BASICS OF PYTHON PROGRAMMING

Dr. K. B. Ramesh

Kripa Drishti Publications, Pune.

BASICS OF PYTHON PROGRAMMING

Dr. K. B. Ramesh
Associate Professor,
EIE Dept.,
RVCE, Bengaluru.

Kripa-Drishti Publications, Pune.

Book Title: **Basics of Python Programming**

Author By: **Dr. K. B. Ramesh**

ISBN: **978-81-19149-45-2**

Published: **Jan 2024**

Publisher:

Kripa-Drishti Publications

A/ 503, Poorva Height, SNO 148/1A/1/1A,
Sus Road, Pashan- 411021, Pune,
Maharashtra, India.
Mob: +91-8007068686
Email: editor@kdpublications.in
Web: https://www.kdpublications.in

© **Copyright Dr. K. B. Ramesh**

All Rights Reserved. No part of this publication can be stored in any retrieval system or reproduced in any form or by any means without the prior written permission of the publisher. Any person who does any unauthorized act about this publication may be liable to criminal prosecution and civil claims for damages. [The responsibility for the facts stated, conclusions reached, etc., is entirely that of the author. The publisher is not responsible for them, whatsoever.]

PREFACE

Python Programming is one of the vital programming languages for engineering students. This book will introduce you to the Python programming language in a simple manner. It's aimed at beginning programmers, but even if you've written programs before and just want to add Python to your list of languages, *Basics of Python Programming* will get you started. This book attempts to provide a simple explanation about the concepts of Python programming language with brief theory and a greater number of examples to illustrate the theoretical concepts. The contents presented in the book are very crisp and concise so that students are able to understand the concepts quickly. This book is for anybody interested in learning what seems to be emerging as the world's most popular computing language, whether or not you have learned any programming before. This book is structured into the following chapters.

Chapter 1: Introduction, Literals, Variables and Operators

Chapter 2: Control Structures

Chapter 3: Functions, Files and Object-Oriented Programming

I have made a sincere and honest effort to transform my handwritten notes into book form.

K. B. RAMESH

INDEX

ISBN: 978-81-19149-45-2

Chapter 1

Introduction, Literals, Variables and Operators

Guido Van Rossum (Born 31st January 1951) is a Dutch programmer best known as the author of the Python programming language.

- Van Rossum thought he needed a name that was unique, and slightly mysterious, so he decided to call the language.
- So he decided to call the language Python.
- The name comes from Rossum's favourite television show, 'Monty Python's Flying Circus', which was first released in 1991.
- The developer is Python software development.
- Python is an interpreter, object-oriented programming high-level language for general-purpose programming.

Computer Programming for Everybody:

In 1999, Van Rosum submitted a funding proposal to DARPA called **"Computer Programming for Everybody"** in which he defined his goals for Python.

a. An easy and intuitive language just as powerful as major competitors.
b. Open source, so anyone can contribute to its development.
c. Code that is as understandable as Simple English.
d. Suitability for every task, allowing for short development times.

Python Overview:

- Scripting language.
- Object-oriented.
- Portable.
- Powerful.
- Easy to learn and use.

- Includes the best features of Java, Pearl, etc.

Advantages of Python:

- System utilities.
- GUIs.
- Internet scripting.
- Embedded scripting.
- Database programming.
- Artificial intelligence.
- Image processing.

Note Summary:

- **Simple:** Allows programmers to concentrate on the solution to the problem rather than the language itself.
- **Platform-Independent:** Python is an open-source project, supported by many individuals. It is a platform-independent, scripted language.
- **Easy to learn.**
- **Versatile:** Python supports development of a wide range of applications.
- **Portable:** The programs work on any of the operating systems.
- Object-oriented and procedure-oriented techniques.
- **Interpreted:** Python is processed at run-time by the interpreter. So, there is no need to compile a program before executing it, we can simply run the program.
- **Dynamic:** Python executes dynamically.
- **Embeddable:** Programmers can embed Python within their C, C++, CORBA and Java programs to give 'scripting' capabilities for users.
- **Extensive libraries:** Python has a huge library that is easily portable across different platforms.
- **Easy maintenance:** Code written in Python is easy to maintain.

Writing and Executing Python Program:

- Download Python from www.python.org

- Python programming versions.

a. Python 1: 1.0, 1.5, 1.5
b. Python 2: 2.0, 2.1, 2.2
c. Python 3: 3.0, 3.1, 3.2

Note:

Python 2 is legacy, Python 3 is the future.

