BVRIT

09 February 2026 09:00

Python:-	Python is a **General purpose** programming language.

1. It was developed by **Guido Van Rossum** in **1991**.
2. The latest version of python is **Python 3.14.3** (2026)

Features of Python:-

1. Easy to learn and analyze
2. High Level programming language
3. Dynamically typed programming language
4. Interpreter based programming language
5. Huge no. of libraries
6. Free and Open Source
7. Platform Independent
8. Large community support

Introduction to Library Functions:-

Library Function:- It is a library which consists of n number of functions.

Function:- Are nothing but **predefine**d things.

For ex:- + --> Each of them having some predefined functionality

- -->

* -->

len(var):- It is a function i.e. used to calculate the **no. of values** present in a container.
And it is used with **multivalued datatype** only.

Similar to this, there are many others functions which are predefined whose task are already defined and that function are called as **Library Functions.**

There are 3 types of Library Functions:-

1. Keywords
2. Operators / Special Symbols
3. Inbuilt Functions

Keywords:-

1. It is a Universal Standard words whose task are predefined by the developer and it is fixed.
2. We can only access this, but cannot modify this.
3. To check keywords -
    a. help("keywords")
    b. import keyword
       keyword.kwlist
4. No. of keywords = 35
5. True, False, None --> **Special Keywords**
    a. Starting with the uppercase character
    b. We can assign this keywords as value also.

Variables:-

Variable is a name given to particular memory location where we stored the value.

OR

It is a container which is used to store the value.

Syntax:-	**var_name = value**

Id() function:-

1. Used to return the integer address.
2. **Syntax:-** id(val/var_name)
3. If no. >256 --> different id generated .

Multiple Variable Creation:-

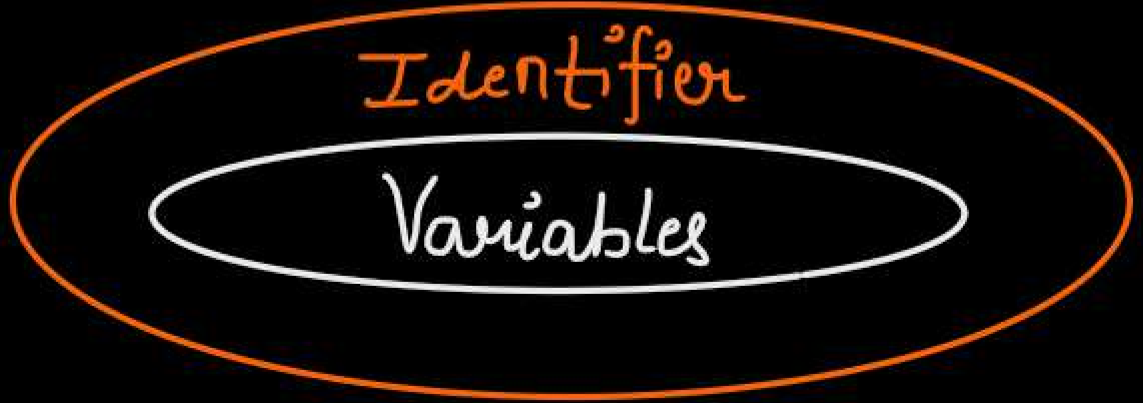
Example:-	a, b, c, d, e	10, 20, 30, 10, 20

Reference Count

Ques:- Can we write anything as variable name?

Identifiers rules:-

Identifier is a variable which is used to identify the value stored in it.
* All Variables are identifiers but all identifiers are not variables.



Rules of Identifiers:-

1. Identifier should not be a keyword.
2. It should not start with numbers.
3. It should not contain any special characters except _ .
4. It should not contain space in between or at the beginning.
5. It can have alphabets or alphanumeric but it always start with alphabets or _
6. According to ISR(Industrial standard rules), it should not cross more than 72 characters limits.

Datatypes:-

Datatype specify the size and type of value i.e. going to stored in a variable.

