**• Introduction and History of Python Programming language**

[Python](https://www.geeksforgeeks.org/python-programming-language/) is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1989 and made available in 1991. It was started firstly as a hobby project because he was looking for an interesting project to keep him occupied during Christmas. The programming language which Python is said to have succeeded is ABC Programming Language, which had the interfacing with the Amoeba Operating System and had the feature of exception handling. He had already helped to create ABC earlier in his career and he had seen some issues with ABC but liked most of the features. After that what he did as really very clever. He had taken the syntax of ABC, and some of its good features. It came with a lot of complaints too, so he fixed those issues completely and had created a good scripting language which had removed all the flaws. The inspiration for the name came from BBC’s TV Show – ‘Monty Python’s Flying Circus’. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C. Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity.

**• Features of Python language and its importance**

**Python** is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming.  
In Python, we don’t need to declare the type of variable because it is a dynamic typed language.  
For example, x=10  
here x can be anything such as String, int etc.

**1.Easy to code:**  
Python is high level programming language.Python is very easy to learn language as compared to other language like c, c#, java script, java etc.It is very easy to code in python language and anybody can learn python basic in few hours or days.It is also developer-friendly language.

**2. Free and Open Source:**  
Python language is freely available. Since, it is open-source, this means that source code is also available to the public.So you can download it as, use it as well as share it.

**3.Object-Oriented Language:**  
One of the key features of python is Object-Oriented programming.Python supports object oriented language and concepts of classes, objects encapsulation etc

**4. GUI Programming Support:**  
Graphical Users interfaces can be made using a module such as PyQt5, PyQt4, wxPython or Tk in python.  
PyQt5 is the most popular option for creating graphical apps with Python.

**5. High-Level Language:**  
Python is a high-level language.When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

**6.Extensible feature:**  
Python is a **Extensible** language.we can write our some python code into c or c++ language and also we can compile that code in c/c++ language.

**7. Python is Portable language:**  
Python language is also a portable language.for example, if we have python code for windows and if we want to run this code on other platform such as Linux, Unix and Mac then we do not need to change it, we can run this code on any platform.

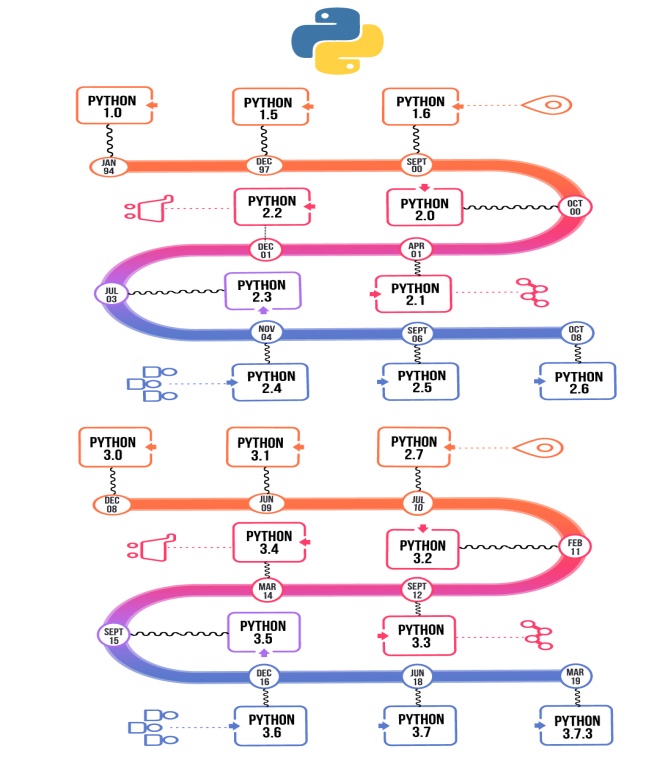
**8. Python is Integrated language:**  
Python is also an Integrated language because we can easily integrated python with other language like c, c++ etc.

**9. Interpreted Language:**  
Python is an Interpreted Language. because python code is executed line by line at a time. like other language c, c++, java etc there is no need to compile python code this makes it easier to debug our code.The source code of python is converted into an immediate form called **bytecode**.

**10. Large Standard Library**  
Python has a large standard library which provides rich set of module and functions so you do not have to write your own code for every single thing.There are many libraries present in python for such as regular expressions, unit-testing, web browsers etc.

**11. Dynamically Typed Language:**  
Python is dynamically-typed language. That means the type (for example- int, double, long etc) for a variable is decided at run time not in advance.because of this feature we don’t need to specify the type of variable.

**• Versions of Python**



**• First application in Python**

print("Hello, world!")

Congratulations! You just wrote your first program in Python.

As you can see, this was a pretty easy task. This is the beauty of Python programming language.

**• Data types in Python**

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

## Python Numbers

Integers, floating point numbers and complex numbers fall under [Python numbers](https://www.programiz.com/python-programming/numbers) category. They are defined as int, float and complex classes in Python.

We can use the type() function to know which class a variable or a value belongs to. Similarly, the isinstance() function is used to check if an object belongs to a particular class.

a = 5

print(a, "is of type", type(a))

a = 2.0

print(a, "is of type", type(a))

a = 1+2j

print(a, "is complex number?", isinstance(1+2j,complex))

Run Code

**Output**

5 is of type <class 'int'>

2.0 is of type <class 'float'>

(1+2j) is complex number? True

Integers can be of any length, it is only limited by the memory available.

