UNIT-I

Programming: Introduction to Programming Concepts with Scratch

Topic 1:Introduction to Programming Concepts with Scratch

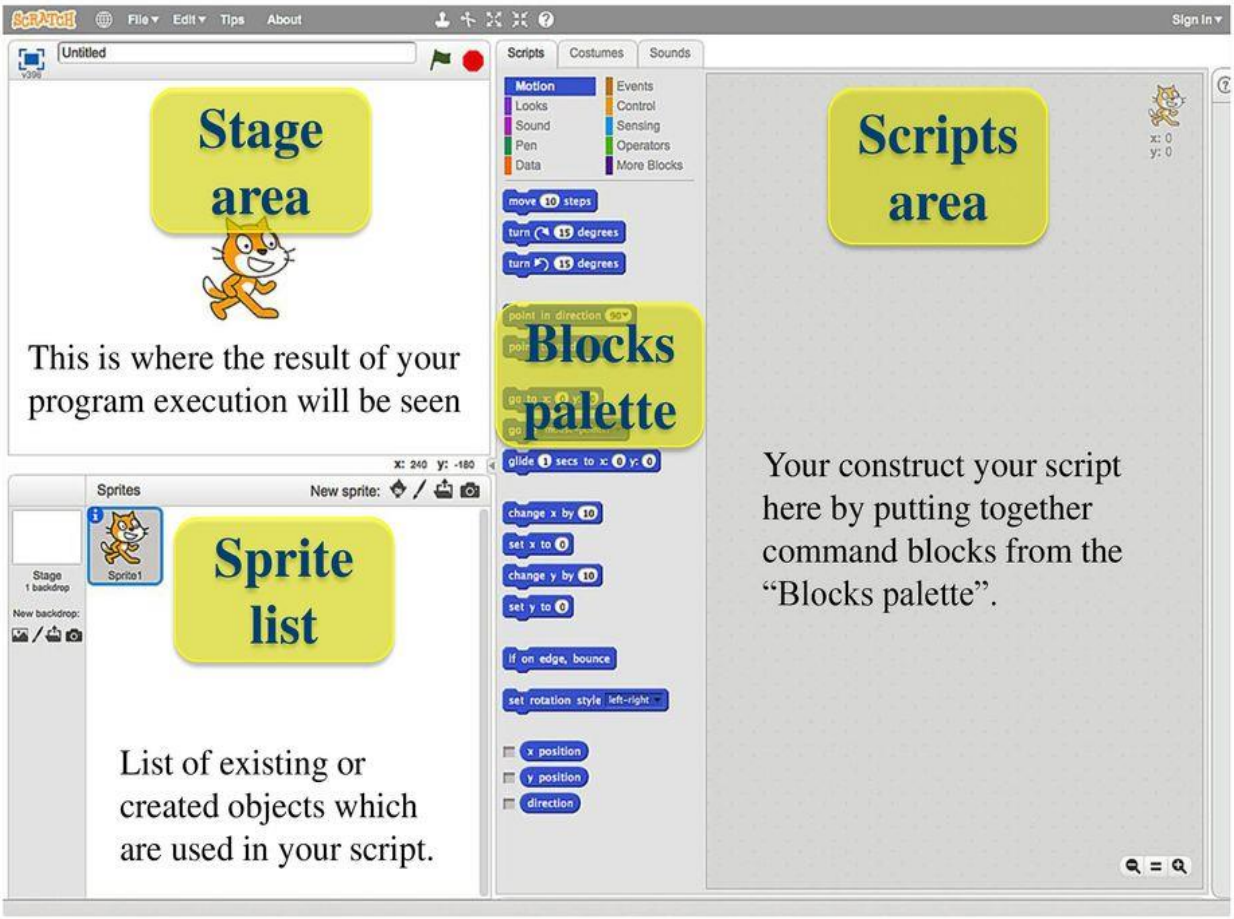
What is Scratch

Scratch is a ***free programmable toolkit*** that enables users to create their own games, animated stories, and interactive art and share their creations with one another over the Internet.

Use of scratch allows you to:

- Create solutions to problems using computers
- Study information
- Invent ***algorithms***
- Write programs to implement the algorithms
- Reuse a lot of existing program and machine parts

Scratch start window



What Scratch can do

- Can make cartoons
- Can create stories
- Can create video games

Elements of Scratch: *objects*

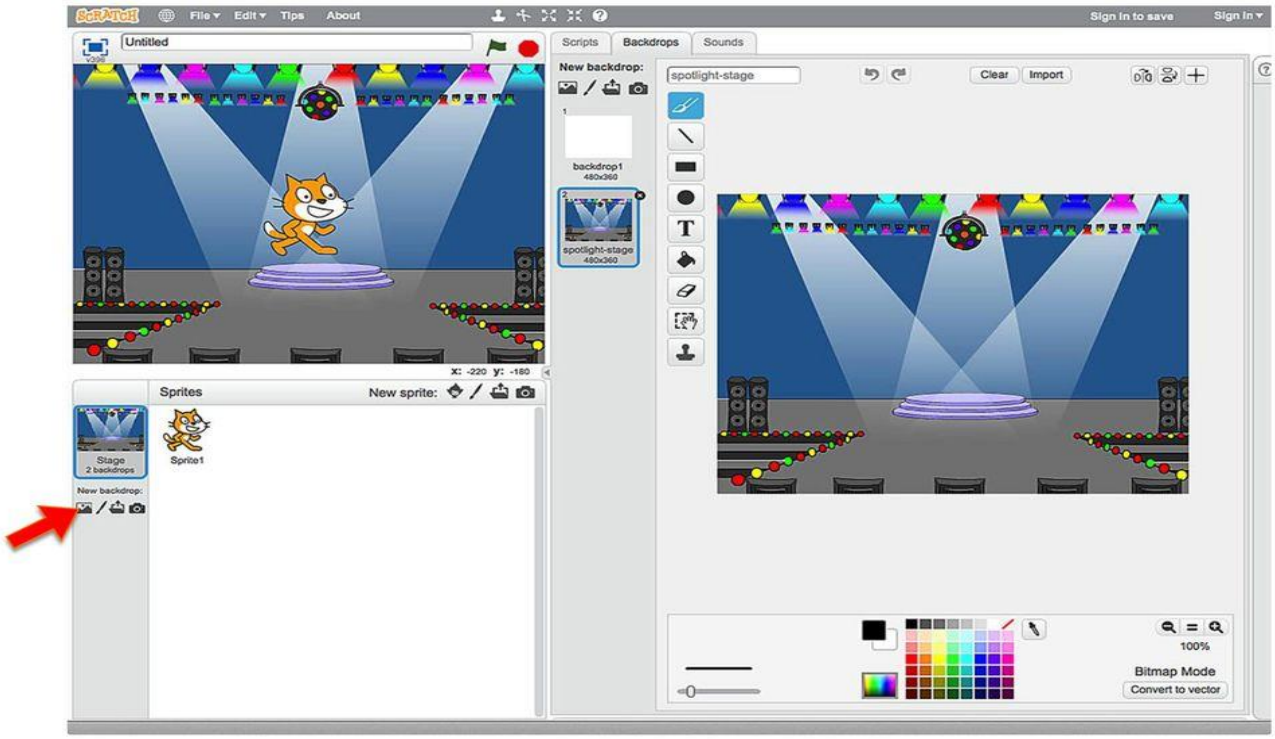
- Colors
- Sounds
- Locations in 2D space
- Sprites
- Costumes
- Variables (to remember the state of things)
- Events: that are broadcast for communication

Your first script

- Make sure to have a sprite selected in the “Sprite list”
- Drag shown blocks into “Script area”.

