

New
syllabus
2021-22

Chapter 5
Basics of
Python
Programming

Informatics Practices

Class XI (As per CBSE Board)

Visit : python.mykvs.in for regular updates



Basics of Python Programming

Structure of a python program

Program

| -> Module -> main program

| -> functios

| ->libraries

| ->Statements -> simple statement

| ->compound statement

| ->expressions -> Operators

| -> expressions

| ---->objects ->data model

Python basics



Python 3.0 was released in 2008. Although this version is supposed to be backward incompatible, later on many of its important features have been back ported to be compatible with version 2.7

Python Character Set

A set of valid characters recognized by python. Python uses the traditional ASCII character set. The latest version recognizes the Unicode character set. The ASCII character set is a subset of the Unicode character set.

Letters :- A-Z,a-z

Digits :- 0-9

Special symbols :- Special symbol available over keyboard

White spaces:- blank space,tab,carriage return,new line, form feed

Other characters:- Unicode

Input and Output



```
var1='Computer Science'  
var2='Informatics Practices'  
print(var1,' and ',var2,')
```

Output :-

Computer Science and Informatics Practices

raw_input() Function In Python allows a user to give input to a program from a keyboard but in the form of string.

NOTE : raw_input() function is deprecated in python 3

e.g.

```
age = int(raw_input('enter your age'))  
percentage = float(raw_input('enter percentage'))
```

input() Function In Python allows a user to give input to a program from a keyboard but returns the value accordingly.

e.g.

```
age = int(input('enter your age'))  
C = age+2 #will not produce any error
```

NOTE : input() function always enter string value in python 3.so on need int(),float() function can be used for data conversion.

Visit : python.mykvs.in for regular updates

Indentation



Indentation refers to the spaces applied at the beginning of a code line. In other programming languages the indentation in code is for readability only, whereas the indentation in Python is very important.

Python uses indentation to indicate a block of code or used in block of codes.

E.g.1

```
if 3 > 2:
```

```
print("Three is greater than two!") //syntax error due to not indented
```

E.g.2

```
if 3 > 2:
```

```
    print("Three is greater than two!") //indented so no error
```



Token

Smallest individual unit in a program is known as token.

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. punctuators

Keywords

Reserve word of the compiler/interpreter which can't be used as identifier.

and	exec	not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Identifiers



A Python identifier is a name used to identify a variable, function, class, module or other object.

* An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

* Python does not allow special characters

* Identifier must not be a keyword of Python.

* Python is a case sensitive programming language.

Thus, Rollnumber and rollnumber are two different identifiers in Python.

Some valid identifiers : Mybook, file123, z2td, date_2, _no



Identifiers-continue

Some additional naming conventions

1. Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
2. Starting an identifier with a single leading underscore indicates that the identifier is private.
3. Starting an identifier with two leading underscores indicates a strong private identifier.
4. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.



Literals

Literals in Python can be defined as number, text, or other data that represent values to be stored in variables.

Example of String Literals in Python

```
name = 'Johni' , fname = "johny"
```

Example of Integer Literals in Python(numeric literal)

```
age = 22
```

Example of Float Literals in Python(numeric literal)

```
height = 6.2
```

Example of Special Literals in Python

```
name = None
```

Escape sequence/Back slash character constants

Escape Sequence	Description
<code>\\</code>	Backslash (\)
<code>\'</code>	Single quote (')
<code>\"</code>	Double quote (")
<code>\a</code>	ASCII Bell (BEL)
<code>\b</code>	ASCII Backspace (BS)
<code>\f</code>	ASCII Formfeed (FF)
<code>\n</code>	ASCII Linefeed (LF)
<code>\r</code>	ASCII Carriage Return (CR)
<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\v</code>	ASCII Vertical Tab (VT)
<code>\ooo</code>	Character with octal value ooo
<code>\xhh</code>	Character with hex value hh



Operators can be defined as symbols that are used to perform operations on operands.

Types of Operators

1. Arithmetic Operators.
2. Relational Operators.
3. Assignment Operators.
4. Logical Operators.
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

Operators continue

1. Arithmetic Operators

Arithmetic Operators are used to perform arithmetic operations like addition, multiplication, division etc.