Based on size of the value:-

Datatype		
Single valued Datatype		Multi valued Datatype
Numeric	**Boolean**	1. String 2. List 3. Tuple 4. Set 5. Dictionary
1. Integer 2. Float 3. Complex	1. Bool	

Single valued Datatype:-

1. **Integer:-**

   Any real no. in range between -inf to +inf without decimal points are called Integer.

Standard Representation	int

type(var/val):-	Used to check the standard representation.

2. **Float:-**

   Any real no. with decimal point.

Standard Representation	float

3. **Complex:-**

   Combination of real and imaginary no.

Syntax	$\pm a \pm bj$

   Real Part ← a ; bj → imaginary part ($j = \sqrt{-1}$)

   Example - 7 + 6.5j

Example - $\underset{\text{Real no.}}{7} + \underset{\text{Imaginary no.}}{6.5j}$

4. Boolean:-

a. It consists of only two types of values i.e. True or False.

b.

True	1
False	0

Standard Representation	bool	

Multivalued Datatype:-

- **String:-**

  Collection of characters enclosed between '', "", """""".

Syntax	var = 'val1val2.....valn'
	var = "val1val2.....valn"
	var = '''val1val2.....valn'''

Standard Representation	str

Example:- s = 'Python 3.14.3@'

Indexing:-

Index:- Sub-address provided to each & every element of any collection

1. It is used to extract particular characters from a given collection
2. It is a sub-address given to each and every block of memory.
3. **Types of indexing:-**

+ve indexing	Traversal from Left to right	Start = 0

-ve indexing	Traversal from Right to left	Start = last index

4.

Syntax:-	Var[index]
Modification	Var[index] = new value

* While doing modification, if controller are not throwing any error	----->	**Mutable Collection**
And if error is generated	----->	**Immutable Collection**

Example:- s = 'Python 3.14.3@'
s[0] = 'A'   -----> TypeError(str object doesn't support item assignment)
5. Because, String is Immutable Datatype.

Methods of string:-

1. upper()

Use	Converts string to uppercase
Syntax	str.upper()
Args	0
Return	str
Example	"hello".upper() → "HELLO"

2. lower()

Use	Converts string to lowercase
Syntax	str.lower()
Args	0
Return	str
Example	"HELLO".lower() → "hello"

3. capitalize()

Use	First character uppercase, rest lowercase
Syntax	str.capitalize()
Args	0
Return	str
Example	"python language".capitalize() → "Python language"

4. title()

Use:	First letter of each word uppercase
Syntax	str.title()
Args	0
Return	str
Example	"python language".title() → "Python Language"

5. isupper()

Use	Checks if all characters are uppercase
Syntax	str.isupper()
Args	0
Return	bool
Example	"HELLO".isupper() → True

6. islower()

Use	Checks if all characters are lowercase
Syntax	str.islower()
Args	0
Return	bool

Example	"hello".islower() → True

7. isalpha()

Use	Checks if string contains only alphabets
Syntax	str.isalpha()
Args	0
Return	bool
Example	"Python".isalpha() → True

8. isdigit()

Use	Checks if string contains only digits
Syntax	str.isdigit()
Args	0
Return	bool
Example	"123".isdigit() → True

9. count()

Use	Counts occurrences of a substring
Syntax	str.count(sub)
Args	1
Return	int
Example	"banana".count("a") → 3

10. replace()

Use	Replaces old value with new value
Syntax	str.replace(old, new)

Args	2
Return	str
Example	"hello world".replace("world","python")

11. split()

Use	Splits string into list
Syntax	str.split(sep)
Args	0 or 1
Return	list
Example	"a,b,c".split(",") → ['a','b','c']

12. strip()

Use	Removes spaces from both ends
Syntax	str.strip()
Args	0
Return	str
Example	" hi ".strip() → "hi"

13. lstrip()

Use	Removes spaces from left
Syntax	str.lstrip()
Args	0
Return	str
Example	" hi".lstrip() → "hi"

14. rstrip()

14. rstrip()

Use	Removes spaces from right
Syntax	str.rstrip()
Args	0
Return	str
Example	"hi ".rstrip() → "hi"

- **List:-**
  a. Collection of Homogeneous and heterogeneous values which are enclosed between [].

  b.