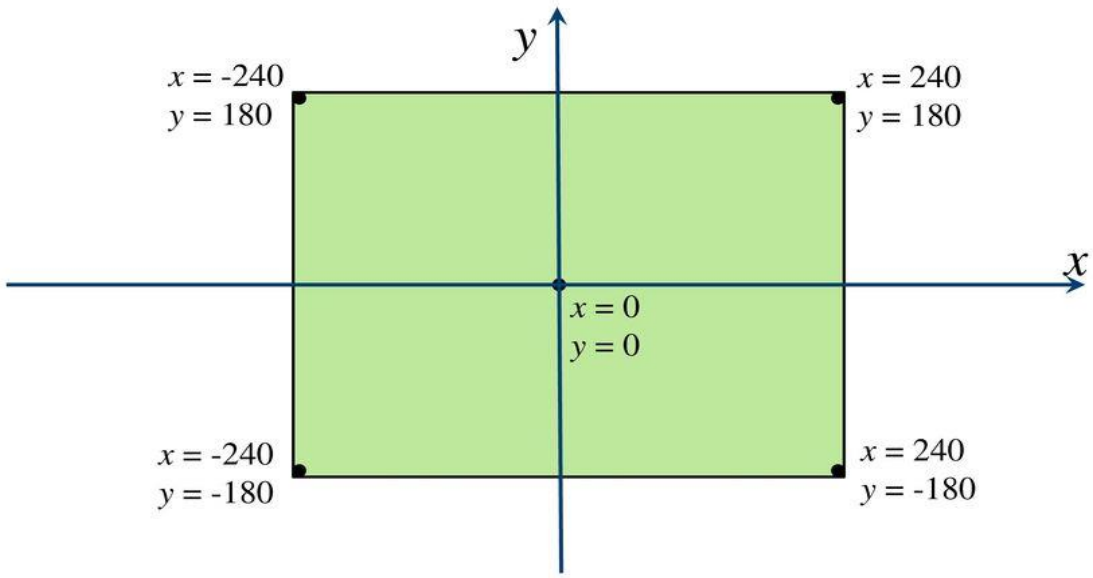
- Click the sprite in the “Stage area”

Add new stage



Stage dimensions

These are *x*- and *y*- coordinates for the stage:



Topic 2: Introduction to Python

Python History:

1. Python is a general purpose high level programming language.
2. Python was developed by Guido Van Rossam in 1989 while working at National Research Institute at Netherlands.
3. But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.
4. Python is recommended as first programming language for beginners.
5. The name Python was selected from the TV Show “The Complete Monty Python's Circus", which was broadcasted in BBC from 1969 to 1974.
6. Guido developed Python language by taking almost all programming features from different languages

    Functional Programming Features from C

    Object Oriented Programming Features from C++

    Scripting Language Features from Perl and Shell Script

    Modular Programming Features from Modula-3

7. Most of syntax in Python Derived from C and ABC languages.

Where we can use Python:

We can use everywhere. The most common important application areas are

- For developing Desktop Applications
- For developing web Applications
- For developing database Applications
- For Network Programming
- For developing games
- For Data Analysis Applications
- For Machine Learning
- For developing Artificial Intelligence Applications
- For IOT ..... etc

Note:

- Internally Google and Youtube use Python coding
- NASA and Newyork Stock Exchange Applications developed by using Python.
- Top Software companies like Google, Microsoft, IBM, Yahoo using Python.

Features of Python:

1. Simple and easy to learn:

Python is a simple programming language. When we read Python program, we can feel like reading English statements. The syntaxes are very simple and only 30+ keywords are available. When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity. We can reduce development and cost of the project.

2. Freeware and Open Source

We can use Python software without any license and it is freeware. Its source code is open, so that we can we can customize based on our requirement.

Eg: Jython is customized version of Python to work with Java Applications.

3. High level Programming Language

Python is high level programming language and hence it is programmer friendly language. Being a programmer we are not required to concentrate low level activities like memory management and security etc..

4. Platform Independent

Once we write a Python program, it can run on any platform without rewriting Once again. Internally PVM is responsible to convert into machine understandable form.

5. Portability

Python programs are portable. I.e we can migrate from one platform to another platform very easily. Python programs will provide same results on any platform.

6. Dynamically Typed

In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language.

But Java, C etc are Statically Typed Languages because we have to provide type at the beginning only. This dynamic typing nature will provide more flexibility to the programmer.

7. Both Procedure Oriented and Object Oriented

Python language supports both Procedure oriented (like C, Pascal etc) and object oriented (like C++,Java) features. Hence we can get benefits of both like security and reusability etc

8. Interpreted

We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation.

If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

9.Extensible

We can use other language programs in Python. The main advantages of this approach are:

We can use already existing legacy non-Python code We can improve performance of the application

10.Embedded

We can use Python programs in any other language programs. i.e we can embed Python programs anywhere.

11. Extensive Library

Python has a rich inbuilt library. Being a programmer we can use this library directly and we are not responsible to implement the functionality.

Flavours of Python

CPython:

It is the standard flavor of Python. It can be used to work with C lanugage Applications

Jython or JPython:

It is for Java Applications. It can run on JVM

IronPython:

It is for C#.Net platform

PyPy:

The main advantage of PyPy is performance will be improved because JIT compiler is available inside PVM.

RubyPython

For Ruby Platforms

AnacondaPython

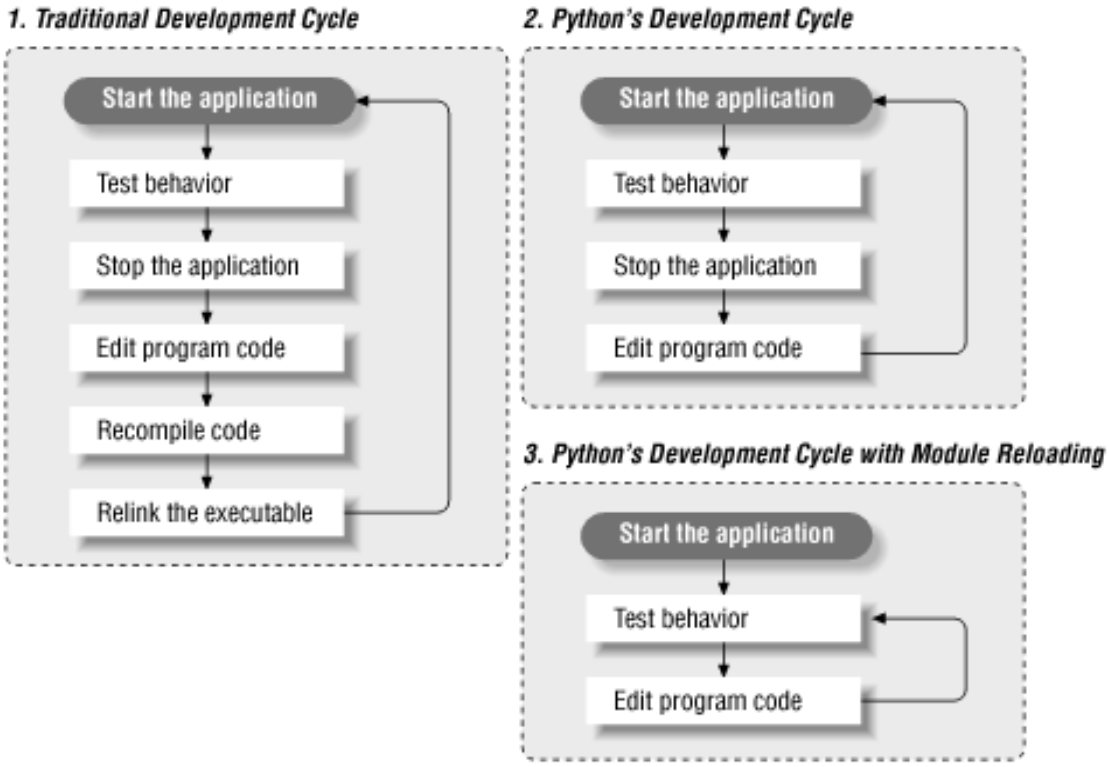
It is specially designed for handling large volume of data processing.