They have different libraries.

- Install IDLE [Integrated Development and Learning Environment]

Comparing Python with C++ Programmes:

A. #include < iostream.h >

```
voids main ()
{
        printf (“Hello, World!”);
}
```

C + + program

B. #include <stdio.h> → pre-processor command

```
voids main ()
{
        printf (“Hello, World!”);
}
```

C program

C. Print (“Hello World”)

D. $x = \text{int}\left(\text{input}\left(\text{"Enter the value of } x / n\text{"}\right)\right)$

$y = \text{int}\left(\text{input}\left(\text{"Enter the value of } y / n\text{"}\right)\right)$

print ("x + y =", x + y)

print ("\n x - y =", x - y)

print ("\n x + y =", x * y)

print ("\n")

For 'i' in range (1, 10):

print (i)

- **The range ()** function defaults to 0 as a starting value.

However, it is possible to specify the starting value by adding a parameter.

a. Range (2, 6) which means values from 2 to 6 (but not including 6).
b. Range (6) is not the values of 0 to 6, but the values 0 to 5.

- **Data and Expressions**

Literals, Variables, Operations, Data types

- **Literal:** a value that is expressed as itself.
- **Example:** The number 25. The string "Hello world" □ Literals.
- **Variable:** Its value can change during the execution of the program.
- **Constant:** A constant retains the same value throughout the program.

NOTE: A literal is a notation for representing a fixed value.
A variable is a storage location associated with a symbolic name.

Examples: 1, 1.5, ‘a’, “abc” → Literals

Variables → x = 123→ Literals

x = 2 + 3 → 2 and 3 are literals

2 + 3 → is an expression

x → is variable

Problem:

What is the difference between variable, constant and literal?

- **Variables:** Name of the locations
  Example: int i = 10; variable
- **Constants:** Same as variables but the only difference is that once the value is assigned to the constant, its value can’t be changed.
  Example: const int i = 10 constant
- **Literals:** are values assigned to variables and constants.

NOTE: Constant are like a variable

- Constants and variables are both tools for storing data in memory. In most languages, we need to mention the type of data we wish to store, but in Python, this is done automatically
- **Operators**: Operators are special symbols in Python that carry out arithmetic and/or logical computation.

Assigning Value to a Variable in Python:

A. website = “Apple.com”

print (website) o/p: Apple.com

B. website = “Apple.com”

website = “programing.com”

print (website)

} o/p: programing.com

C. Assigning multiple values to multiple variables

a, b, c = 5, 3.2, “Hello”

print (a)
print (b)
print (c)

} o/p: 5
3.2
Hello

D. Assigning the same value to multiple variables

x = y = z = “same”

print (x)

print (y)

print (z)

E. Valid Program

x = 10 = 21.54 z = “Hello python” w = “Hello” Print (x, y, z, w)	print (x) print (y) print (z) print (w) a = b = c = d = 100 Print (a, b, c, d)	a=b= c=d = “I.T Dept” print (a, b, c, d)

P5. Py
PI = 3.14

P6. Py
PI = 3.14
PI = 2.00
Print (P5.PI)

} o/p: 3.14

NOTE:

- In reality, we don't use constants in Python.
- Use capital letters to declare a constant.

Literals:

- **Numeric literals:** Binary, octal, decimal, hexa decimal, float, complex
- **String literals**
- **Boolean literals**
- **Special literals:** 'None' literal
- **Literal collection:** List, Tuple, Dict and set literals

String Literals:

- A string literal is a sequence of characters surrounded by quotes.
- We can use both single, double and triple quotes for a string.

a. **Boolean literals:** A boolean literal can have any of the two values: true or false. In Python, true represents the value as 1 and false as 0.

```
x = (1 = = true)          b = false + 10
y = (0 = = false)         print (x, y, a, b)   } o/p: true true, 5 10
a = true + 4
```

Program:

```
a = 10

b = 20

print (a + b) → o/p: 30
```

a = “10”

b = “20”

Print (a + b) → o/p: 1020

Program:

x = int (input ("Enter the value of x / n))

sum = 0

```
for i in range (x + 1): Sum = sum + I
print (sum)
```

```
for i in range (6):
print (i)
```

```
y = [1, 2, 3, 4, 5, 6, 7]
for i in y:
print (i)
```

```
y = [1, 2, 3, 4, 5, 6, 7,
“Ramesh”]
if i == 8: print (i)
break
print (i)
else:
print (“Finally finished”)
print (“The operation is
over”)
```

Factorial of Number:

x = int (input ("Enter the number \ n"))

factorial = 1

for i in range (1, x + 1)

factorial = factorial * i

print (factorial)

Python Operators:

Special symbol that carries out computation.