Homogeneous collection	Same datatype of each value
Heterogeneous collection	Different datatypes collection
Syntax:-	Var = [val, val2, val3, …., val n]
Standard Representation	list

  c. List is Mutable Datatype

Methods of list:-

1. append()

Use	Adds element at the end of list
Syntax	list.append(value)
Args	1
Return	None
Example	[1,2].append(3) → [1,2,3]

2. extend()

Use	Adds multiple elements to list

Syntax	list.extend(iterable)
Args	1
Return	None
Example	[1,2].extend([3,4]) → [1,2,3,4]

3. insert()

Use	Inserts element at given index
Syntax	list.insert(index, value)
Args	2
Return	None
Example	[1,2,3].insert(1,100) → [1,100,2,3]

4. remove()

Use	Removes first occurrence of value if present otherwise error
Syntax	list.remove(value)
Args	1
Return	None
Example	[1,2,3].remove(2) → [1,3]

5. pop()

Use	Removes element using index
Syntax	list.pop(index)
Args	0 or 1
Return	removed element
Example	[1,2,3].pop() → 3

6. clear()

Use	Removes all elements from list
Syntax	list.clear()
Args	0
Return	None
Example	[1,2].clear() → []

7. index()

Use	Returns index of given value if present otherwise error
Syntax	list.index(value)
Args	1
Return	int
Example	[10,20,30].index(20) → 1

8. count()

Use	Counts occurrences of value
Syntax	list.count(value)
Args	1
Return	int
Example	[1,1,2].count(1) → 2

9. sort()

Use	Sorts list in ascending order
Syntax	list.sort()
Args	0
Return	None

Example	`a = [3, 1, 2]`
	`a.sort()`
	`print(a)`
	→ [1,2,3]

10. reverse()

Use	Reverses the list
Syntax	list.reverse()
Args	0
Return	None
Example	[1,2,3].reverse() → [3,2,1]

3. Tuple:-

a. Collection of Homogeneous and heterogeneous values which are enclosed between ().

Homogeneous collection	Same datatype of each value
Heterogeneous collection	Different datatypes collection

b.

Syntax:-	Var = (val, val2, val3, …., val n)
Standard Representation	tuple
Example	t = (10, 20.5, 'Python', True)

c. Tuple is Immutable Datatype

Methods of Tuple:-

i. count()

Use	Counts occurrences of a value
Syntax	tuple.count(value)
Args	1

Args	1
Return	int
Example	(1,2,2,3).count(2) → 2

ii. **index()**

Use	Returns index of first occurrence of value
Syntax	tuple.index(value)
Args	1
Return	int
Example	(10,20,30).index(20) → 1

4. Dictionary:-

a. It is used to store the multiple values in the form of key-value pairs.

b.

Syntax	Var = {key1:val1, key2:val2, key3:val3, …, keyn:valn}

c. Key and value are separated by the :(colons) and the whole element is separated by ,(comma).

d. Indexing and slicing is not present in dictionary.

e. Key should be unique, immutable value and acts as index.

f.

Standard representation	dict
Default value	{}

g. There are 2 layers in dictionary -

i.

Key Layer	Only visible
Value Layer	Hidden

Methods of dictionary:-

i. get()

i. get()

Use	Returns value for given key (no error if key not found)
Syntax	dict.get(search_key, default_value)
Args	1 or 2
Return	value / None
Example	{"a":10,"b":20}.get("a") → 10

ii. keys()

Use	Returns all keys of dictionary
Syntax	dict.keys()
Args	0
Return	dict_keys
Example	{"a":10,"b":20}.keys() → dict_keys(['a','b'])

iii. values()

Use	Returns all values of dictionary
Syntax	dict.values()
Args	0
Return	dict_values
Example	{"a":10,"b":20}.values() → dict_values([10,20])

iv. items()

Use	Returns key-value pairs as tuples
Syntax	dict.items()
Args	0
Return	dict_items

Example	{"a":10,"b":20}.items() → dict_items([('a',10),('b',20)])

v. pop()

Use	Removes and returns value of given key
Syntax	dict.pop(key)
Args	1
Return	value
Example	{"a":10,"b":20}.pop("a") → 10

vi. clear()

Use	Removes all elements from dictionary
Syntax	dict.clear()
Args	0
Return	None
Example	d = {"a":10}; d.clear() → {}

Ques: Difference between mutable datatype and immutable datatype?