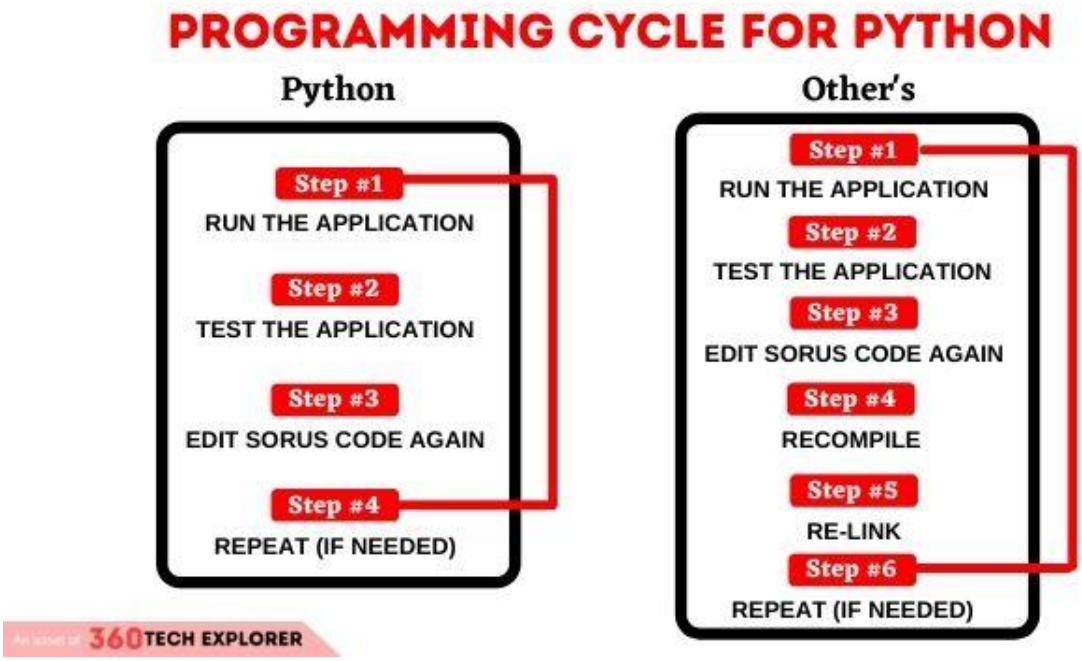
Career paths after learning python

- Web Developer
- Research Analyst
- Data Scientist
- Server side programmer

Topic 3: Python Program Development Cycle

Python's development cycle is dramatically shorter than that of traditional tools. In Python, there are no compile or link steps -- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.





1.Python's programming cycle is dramatically shorter than that of traditional programming cycle.

2. In Python, there are no compile or link steps.

3. Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.

4. In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.

Topic 4: Input, Processing, and Output

- Most useful programs accept inputs from some source, process these inputs, and then finally output results to some destination.
- In terminal-based interactive programs, the input source is the keyboard, and the output destination is the terminal display. 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.The programmer can also force the output of a value by using the print function. The simplest form for using this function looks like the following:

print(<expression>)

When running the print function, Python first evaluates the expression and then displays its value. In the example shown earlier, print was used to display some text. The following is another example:

```
>>> print ("Hi there")
 Hi there
```

The input function does the following:

1. Displays a prompt for the input. In this example, the prompt is "Enter your name: ".
2. Receives a string of keystrokes, called characters, entered at the keyboard and returns the string to the shell.

The string returned by the function in our example is saved by assigning it to the variable name. The form of an assignment statement with the input function is the following:

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

A variable identifier, or variable for short, is just a name for a value. When a variable receives its value in an input statement, the variable then refers to this value. If the user enters the name "Ken Lambert" in our last example, the value of the variable name can be viewed as follows:

```
>>> name
'Ken Lambert'
```

Function	What It Does
`float(<a string of digits>)`	Converts a string of digits to a floating-point value.
`int(<a string of digits>)`	Converts a string of digits to an integer value.
`input(<a string prompt>)`	Displays the string prompt and waits for keyboard input. Returns the string of characters entered by the user.
`print(<expression>, ...,<expression>)`	Evaluates the expressions and displays them, separated by one space, in the console window.
`<string 1> + <string 2>`	Glues the two strings together and returns the result.

Table 1-2 Basic Python functions for input and output

Topic 5: Displaying Output with the Print Function

We can use print() function to display output.

Form-1:

print() without any argument Just it prints new line character

Form-2:

1) print(String):

```
print("Hello World")
```

2) We can use escape characters also

```
print("Hello \n World")
print("Hello\tWorld")
```

3) We can use repetetion operator (*) in the string

```
print(10*"Hello")
print("Hello"*10)
```

4) We can use + operator also

```
print("Hello"+"World")
```

Note:

If both arguments are String type then + operator acts as concatenation operator.

If one argument is string type and second is any other type like int then we will get Error

If both arguments are number type then + operator acts as arithmetic addition operator.

Example:

1) print("Hello"+"World")

2) print("Hello","World")

HelloWorld

Hello World

Form-3: print() with variable number of arguments:

Example:

```
a,b,c=10,20,30
print("The Values are :",a,b,c)
```

Output

The Values are : 10 20 30

By default output values are seperated by space.If we want we can specify seperator by using "sep" attribute

Example:

```
a,b,c=10,20,30
print(a,b,c,sep=',')
print(a,b,c,sep=':')
```

```
D:\Python_classes>py test.py
10,20,30
10:20:30
```

Form-4:print() with end attribute:

```
print("Hello")
print("Aditya")
print("College")
```

Output:

```
Hello
Aditya
College
```

If we want output in the same line with space.

```
print("Hello",end=' ')
print("Aditya",end=' ')
print("College")
```

Output: Hello Aditya College

Note: The default value for end attribute is \n, which is nothing but new line character.

Form-5: print(object) statement:We can pass any object (like list,tuple,set etc)as argument to the print() statement.

Eg:

```
l=[10,20,30,40]
t=(10,20,30,40)
print(l)
print(t)
```

Form-6: print(String,variable list):

We can use print() statement with String and any number of arguments.