Example: 2 + 3

2, 3 → Operands

+ → Operator

- Arithmetic operators
- Comparison operators or Relational operators
- Assignment operators
- Logical operators or Bitwise operators
- Membership operators
- Identity operators
- Operator precedence

Arithmetic Operators:

Perform various arithmetic operations like addition, subtraction, multiplication, division, % modulus, exponent, etc.

Note: x%y → Remainder of x and y

Floor division: x//y → Division that results in the number adjusted to the left in the number line.

Exponent: x * * y → Left operand raised to the power of right [x to the power y]

Program

x = int (input (“enter the value of x\n”))

y = int (input (“enter the value of y\n”))

```
print ("x + y", x + y) # Addition

print ("x - y", x - y) # Subtraction

print ("x * y", x * y) # Multiplication

print ("x/y", x/y) # Division

print ("x //y = ", x // y) # Floor division

print ("x % y = ", x % y) # Modulus

print ("x ** y", x ** y)

print ("All operations executed successfully")
```

Comparison Operators:

Comparison operators are used to compare values. It either returns true or false according to the condition.

Example: >, <, = =, ! =, >=, < =

Example:

```
print (x < y)

print (x > y)

print (x = = y)

print (x < = y)

print (x > = y)

print (x! = y)
```

Example:

```
x = int (input ("enter the value of x/n")

y = int (input ("enter the value of y/n")
```

```
if x = = y;

print (x = = y)

else

print (x! = y)
```

Logical Operators:

They include, AND, OR and NOT operators

Example: x and y → True if both the operands are true

x or y → True if either of the operands is true

not x → Complements the operand

- The logical operators are used to compare Boolean expressions.
- The result of the Boolean expression is always a Boolean that is true or false.
- The logical operators are logical AND (&&), logical OR (||) and logical NOT(!)

```
x = int (input ("enter the value of x/n"))

y = int (input ("enter the value of y/n"))

if x = = y;

        print ("x is equal to y")

else if x < y

        print ("x is less than y")

# else if x > y
```

```
            print ("x is greater than y")
# else: print ("x is greater than y")
      print ("The operation is over")
      x = int (input ("enter the value of x/n")
      y = int (input ("enter the value of y/n")
      if x = = y; print ("x is equal to y"),
      print (x = = y)
      else print ("x is not equal to y")
      if x < = y: print ("x is less than or equal to y)
      else: print ("x is not less than or equal to y")
      if x > = y' print ("x is greater than equal to y")
else: print ("x is not greater than or equal to y")
if x < y: print ("x is not less than y")
if x > y: print ("x is greater than y")
else: print ("x is not greater than y")
if x! = y: print ("x is not equal to y")
else: print ("x is equal to y")
print ("The operations is over")
```

Sample Program:

1. x = 10

y = 20

if (x < y and x = = 10); print ("true")

print ("OK")

Output

True

OK

2. x = 10

y = 20

if (x < y and x! = 10); print ("True")

print ("OK")

Output

OK

3. x = 10

y = 20

if (x > y or x = = 10); print ("True")

print ("OK")

Output

True

OK

4. x = 10

y = 20

if (x > y or x! = 10); print ("False")

print ("OK")

Output

OK

5. x = 10

y = 20

if (x > y or y = = 20);

6. x = 10

y = 20

if (not (x > y));

print (“True”)

print (“OK”)

Output

True

OK

print (“True”)

print (“OK”)

Output

True

OK

7. x = 10

y = 20

if (not (x = = 10));

print (“True”)

print (“OK”)

Output

OK

8. x = 10

y = 20

print (x < y and x = = 10)

print (x < y and x! = 10)

print (x < y or x = = 10)

print (x! = 10 or y! = 20)

print (not (x > y))

print (not (x < y)) → False

Output

True

False

True

True

False

True

Assignment Operators:

=, += (add and assign operator), =, *=, /=, %=, **=

x = 10;

x+ = 10; x = x + 10

x – = 10; x = x – 10

x * = 10; x = x * 10 x/=10; x = x/10

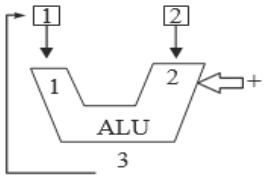
x// = 10; x = x//10

x * * = 10; x = x * * 10

x = 1

y = 2

print (x)



x = x + y

print (x)

or

x = 1

y = 2 print (x)

x+ = y; x = x + y x = 10

x+ = 10

print(x)

Sample Program

```
x = 10

y = 20

z = 0

z = x + y; 30

print (z)

z + = x ; 40

print (z)

z * = x; 400

print (z)

z/ = x; 40.000

print (z)

z = 3

z% = x

print (z); 3
```

Bitwise Operators:

Operators do operations on bits

(1) Example: (20) = 1 0 1 0 0 → five bits

Integer = 0 0 0 1 0 1 0 0

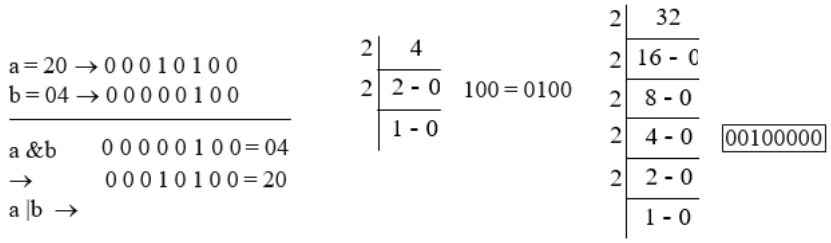
2	20
2	10 - 0
2	5 - 0
2	2 - 0
	1 - 0

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	20 in binary form
0	0	0	1	0	1	0	1	⇨ 16 + 4 = 20

* Binary representation of the value 20.

There are six bitwise operators

A. Bitwise AND: & Example: a & b
B. Bitwise OR: 1 Example: a/b
C. Bitwise XOR: ^ Example: a ^ b
D. Bitwise complement
E. Bitwise left shift <<
F. Bitwise right shift



a = 20
b = 4
print (a & b)
output = 04

a = 16;
b = 32;
print (a & b)
output = 0

0001 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0

a = 20
b = 4
a ^ b = 16

0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 0
a ^ b 0 0 0 1 0 0 0 0 = 16

0. 0 = 0
0. 1 = 1
1. 0 = 1
1. 1 = 0
Same input → output is zero
Different input → output is 1

Note: bin (10) = '0b1010'

bin (128) = '0b10000000'

Identity Operator:

They include, is, is not

- Identity operators compare the memory locations of two objects.

ISBN: 978-81-19149-45-2

Chapter 2

Control Structures

- Control structures are used to alter the order of execution of a program.
- There are Three types of control structures

(1) Decision Making Statements:

→ if statement

→ if … else statement

→ if … else if … else statement

→ nested if statement

Example:

```
a = 33

b = 200

if b > a;

print (“b is greater than a”)

print (b > a)
```

Output

```
b is greater than a

True
```

Note: Python rules on indentation, using whitespace, to define a set of statements of functions.

Example: Slope in the code

If statement without indentation

```
if b > a;
print (b > a)
```

Indentation error: expected an indented blank/space.

```
a = 200
b = 33
if b > a;
    print ("b is greater than a")
elif a = =b;
    print ("a is greater than b")
    print ("OK")
```

Nested if statements:

Example: To determine whether a number is positive, negative or zero using nested if statements.

```
# num = input ("enter a number")
Num = int(input ("enter a number")
if num > = 0;
```

```
if num = = 0;
        print (“Zero”)
else
        print (“Positive number”)
else
        print (“Negative number”)
print (“OK”)
```

(2) Python Loops:

In some situations, in programming, it is required to repeat some set of statements to attain the required results.

This repetition can be achieved by using a loop control structure.

Types of loops:

→ while

→ for

→ nested loop: loop inside another loop

Nested loop:

Program to compute the factorial of first n natural numbers.

(3) Python Control Statement:

→ Break: Break the loop immediately when the condition is true.

→ Continue: Returns the control to the beginning of the loop.

→ Pass: It represents a null operation; nothing happens when it executes.

- **Python Native Data Types:**

→ **Standard data types:** To handle group of data.