5. Set:-

a. It is an unordered and mutable collection of immutable values i.e. stored in {}.
b. Indexing and Slicing can't be performed because it is unordered.
c. Keeps only unique values.
d. It can only store - int, float, complex, bool, str, tuple.
e.

Standard Representation	set()
Default Value	set()

Methods of Set:-

i. add()

Use	Adds an element to the set
Syntax	set.add(value)
Args	1
Return	None
Example	s = {1,2}; s.add(3) → {1,2,3}

ii. copy()

Use	Returns a copy of the set
Syntax	set.copy()
Args	0
Return	set
Example	s = {1,2}; s.copy() → {1,2}

iii. pop()

Use	Removes and returns a random element
Syntax	set.pop()
Args	0
Return	element
Example	{1,2,3}.pop() → 1 (any element)

iv. remove()

Use	Removes specified element (error if not found)
Syntax	set.remove(value)
Args	1
Return	None

Example	{1,2,3}.remove(2) → {1,3}

v. discard()

Use	Removes specified element (no error if not found)
Syntax	set.discard(value)
Args	1
Return	None
Example	{1,2,3}.discard(5) → {1,2,3}

Operator:-	Used to perform operation on operands

Types of Operator:-

1. Arithmetic Operator (+, -, *, %, //, **)
2. Logical Operator (or, not, and)
3. Assignment Operator (=, +=, -=, *=, /=, //=, %=, **=)
4. Relational Operator (<, >, <=, >=, ==, !=)
5. Membership Operator (in, not in)
6. Bitwise Operator (&, |, ~, ^, <<, >>)
7. Identity Operator (is, is not)

1. Arithmetic Operators (+, -, *, %, //, **)

```
a = 10
b = 3

print(a + b)   # Addition → 13
print(a - b)   # Subtraction → 7
print(a * b)   # Multiplication → 30
print(a * b)  # Multiplication → 30
print(a % b)  # Modulus (remainder) → 1
print(a // b)  # Floor division → 3
print(a ** b)  # Exponentiation → 1000
```

2. Logical Operators (and, or, not)

```
x = True
y = False

print(x and y)  # True if both are True → False
print(x or y)   # True if any one is True → True
print(not x)    # Negation → False
```

3. Assignment Operators (=, +=, -=, *=, /=, //=, %=, **=)

```
c = 10        # Assignment
c += 5        # c = c + 5 → 15
c -= 2        # c = c - 2 → 13
c *= 2        # c = c * 2 → 26
c /= 2        # c = c / 2 → 13.0
c //= 2       # c = c // 2 → 6.0
c %= 4        # c = c % 4 → 2.0
c **= 3       # c = c ** 3 → 8.0
print(c)
```

4. Relational Operators (<, >, <=, >=, ==, !=)

```
p = 5
q = 10

print(p < q)   # Less than → True
```

4. Relational Operators (<, >, <=, >=, ==, !=)

```
p = 5
q = 10

print(p < q)    # Less than → True
print(p > q)    # Greater than → False
print(p <= q)   # Less than or equal → True
print(p >= q)   # Greater than or equal → False
print(p == q)   # Equal to → False
print(p != q)   # Not equal to → True
```

5. Membership Operators (in, not in)

```
lst = [1, 2, 3, 4]

print(2 in lst)       # Checks if 2 exists in list → True
print(5 not in lst)  # Checks if 5 not in list → True
```

6. Bitwise Operators (&, |, ~, ^, <<, >>)

```
m = 5    # 0101
n = 3    # 0011

print(m & n)   # AND → 1
print(m | n)   # OR → 7
print(~m)      # NOT → -6
print(m ^ n)   # XOR → 6
print(m << 1)  # Left shift → 10
print(m >> 1)  # Right shift → 2
```

7. Identity Operators (is, is not)

7. Identity Operators (is, is not)

```
r = [1, 2, 3]
s = r # [1, 2, 3]
t = [1, 2, 3]

print(r is s)       # Same object → True
print(r is t)       # Different object → False
print(r is not t)   # Not same object → True
```

input(): Used to get input from the user.