Eg:

```
s="Srinu"
a=34
s1="java"
s2="Python"
print("Hello",s,"Your Age is",a)
print("You are teaching",s1,"and",s2)
```

Output:

1) Hello Srinu Your Age is 34

2) You are teaching java and Python

Form-7: print(formatted string):

%i====>int

%d====>int

%f=====>float

%s======>String type

Syntax:

print("formatted string" %(variable list))

Eg 1:

```
a=10
b=20
c=30
print("a value is %i" %a)
```

```
print("b value is %d and c value is %d" %(b,c))
```

Output

```
a value is 10
b value is 20 and c value is 30
```

Form-8: print() with replacement operator {}

Eg:

```
name="Srinu"
salary=30000
frnd="Sunny"
print("Hello {0} your salary is {1} and Your Friend {2} is waiting".format(name,salary,frnd))
print("Hello {x} your salary is {y} and Your Friend {z} is waiting".format(x=name,y=salary,z=frnd))
```

Output

```
Hello Srinu your salary is 10000 and Your Friend Sunny is waiting
Hello Srinu your salary is 10000 and Your Friend Sunny is waiting
```

Topic 6: Comments, Variables

Introduction to Python Comments:

We may wish to describe the code we develop. We might wish to take notes of why a section of script functions, for instance. The Python interpreter overlooks the remarks and solely interprets the script when running a program. Single-line comments, multi-line comments, and documentation strings are the 3 types of comments in Python.

Advantages of Using Comments

- Our code is more comprehensible when we use comments in it. It assists us in recalling why specific sections of code were created by making the program more understandable.
- Aside from that, we can leverage comments to overlook specific code while evaluating other code sections.

Types of Comments in Python

In Python, there are 3 types of comments. They are described below:

Single-Line Comments

Single-line remarks in Python have shown to be effective for providing quick descriptions for parameters, function definitions, and expressions. A single-line comment of Python is the one that has a hashtag # at the beginning of it and continues until the finish of the line.

```
# This code is to show an example of a single-line comment
print( 'This statement does not have a hashtag before it' )
```

Output:

This statement does not have a hashtag before it

The following is the comment:

```
# This code is to show an example of a single-line comment
```

Multi-Line Comments

Python does not provide the facility for multi-line comments. However, there are indeed many ways to create multi-line comments.

With Multiple Hashtags (#)

it is a

comment

extending to multiple lines

In this case, each line is considered a comment, and they are all omitted.

Python Docstring

The strings enclosed in triple quotes that come immediately after the defined function are called Python docstring.

```
# Code to show how we use docstrings in Python

def add(x, y):
    """This function adds the values of x and y"""
    return x + y

# Displaying the docstring of the add function
print( add.__doc__ )
```

Output:

This function adds the values of x and y

Python Variables

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

Identifier Naming:

1. The first character of the variable must be an alphabet or underscore (_).
2. All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
3. Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
4. Identifier name must not be similar to any keyword defined in the language.
5. Identifier names are case sensitive; for example, my name, and MyName is not the same.
6. Examples of valid identifiers: a123, _n, n_9, etc.
7. Examples of invalid identifiers: 1a, n%4, n 9, etc.

Let's understand the following example

a = 50

In the above image, the variable a refers to an integer object.

Suppose we assign the integer value 50 to a new variable b.

a = 50

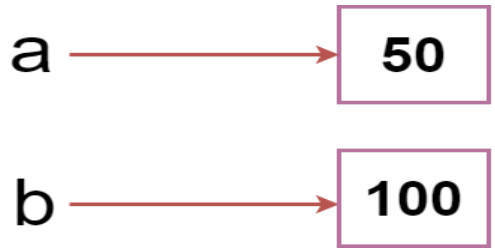
b = a

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

a = 50

b =100



Variable Names:

We have already discussed how to declare the valid variable. Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

```
name = "Devansh"
age = 20
marks = 80.50

print(name)
print(age)
print(marks)
```

Output:

```
Devansh
20
80.5
```

The multi-word keywords can be created by the following method.

Camel Case - In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.

Pascal Case - It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.

Snake Case - In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Topic 7:Reading Input from the Keyboard

```
# input() function is used to read the data from a keyboard.
# every input value is treated as str type only.
```

```
# syntax:  variable=input(string)
'''
a=input("Enter a number\n")
print(a)
print(type(a))'''

'''x=input("Enter first number\n")
y=input("Enter second number\n")
z=x+y
print("sum is:",z)'''
```

Ex:1

Write a program to read 5 subject marks and find the sum and average of them?

```
# Find sum and Average
'''m1=int(input("Enter sub1 marks\n"))
m2=int(input("Enter sub2 marks\n"))
m3=int(input("Enter sub3 marks\n"))
m4=int(input("Enter sub4 marks\n"))
m5=int(input("Enter sub5 marks\n"))
tot=m1+m2+m3+m4+m5
avg=tot/5
print("Total Marks is:",tot,end='\t')
print("Average Marks is:",avg)'''
```

Ex:2

Write a program to read two hero names and number of hits, avg, flop movies as an input.
Now calculate the total points he gained. if movie is hit = 10, avg= 5, flop= -5

```
#find the total points gained by a hero
'''
hero1=input("Enter your Favourite hero1\n")
h1=int(input("Enter number of hit films\n"))
avg1=int(input("Enter number of avg films\n"))
flop1=int(input("Enter number of flop films\n"))

hero2=input("Enter your Favourite hero2\n")
h2=int(input("Enter number of hit films\n"))
avg2=int(input("Enter number of avg films\n"))
flop2=int(input("Enter number of flop films\n"))

Tot1=h1*10+avg1*5+flop1*-5
Tot2=h2*10+avg2*5+flop2*-5

print("Hero 1 points is:",Tot1)
print("Hero2 points is:",Tot2)  '''
```

Topic 8: Performing Calculations

Python Programming

Python Program to Make a Simple Calculator:

In this example you will learn to create a simple calculator that can add, subtract, multiply or divide depending upon the input from the user.

```
# Program make a simple calculator
# This function adds two numbersdef add(x, y):
    return x + y
# This function subtracts two numbersdef subtract(x, y):
    return x - y
# This function multiplies two numbersdef multiply(x, y):
    return x * y
# This function divides two numbersdef divide(x, y):
    return x / y

print("Select operation.")print("1.Add")print("2.Subtract")print("3.Multiply")print("4.Divide")
while True:
    # take input from the user
    choice = input("Enter choice(1/2/3/4): ")

    # check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # check if user wants another calculation
        # break the while loop if answer is no
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
          break

    else:
        print("Invalid Input")
```

Output

```
Select operation.
1.Add
```

2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 3
Enter first number: 15
Enter second number: 14
15.0 * 14.0 = 210.0
Let's do next calculation? (yes/no): no

Topic 9: Operators

An operator is a symbol that specifies an operation to be performed on the operands. The data items that an operator acts upon are called operands. The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and exponentiation. Example: 20+32

In this example, 20 and 32 are operands and + is an operator

Python provides a variety of operators, which are described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators
- Arithmetic Operators

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
* (Multiplication)	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
** (Exponent)	It is an exponent operator represented as it calculates the first operand power to the second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator:

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Bitwise Operators:

The bitwise operators perform bit by bit operation on the values of the two operands. Consider the following example.

For example,

if a = 7

b = 6

then, binary (a) = 0111

binary (b) = 0110

hence, a & b = 0011

a | b = 0111

a ^ b = 0100

~ a = 1000

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different; otherwise, the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

Logical Operators:

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Membership Operators:

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both sides do not point to the same object.