1) Number: Numeric data type Example: int, float, complex

2) String → sequence data types

3) List : List of items → sequence data types

4) Tuples: → sequence data types

5) Dictionary

6) Set

7) Boolean data type

- **List:** General purpose; most widely used data structure, grow and shrink size as needed; sequence type, sortable.
- **Tuple:** Immutable (cannot be modified), useful for fixed data, faster than lists, and sequence type.

Note:

- Lists are ordered sequences of values.
- Tuples are ordered, immutable sequences of values.
- Sets are unordered groups.
- Dictionaries are unordered elements of key-value pairs.

Note: List is similar to array in 'c' language but the only difference is that, array is a → group of similar elements where as list is a group of dissimilar elements.

Sample Program:

```
List = [1, 2, 3, 4, 5, “Biomedical”, ‘Ramesh’, “chandana”, “Niru”]

print (list)

for i in list

print (i)

print (“OK”)

print (“The operation is successful”)

print (list [0])

print (list [1])

print (list [2])

print (list [3])

list [0] = 10

list [1] = 20

print (list)

list.append (29)

list.extend ([76, 23, 15])

print (list)

list = [12, 13, 14, 15, 16, 17, 18, 19]

print [(len (list)]
```

```
print [max (list)]

print [(min (list)]

num = [I for i in range (10)]

print (num)
```

Note: The elements in a list can be altered whereas tuples are immutable and x cannot be changed.

```
# list = [10, 20, 30, 40, 50]

print (list)

list = [29, 28, 27, 26, 25, 24]

list.sort()

print (list)

list.remove (24)

print (list)

list.remove(25)

print (list)

        print (list.index (27))

        list.clear ()

        print (list)
```

Sample Program:

```
tuples = (10, 20, 30)

print(tuples)

for i in tuples:

        print (i)

tuple 2 = (1, “Ramesh”, “Chandu”, “Niru”)

        print (tuple 2)

tuple 3 = (100, [10, 20, 30], 1, (“Ramesh”, “Chandu”, “Niru”))

        print (tuple 3)

tuple 3 (tuple 1, tuple 2)

x, y = tuple 3

print (x)

print (y)

print (sum (tuple 1))

print (max (tuple 1))

print (min (tuple 1))
```

Note: Tuples are similar to the lists the only difference is that tuples are immutable. Due to this,

the tuple possesses certain advantages in Python over lists.

- Set: is an unordered collection of elements

a. No duplicity in set alike tuple
b. Mutable
c. Can group certain heterogeneous types of data.
d. Used to perform all mathematical set operations such as union, intersection, difference, etc.,

```
dataset1 = {10, 20, 30, 40}

dataset2 = {50, 60, 70, 80}

print(dataset1, dataset2)
```

 # Sample Program:

```
Dict = {“Name” : “Ramesh”, “College” : “RV”, “Year” : “1994”}

print(dict)

x = dict [“Name”]

print(x)

print(dict[“College”]

print(dict[“Year”]
```

 # Sample Program:

```
list = [1, 2, 3, 4, 5, “Ramesh”, “Chandu”, “Niru”]

print (list)

for i in list:

print (i)

print (len (list))
```

```
x = len (list)
print (x)
list 1 = [1, 20, 13, 14, 17]
print (max(list1))
print (max(list1))
print (min(list1))
list1.append (100)
print (list 1)
list1.extend ([200, 300, 400, 500)]
print (list 1)
list 1. remove (500)
print (list 1)
list 2 = [i for i in range (1, 10)]
print (list 2)
for i in list2:
print (i)
print (“OK”)
print (sum (list 2))
list 2. Reverse ()
```

print (list 2)

print (list, list 1, list 2)

- **Indexing**

a. The index of elements of a list starts from 0.
b. Example: If a list contains 10 elements, then its index will vary from 0 to 9.
c. If a user tries to access an element from a list beyond the range, it will result in an **Index error**.
d. **Type error:** If the user tries to access a list element using floating points indexing.

- **Tranversing a list**

a. Accessing or visiting elements of a list.
b. The methods to access the elements of a list.

(1) Indexing (2) Negative Indexing (3) Slicing

Note: Indexing operator / subscript operator → [ ]

Sample Program:

n = int (input (“enter the number”)

x = [i + 10 for i in range (n + 1)]

print (x)

for i in range (len (x)):

print (“x [“, I,” = “, x [i])

x = [10, 20, 30, 40]

print (x)

print (x[-1])

print (x[-2])

print (x[-3])

print (x[-4])

print (x[-5]) → Index Error

- **Slicing:**

a. The slicing operator is used to access the elements of a list within a specific range.
b. [:] is the slicing operator.
c. The slicing operator is used with different ranges of positive as well as negative indexing.
d. The syntax of the slicing operator is [beg: end], the end is excluded from the range.

