UNIT -1

(PART 1)

INTRODUCTION TO PYTHON

SYLLABUS :

Introduction: Introduction to Python, Program Development Cycle, Input, Processing, and Output, Displaying Output with the Print Function, Comments, Variables, Reading Input from the Keyboard, Performing Calculations, Operators. Type conversions, Expressions, More about Data Output.

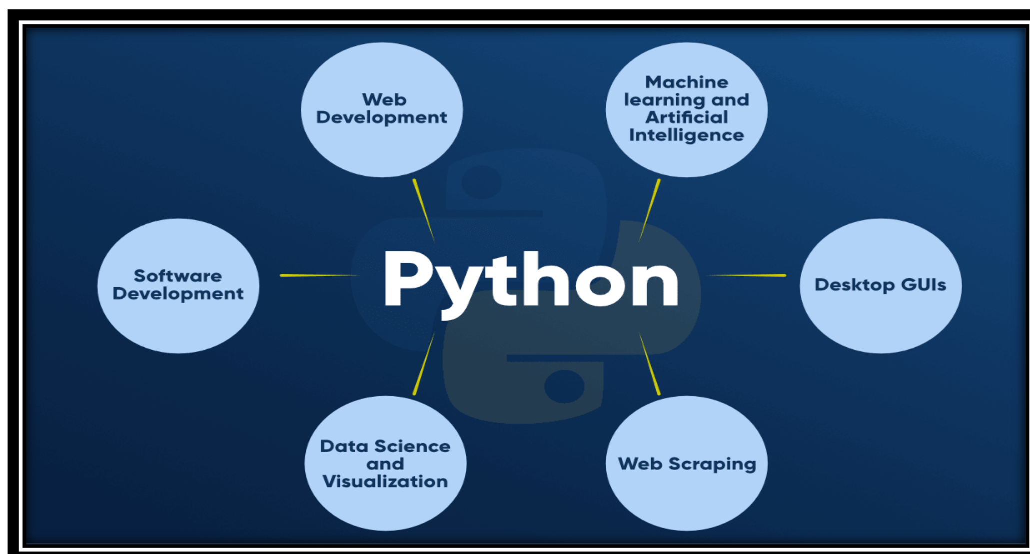
Data Types, and Expression: Strings Assignment, and Comment, Numeric Data Types and Character Sets, Using functions and Modules.

Decision Structures and Boolean Logic: if, if-else, if-elif-else Statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables. Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.

Introduction to Python

Python was developed by Guido Van Rossum in 1991 at National Research Institute for Mathematics and Computer Science in the Netherlands.

Application Areas of Python:



Features of Python :

- **Simple**

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- Python is a simple programming language. Reading a program written in python feels almost like reading English.
- Programmers can concentrate on the solution rather than the language.

➢ **Easy to learn**

- A python program is clearly defined and easily readable. The structure of the program is very simple.

➢ **Free and open source**

- Python is an example of open source software.
- Anyone can freely distribute it, read the source code, edit it, and even use the code to write new programs

➢ **High-level Language**

- When writing programs in python, the programmers don't have to worry about the low-level details like managing memory used by the program.

➢ **Portable**

- Python is a portable language. The programs work on any of the operating systems like Linux, Windows, Macintosh, Solaris, Palm OS..etc.

➢ **Interpreted**

- Python is processed at runtime by the interpreter. No need to compile the program before executing it.
- Python converts the source code to intermediate form called byte code, which is translated into the native language of your computer so that it can be executed.

➢ **Object Oriented**

- Python supports procedure-oriented programming as well as object-oriented programming.
- In procedure-oriented language, the program is built around procedures or functions which are nothing but reusable pieces of programs.

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

– In object oriented languages, the program is built around objects which combine data and functionality.

➢ **Extensible**

– You can write some of your python code in other languages . It can be extended to other languages.

➢ **Extensive Libraries**

– Python has a huge library functions are compatible on UNIX, Windows, Macintosh and allow programmers to perform wide range of applications varying from text processing, maintaining databases, to GUI programming.

Program Development Cycle:

- Computer scientists refer to the process of planning and organizing a program as **software development.**
- One version of software development is known as the **waterfall model.**

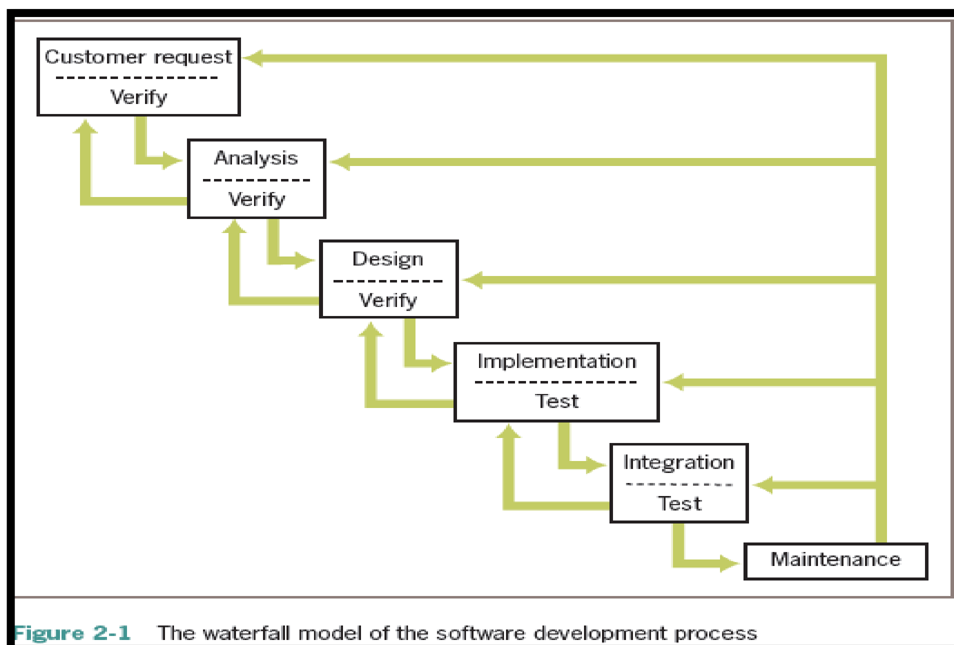


Figure 2-1 The waterfall model of the software development process

- The waterfall model consists of several phases:

1. **Customer request—**

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

In this phase, the programmers receive a broad statement of a problem that is potentially amendable to a computerized solution. This step is also called the user requirements phase.