Operators	Description	Example
+	perform addition of two number	$x=a+b$
-	perform subtraction of two number	$x=a-b$
/	perform division of two number	$x=a/b$
*	perform multiplication of two number	$x=a*b$
%	Modulus = returns remainder	$x=a\%b$
//	Floor Division = remove digits after the decimal point	$x=a//b$
**	Exponent = perform raise to power	$x=a**b$

Operator continue



Arithmetic operator continue

e.g.

```
x = 5
```

```
y = 4
```

```
print('x + y =',x+y)
```

```
print('x - y =',x-y)
```

```
print('x * y =',x*y)
```

```
print('x / y =',x/y)
```

```
print('x // y =',x//y)
```

```
print('x ** y =',x**y)
```

OUTPUT

```
('x + y =', 9)
```

```
('x - y =', 1)
```

```
('x * y =', 20)
```

```
('x / y =', 1)
```

```
('x // y =', 1)
```

```
('x ** y =', 625)
```

- Write a program in python to calculate the simple interest based on entered amount ,rate and time



Operator continue

EMI Calculator program in Python

```
def emi_calculator(p, r, t):  
    r = r / (12 * 100) # one month interest  
    t = t * 12 # one month period  
    emi = (p * r * pow(1 + r, t)) / (pow(1 + r, t) - 1)  
    return emi
```

driver code

```
principal = 10000;  
rate = 10;  
time = 2;  
emi = emi_calculator(principal, rate, time);  
print("Monthly EMI is= ", emi)
```

Operator continue



Arithmetic operator continue

How to calculate GST

GST (Goods and Services Tax) which is included in netprice of product for get GST % first need to calculate GST Amount by subtract original cost from Netprice and then apply

GST % formula = (GST_Amount*100) / original_cost

Python3 Program to compute GST from original and net prices.

```
def Calculate_GST(org_cost, N_price):
```

```
# return value after calculate GST%
```

```
    return (((N_price - org_cost) * 100) / org_cost);
```

Driver program to test above functions

```
org_cost = 100
```

```
N_price = 120
```

```
print("GST = ",end="")
```

```
print(round(Calculate_GST(org_cost, N_price)),end="")
```

```
print("%")
```

*** Write a Python program to calculate the standard deviation**



Operators continue

2. Relational Operators/Comparison Operator

Relational Operators are used to compare the values.

Operators	Description	Example
==	Equal to, return true if a equals to b	a == b
!=	Not equal, return true if a is not equals to b	a != b
>	Greater than, return true if a is greater than b	a > b
>=	Greater than or equal to , return true if a is greater than b or a is equals to b	a >= b
<	Less than, return true if a is less than b	a < b
<=	Less than or equal to , return true if a is less than b or a is equals to b	a <= b

Operator continue



Comparison operators continue

e.g.

```
x = 101
```

```
y = 121
```

```
print('x > y is',x>y)
```

```
print('x < y is',x<y)
```

```
print('x == y is',x==y)
```

```
print('x != y is',x!=y)
```

```
print('x >= y is',x>=y)
```

```
print('x <= y is',x<=y)
```

Output

```
('x > y is', False)
```

```
('x < y is', True)
```

```
('x == y is', False)
```

```
('x != y is', True)
```

```
('x >= y is', False)
```

```
('x <= y is', True)
```



Operators continue

3. Assignment Operators

Used to assign values to the variables.

Operators	Description	Example
=	Assigns values from right side operands to left side operand	a=b
+=	Add 2 numbers and assigns the result to left operand.	a+=b
/=	Divides 2 numbers and assigns the result to left operand.	a/=b
=	Multiply 2 numbers and assigns the result to left operand.	A=b
-=	Subtracts 2 numbers and assigns the result to left operand.	A-=b
%=	modulus 2 numbers and assigns the result to left operand.	a%=b
//=	Perform floor division on 2 numbers and assigns the result to left operand.	a//=b
=	calculate power on operators and assigns the result to left operand.	a=b

Operators continue

4. Logical Operators

Logical Operators are used to perform logical operations on the given two variables or values.

Operators	Description	Example
and	return true if both condition are true	x and y
or	return true if either or both condition are true	x or y
not	reverse the condition	not(a>b)

```
a=30
```

```
b=20
```

```
if(a==30 and b==20):
```

```
    print('hello')
```

Output :-

```
hello
```

Operators continue

6. Membership Operators

The membership operators in Python are used to validate whether a value is found within a sequence such as strings, lists, or tuples.

Operators	Description	Example
in	return true if value exists in the sequence, else false.	a in list
not in	return true if value does not exists in the sequence, else false.	a not in list

E.g.

```
a = 22
```

```
list = [22,99,27,31]
```

```
In_Ans = a in list
```

```
NotIn_Ans = a not in list
```

```
print(In_Ans)
```

```
print(NotIn_Ans)
```

Output :-

```
True
```

```
False
```

Operators continue

7. Identity Operators

Identity operators in Python compare the memory locations of two objects.