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Topic 10: Type conversions

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
5. In Type Casting, loss of data may occur as we enforce the object to a specific data type.

 # Now we need to go for the type casting

int(), float(), bool(), str(), complex() - Type casting functions

'''x=input("Enter first number\n") #- '10'

y=input("Enter second number\n") #- '20'

z=int(x)+int(y)

print("sum is:",z) '''

 # int() - We can use this function to convert values from other types to int

#print(int(124.35)) - 124

#print(int("10")) - 10

#print(int(True)) - 1

#print(int(False)) - 0

#print(int("10.5")) - Value Error

#print(int("ten")) - Error

#print(int("0B111")) - Error

#print(int(10+5j)) - Error'''

Note:

1. We can convert from any type to int except complex type.

2. If we want to convert str type to int type, compulsary str should contain only integral value and should be specified in base-10 '"

 # float() - We can use float() function to convert other type values to float type.

#print(float(10)) - 10.0

#print(float(True)) - 1.0

#print(float(False)) - 0.0

#print(float("10")) - 10.0

#print(float("10.5")) - 10.5

#print(float("ten")) - Error

#print(float("0B1111")) - Error

#print(float(10+5j)) - Error'"

Note:

1. We can convert any type value to float type except complex type.
2. Whenever we are trying to convert str type to float type compulsary str should be either integral or floating point literal and should be specified only in base-10.'"

 # bool() - We can use this function to convert other type values to bool type.

'"

bool(0)==>False

bool(1)==>True

bool(10)===>True

bool(10.5)===>True

bool(0.178)==>True

bool(0.0)==>False

bool(10-2j)==>True

bool(0+1.5j)==>True

bool(0+0j)==>False

bool("True")==>True

bool("False")==>True

bool("")==>False'''

 # str() - We can use this method to convert other type values to str type

'''

print(str(10)) - '10'

print(str(10.5)) -'10.5'

print(str(10+5j)) - '10+5j'

print(str(True)) - 'True""

 # complex() - We can use complex() function to convert other types to complex type.

'''complex(10)==>10+0j

complex(10.5)===>10.5+0j

complex(True)==>1+0j

complex(False)==>0j

complex("10")==>10+0j

complex("10.5")==>10.5+0j

complex("ten")

ValueError: complex() arg is a malformed string

complex(10,-2)==>10-2j

complex(True,False)==>1+0j '''

Topic 11: Expressions in Python

An arithmetic expression consists of operands and operators combined in a manner that is already familiar to you from learning algebra. Table 2-6 shows several arithmetic operators and gives examples of how you might use them in Python code.

Operator	Meaning	Syntax
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

Table 2-6 Arithmetic operators

The precedence rules you learned in algebra apply during the evaluation of arithmetic expressions in Python:

• Exponentiation has the highest precedence and is evaluated first.

• Unary negation is evaluated next, before multiplication, division, and remainder.

• Multiplication, both types of division, and remainder are evaluated before addition and subtraction.

• Addition and subtraction are evaluated before assignment.

• With two exceptions, operations of equal precedence are left associative, so they are evaluated from left to right. Exponentiation and assignment operations are right associative, so consecutive instances of these are evaluated from right to left.

• You can use parentheses to change the order of evaluation.

Table 2-7 shows some arithmetic expressions and their values.

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`	

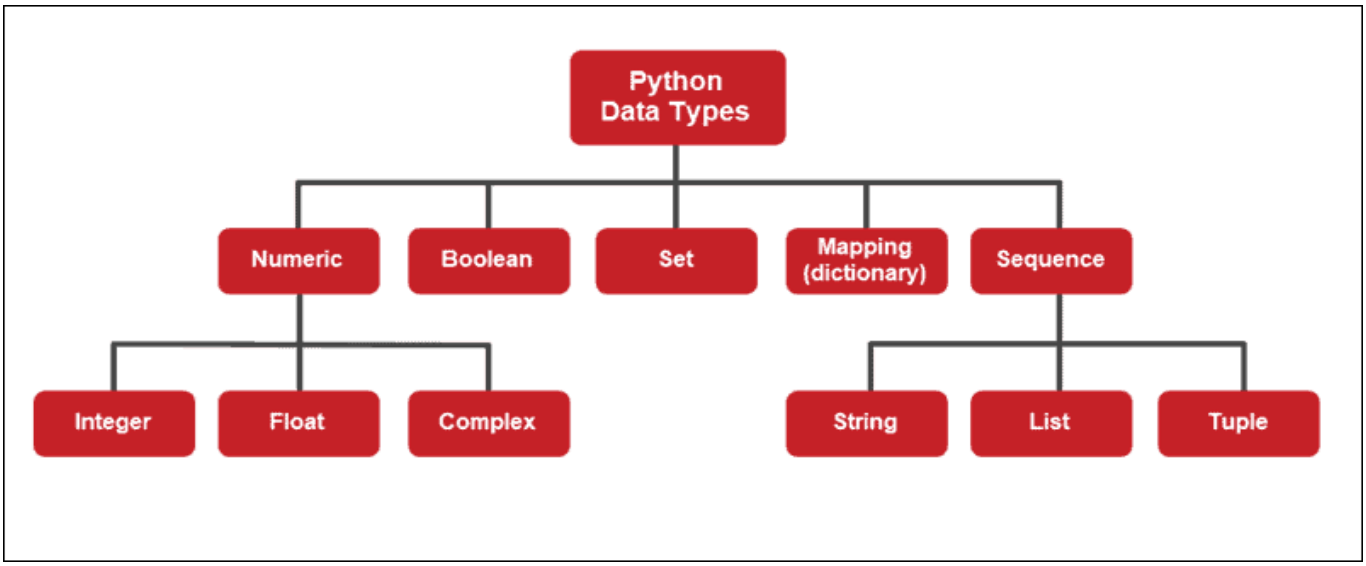
Table 2-7 Some arithmetic expressions and their values

Topic 12: Data Types in Python

Data Type represent the type of data present inside a variable.

• In Python we are not required to specify the type explicitly. Based on value provided the type will be assigned automatically.

• Hence Python is Dynamically Typed Language.



In Python the following data types are considered as Fundamental Data types

- int
- float
- complex
- bool
- str

int Data Type:

We can use int data type to represent whole numbers (integral values)

Eg:

```
a=10
type(a) #int
```

We can represent int values in the following ways

1. Decimal form
2. Binary form
3. Octal form
4. Hexa decimal form

1. Decimal form(base-10):

It is the default number system in Python

The allowed digits are: 0 to 9

Eg: a =10

2. Binary form(Base-2):

The allowed digits are : 0 & 1

Literal value should be prefixed with 0b or 0B

Eg: a = 0B1111

a =0B123

a=b111

3. Octal Form(Base-8):

The allowed digits are : 0 to 7

Literal value should be prefixed with 0o or 0O.

Eg: a=0o123

a=0o786

4. Hexa Decimal Form(Base-16):

The allowed digits are : 0 to 9, a-f (both lower and upper cases are allowed)

Literal value should be prefixed with 0x or 0X

Eg:

a =0XFACE

a=0XBeef

a =0XBeer

Note: Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal

form.

a=10

b=0o10

c=0X10

d=0B10

print(a) - 10

print(b) - 8

print(c) - 16

print(d) - 2

Base Conversions:

Python provide the following in-built functions for base conversions.