2. **Analysis—**

   The programmers determine what the program will do. This is sometimes viewed as a process of clarifying the specifications for the problem.

3. **Design—**

   The programmers determine how the program will do its task.

4. **Implementation**—

   The programmers write the program. This step is also called the coding phase.

5. **Integratio**n—

   Large programs have many parts. In the integration phase, these parts are brought together into a smoothly functioning whole, usually not an easy task.

6. **Maintenance**—

   Programs usually have a long life; a life span of 5 to 15 years is common for software.

   During this time, requirements change, errors are detected, and minor or major modifications are made.

- There is more to software development than writing code.
- If you want to reduce the overall cost of software development, write programs that are easy to maintain.
  - This requires thorough analysis, careful design, and a good coding style.
- Keep in mind that mistakes found early are much less expensive to correct than those found late.
- Following figure illustrates some relative costs of repairing mistakes when found in different phases.

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

– These are not just financial costs but also costs in time and effort.

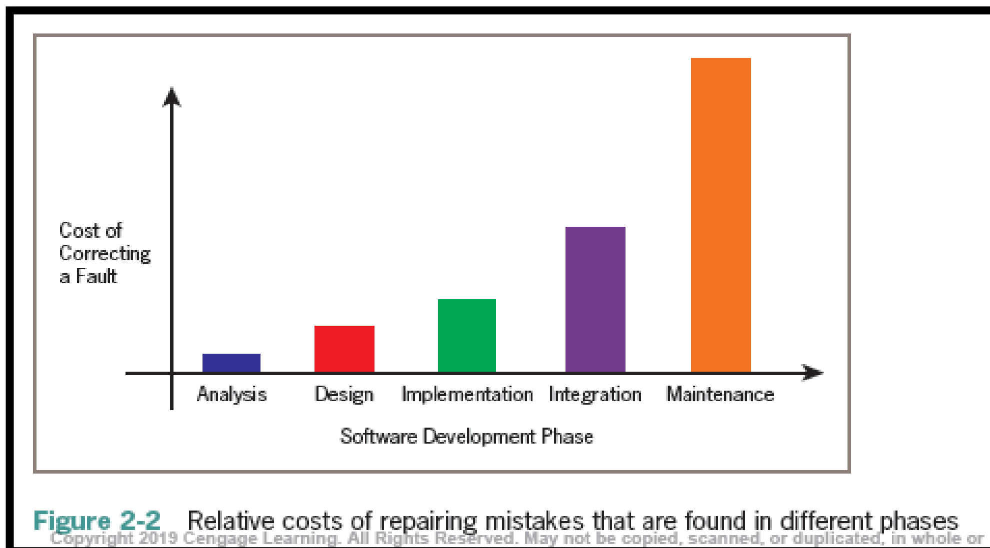


Figure 2-2 Relative costs of repairing mistakes that are found in different phases

Copyright 2019 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or i

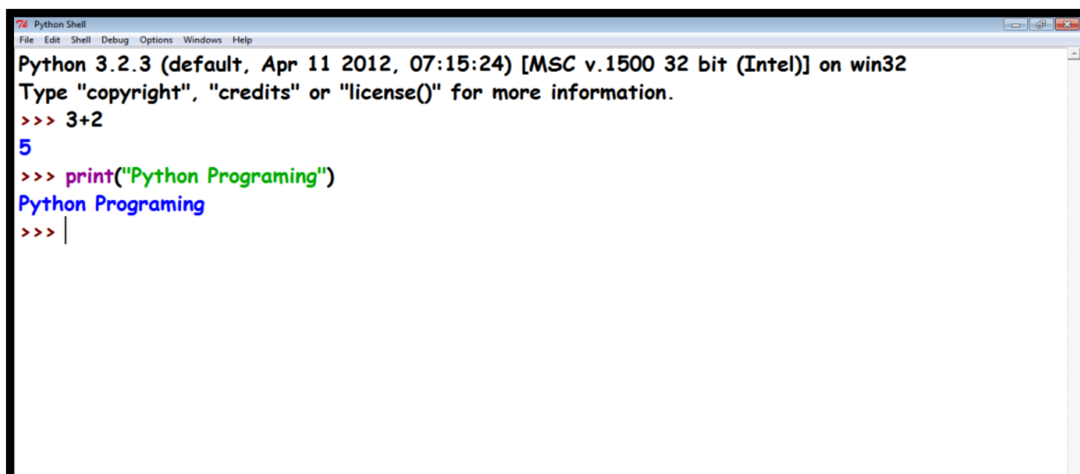
Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Getting Started with Python Programming:

- Python is a high-level, general-purpose programming language for solving problems on modern computer systems.
- The language and many supporting tools are free, and Python programs can run on any operating system.
- You can download Python, its documentation, and related materials from www.python.org.

Running Code in the Interactive Shell :

- You can run simple Python expressions and statements in an interactive programming environment called the **shell.**
- The easiest way to open a Python shell is to launch the **IDLE** (Integrated DeveLopment Environment).
  - This is an integrated program development environment that comes with the Python installation.
  - When you do this, a window named Python Shell opens.
- A shell window contains an opening message followed by the special symbol >>>, called **a shell prompt**.
- When you enter an expression or statement, Python evaluates it and displays its result, if there is one, followed by a new prompt.



Surya Lakshmi Kantham Vinti
CSE Dept
ACET

COLOR CODING OF PYTHON PROGRAM ELEMENTS:

Color	Type of Element	Examples
Black	Inputs in the IDLE shell Numbers Operator symbols Variable, function, and method references Punctuation marks	`67, +, name, y = factorial(x)`
Blue	Outputs in the IDLE shell Function, class, and method names in definitions	`'Ken Lambert',` `def factorial(n)`
Green	Strings	`"Ken Lambert"`
Orange	Keywords	`def, if, while`
Purple	Built-in function names	`abs, round, int`
Red	Program comments Error messages in the IDLE shell	`# Output the results` `ZeroDivisionError: division by zero`

Table 1-1 Color-coding of Python program elements in IDLE

Input, Processing, and Output :

- The Python shell itself is such a program;
  - its inputs are Python expressions or statements.
  - Its processing evaluates these items.
  - Its outputs are the results displayed in the shell.