Syntax	var = input("msg")

i. Always considers the input in the form of string.
Ii. Using input(), we cannot directly insert tuple, list, dict, or set.

To overcome this issue, eval() is used.

eval():

Used to take user input and automatically convert it into its actual datatype.

Syntax	var = eval(input("msg"))

Example	var = eval(input("Enter value: "))

Input → [1,2,3,4]
Output → [1, 2, 3, 4]

Control Flow Statements:-

1. It is used to control the flow of execution of program.
2. **Types:-**
   a. Decisional / Conditional Control Statements (if, if-else, elif, nested-if)
   b. Looping Control Statements (while, for)

Decisional / Conditional Control Statements:-

These are the statements which is used to perform some operation **--> condition --> if satisfied**

a. Normal If

```
marks = 75

if marks >= 40:
    print("Pass")
```

b. if - else

```
marks = 75

if marks >= 40:
    print("Pass")
else:
    print("Fail")
```

c. elif

```
marks = 75

if marks >= 75:
    print("Distinction")
elif marks >= 40:
    print("Pass")
else:
    print("Fail")
```

d. nested if

```
marks = 75

if marks >= 40:
    if marks >= 75:
        print("Pass with Distinction")
    else:
        print("Pass")
else:
    print("Fail")
```

Looping Control Statements:-

These are the statements which is used to perform some task repeatedly.

a. WHILE LOOP:-

- Used to perform some task again and again until the given condition satisfied.
- No. of iterations are not known
- It will not works with set and dictionary
- **Syntax:**

```
Initialization
while condition:



        Updation
```

- Initialization & Updation are mandatory.
- Example 1:- While loop **without updation**

```
i = 1     # Initialization

while i <= 5:
    print(i)
```

- Example 2:- While loop **with updation**

```
i = 1     # Initialization

while i <= 5:
    print(i)
```

- Example 2:- While loop **with updation**

```
i = 1     # Initialization

while i <= 5:
   print(i)
   i = i + 1   # Updation
```

b. FOR LOOP:-

- For loops works with any of the collection datatype
- By default, for loops go till the length of the collection and iterate over each values.
- **Syntax:-**

```
for var in collection:
```

- **Examples (for loop)**

```
for i in range(1, 6):
print(i)


for ch in "Python":
print(ch)
```

- **range():-**

  1. range() is used to generate a sequence of numbers.
  2. Does **not store values**, it generates them one by one

Syntax:	range(start, stop, step)

• start → starting value (default = 0)

- stop → ending value (excluded)
- step → increment/decrement (default = 1)

Examples (range):-

```
print(list(range(5)))
print(list(range(1, 10, 2)))
```

Transfer Control Statements:-

1. **break:**	Terminated the execution of a loop
2. **continue**	Used to skip rest of the code
3. **pass**	Used as a **null statement** when no action is required

Code for break:-

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

Code for continue-

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Code for pass:-

Code for pass:-

```
def main():
    pass
```

Slicing:-

Slicing is a process where we are extracting a group of characters from collections.

Syntax	var[start_idx: end_idx: updation]

By default,

updation (step)	1
Left → Right	end_idx + 1
Right → Left	end_idx - 1

String Slicing Example:-

```
s = "BVRIT College"
s[0:5]
s[6:12]
```

Reverse the string

```
s[::-1]
```

Skip 1 character from "College"

```
s[6:13:2]
```

Tuple Example:-

```
st = ('Indore', 'Pune', 'Goa', 'Delhi')
st[-4:-1]
```

Nested Tuple