- **List methods:**

Examples: append, extend, remove, sort, reverse, etc.,

- **List Functions**

Examples: comp (list1, list2), len(list), max(list), min(list), etc

Sample Program:

x = [10, 20, 30, 40]

```
x.append (50)

x.extend ([60, 70, 80])

x.remove (10)

x.reverse ()

x.sort ()

print (len(x))

print (max (x))

print (min (x))
```

- **List Comprehension**

a. It is a very useful feature in the Python language.
b. It provides an extremely efficient way to create a new list from the existing one.

Sample Program:

```
even = [i for i in range (50) if i%2 = =0]

print (even)

odd = [i for i in range (50) if i%2! = 0]

print (odd)
```

- **List Membership Test**

a. The most important operation is the searching.
b. The 'in' operator is used to search for an elements in the list. It returns true the if element exists, otherwise false.

Sample Program:

```
x = [10, 20, 30, 40, 50]

print (20 in x)

print (60 in x)
```

→ The 'in' operator is also used to iterate through the list using a for loop.

Sample Program:

```
for city in [" ", " ", " "];

print ("I visited", city)

x = [1, 2, 3, 4]

print (x)

for i in x:

print (i)

for i in range (len (x)) : print ("x [", i, ]", x [i])

print (2 in x)

print (5 not in x)
```

Sample Program

```
x = int (input ("enter the number"))

list1 = [i for i in range (x + 1)]

print (list1)
```

even = [i for i in range (x + 1) if i%2 == 0]

print (“Even = “, even)

odd = [i for i in range (x + 1) if i%2 ! = 0]

print(“odd” = ”, odd)

print (“OK”)

Alternative Statement:

even = [i for i list 1 if i%2 = = 0]

odd = [i for i list 1 if i%2! = 0]

 # Sample Program:

x = {“a” : 4, “b” : 2, “c” : 3, “d” : 4} → Dictionary

print (x)

y = x.items ( )

print (y)

for key, val in list (x.items ( )):

print (key, val)

for key, val in list (x.items ()):

print (val, key)

ISBN: 978-81-19149-45-2

Chapter 3

Functions, Files and Object Oriented Programming

*** Pass by value vs pass by reference:**

→ The two-way communication between the function caller and function definition is achieved through arguments.

→ **In Python language arguments can be passed by value and by object reference.**

 # Sample Program: (Pass by Value)

```
def update (x):

x = [10, 20, 30]

print (“Inside list”)

print (x)

return x = [1, 2, 3]

update (x)

print (“outside list”)

print (x)

print (“OK”)
```

This concept is known as pass-by value, in this case, if any modifications are made to the values in the function definition, then it does not have any effect on the arguments of the caller function.

 # Sample Program: (Pass by Object Reference)

```
def update (x):

x.extend ([100, 200, 300])

print ("outside list")

print (x)

print ("OK") r

eturn x = [1, 2, 3]

update list
```

In this program, the new values are appended after the old ones and a similar result is printed both in the function definition and after the function call.

Python Anonymous Function: (Lambda Function)

→ def keyword is to define the normal functions.

→ Lamda keyword is to create an anonymous function.

Syntax: Lambda arguments: Expression

Lambda arg1, arg2, arg3, … argn; expression

→ Python has two tools for building functions

(1) Def

(2) Lambda

→ Python allows to create anonymous function 'x' function having no names using a facility called lambda function.

→ Lambda functions are small functions, usually not more than a line.

→ The result of the expression is the value when the lambda is applied to an argument.