Output :

- The programmer can force the output of a value by using the print function.
- It can take any of the following forms:

1) **print(<expression>)**

Here <expression> can be replaced with an expression that need to be evaluated or any string that can be printed.

```
>>> print(3+4)
7
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
>>> print("Hello CSE!!")
Hello CSE!!
```

2) **print(<expression>,..., <expression>)**

- Note the ellipsis (...) in this syntax example. The ellipsis indicates that you could include multiple expressions after the first one.
- The print function evaluates the expressions and displays their results, separated by single spaces, on one line.

```
>>> print(3+4,5*9,"hai cse")
  7  45   hai cse
```

3) **print(<expression>, end = "")**

- Whether it outputs one or multiple expressions, the print function always ends its output with a **newline.**
- To begin the next output on the same line as the previous one, you can place the expression end = "", which says "end the line with an empty string instead of a newline," at the end of the list of expressions

```
>>> def hai():
        print("hai")
        print("hello")
>>> hai()
hai
hello
>>> def hai():
        print("hai",end="")
        print("hello")
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
>>> hai()
Haihello
```

4) print(<expression>,...,<expression>, sep='<literal>')

sep stands for **separator** and is assigned a single space (' ') by default. It determines the value to join elements with.

```
>>>print(3+4,"hai",5*7)
7 hai 35
>>> print(3+4,"hai",5*7,sep='-->')
7-->hai-->35
>>>print(3+4,"hai",5*7,sep='\n')
7
hai
35
```

Input :

- you'll often want your programs to ask the user for input. You can do this by using the **input function**.
- This function causes the program to stop and wait for the user to enter a value from the keyboard.
- The form of an assignment statement with the input function is the following:

<variable identifier> = input(<a string prompt>)

- The input function does the following:

1. Displays a prompt for the input. In this example, the prompt is "Enter your name: ".

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

2. Receives a string of keystrokes, called characters, entered at the keyboard and returns the string to the shell.

```
>>> name=input("Enter a name")

Enter a nameSurya

>>> name

'Surya'

>>> a=input("Enter a number")

Enter a number9

>>> a

'9'                  # 9 is treated as String

>>> b=input("Enter a real number")

Enter a real number9.563

>>> b

'9.563'              #9.563  is also treated as string
```

- The input function always builds a string from the user’s keystrokes and returns it to the program.
- After inputting strings that represent numbers, the programmer must convert them from strings to the appropriate numeric types.
- In Python, there are two type conversion functions for this purpose, called
  - **int (for integers) and**
  - **float (for floating point** numbers).

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
a=input("Enter a number")
Enter a number9
>>> a
'9'
a=int(input("Enter a number"))
Enter a number9
>>> a
9
```

```
>>> b=input("Enter a real number")
Enter a real number9.563
>>> b
'9.563'
>>> b=float(input("Enter a real number"))
Enter a real number9.563
>>> b
9.563
```

Editing, Saving, and Running a Script :

- To compose and execute programs in this manner, you perform the following steps:

  1. Select the option **New Window from the File menu of the shell window.**

  2. In the new window, enter Python expressions or statements on separate lines, in the order in which you want Python to execute them.

  3. At any point, you may save the file by selecting File/Save. If you do this, you should use a **. py extension.**

  For example, your first program file might be named myprogram.py.

  4. To run this file of code as a Python script, select **Run Module from the Run menu** or press the F5 key.

Comments :

- A comment is a piece of program text that the computer ignores but that provides useful documentation to programmers.

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- At the very least, the author of a program can include his or her name and a brief statement about the program's purpose at the beginning of the program file.
- end-of-line comments can document a program. These comments begin with the # symbol and extend to the end of a line. An end-of-line comment might explain the purpose of a variable or the strategy used by a piece of code.

  >>> RATE = 0.85 # Conversion rate for Canadian to US dollars

It's always better on a programmer's side to:

1) Begin a program with a statement of its purpose and other information that would help orient a programmer called on to modify the program at some future date.

2) Accompany a variable definition with a comment that explains the variable's purpose.

3) Precede major segments of code with brief comments that explain their purpose.

4) Include comments to explain the workings of complex or tricky sections of code

Variables :

- **Variable associates a name with a value, making it easy to** remember and use the value later in a program.
- Variables serve two important purposes in a program.
  - They help the programmer keep track of data that change over time.
  - They also allow the programmer to refer to a complex piece of information with a simple name

How to frame a Variable name(identifier) :

- 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.

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- 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 interestRate is slightly easier to read than the name interestrate
- 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.**
- Variables receive their initial values and can be reset to new values with an **assignment statement.**

SYNTAX :

The simplest form of an assignment statement is the following:

<variable name> = <expression>

  - The Python interpreter first evaluates the expression on the right side of the assignment symbol and then binds the variable name on the left side to this value.
  - When this happens to the variable name for the first time, it is called **defining or initializing the variable.**

```
>>>9rate=78     #Started with number

SyntaxError: invalid syntax

>>> rate=78

>>> rate

78

>>> _rate=47

>>> rate

78
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
>>> _rate
47
>>> @si=98.45          #Started with special symbol
SyntaxError: invalid syntax
>>> si=98.45
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Python Operators :

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators
  - Assignment operators

1. Arithmetic operators:

Some basic arithmetic operators are +,-,*,/,%,** and //

We can apply these operators in numbers as well as variables to perform corresponding operations

Let a= 10, b= 20

Operator	Description	Example	Output
+	It is used to perform addition operation (add the operands)	a+b	30
-	Perform substation operation (subtract the operands)	a-b	-10
*	Perform the multiplication operation	a*b	200
/	It returns quotient	a/b	0.5
%	It returns the reminder	b%a	0
**	Perform power calculation(Exponent)	a**b	10^{20}
//	Floor division	a//b	20

<u>Example:</u>

```
a = float (input (“enter a number”))
b= float (input (“enter b value “))
print (“Addition of “,a,” and “, b,” is “,a+b)
print (“Subtraction of “,a,” and “,b,” is “,a-b)
print(“Product of “,a,” and “,b,” is “,a*b)
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
print ("Division of ",a," and ",b," is ",a/b)
print("Modulus division of ",b," and ",a," is ",b%a)
print("Floor division of ",a," and ",b," is ",a//b)
print("Exponent of ",a," and ",b," is ",a**b)
```

2) **Comparison Operator: (Relational Operators):**

- It is used to compare the values on its either sides (Left and Right) of the operator and determines the relation between them.
- Let a = 100, b = 200

Operator	Description	Example	Output
==	It returns true if the two values on either side of the operator are exactly same	a==b	FALSE
!=	It returns true if the two values on either side of the operator are not same	a!=b	TRUE
<	It returns true if the value at LHS of the operator is less than the value at the RHS of the operator, otherwise false	a<b	TRUE
>	It returns true if the value at LHS of the operator is greater than the value at the RHS of the operator, otherwise false	a>b	FALSE
<=	It returns true if the value at LHS of the operator is less than or equal to the value at the RHS of the operator, otherwise false	a<=b	TRUE
>=	It returns true if the value at LHS of the operator is greater than or equal to the value at the RHS of the operator, otherwise false	a>=b	FALSE

Example:

```
a = float(input("Enter the value of a: "))
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
b = float(input(“Enter the value of b: “))

print(a,” == “,b,a==b)

print(a,” != “,b,a!=b)

print(a,” < “,b,a<b)

print(a,” > “,b,a>b)

print(a,” <= “,b,a<=b)

print(a,” >= “,b,a>=b)
```

3) Logical Operators:

- Logical operators are used to evaluate two or more expressions with relational operators.
- Python supports 3 logical Operators

1. Logical AND (and)
2. Logical OR (or)
3. Logical NOT (not)

Logical AND (and): If expression on both sides of the logical operator are true, then the whole expression is true.

