!**'

>>> **"Hello there!"**

'Hello there!' >>> '' '' >>> ""

’’

Variables and the Assignment Statement

variable associates a name with a value, making it easy to remember and use the value later in a program. You need to be mindful of a few rules when choosing names for your variables. For example, some names, such as if, def, and import, are reserved for other purposes and thus cannot be used for variable names. In general, a variable name must begin with either a letter or an underscore (_), and can contain any number of letters, digits, or other underscores.

Python variable names are case sensitive; thus, the variable WEIGHT is a different name from the variable weight. Python programmers typically use lowercase letters for variable names, but in the case of variable names that consist of more than one word, it’s common to begin each word in the variable name (except for the first one) with an uppercase letter. This makes the variable name easier to read. For example, the name interest Rate is slightly easier to read than the name interest rate.

<Variable name>=<Expression>

Programmers use all uppercase letters for the names of variables that contain values that the program never changes. Such variables are known as symbolic constants. Examples of symbolic constants in the tax calculator case study are TAX_RATE and STANDARD_DEDUCTION. Variables receive their initial values and can be reset to new values with an assignment statement.

>>> **firstName = "Ken"**

>>> **secondName = "Lambert"**

>>> **fullName = firstName + " " + secondName**

>>> **fullName 'Ken Lambert'**

Numeric Data Types and Character Sets

Integers As you learned in mathematics, the integers include 0, the positive whole numbers, and the negative whole numbers. Integer literals in a Python program are written without commas, and a leading negative sign indicates a negative value. Although the range of integers is infinite, a real computer’s memory places a limit on the magnitude of the largest positive and negative integers. The most common implementation of the int data type in many programming languages consists of the integers from 22 2,147,483,648 (2) 31 to 2,147,483,647 (2 2 1) 31 . However, the magnitude of a Python integer is much larger and is limited only by the memory of your computer. As an experiment, try evaluating the expression 2147483647 ** 100, which raises the largest positive int value to the 100th power. You will see a number that contains many lines of digits! Floating-Point Numbers Areal number in mathematics, such as the value of p (3.1416...), consists of a whole number, a decimal point, and a fractional part.

Real numbers have infinite precision, which means that the digits in the fractional part can continue forever. Like the integers, real numbers also have an infinite range.

However, because a computer's memory is not infinitely large, a computer's memory limits not only the range but also the precision that can be represented for real numbers. Python uses floating-point numbers to represent real numbers.

Values of the most common implementation of Python's float type range from approximately 210308 to 10308 and have 16 digits of precision.

A floating-point number can be written using either ordinary decimal notation or scientific notation. Scientific notation is often useful for mentioning very large numbers.

Character Sets

Some programming languages use different data types for strings and individual characters. In Python, character literals look just like string literals and are of the string type. To mark the difference in this book, we use single quotes to enclose single-character strings, and double quotes to enclose multi-character strings. Thus, we refer to 'H' as a character and "Hi!" as a string, even though they are both technically Python strings, and both are color-coded in green in this text. As you learned in Chapter 1, all data and instructions in a program are translated to binary numbers before being run on a real computer.To support this translation, the characters in a string each map to an integer value.

This mapping is defined in character sets, among them the ASCII set and the Unicode set. (The term ASCII stands for American Standard Code for Information Interchange.) In the 1960s, the original ASCII set encoded each keyboard character and several control characters using the integers from 0 through 127. An example of a control character is Control1D, which is the command to terminate a shell window. As new function keys and some international characters were added to keyboards, the ASCII set doubled in size to 256 distinct values in the mid-1980s. Then, when characters and symbols were added from languages other than English, the Unicode set was created to support 65,536 values in the early 1990s. Unicode supports more than 128,000 values at the present time

Expressions

As we have seen, a literal evaluates to itself, whereas a variable reference evaluates to the variable's current value. Expressions provide an easy way to perform operations on data values to produce other data values. You saw strings used in expressions earlier.

Arithmetic Expressions An arithmetic expression consists of operands and operators combined in a manner that is already familiar to you from learning algebra.