1. bin()
2. oct()
3. hex()

bin():We can use bin() to convert from any base to binary.

Eg:

```
1)    >>> bin(15)
2)  '0b1111'
3)  >>> bin(0o11)
4)  '0b1001'
5)  >>> bin(0X10)
6)  '0b10000'
```

oct():We can use oct() to convert from any base to octal

Eg:

```
1)  >>> oct(10)
2)  '0o12'
3)  >>> oct(0B1111)
4)  '0o17'
5)  >>> oct(0X123)
6)  '0o443'
```

hex():We can use hex() to convert from any base to hexa decimal

Eg:

```
1)  >>> hex(100)
2)  '0x64'
3)  >>> hex(0B111111)
4)  '0x3f'
5)  >>> hex(0o12345)
6)  '0x14e5'
```

float Data Type:

We can use float data type to represent floating point values (decimal values).

Eg: f=1.234

type(f) float

We can also represent floating point values by using exponential form (scientific notation)

Eg: f=1.2e3

print(f) 1200.0

instead of 'e' we can use 'E'

The main advantage of exponential form is we can represent big values in less memory.

Note:

• We can represent int values in decimal, binary, octal and hexa decimal forms.

But we can represent float values only by using decimal form.

>>> f=0o123.456

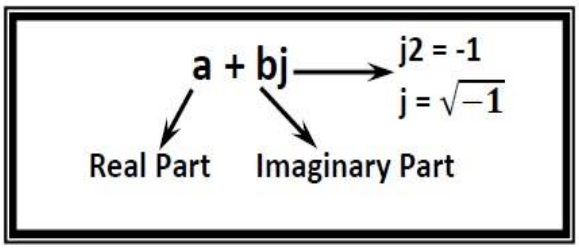
SyntaxError: invalid syntax

>>> f=0X123.456

SyntaxError: invalid syntax

complex Data Type:

A complex number is of the form



a and b contain integers or floating point values Eg:

3+5j

10+5.5j

0.5+0.1j

bool Data Type

- We can use this data type to represent boolean values.
- The only allowed values for this data type are: True and False
- Internally Python represents True as 1 and False as 0

b=True type(b) =>bool Eg:

a=10 b=20

c=a<b

print(c)==>True

True+True==>2

True-False==>1

str represents String data type.

str data Type:

A String is a sequence of characters enclosed within single quotes or double quotes.

s1= 'aditya'

s1= "aditya"

By using single quotes or double quotes we cannot represent multi line string Literals.

s1= "Aditya
College"
For this requirement we should go for triple single quotes(''') or triple double quotes(""") s1= '''Aditya
College''' s1= """Aditya
College"""

Expressions

Topic 13: Strings Assignment and Comment

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 programming, a data type consists of a set of values and a set of operations that can be performed on those values. A literal is the way a value of a data type looks to a programmer. The programmer can use a literal in a program to mention a data value. When the Python interpreter evaluates a literal, the value it returns is simply that literal. Table 2-2 shows example literals of several Python data types.

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"`

Table 2-2 Literals for some Python data types

The first two data types listed in Table 2-2, `int` and `float`, are called **numeric data types**, because they represent numbers. You'll learn more about numeric data types later in this chapter. For now, we will focus on character strings—which are often referred to simply as strings.

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!'
>>> ''
''
>>> ""
''
```

The last two string literals ('' and "") represent the empty string.

Escape Sequences

The newline character \n is called an escape sequence. Escape sequences are the way Python expresses special characters, such as the tab, the newline, and the backspace (delete key), as literals. Table 2-3 lists some escape sequences in Python.

Escape Sequence	Meaning
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

Table 2-3 Some escape sequences in Python

Because the backslash is used for escape sequences, it must be escaped to appear as a literal character in a string. Thus, `print('\\')` would display a single \ character.

String Concatenation

You can join two or more strings to form a new string using the concatenation operator +.

Here is an example:

```
>>> "Hi " + "there, " + "Ken!"
'Hi there. Ken!'
```

Program Comments and Docstrings

We conclude this subsection on strings with a discussion of program comments. A comment is a piece of program text that the computer ignores but that provides useful documentation to programmers.

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. This type of comment, called a

docstring, is a multi-line string of the form discussed earlier in this section. Here is a docstring that begins a typical program for a lab session:

"""
Program: circle.py
Author: Ken Lambert
Last date modified: 10/10/17
The purpose of this program is to compute the area of a
circle. The input is an integer or floating-point number
representing the radius of the circle. The output is a
floating-point number labeled as the area of the circle.
"""

In addition to docstrings, end-of-line comments can document a program.

Topic 14:Numeric Data Types and Character Sets

The first applications of computers were created to crunch numbers. Although text and media processing have lately been of increasing importance, the use of numbers in many applications is still very important. In this section, we give a brief overview of numeric data types and their cousins, 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.

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

A real 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.

Decimal Notation	Scientific Notation	Meaning
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e–1	3.78×10^{-1}
0.00378	3.78e–3	3.78×10^{-3}

Character Sets:

Table 2-5 shows the mapping of character values to the first 128 ASCII codes. The digits in the left column represent the leftmost digits of an ASCII code, and the digits in the top row are the rightmost digits. Thus, the ASCII code of the character 'R' at row 8, column 2 is 82.

	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		

Ex:

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
'A'
>>> chr(66)
'B'
```

Topic 15: Using Functions and Modules

Python includes many useful functions, which are organized in libraries of code called modules. In this section, we examine the use of functions and modules.

Calling Functions: Arguments and Return Values:

- A function is a chunk of code that can be called by name to perform a task. Functions often require arguments, that is, specific data values, to perform their tasks.
- Names that refer to arguments are also known as parameters. When a function completes its task (which is usually some kind of computation), the function may send a result back to the part of the program that called that function in the first place.
- The process of sending a result back to another part of a program is known as returning a value.

The math Module:

Functions and other resources are coded in components called modules. Functions like abs and round from the __builtin__ module are always available for use, whereas the programmer must explicitly import other functions from the modules where they are defined.

The math module includes several functions that perform basic mathematical operations.

The next code session imports the math module and lists a directory of its resources:

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
```

'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau',
'trunc']

The Main Module

In the case study, earlier in this chapter, we showed how to write documentation for a Python script. To differentiate this script from the other modules in a program (and there could be many), we call it the main module. Like any module, the main module can also be imported. Instead of launching the script from a terminal prompt or loading it into the
shell from IDLE, you can start IDLE from the terminal prompt and import the script as a module.

Let's do that with the taxform.py script, as follows:

```
>>> import taxform
Enter the gross income: 120000
Enter the number of dependents: 2
The income tax is $20800.0
```

Program Format and Structure

This is a good time to step back and get a sense of the overall format and structure of simple Python programs. It's a good idea to structure your programs as follows:

• Start with an introductory comment stating the author's name, the purpose of the program, and other relevant information. This information should be in the form of a docstring.
• Then, include statements that do the following:
• Import any modules needed by the program.
• Initialize important variables, suitably commented.
• Prompt the user for input data and save the input data in variables.
• Process the inputs to produce the results.
• Display the results.

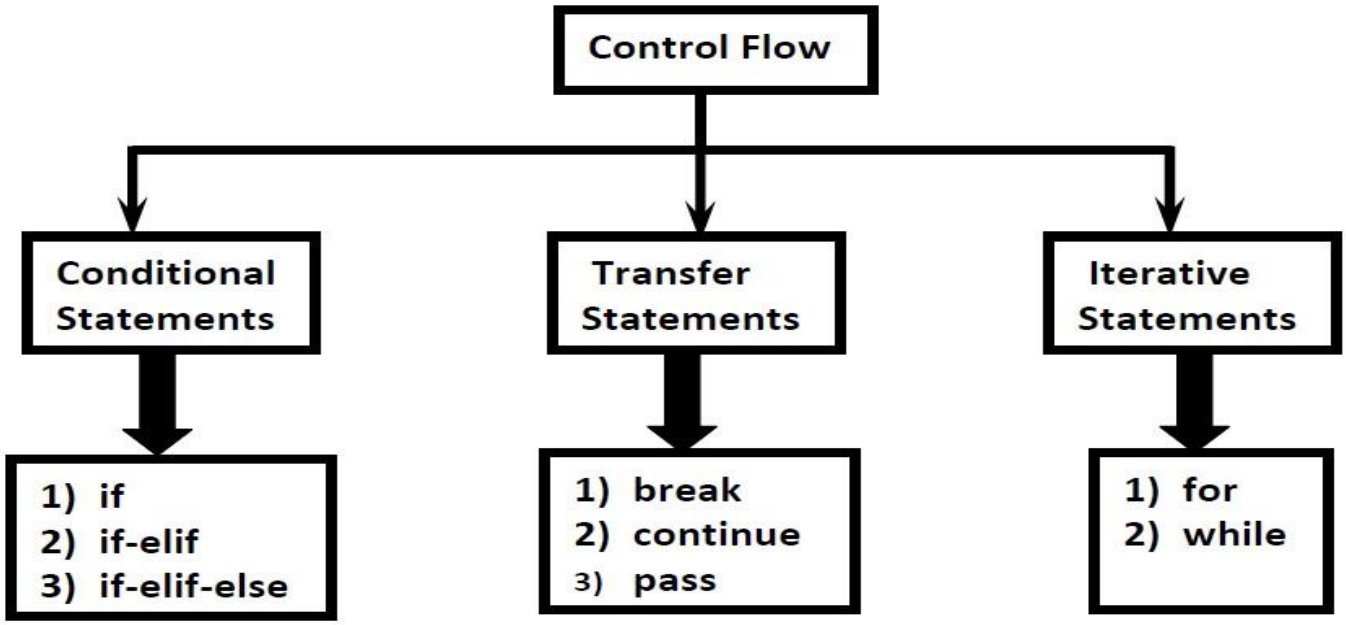
Running a Script from a Terminal Command Prompt

Thus far in this book, we have been developing and running Python programs experimentally in IDLE. When a program's development and testing are finished, the program can be released to others to run on their computers. Python must be installed on a user's computer, but the user need not run IDLE to run a Python script.

Decision Structures and Boolean Logic

Topic 16:if, if-else, if-elif-else Statements, Nested Decision Structures Comparing Strings, Logical Operators, Boolean Variables.

Flow control describes the order in which statements will be executed at runtime.



1) If Syntax:

if condition : statement

or

if condition : statement-1 statement-2 statement-3

Eg:

```
1)  name=input("Enter Name:")
2)  if name=="durga" :
3)      print("Hello Durga Good Morning")
4)  print("How are you!!!")
5)
6)  D:\Python_classes>py test.py
7)  Enter Name:durga
8)  Hello Durga Good Morning
9)  How are you!!!
10)
11) D:\Python_classes>py test.py
12) Enter Name:Ravi
13) How are you!!!
```

if - else

Syntax:

if condition :

Action-1

else :

Action-2

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

```
1)  name=input("Enter Name:")
2)  if name=="durga" :
3)      print("Hello Durga Good Morning")
4)  else:
5)      print("Hello Guest Good Moring")
6)  print("How are you!!!")
7)
8)  D:\Python_classes>py test.py
9)  Enter Name:durga
10) Hello Durga Good Morning
11) How are you!!!
```

```
13) D:\Python_classes>py test.py
14) Enter Name:Ravi
15) Hello Guest Good Moring
16) How are you!!!
```

if-elif - else

Syntax:

```
if condition1:
    Action-1
elif condition2:
    Action-2
elif condition3:
    Action-3
 elif condition4:
    Action-4
    ...
else:
    Default Action
```

Based condition the corresponding action will be executed.

```
1) n1=int(input("Enter First Number:"))
2) n2=int(input("Enter Second Number:"))
3) n3=int(input("Enter Third Number:"))
4) if n1>n2 and n1>n3:
5)     print("Biggest Number is:",n1)
6) elif n2>n3:
7)     print("Biggest Number is:",n2)
8) else :
9)     print("Biggest Number is:",n3)
```

```
11) D:\Python_classes>py test.py
12) Enter First Number:10
13) Enter Second Number:20
14) Enter Third Number:30
15) Biggest Number is: 30
```

Topic 17:Repetition Structures: Introduction, while loop, for loop,

If we want to execute a group of statements multiple times then we should go for Iterative statements.

Python supports 2 types of iterative statements.

1. for loop
2. while loop

for loop:

If we want to execute some action for every element present in some sequence(it may be string or collection)then we should go for for loop.

Syntax:

for x in sequence :

body

where sequence can be string or any collection.

Body will be executed for every element present in the sequence.

for loop:

Example1: To print Hello 10 times

```
1) for x in range(10) :
2)     print("Hello")
```

Example2: To display numbers from 0 to 10

```
for x in range(11):
    print(x)
```

Example3: To display odd numbers from 0 to 20

```
for x in range(21):
    if(x%2!=0):
        print(x)
```

Example4: To display numbers from 10 to 1 in descending order

```
for i in range(10,0,-1):
    print(i)
```

Example5: To print characters present in the given string :

```
1 s=input("Enter a String\n")
2 for ch in s:
3     print(ch)
```

```
D:\CSE-C\Programs>python loops.py
Enter a String
ADITYA
A
D
I
T
Y
A
```

while loop:

If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syntax:

```
while condition:
```

Body

while loop:

Example1: To print numbers from 1 to 10 by using while loop

Code:

```
x=1
while x<=10:
    print(x)
    x=x+1
```

Example2: To display the sum of first n numbers

```
1) n=int(input("Enter number:"))
2) sum=0
3) i=1
4) while i<=n:
5)     sum=sum+i
6)     i=i+1
7) print("The sum of first",n,"numbers is :",sum)
```

nested loop:sometimes we can take a loop inside another loop is known as nested loop

```
1)  for i in range(4):
2)      for j in range(4):
3)          print("i=",i,"   j=",j)
```

```
5)  Output
6)  D:\Python_classes>py test.py
7)  i= 0   j= 0
8)  i= 0   j= 1
9)  i= 0   j= 2
10) i= 0   j= 3
11) i= 1   j= 0
12) i= 1   j= 1
13) i= 1   j= 2
14) i= 1   j= 3
15) i= 2   j= 0
16) i= 2   j= 1
17) i= 2   j= 2
18) i= 2   j= 3
19) i= 3   j= 0
20) i= 3   j= 1
21) i= 3   j= 2
22) i= 3   j= 3
```

Ex: Write a program to display *'s in Right angled triangle form.