Truth Table:

Exp 1	Exp 2	Exp 1 and Exp2
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Example:

Let a = 10, b = 20, c = 30, d = 40

(a<d) and (b>c)

(True) and (False)

Result = False

Logical OR (or): If one or both the expressions is true then the whole expression is true.

Truth Table:

Exp 1	Exp 2	Exp1 or Exp2
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

Example: Let a = 10, b = 20, c = 30, d = 40

(a<d) or (b>c)

(True) or (False)

Result = True

Logical NOT (not):

- Logical NOT operator takes a single expression and negates the value of expression.
- Logical NOT produces a zero if the expression evaluates to a non-zero and produces 1 if the expression produces zero.

Truth Table:

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Expression	!(Expression)
T (1)	F (0)
F (0)	T (1)

Example: Let a = 10, b = 20

not (a>b)

not (False)

Result : True

4) Identity Operators:

These operators compare the memory locations of two objects. Python supports two types of Identity Operators

- **is** operator: It returns true if operands (or) values on both sides of the operator point to the same object and False otherwise.

  Example: If a is b – it returns 1 if id (a) is same as id (b)

- **is not** operator : It returns true if operands (or) values on both sides of the operator do not point to the same object and False otherwise.

  Example: If a is not b – it returns 1 if id (a) is not same as id (b)

5) Membership Operators:

Python supports two types of membership operators

- **in**
- **not in**

The names itself suggests that, test for membership in a sequence such as string, list, tuple.

in – The operator returns true if a variable is found in the specified sequence and false otherwise.

Example: a in nums - it returns True if ‘a’ is present in the nums

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

not in – The operator returns true if a variable is not found in the sequence.

Example: a not in nums - it returns 1 if ‘a’ is not present in the nums

```
>>>name="Surya"
>>> 'u' in name
True
>>> 'b' in name
False
>>> m=[45,67,12,42,55,564,21]
>>> 45 in m
True
>>> 34 in m
False
>>> 45 not in m
False
>>> 34 not in m
True
```

6) Bitwise Operators :

Bitwise Operators perform operations at bit level. These operators include

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise NOT (~)
- Bitwise XOR (^)
- Bitwise Shift

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Bitwise operators expect their operands to be of integers and track them as a sequence of bits.

Bitwise AND (&):

The bit in the first operand is Anded with the corresponding bit in the second operand. If both the bits are 1 then the corresponding bit result is 1 and 0 otherwise**.**

Truth Table:

Bit1	Bit2	Bit1 & Bit2
1	1	1
1	0	0
0	1	0
0	0	0

Example:

1010101010

&

1111010101

1010000000

Bitwise OR (|):

The bit in the first operand is Ored with the corresponding bit in the second operand. If either of the bits are 1 then the corresponding bit result is 1 and 0 otherwise.

Truth Table:

Bit1	Bit2	Bit1 & Bit2
1	1	1
1	0	1

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

0	1	1
0	0	0

Example:

1 0 1 0 1 0 1 0 1 0

|

1 1 1 1 0 1 0 1 0 1

1 1 1 1 1 1 1 1 1 1

Bitwise XOR (^):

The bit in the first operand is XORed with the corresponding bit in the second operand. If one of the bits is 1, then the corresponding bit result is 1 otherwise 0

Truth Table:

Bit1	Bit2	Bit1 & Bit2
1	1	0
1	0	1
0	1	1
0	0	0

Example:

1 0 1 0 1 0 1 0 1 0

^

1 1 1 1 0 1 0 1 0 1

0 1 0 1 1 1 1 1 1 1

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Bitwise NOT (~):

It is a unary operation, which performs logical negation on each bit of the operand. It produces 1’s compliment of the given value

Truth Table:

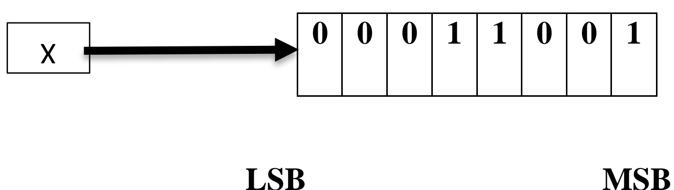
Bit	~(Bit)
0	1
1	0

Example:

~ (10101010) = 01010101

Shift Operators:

- Python supports two bitwise shift operators
  1. Shift Left (<<)
  2. Shift Right (>>)
- These operators are used to shift bits to the left or right.
- Syntax of shift operation is <u>operand op num</u>
- Let x = 00011010