Arithmetic Expressions

An **arithmetic expression** consists of operands and operators combined in a manner that is familiar to the algebra you learn in math classes.

OPERATOR	MEANING	SYNTAX
-	Negation	`-a`
**	Exponentiation	`a ** b`
*	Multiplication	`a * b`
/	Division	`a / b`
%	Remainder or modulus	`a % b`
+	Addition	`a + b`
-	Subtraction	`a - b`

EXPRESSION	EVALUATION	VALUE
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`-(3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

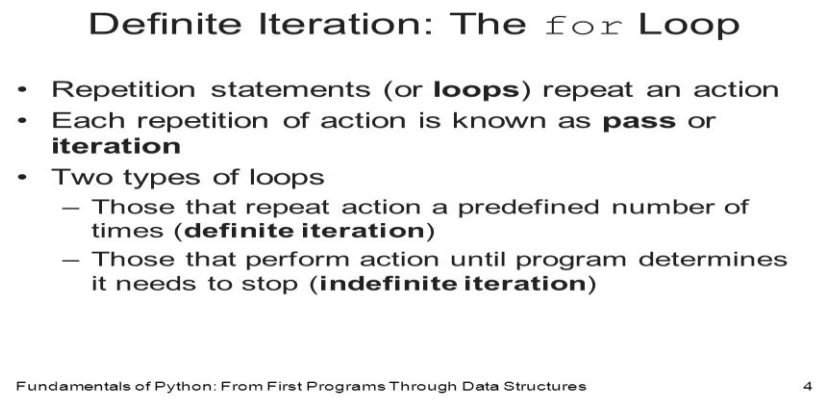
The precedence rules (AKA PEMDAS) apply in Python arithmetic expressions.

- Operations within parentheses get simplified first
- Exponents
- Multiplication
- Division (remainders are also evaluated)
- Addition
- Subtraction

There are two exceptions. Operations of equal precedence are left associative, meaning that they are evaluated from left to right. Exponentiation and assignment operations are right associative, so they are read from right to left. Utilizing parentheses would change the order of operations.

Definite Iteration: The for Loop

control statements with repetition statements, also known as loops, which repeat an action. Each repetition of the action is known as a pass or an iteration. There are two types of loops—those that repeat an action a predefined number of times (definite iteration) and those that perform the action until the program determines that it needs to stop (indefinite iteration). In this section, we examine Python's for loop, the control statement that most easily supports definite iteration




```
>>> for eachPass in range(4):

print("It's alive!", end = " ")
```

It's alive! It's alive! It's alive! It's alive!

This loop repeatedly calls one function—the print function. The constant 4 on the first line tells the loop how many times to call this function. If we want to print 10 or 100 exclamations, we just change the 4 to 10 or to 100. The form of this type of for loop is for in range(): . . The first line of code in a loop is sometimes called the loop header.

```
>>> number = 2

>>> exponent = 3

>>> product = 1

>>> for eachPass in range(exponent):

 product = product * number

print(product, end = " ")

2 4 8

>>> product 8
```

Selection: if and if-else Statements

We have seen that computers can plow through long sequences of instructions and that they can do so repeatedly. However, not all problems can be solved in this manner. In some cases, instead of moving straight ahead to execute the next instruction, the computer might be faced with two alternative courses of action. The computer must pause to examine or test a condition, which expresses a hypothesis about the state of its world at that point in time. If the condition is true, the computer executes the first alternative action and skips the second alternative. If the condition is false, the computer skips the first alternative action and executes the second alternative. In other words, instead of moving blindly ahead, the computer exercises some intelligence by responding to conditions in its environment. In this section, we explore several types of selection statements, or control statements, that allow a computer to make choices. But first, we need to examine how a computer can test conditions.