Example: Product = lambda a, b, c, d: a * b * c * d

prod = product (1, 2, 3, 4)

print (prod)

Here, lambda is just a single line statement, that performs the intended task and the result is assigned to the product variable.

product_1 = lambda a, b, c, d: a * b * c * d

product_2 = lambda a, b, c, d: a + b + c + d

prod_1 = product_1 (1, 2, 3, 4)

prod_2 = product_2 (1, 2, 3, 4)

print (prod_1)

print (prod_2)

print (“OK”)

Sample Program on Lambda Function

Product = lambda a, b, c, d: a * b * c * d

Add = lambda a, b, c, d: a + b + c + d

Prod = product (2, 3, 4, 5)

Add = add (2, 3, 4, 5)

Print (“product of numbers is”, prod)

Print (“addition of numbers is”, add)

Print (“ok done”)

#Sample Program

def funct_1(x):

x.exten ([20, 30, 40])

print (x)

print (“Inside the function”)

for i in x:

print (i)

return x = [1, 2, 3, 4]

funct_1(x)

print (“outside the function”)

print (x)

print (“OK”)

*** Recursion:** Program to compute the factorial of a number using recursion.

def fact(n):

if (n = = 0):

return (1)

```
    else:
            return (n * fact (n – 1))
num = (input (input (“enter the number”))
result = fact (num)
print (“The factorial of a given number is”, result)
print (“ok, done”)
```

* Scope and lifetime of variables

```
val = 0
def scope (val):
    print (val)
        return scope (10)
print (val)
```

The scope of a variable can either be local or global. It is a region of the program where it is recognized.

→ The lifetime of a local variable is as long as that function executes.

→ The lifetime of a global variable is as long as the whole program executes.

```
val = 0 def = fun_1(val):
    print (“value = “, val)
    return
```

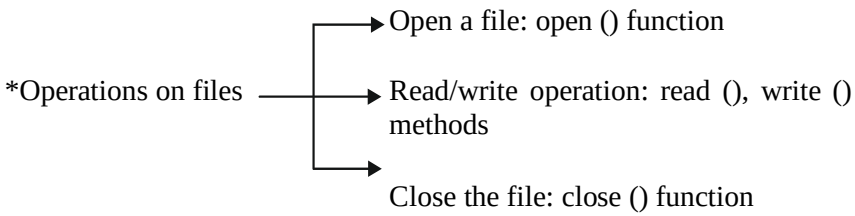
num = int (input (“enter the value of val:”))

func_1 (num)

print (“value = “, val)

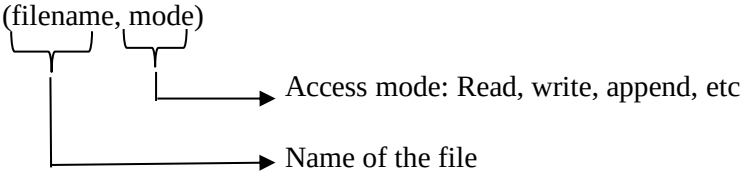
print (“OK, done”)

*** Files:** Collection of data.



* Opening the file

→ open (filename, mode)



→ file object = open (file_name, access_mode)

→ **Example:** text_file = open ("sample.txt", 'w')

File object

Name of the file

 # Sample Program

```
text_file = open (“sample.txt”, ‘w’)

text_file. Write (“Hello”)

text_file close ()

text_file = open (“sample.txt”, ‘a’)

text_file.write (“welcome to IT Department”) t

ext_file close ()

text_file = open (“sample.tex”, ‘r’)

print(text_file.read())

text_file close ()
```

*** Object Oriented Programming**

→ Encapsulation ——→ Class

→ Data hiding and abstraction ——→ OBject

→ Inheritance

→ Polymorphism ——→ Overloading

→ Dynamic binding ——→ Overriding

 # Sample Program

```
# Function

def display (name)

        print (“Hello”, name)

        return

# main

display (“sam”)

display (“Ram”)
```

Example of procedural-oriented approach.

In this approach, the functions are the main aspect.

*** Object Oriented Programming:** In this approach, the class and objects are the main aspects.

* **Class:** Blueprint/template of an object. Using class we can create many objects.

*** Example:**

→ Object: House

→ Class: Sketch/prototype/plan

→ Using the plan we can build many houses, in a similar way

→ Using class, we can create many objects.

 # Sample Program

Class person:

def display (self):

Refers to object: display (person1)

No need to send object externally

print (“Hello”)

person1 = person ()

person1.display ()

 # Program – 2

Class person:

def_init_ (self, name):

self.name = name

def display (self):

print (“Hello”, sefl.name)

person1 = person(“Ram”)

person1.display()

 # Program_3:

Class person:

```
def_int_ (self, id, name, salary):
        self.id = id
        self.name = name
        self.salary = salary
def display (self)
        print (“Id = “, self.id, “|n”, “Name = “, self.name, “|n”, “salary = “, self.salary)
person1 = person (1, “Ramesh”, 3000)
person1.display()
print (“OK, done”)
```