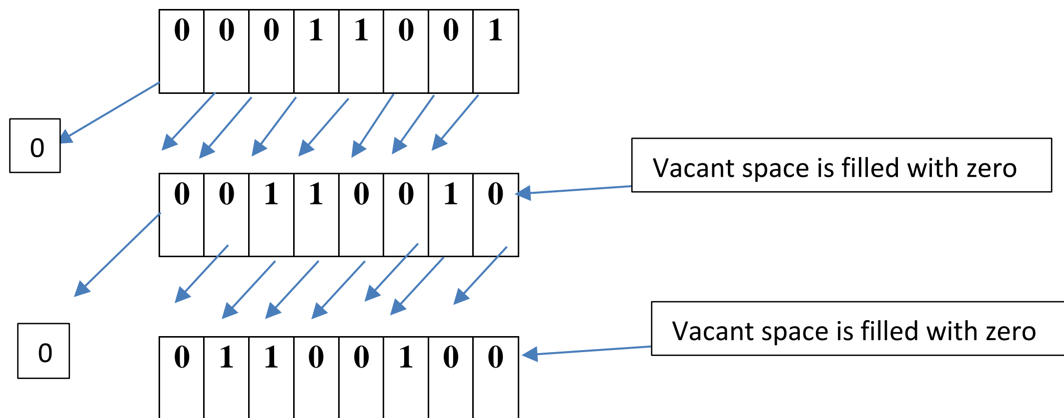


LSB **MSB**

- **Example 1: shift Left**

  x=3

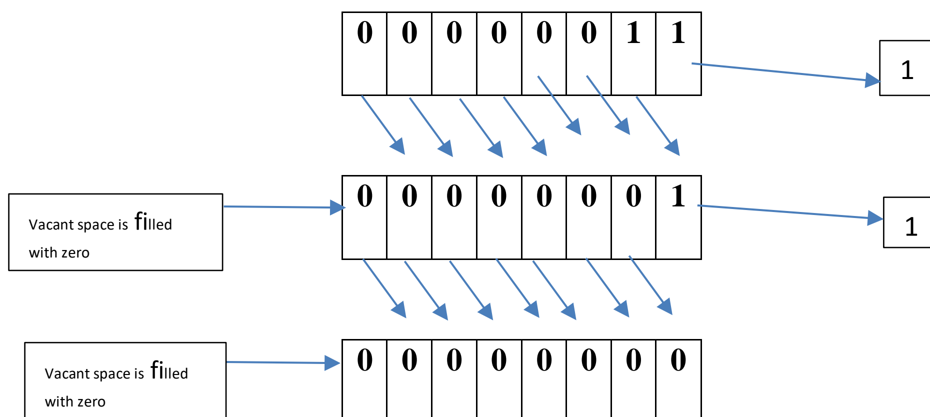
  x<<2



- **Example 1: shift Right**

x=3

x>>2




```
>>> x=5
>>> y=3
>>> x&y
1
>>> x|y
7
>>> x^y
6
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
>>> ~x
-6
>>> x<<1
10
>>> x>>1
2
>>> x>>2
1
```

7) Assignment and In place OR shortcut operators

Assignment Operator: It is used to assign a value to the operand.

Syntax: variablename=value

variablename=Expression

Example 1 : a=10 Example 2: c=a+b

b=20, c=30

Inplace Operator:

In place operators are called as shortcut operators that includes +=,-=, *=, /=, %=, //=, **= allow you to write code like num = num + 10 more concisely as num+ = 3

Operator	Description	Example
=	Assign right side value to the left hand side variable	a = b
+=	Add and assign	a+=b=> a=a+b

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

-=	Subtract and assign	a-=b=> a=a-b
*=	Multiply and assign	a*=b=> a=a*b
/=	Divide and assign	a/=b=> a=a/b
%=	Modulus and assign	a%=b=> a=a%b
//=	Floor division and assign	a//=b=> a=a//b
**=	Exponent and assign	a**=b=> a=a**b

Unary Operators:

- Acts on single operand
- Python supports single minus (-)
- When an operand is preceded by a minus sign, the unary operator negates its value.

  Example:

  ```
  b=10
  print(-b)
  ```

Operator Precedence and Associativity:

The following table is the list from higher precedence to lower precedence.

**	Exponentiation
~, +, -	Unary Operator
*, /, //, %	Multiply, division, floor division, modulus division
+, -	Addition and subtraction

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

>>, <<	Right and left bitwise shift operator
&	Bitwise AND
^, |	Bitwise XOR and regular OR
<=, >=, <, >	Comparison operator
<>, ;, ==, !=	Equality operator
=, %=, +=, -=	Assignment operators
Is, is not	Identity operators
In, not in	Membership operators
NOT, OR, AND	Logical Operators

Expressions in Python:

- An expression is any logical combination of symbols (like variables, constants and operators) that represent a value.
- Every language has some set of rules to define whether the expression is valid/invalid.
- In python, an expression must have at least have one operand can have one or more operators.

Example:a+b*c-5

In above expression

" +, *, - " are operators

" a, b, c " are operators

" 5 " is operators

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- Valid Expression:

x = a/b

y = a*b

z = a^b

x = a>b

- Invalid Expressions:a+<y++

<u>Types of Expression:</u>

Python supports different types of expressions that can be classified as follows

- Based on position of operators in an expression
- Based on the datatype of the result obtained on evaluating an expression

- **Based on position of operators in an expression**

These type of expressions include:

Infix expression: In this type of expression the operator is placed in between the operands

Example: a=b-c

Prefix expression: In this type of expression the operator is placed before the operands

Example: a=-bc

Post fix expression: In this type of expression the operator is placed after the operands

Example: a=bc-

- **Based on the data type of the result obtained on evaluating an expression:**

These type of expressions include:

Constant Expression:

This type of expression involves only constants.

Example: 8+9-2

Floating Point Expression:

This type of expression produces floating point results

Example: a = 10, b = 5

a*b/2

Integer Expression:

This type of expression produces integer result after evaluating the expression.

Example: a = 10

b = 5

c = a*b

Relational Expression:

This type of expression returns either true or false

Example: c = a>b

Result: **True**

Logical Expression:

This type of expression combines two or more relational expressions and return a value as true or false

Example: (a>b) and (a!=b)

False and True

Result:True

Bitwise Expressions:

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

This type of expressions manipulate data at bit level.

Example: x = y&z

Assignment Expression:

In this type expression will assign a value or expression to a variable

Example: c=a+b

c=10

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

UNIT -1

PART -2

1. Standard Data Types

- The data stored in memory can be of many types.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types-
    - Numbers
    - String
    - List
    - Tuple
    - Dictionary

1.1 Python Numbers

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports different numerical types −
    - int (signed integers)
    - float (floating point real values)
    - complex (complex numbers)
    - A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are real numbers and j is the imaginary unit.
    - ***bool*** (Boolean values)
        - True False
- A number is also an **immutable type**, meaning that changing or updating its value results in a newly allocated object.

1.1.1 How to Create and Assign Numbers (Number objects)

- Creating numbers is as simple as assigning a value to a variable:

anInt = 1

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

aFloat = 3.1415926535897932384626433832795

aComplex = 1.23+4.56J

1.1.2 How to Update Numbers :

- update an existing number by (re)assigning a variable to another number.
- Every time you assign another number to a variable, you are creating a new object and assigning it
    - anInt += 1

1.1.3 How to Remove Numbers :

- delete a reference to a number object, just use the **del statement**
    - del anInt
    - del aLong, aFloat, aComplex

i) Standard (Regular or Plain) Integers

- Most machines (32-bit) running Python will provide a range of -2^31 to 2^31 -1, that is -2, 147,483,648 to 2,147,483,647.
- If Python is compiled on a 64-bit system with a 64-bit compiler, then the integers for that system will be 64-bit.
- Examples of Python integers:

0101 84 -237 0x80 (mention x for hexadeximal)

017 -680 -0x92 0o234 (mention o for octal)

ii) Boolean

- Objects of this type have two possible values, True or False.

```
>>>t=4>8
>>>type(t)
<class 'bool'>
```

iii) Floating Point Numbers :

- Floats in Python are values that can be represented in straightforward decimal or scientific notations.