If Statement

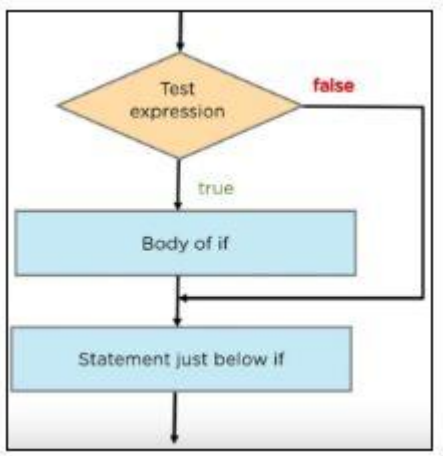
It uses the if keyword, followed by the condition.

Syntax:

if condition:

#statement to execute if the condition is true

Below is the entire workflow of how if statements work:



First, the test expression is checked. If the expression is true, the body of the if the statement is executed. If it is false, the statement present after the if statement is executed. In either case, any line of code present outside if the statement is evaluated by default.

To understand this better, we'll use an example:

a=20

if a>50:

print("This is the if body")

print("This is outside the if block")

Since 20 is not greater than 50, the statement present inside the if block will not execute. Instead, the statement present outside the if block is executed.

```
a=20
if a>50:
    print("This is the if body")
print("This is outside the if block")
```

```
This is outside the if block
```

In the code below, both the print statements will be executed since a is greater than 50.

```
a=60
if a>50:
    print("This is the if body")
print("This is outside the if block")
```

```
This is the if body
This is outside the if block
```

So far, we could specify the statements that will be executed if a condition is true. Now, if you want to evaluate statements that determine whether a condition is actual and if a separate set of statements is false, you can use the if-else conditional statement

If-Else Statement

The if-else statement is used to execute both the true part and the false part of a given condition. If the condition is true, the if block code is executed and if the condition is false, the else block code is executed.

Syntax:

if(condition):

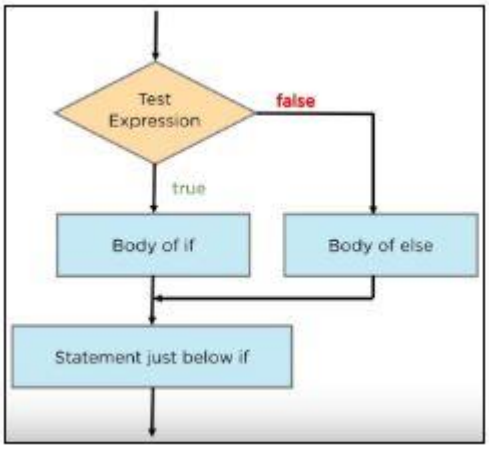
#Executes this block if the condition is true

else:

#Executes this block if the condition is false

You should note here that Python uses indentation in both the blocks to define the scope of the code. Other programming languages often use curly brackets for this purpose.

Below is the entire workflow of how the if-else statement works.



First, the test expression is checked. If it is true, the statements present in the body of the if block will execute. Next, the statements present below the if block is executed. In case the test expression has false results, the statements present in the else body are executed, and then the statements below the if-else are executed.

```
i=20
if i%2==0:
    print("This is the if block")
    print("i is an even number")
else:
    print("This is the else block")
    print("i is an odd number")
```

```
This is the if block
i is an even number
```

The following is an example that better illustrates how if-else works:

```
i=23
if i%2==0:
    print("This is the if block")
    print("i is an even number")
else:
    print("This is the else block")
    print("i is an odd number")
```

```
This is the else block
i is an odd number
```

Since the value of "i" is divisible by two, the if statement is executed.

Since the value of "i" is not divisible by two, the else statement is executed.

Let us now look at what a nested IF statement is and how it works.

If Statement

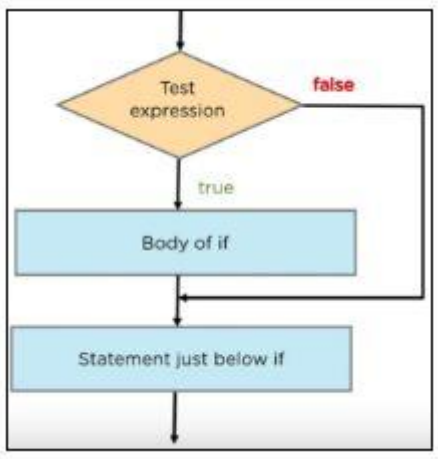
It uses the if keyword, followed by the condition.