Operators	Description	Example
is	returns true if two variables point the same object, else false	a is b
is not	returns true if two variables point the different object, else false	a is not b

e.g.

```
a = 34
```

```
b=34
```

```
if (a is b):
```

```
    print('both a and b has same identity')
```

```
else:
```

```
    print('a and b has different identity')
```

```
b=99
```

```
if (a is b):
```

```
    print('both a and b has same identity')
```

```
else:
```

```
    print('a and b has different identity')
```

Output :-

both a and b has same identity

a and b has different identity

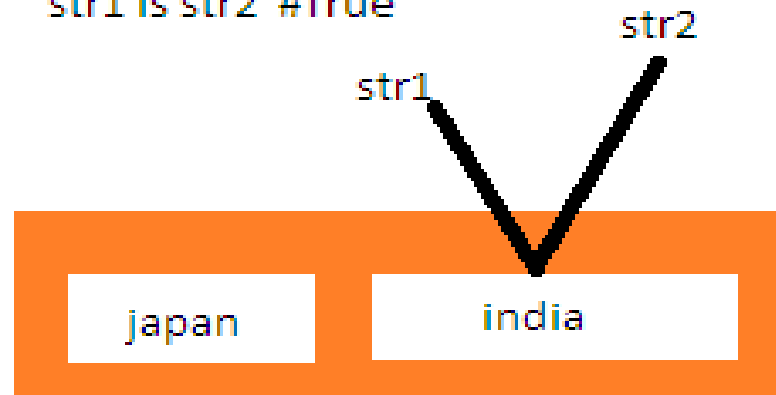
memory address of a variable in python

```
str1="india"
```

```
str2="india"
```

```
str1 == str2 #True
```

```
str1 is str2 #True
```



Operator continue

Operators Precedence :

highest precedence to lowest precedence table. Precedence is used to decide ,which operator to be taken first for evaluation when two or more operators comes in an expression.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus,minus(method names for the last two are+@and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //=- = += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators



Punctuators

Used to implement the grammatical and structure of a Syntax. Following are the python punctuators.

‘ “ # \
() [] { } @
, : . \ = ;
+= -= *= /= //= %=
&= |= ^= >>= <<= **=



Barebone of a python program

```
#function definition      comment
def keyArgFunc(empname, emprole):
    print ("Emp Name: ", empname)
    print ("Emp Role: ", emprole)
    return;
```

Function

indentation

```
A = 20
print("Calling in proper sequence")
keyArgFunc(empname = "Nick",emprole = "Manager" )
print("Calling in opposite sequence")
keyArgFunc(emprole = "Manager",empname = "Nick")
```

expression

statements

A python program contain the following components

- Expression
- Statement
- Comments
- Function
- Block & n indentation



Barebone of a python program

- a. **Expression** : - which is evaluated and produce result. E.g. $(20 + 4) / 4$
- b. **Statement** :- instruction that does something.

e.g

```
a = 20
```

```
print("Calling in proper sequence")
```

- c. **Comments** : which is readable for programmer but ignored by python interpreter
 - i. Single line comment: Which begins with # sign.
 - ii. Multi line comment (docstring): either write multiple line beginning with # sign or use triple quoted multiple line. E.g.

```
'''this is my
```

```
first
```

```
python multiline comment
```

```
'''
```

- d. **Function**

a code that has some name and it can be reused.e.g. keyArgFunc in above program

- d. **Block & indentation** : group of statements is block.indentation at same level create a block.e.g. all 3 statement of keyArgFunc function



Variables

Variable is a name given to a memory location. A variable can consider as a container which holds value. Python is a type infer language that means you don't need to specify the datatype of variable. Python automatically get variable datatype depending upon the value assigned to the variable.

Assigning Values To Variable

```
name = 'python' # String Data Type
sum = None # a variable without value
a = 23      # Integer
b = 6.2     # Float
sum = a + b
print (sum)
```

Multiple Assignment: assign a single value to many variables

```
a = b = c = 1 # single value to multiple variable
a,b = 1,2 # multiple value to multiple variable
a,b = b,a # value of a and b is swaped
```

Variables



Variable Scope And Lifetime in Python Program

1. Local Variable

```
def fun():  
    x=8  
    print(x)
```

```
fun()  
print(x) #error will be shown
```

2. Global Variable

```
x = 8  
def fun():  
    print(x) # Calling variable 'x' inside fun()
```

```
fun()  
print(x) # Calling variable 'x' outside fun()
```



Dynamic typing

Data type of a variable depend/change upon the value assigned to a variable on each next statement.


```
X = 25          # integer type
```

```
X = "python"   # x variable data type change to string on just next line
```

Now programmer should be aware that not to write like this:

```
Y = X / 5      # error !! String cannot be divided
```

Constants



A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

In Python, constants are usually declared and assigned in a module. Here, the module is a new file containing variables, functions, etc which is imported to the main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Create a constant.py:

```
PI = 3.14
```

Create a main.py:

```
import constant  
print(constant.PI)
```

Note: In reality, we can not create constants in Python. Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment, so we can change it's value

Input and Output



print() Function In Python is used to print output on the screen.

Syntax of Print Function - `print(expression/variable)`

e.g.

```
print(122)
```

Output :-

122

```
print('hello India')
```

Output :-

hello India

```
print('Computer','Science')
```

```
print('Computer','Science',sep=' & ')
```

```
print('Computer','Science',sep=' & ',end='.')
```

Output :-

Computer Science

Computer & Science

Computer & Science.