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- These 8-byte (64-bit) values conform to the IEEE 754 definition (52M/11E/1S) where 52 bits are allocated to the mantissa, 11 bits to the exponent , and the final bit to the sign.
- Floating point values are denoted by a decimal point (.) in the appropriate place and an optional "e“ suffix representing scientific notation.
- Here are some floating point values:

  0.0 -777. 1 -5.555567119 9.6e3 9.384e-23

iv) Complex Numbers

- A complex number is any ordered pair of floating point real numbers (x, y) denoted by x + yj where x is the real part and y is the imaginary part of a complex number.
- Facts about Python's support of complex numbers:
  - Imaginary numbers by themselves are not supported in Python (they are paired with a real part of 0.0 to make a complex number)
  - Complex numbers are made up of real and imaginary parts
  - Syntax for a complex number: *real+imagj*
  - Both real and imaginary components are floating point values
  - Imaginary part is suffixed with letter "J" lowercase (j) or uppercase (J)
- The following are examples of complex numbers:

  64.375+1j 4.23-8.5j 0.23-8.55j

1.2 Strings:

- Strings are sequence of individual characters enclosed in single or double quotes.

  Ex :- >>>str=“hai how are u?”

  >>> new= ‘this is another example’

- Strings are immutable, meaning that changing an element of a string requires creating a new string.

1.2.1 How to Create and Assign Strings

- Creating strings is as simple as using a scalar value and assign it to a variable:

```
>>> aString = 'Hello World!'       # using single quotes
```

```
>>> another = "Python is cool!"   # double quotes
>>> print(aString)                # print, no quotes!
Hello World!
>>> another                       # no print, quotes!
'Python is cool!'
```

1.2.2 **How to Access Values in Strings**

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring:

```
>>> aString = 'Hello World!'
>>> aString[0]
'H'
>>> aString[1:5]
'ello'
>>> aString[6:]
'World!'
```

1.2.3 How to Update Strings

- You can "update" an existing string by (re)assigning a variable to another string.

```
>>>aString='Hello World'
>>> aString = aString[:6] + 'Python!'
>>> aString
'Hello Python!'
>>> aString = 'different string altogether'
>>> aString
'different string altogether'
```

1.2.4 How to Remove Characters and Strings

- Strings are immutable, so you cannot remove individual characters from an existing string.
- To clear or remove a string, you assign an empty string or use the del statement, respectively:

  >>> aString = ''

  >>> aString

  ‘’

  >>> del aString

EXAMPLES :

str = 'Hello World!'

>>> print (str) # Prints complete string

Hello World!

>>> print (str[0]) # Prints first character of the string

H

>>> print (str[2:5]) # Prints characters starting from 3rd to 5th

llo

>>> print (str[2:]) # Prints string starting from 3rd character

llo World!

>>> print (str * 2) # Prints string two times

Hello World!Hello World!

>>> print (str + "TEST") # Prints concatenated string

Hello World!TEST

1.3 LISTS :

- Lists provide sequential storage through an index offset and access to single or consecutive elements through slices.
- Lists are flexible container objects that hold an arbitrary number of Python objects

- >>>a=[45,7,9,78]

  >>>a

  [45,7,9,78]

1.3.1 How to Create and Assign Lists

- Creating lists is as simple as assigning a value to a variable.
- Lists are delimited by surrounding square brackets ([ ])

>>>a=[8,9,'a',6.8,"hello"]

>>>a

[8, 9, 'a', 6.8, 'hello']

>>>b=[8,9,'a',6.8,[66,"bye"],"hello"]

>>>b

[8, 9, 'a', 6.8, [66, 'bye'], 'hello']

1.3.2 How to Access Values in Lists

- Slicing works similar to strings; use the square bracket slice operator ([ ]) along with the index or indices.

>>>aList = [123, 'abc', 4.56, ['inner', 'list'], 7-9j]

>>> aList[0]

123

>>> aList[1:4]

['abc', 4.56, ['inner', 'list']]

>>> aList[:3]

[123, 'abc', 4.56]

>>> aList[3][1]

'list'

1.3.3 How to Update Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method:

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
aList
[123, 'abc', 4.56, ['inner', 'list'], (7-9j)]
>>> aList[2]
4.56
>>> aList[2] = 'float replacer'
>>> aList
[123, 'abc', 'float replacer', ['inner', 'list'], (7-9j)]
>>>aList.append(84)
>>> aList
[123, 'abc', 'float replacer', ['inner', 'list'], (7-9j),84]
```

1.3.4 How to Remove List Elements and Lists

- To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

```
>>> aList
[123, 'abc', 'float replacer', ['inner', 'list'], (7-9j)]
>>> del  aList[1]
>>> aList
[123, 'float replacer', ['inner', 'list'], (7-9j)]
>>> aList.remove(123)
>>> aList
['float replacer', ['inner', 'list'], (7-9j)]
```

1.4 Tuples :

- Tuples are another container type extremely similar in nature to lists.
- The only visible difference between tuples and lists is that tuples use parentheses () and lists use square brackets[].

- Functionally, there is a more significant difference, and that is the fact that tuples are **immutable**. Because of this, tuples can do something that lists cannot do . . . be a dictionary key.

1.4.1 How to Create and Assign Tuples

- Creating and assigning tuples are practically identical to creating and assigning lists, i.e., enclosing objects in paranthesis() and assigning the tuple to an identifier.

```
>>> a=(34,56,"hai",21.54)

>>> a

(34, 56, 'hai', 21.54)

>>> tuple("hai")

('h', 'a', 'i')

>>> a=(34,(56,"hai"),21.54)

>>> a

(34, (56, 'hai'), 21.54)
```

1.4.2 How to Access Values in Tuples

- Slicing works similarly to lists. Use the square bracket slice operator ([ ]) along with the index or indices.

```
>>> a=(34,56,"hai",21.54)

>>> a[2]

'hai'

>>> a[1:4]

(56, 'hai', 21.54)
```

1.4.3 How to Update Tuples

- Like numbers and strings, tuples are immutable, which means you cannot update them or change values of tuple elements.
- We were able to take portions of an existing tuple to create a new tuple.