Syntax:

if condition:

#statement to execute if the condition is true

Below is the entire workflow of how if statements work:



First, the test expression is checked. If the expression is true, the body of the if the statement is executed. If it is false, the statement present after the if statement is executed. In either case, any line of code present outside if the statement is evaluated by default.

To understand this better, we'll use an example:

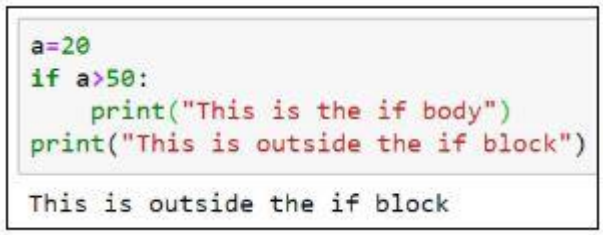
```
a=20

if a>50:

    print("This is the if body")

print("This is outside the if block")
```

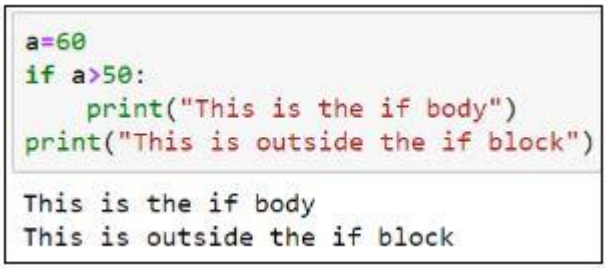
Since 20 is not greater than 50, the statement present inside the if block will not execute. Instead, the statement present outside the if block is executed.



```
a=20
if a>50:
    print("This is the if body")
print("This is outside the if block")
```

```
This is outside the if block
```

In the code below, both the print statements will be executed since a is greater than 50.



```
a=60
if a>50:
    print("This is the if body")
print("This is outside the if block")
```

```
This is the if body
This is outside the if block
```

So far, we could specify the statements that will be executed if a condition is true. Now, if you want to evaluate statements that determine whether a condition is actual and if a separate set of statements is false, you can use the if-else conditional statement.

If-Else Statement

The if-else statement is used to execute both the true part and the false part of a given condition. If the condition is true, the if block code is executed and if the condition is false, the else block code is executed.

Syntax:

if(condition):

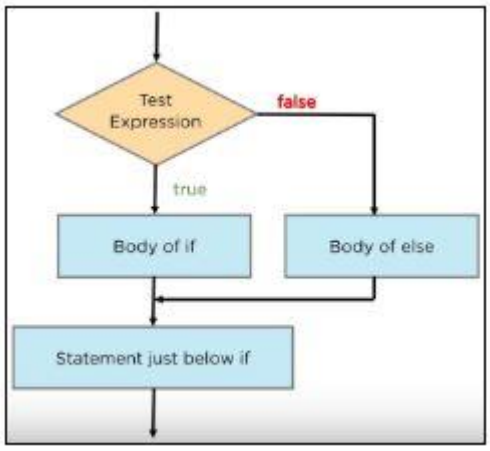
#Executes this block if the condition is true

else:

#Executes this block if the condition is false

You should note here that Python uses indentation in both the blocks to define the scope of the code. Other programming languages often use curly brackets for this purpose.

Below is the entire workflow of how the if-else statement works.



First, the test expression is checked. If it is true, the statements present in the body of the if block will execute. Next, the statements present below the if block is executed. In case the test expression has false results, the statements present in the else body are executed, and then the statements below the if-else are executed.

```
i=20
if i%2==0:
    print("This is the if block")
    print("i is an even number")
else:
    print("This is the else block")
    print("i is an odd number")
```

```
This is the if block
i is an even number
```

The following is an example that better illustrates how if-else works:

```
i=23
if i%2==0:
    print("This is the if block")
    print("i is an even number")
else:
    print("This is the else block")
    print("i is an odd number")
```

```
This is the else block
i is an odd number
```

Since the value of “i” is divisible by two, the if statement is executed.