```
9)  n = int(input("Enter number of rows:"))
10) for i in range(1,n+1):
11)     for j in range(1,i+1):
12)         print("*",end=" ")
13)     print()
```

```
1) *
2) * *
3) * * *
4) * * * *
5) * * * * *
6) * * * * * *
7) * * * * * * *
```

Transfer Statements:

We can use break statement inside loops to break loop execution based on some condition.

condition.

```
1) for i in range(10):
2)     if i==7:
3)         print("processing is enough..plz break")
4)         break
5)     print(i)
```

```
7)  D:\Python_classes>py test.py
8)  0
9)  1
10) 2
11) 3
12) 4
13) 5
14) 6
15) processing is enough..plz break
```

2) Continue:

We can use continue statement to skip current iteration and continue next iteration.

Example: To print odd numbers in the range 0 to 9

```
1)  for i in range(10):
2)      if i%2==0:
3)          continue
4)      print(i)
```

```
6)  D:\Python_classes>py test.py
7)  1
8)  3
9)  5
10) 7
11) 9
```

Topic :18 Calculating a Running Total, Input Validation Loops, Nested Loops Example programs

#1. Program to find the biggest of two numbers

```
'''n1=int(input("Enter a first number1\n"))
n2=int(input("Enter a second number\n"))
if n1>n2:
        print("n1 is bigger")
else:
        print("n2 is bigger")'''
```

#2. Program to find the who is the best hero.

```
'''hero1=input("enter first hero name\n")
h1=int(input("Enter number of hit movies\n"))
a1=int(input("Enter number of avg movies\n"))
f1=int(input("Enter number of flop movies\n"))

hero2=input("enter second hero name\n")
h2=int(input("Enter number of hit movies\n"))
a2=int(input("Enter number of avg movies\n"))
f2=int(input("Enter number of flop movies\n"))
```

```
tot1=h1*10+a1*5+f1*-5
tot2=h2*10+a2*5+f2*-5

if tot1>tot2:
        print(hero1,"is the best hero with",tot1)
else:
        print(hero2,"is the best hero with",tot2)"'
```

#3. Implement a Python script for checking whether the citizen is eligible to cast vote or not.

```
'"age=int(input("Enter your age\n"))

if age>=18:
        print("you are eligible for vote cast")
else:
        print("You are not eligible for vote")
'"
```

'''4. Five Bikers Compete in a race such that they drive at a constant speed which may or may not be the same as the other. To qualify the race, the speed of a racer must be more than the average speed of all 5 racers. Take as input the speed of each racer and print back the speed of qualifying racers.

```
        '"
'"s1=int(input("Enter biker1 speed\n"))
s2=int(input("Enter biker2 speed\n"))
s3=int(input("Enter biker3 speed\n"))
s4=int(input("Enter biker4 speed\n"))
s5=int(input("Enter biker5 speed\n"))

s=s1+s2+s3+s4+s5
a=s/5

if s1>a:
        print("Biker1 is qualified")
```

```
if s2>a:
        print("Biker2 is qualified")
if s3>a:
        print("Biker3 is qualified")
if s4>a:
        print("Biker4 is qualified")
if s5>a:
        print("Biker5 is qualified")
'''
```

#5. Program to find the Color based on given character. Ex: R -> Red , W-> White

#6. Program to display the month name based given number. Ex: 3 -> March

```
'''color=input("Enter a Color code\n")

if color=='G':
        print("Green")
elif color=='R':
        print("Red")
elif color=='B':
        print("Blue")
elif color=='V':
        print("Violet")
elif color=='Y':
        print("Yellow")
elif color=='O':
        print("Orange")
elif color=='I':
        print("Indigo")
else:
        print("Please enter a valid color code\n")'''
```

#Example for nested if

'''

```
n=int(input("Enter a number between 0 to 100\n"))
if n<=100:
        if n%2==0:
                print(n,"is even and with in the range")
        else:
                print(n," is with in the range but not even")
else:
        print(n,"is not with in the range")'''
```

#range(start,end,step)

1.'''

```
for i in range(0,101,2):
        print(i,end=' ')
print()
```

```
2.for d in range(1,100,2):
        print(d,end=' ')
print()
```

```
3.for x in range(100,0,-1):
        print(x,end=' ')
print()
print()
```

```
4.for p in range(100,0,-2):
        print(p,end=' ')'''
```

5.'''name=input("Enter you name\n");

```
#name='adityaengineeringcollege'
count=0
for ch in name:
        count=count+1
print(count)'''

6.'''list1=[10,20,30,40,50]
sum=0
for i in list1:
        print(i,end=' ')
        sum=sum+i
print("sum is:",sum)'''

7.'''x=1
while x<=100:
        if x%2==0:
                print(x,end=' ')
        x=x+1'''
```

#break and continue

#Iterative Statements

 # program to display numbers from 1 to 100 using for loop

```
'''for x in range(1,101):
        print(x)'''
```

 # program to display even numbers upto given number

```
'''n=int(input("Enter a Number\n"))
for i in range(0,n+1,2):
        print(i,end=' ')'''
```

 # Program to display prime numbers upto given number

```
'''sum=0
n=int(input("Enter a number\n"))    # n=10
for j in range(2,n+1):
```

```
        count=0
        for i in range(1,j+1):          # [1,2,3,.....10]
                if j%i==0:
                        count=count+1
        if count==2:
                sum=sum+j
                print(j,end=' ')
print(sum)
'''
```

 # Program to find the sum of factors of a given number

```
'''sum=0
n=int(input("Enter a numbe\n"))  #10
i=1
while i<=n:
        if n%i==0:
                sum=sum+i
        i=i+1
print(sum)'''
```

 # Program to find the sum of digits of a given number

```
'''sum=0
n=int(input("Enter a number\n"))
while n>0:
        r=n%10
        sum=sum+r
        n=n//10
print(sum)'''
```

 # Program to find the reverse of a given number

```
'''sum=0
n=int(input("Enter a number\n"))
m=n
while n>0:
        r=n%10
```

```
        sum=sum*10+r
        n=n//10
if m==sum:
        print("palindrome")
else:
        print("not a palindrome")'''
```

 # Program to find the armstrong number of a given number

```
'''sum=0
n=int(input("Enter a number\n"))
m=n
while n>0:
        r=n%10
        sum=sum+r*r*r
        n=n//10
if m==sum:
        print("Armstrong")
else:
        print("Not a Armstrong number")'''
```

 # Program to find the number of digits of a given number

```
import math
n=int(input())
d=math.floor(math.log10(n))+1
print("number of digits = ",d)
```

#Implement a Python script that prompts the user for a number,
#and prints that number in words.

```
'''Example:
Input : 453
Ouput : Four Five Three
Input : 1000
Ouput : One Zero Zero Zero'''
```

```
import math
n=int(input())
d=math.floor(math.log10(n))+1

while d>0:
    a=(int)(n//math.pow(10,d-1))    # 453//100 =4
    if a==1:
        print("One",end=' ')
    elif a==2:
        print("Two",end=' ')
    elif a==3:
        print("Three",end=' ')
    elif a==4:
        print("Four",end=' ')
    elif a==5:
        print("Five",end=' ')
    elif a==6:
        print("Six",end=' ')
    elif a==7:
        print("Seven",end=' ')
    elif a==8:
        print("Eight",end=' ')
    elif a==9:
        print("Nine",end=' ')
    elif a==0:
        print("Zero",end=' ')
    n=n%math.pow(10,d-1)   # 453%100 =53
    d=d-1
```