 # Program_3

Class demo:

```
        count = 0
        def_init_(self):
                dem.count+ = 1
        demo1 = demo ()
        demo2 = demo ()
        demo3 = demo ()
```

```
    print ("The number of objects created is", demo.count)
    print ("OK")
```

 # Program_4:

Class demo:

```
    def_init_ (self, x, y):
        self.x=x
        self.y=y
    def add(self):
        print ("addition = ", self.x + self.y)
    def sub(self):
        print ("subtraction = ", self.x – self.y)
    def mul(self):
        print ("Multiplication =", self.x * self.y)
    demo1 = demo (2, 3)
    demo1 = add ()
    demo2.sub ()
    demo3.mul ()
    print("done")
```

 # Sample Program

Class demo:

```
    def get_data(self):
        self.x = int (input (“enter the first number:”))
        self.y = int(input(“enter the second number:”))
    def display_data(self):
        print (“x=”, self.x)
        print (“y=”, self.y)
    demo_1 = demo ()
    demo1.get_data ()
    demo_1. display data ()
```

 # Sample Program

Class done:

```
    no_of_objects = 0
    def_init_(self):
        demo.no_of_objects + = 1
    def get_data(self):
        self.x = int (input (“enter the first number:”))
        self.y = int(input(“enter the second number:”))
```

```
def display_date(self):
        print ("x=", self.x)
        print ("y=", self.y)
demo1 = demo ()
demo2 = demo ()
demo3 = demo ()
demo4 = demo ()
demo1 get_data ()
demo1.display_data ()
demo2.get_data ()
demo3.get_data ()
demo4.get_data ()
demo2.display_data ()
demo3.display_data ()
demo4.display_data ()
print ("The number of objects created", demo.no_of_objects)
```

 # Sample Program

```
Class demo:
        no_of_objects = 0
        def_initi_(self):
                self.id = int (input (“enter the ID”))
                self.name = int (input (“enter the name”))
                demo.no_of_objects + = 1
        def display(self):
                print (“ID: = “, self.id, “|n”, “Name: =”, self.name)
demo_1 = demo () demo_1. display ()
demo_2 = demo () demo_2. display ()
demo_3 = demo () demo_3. display ()
demo_4 = demo () demo_4. display ()
print (“The number of objects is created: =”, demo.no_of_objects)
```

ABOUT THE BOOK

Python is a powerful general-purpose programming language. It's now one of the most popular and widely used programming languages in the world. It is used in web development, data science, creating software prototypes, data analytics, machine learning design and so on.

The Python language has the following features.

- Easy To Learn & Use While Coding
- Extensible Feature
- Interpreted Language
- Expressive Language
- Cross-Platform Portable Language
- Dynamic Memory Allocation
- High-Level Interpreted Language
- Graphical User Interface (GUI) Support:
- Object-Oriented Language
- Open Source Programming Language
- Large Standard Library
- Easy To Integrate
- Embeddable

Basics of Python Programming is written for students who are beginners in the field of computer programming languages. This book presents a simple approach with examples of the concepts of Python Programming for students. In this book used natural language expressions instead of the traditional shortened words of the programming world. This book has been written with the vision to provide students with a textbook that can be easily understood simply.

ABOUT THE AUTHOR

Dr. K. B. Ramesh is currently working as an Associate Professor in the Department of Electronics and Instrumentation Engineering at RV College of Engineering, Bengaluru, India. He earned his PhD in Engineering from Kuvempu University, Karnataka, India. He has 29-plus years of teaching experience in RVCE. He has published several papers in various reputed International Journals and Conferences. He is serving as paper reviewer and session chair for various prestigious International conferences. He is also member of CSI and ISTE. He has also served on the panel of Academic Bodies of Universities and Autonomous Colleges as a BOS and BOE member. Concerning research activity five research scholars have obtained Ph.D. degrees under his guidance and one research scholar are pursuing research work. His current research interests are mainly focused on Biomedical Instrumentation, Embedded system design, signal processing and computer architecture.

KRIPA DRISHTI PUBLICATIONS

Price: ₹400

Kripa-Drishti Publications
A-503 Poorva Heights, Pashan-Sus Road, Near Sai Chowk,
Pune – 411021, Maharashtra, India.
Mob: +91 8007068686
Email: editor@kdpublications.in
Web: https://www.kdpublications.in

ISBN: 978-81-19149-45-2

9 788119 149452