Since the value of “i” is not divisible by two, the else statement is executed.

Let us now look at what a nested IF statement is and how it works.

Nested IF Statement

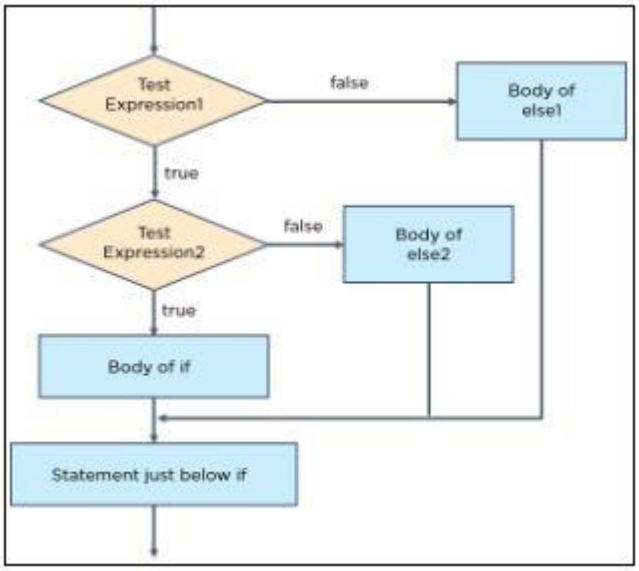
When an if a statement is present inside another if statement, it is called a nested IF statement. This situation occurs when you have to filter a variable multiple times.

Syntax:

```
if (condition1):
    #Executes if condition1 is true
    if (condition2):
        #Executes if condition2 is true
    #Condition2 ends here
#Condition1 ends here
```

In nested IF statements, you should always take care of the indentation to define the scope of each statement. You can have as many levels of nesting as required, but it makes the program less optimized, and as a result, can be more complex to read and understand. Therefore, you should always try to minimize the use of nested IF statements.

The workflow below demonstrates how nested IF statements work:



The following is another example that shows how nested IF works: We have a number, and we're going to check if the number is greater or less than 25. If the number is less than 25, we'll check if it is an odd number or an even number. If the number is greater than 25, we will print that the number is greater than 25.

```
c=21
if c<25:
    if c%2==0:
        print("c is an even number less than 25")
    else:
        print("c is an odd number less than 25")
else:
    print("c is greater than 25")
```

```
c is an odd number less than 25
```

```
c=50
if c<25:
    if c%2==0:
        print("c is an even number less than 25")
    else:
        print("c is an odd number less than 25")
else:
    print("c is greater than 25")
```

```
c is greater than 25
```

So far, with IF and if-else, we have only seen a binary approach. Suppose we have a problem that has multiple conditions. In this scenario, the if-elif-else statement comes to the rescue.

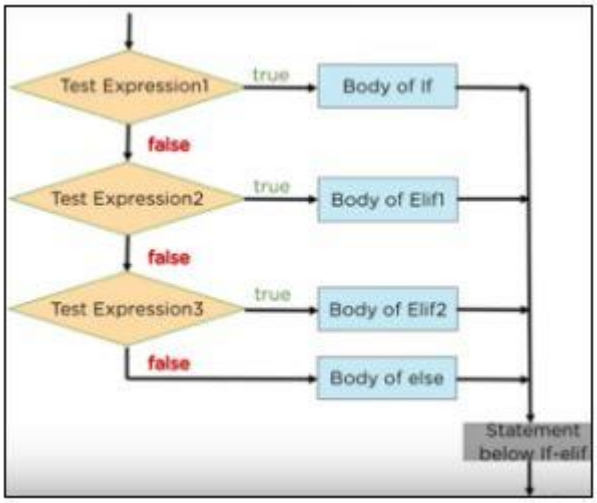
If-Elif-Else Statement

It checks the if statement condition. If that is false, the elif statement is evaluated. In case the elif condition is false, the else statement is evaluated.