```
>>> a

(34, 56, 'hai', 21.54)

>>> b=(a[2],a[1],a[0])
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
>>> b
('hai', 56, 34)
```

1.4.4 How to Remove Tuple Elements and Tuples

- Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.
- To explicitly remove an entire tuple, just use the **del** statement to reduce an object's reference count

  del aTuple

1.5 Dictionary :

- Dictionaries are the sole mapping type in Python. Mapping objects have a one-to-many correspondence between *hashable values (keys) and the objects they represent (values).*
- *They* can be generally considered as *mutable hash tables.*
- *A dictionary object itself is mutable and is yet* another container type that can store any number of Python objects, including other container types.

1.5.1 How to Create and Assign Dictionaries

- The syntax of a dictionary entry is *key:value . Also,* dictionary entries are enclosed in braces ({ })
- Creating dictionaries simply involves assigning a dictionary to a variable, regardless of whether the dictionary has elements or not:

```
>>> dict1={}
>>> dict2={"name":"Surya","last":"vinti","college":"CMREC"}
>>> dict1
{}
>>> dict2
{'name': 'Surya', 'last': 'vinti', 'college': 'CMREC'}
>>> dict3={1:23,3:45,2:98}
>>> dict3
{1: 23, 3: 45, 2: 98}
```

1.5.2 How to Access Values in Dictionaries

- To traverse a dictionary (normally by key):

```
>>> dict3={1:23,3:45,2:98}
>>> dict3[1]
23
>>> dict2
{'name': 'Surya', 'last': 'vinti', 'college': 'CMREC'}
>>> dict2["name"]
'Surya'
```

1.5.3 How to Update Dictionaries

- You can update a dictionary by adding a new entry or element (i.e., a key-value pair), modifying an existing entry, or deleting an existing entry

```
>>> dict2= {'name': 'Surya', 'last': 'vinti', 'college': 'CMREC'}
>>> dict2["name"]="Lakshmi"
>>> dict2
{'name': 'Lakshmi', 'last': 'vinti', 'college': 'CMREC'}
>>> dict3={1:23,3:45,2:98}
>>> dict3[3]=45.7
>>> dict3
{1: 23, 3: 45.7, 2: 98}
```

1.5.4 How to Remove Dictionary Elements and Dictionaries

- Either remove individual dictionary elements or clear the entire contents of a dictionary

```
>>> dict3
{1: 23, 3: 45.7, 2: 98}
>>> del dict3[2]
>>> dict3
{1: 23, 3: 45.7}
```

```
>>> del dict3

>>> dict3

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

NameError: name 'dict3' is not defined
```

2. Character Sets

- In Python, character literals look just like string literals and are of the string type.
- 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**

2.1 ASCII:

- 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.

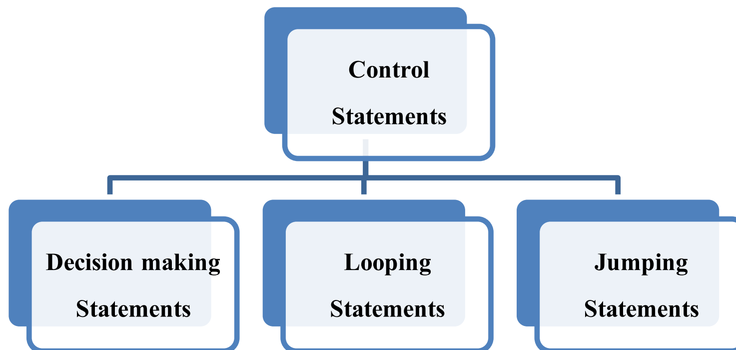
	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	I	j	k	l	m
11	n	o	P	q	r	S	t	u	v	w
12	X	y	z	{	|	}	~	DEL		

2.2 Unicode :

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

- 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.

3. CONTROL STATEMENTS



3.1 Decision making Statements :

- Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.
- Python programming language assumes any non-zero and non-null values as TRUE, and any zero or null values as FALSE value.
- Decision making Statements:
  - if
  - if else
  - if-elif-else
  - nested if

3.1.1 Simple if

- if statement contains a logical/boolean expression using which the data is compared and a decision is made based on the result of the comparison.

- Syntax

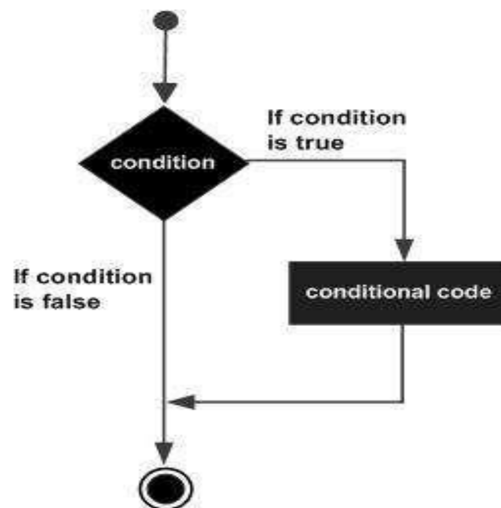
Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
if  expression:

        statement(s)

statement-x
```

- If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.
- In Python, statements in a block are uniformly indented after the : symbol.
- If boolean expression evaluates to FALSE, then the first set of code after the end of block is executed.




```
>>>if 5>3:
        print("hai")
```

Output:

hai

3.1.2 if-else

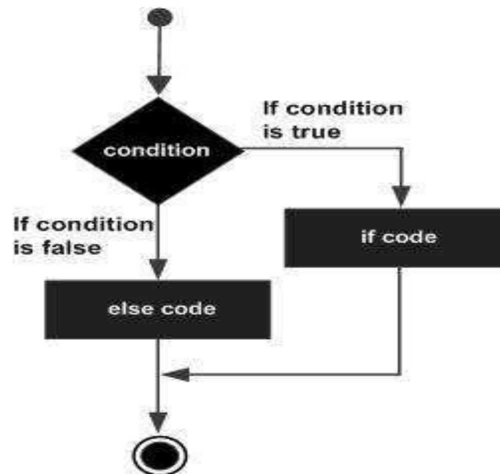
- The if-else statement is the most common type of selection statement. It is also called a two-way selection statement, because it directs the computer to make a choice between two alternative courses of action
- Python syntax for the if-else statement:

```
if <condition>:
```


Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
        <sequence of statements–1>
else:
        <sequence of statements–2>
```

- The condition in the if-else statement must be a Boolean expression—that is, an expression that evaluates to either true or false



Code:

```
n=int(input("Enter a
number"))

if n>0:
    print("Number is +ve")
else:
    print("Number is -ve")
print("Out of if else")
```

Output:

```
>>> ifelse()
Enter a number3
Number is +ve
Out of if else
>>> ifelse()
Enter a number-5
Number is -ve
Out of if else
```

3.1.3. Multi-Way if Statement(if-elif-else)

- The process of testing several conditions and responding accordingly can be described in code by a multi-way selection statement.
- The syntax of the multi-way if statement is the following:

```
if <condition-1>:
```

<sequence of statements-1>

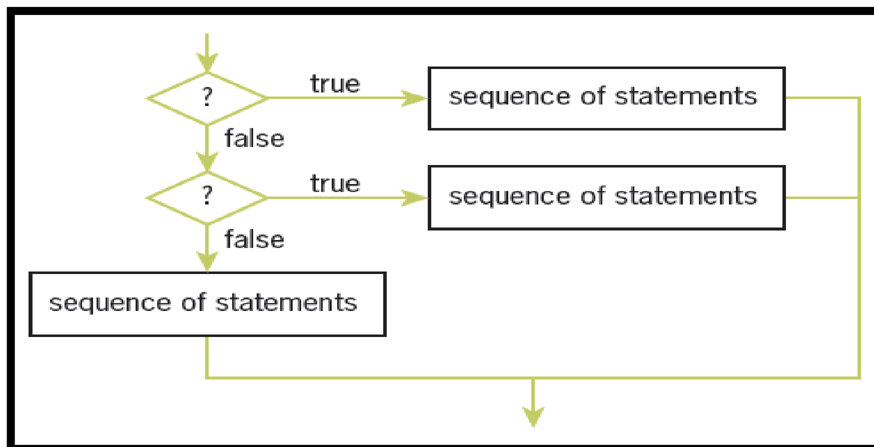
elif <condition-*n*>:

<sequence of statements-*n*>

else:

<default sequence of statements>

- The multi-way if statement considers each condition until one evaluates to True or they all evaluate to False.
- When a condition evaluates to True, the corresponding action is performed and control skips to the end of the entire selection statement.
- If no condition evaluates to True, then the action after the trailing else is performed.



Code:

```
n=int(input("Enter a number"))

  if n>0:
    print("Number is +ve")
  elif n<0:
    print("Number is -ve")
  else:
    print("Number is neither
  +ve nor -ve")
```

Output :

```
>>> ifelif()
Enter a number5
Number is +ve
>>> ifelif()
Enter a number-4
Number is -ve
>>> ifelif()
Enter a number0
Number is neither +ve nor -ve
```

3.1.4 Nested if -else

- Nested if-else statements means an if-else statement inside another if/else statement or both.

- Syntax:

```
if <expression1>:
        if <expression2>:
            statement block1
        else:
            statement block2
else:
        if <expression3>:
            statement block3
        else:
            statement block4
```

Code

```
a=int(input("Enter 1st number:"))
b=int(input("Enter 2nd number:"))
c=int(input("Enter 3rd number:"))
if a>b:
    if a>c:
        print(a,"is biggest")
    else:
        print(c,"is biggest")
else:
    if b>c:
        print(b,"is biggest")
    else:
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

```
        print(c,"is biggest")
```

Output:

nestedif()

Enter 1st number:5

Enter 2nd number:2

Enter 3rd number:8

8 is biggest

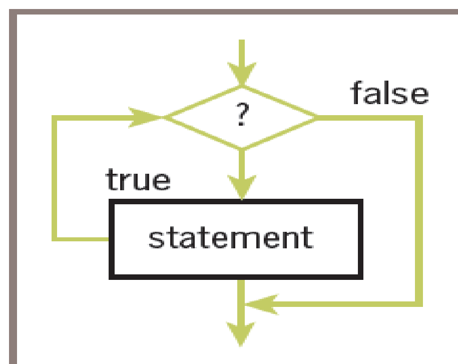
3.2 Looping Statements

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times.
- Looping Statements supported by Python :
    - for
    - while

3.2.1 while loop

- The while loop is also called an entry-control loop, because its condition is tested at the top of the loop.
- This implies that the statements within the loop can execute zero or more times.
- Syntax:

```
while <condition>:
        <sequence of statements>
```



Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Code :

```
i=1
while(i<=10):
    print(i,end=" ")
    i=i+1
```

Output:

1 2 3 4 5 6 7 8 9 10

3.2.2 for loop :

- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- The basic form of for loop is

```
for <variable> in range(<lowerbound>,<upperbound+1)):
    <statement-1>
    .
    .
    <statement-n>
```

Examples:

```
>>>for i in "Surya":
        print(i,end=",")
S,u,r,y,a,
>>>for i in range(5,10):
        print(i)
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

5

6

7

8

9

Nested Loops :

- Python programming language allows the use of one loop inside another loop. The following section shows a few examples to illustrate the concept.
- Syntax for nested for loop:

```
for iterating_var in sequence:
        for iterating_var in sequence:
            statement(s) –block1
        statement(s)- block2
```

- The syntax for a nested while loop statement in Python programming language is as follows:

```
while expression:
        while expression:
            statement(s)-block1
        statement(s)-block2
```

Examples:

```
>>>for i in range(1,4):
          for j in range(1,4):
               print("*" ,end=' ')
          print("\r")
```

Output:

```
* * *
* * *
* * *
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

4 Input Validation : Calculating a Running Total

```
vchoice=["y","Y"]
invchoice=["n","N"]
choice=vchoice+invchoice
num=[]
s=0
ch="y"
while ch in vchoice:
    n=int(input("Enter a number:"))
    num.append(n)
    s=s+n
    ch=input("Do u want to enter another number:[Y/N]")
    while ch not in choice:
        ch=input("Enter a proper choice:")
print("All the elements u entered till now")
for i in num:
    print(i,end=" ")
print("Sum of all the numbers u entered are:",s)
```

OUTPUT:

```
Enter a number:5
Do u want to enter another number:[Y/N]y
Enter a number:6
Do u want to enter another number:[Y/N]Y
Enter a number:2
Do u want to enter another number:[Y/N]u
Enter a proper choice:i
```

Surya Lakshmi Kantham Vinti
CSE Dept
ACET

Enter a proper choice:o

Enter a proper choice:y

Enter a number:87

Do u want to enter another number:[Y/N]n

All the elements u entered till now

5 6 2 87 Sum of all the numbers u entered are: 100

Surya Lakshmi Kantham Vinti
CSE Dept
ACET