Syntax:

```
if (condition):
    statement
elif (condition):
    statement
else:
    Statement
```

Below is a flowchart that shows how the if-elif-else ladder works. The Test Expression1 is checked. If that proves true, the body of if is evaluated. If it is false, then the control moves to the proceeding Test Expression2. If it's true, the body of elif1 is executed. If it's false, the test expression3 is checked. If true, the body of elif2 is executed. If it is false, the body of else is evaluated. Any statement below in if-elif is then checked.



The program below uses the if-elif-else ladder to check if a letter is a vowel or a consonant.

```
var='z'
if var=='a':
    print("This is the vowel a")
elif var=='e':
    print("This is the vowel e")
elif var=='i':
    print("This is the vowel i")
elif var=='o':
    print("This is the vowel o")
elif var=='u':
    print("This is the vowel u")
else:
    print("This is a consonant")
```

```
This is a consonant
```

```
var='e'
if var=='a':
    print("This is the vowel a")
elif var=='e':
    print("This is the vowel e")
elif var=='i':
    print("This is the vowel i")
elif var=='o':
    print("This is the vowel o")
elif var=='u':
    print("This is the vowel u")
else:
    print("This is a consonant")
```

```
This is the vowel e
```

Conditional Iteration: The while Loop

. A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is −

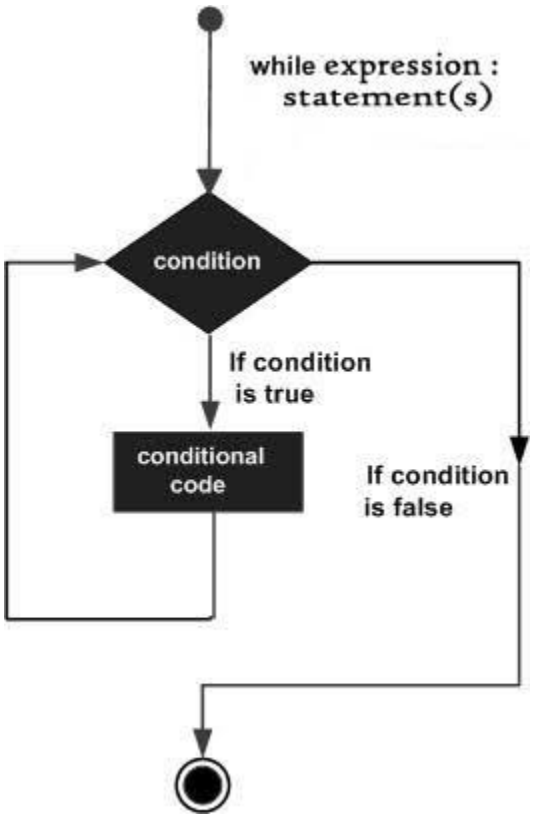
while expression:
  statement(s)

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Flow Diagram



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

Live Demo

```
#!/usr/bin/python

count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

When the above code is executed, it produces the following result −

The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7

```
The count is: 8
Good bye!
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than 9. With each iteration, the current value of the index count is displayed and then increased by 1.

The Infinite Loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

```
#!/usr/bin/python

var = 1
while var == 1 :  # This constructs an infinite loop
   num = raw_input("Enter a number  :")
   print "You entered: ", num

print "Good bye!"
```

When the above code is executed, it produces the following result −

```
Enter a number  :20
You entered:  20
Enter a number  :29
You entered:  29
Enter a number  :3
You entered:  3
Enter a number between :Traceback (most recent call last):
   File "test.py", line 5, in <module>
      num = raw_input("Enter a number :")
KeyboardInterrupt
```

Above example goes in an infinite loop and you need to use CTRL+C to exit the program.

Using else Statement with While Loop

Python supports to have an **else** statement associated with a loop statement.

- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

```
#!/usr/bin/python

count = 0
while count < 5:
   print count, " is  less than 5"
   count = count + 1
else:
   print count, " is not less than 5"
```

When the above code is executed, it produces the following result −

0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5