A floating-point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is an integer, 1.0 is a floating-point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

>>> a = 1234567890123456789

>>> a

1234567890123456789

>>> b = 0.1234567890123456789

>>> b

0.12345678901234568

>>> c = 1+2j

>>> c

(1+2j)

**Python List**

[List](https://www.programiz.com/python-programming/list) is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

a = [1, 2.2, 'python']

We can use the slicing operator [ ] to extract an item or a range of items from a list. The index starts from 0 in Python.

a = [5,10,15,20,25,30,35,40]

# a[2] = 15

print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]

print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]

print("a[5:] = ", a[5:])

**Output**

a[2] = 15

a[0:3] = [5, 10, 15]

a[5:] = [30, 35, 40]

Lists are mutable, meaning, the value of elements of a list can be altered.

a = [1, 2, 3]

a[2] = 4

print(a)

**Output**

[1, 2, 4]

**Python Tuple**

[Tuple](https://www.programiz.com/python-programming/tuple) is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

t = (5,'program', 1+3j)

We can use the slicing operator [] to extract items but we cannot change its value.

t = (5,'program', 1+3j)

# t[1] = 'program'

print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))

print("t[0:3] = ", t[0:3])

# Generates error

# Tuples are immutable

t[0] = 10

Run Code

**Output**

t[1] = program

t[0:3] = (5, 'program', (1+3j))

Traceback (most recent call last):

File "test.py", line 11, in <module>

t[0] = 10

TypeError: 'tuple' object does not support item assignment

**Python Strings**

[String](https://www.programiz.com/python-programming/string) is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

s = "This is a string"

print(s)

s = '''A multiline

string'''

print(s)

Run Code

**Output**

This is a string

A multiline

string

Just like a list and tuple, the slicing operator [ ] can be used with strings. Strings, however, are immutable.

s = 'Hello world!'

# s[4] = 'o'

print("s[4] = ", s[4])

# s[6:11] = 'world'

print("s[6:11] = ", s[6:11])

# Generates error

# Strings are immutable in Python

s[5] ='d'

Run Code

**Output**

s[4] = o

s[6:11] = world

Traceback (most recent call last):

File "<string>", line 11, in <module>

TypeError: 'str' object does not support item assignment

**Python Set**

[Set](https://www.programiz.com/python-programming/set) is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

a = {5,2,3,1,4}

# printing set variable

print("a = ", a)

# data type of variable a

print(type(a))

Run Code

**Output**

a = {1, 2, 3, 4, 5}

<class 'set'>

We can perform set operations like union, intersection on two sets. Sets have unique values. They eliminate duplicates.

a = {1,2,2,3,3,3}

print(a)

Run Code

**Output**

{1, 2, 3}

Since, set are unordered collection, indexing has no meaning. Hence, the slicing operator [] does not work.

>>> a = {1,2,3}

>>> a[1]

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

TypeError: 'set' object does not support indexing

**Python Dictionary**

[Dictionary](https://www.programiz.com/python-programming/dictionary) is an unordered collection of key-value pairs.

It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

>>> d = {1:'value','key':2}

>>> type(d)

<class 'dict'>

We use key to retrieve the respective value. But not the other way around.

d = {1:'value','key':2}

print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error

print("d[2] = ", d[2]);

Run Code

**Output**

<class 'dict'>

d[1] = value

d['key'] = 2

Traceback (most recent call last):

File "<string>", line 9, in <module>

KeyError: 2

**Conversion between data types**

We can convert between different data types by using different type conversion functions like int(), float(), str(), etc.

>>> float(5)

5.0

Conversion from float to int will truncate the value (make it closer to zero).

>>> int(10.6)

10

>>> int(-10.6)

-10

Conversion to and from string must contain compatible values.

>>> float('2.5')

2.5

>>> str(25)

'25'

>>> int('1p')

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '1p'

We can even convert one sequence to another.

>>> set([1,2,3])

{1, 2, 3}

>>> tuple({5,6,7})

(5, 6, 7)

>>> list('hello')

['h', 'e', 'l', 'l', 'o']

To convert to dictionary, each element must be a pair:

>>> dict([[1,2],[3,4]])

{1: 2, 3: 4}

>>> dict([(3,26),(4,44)])

{3: 26, 4: 44}

**• Variables and its types**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

## Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example −

[Live Demo](http://tpcg.io/Eh9FoM)

#!/usr/bin/python

counter = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

print counter

print miles

print name

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result −

100

1000.0

John

## Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −

a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −

a,b,c = 1,2,"john"

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

**• Operators and its types**

## What are operators in python?

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example:

>>> 2+3

5

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

## Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| + | Add two operands or unary plus | x + y+ 2 |
| - | Subtract right operand from the left or unary minus | x - y- 2 |
| \* | Multiply two operands | x \* y |
| / | Divide left operand by the right one (always results into float) | x / y |
| % | Modulus - remainder of the division of left operand by the right | x % y (remainder of x/y) |
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y |
| \*\* | Exponent - left operand raised to the power of right | x\*\*y (x to the power y) |

### Example 1: Arithmetic operators in Python

x = 15

y = 4

# Output: x + y = 19

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

# Output: x - y = 11

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

# Output: x \* y = 60

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

# Output: x / y = 3.75

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

# Output: x // y = 3

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

# Output: x \*\* y = 50625

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

Run Code

**Output**

x + y = 19

x - y = 11

x \* y = 60

x / y = 3.75

x // y = 3

x \*\* y = 50625

## Comparison operators

Comparison operators are used to compare values. It returns either True or False according to the condition.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| > | Greater than - True if left operand is greater than the right | x > y |
| < | Less than - True if left operand is less than the right | x < y |
| == | Equal to - True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to - True if left operand is less than or equal to the right | x <= y |

### Example 2: Comparison operators in Python

x = 10

y = 12

# Output: x > y is False

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

# Output: x < y is True

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

# Output: x == y is False

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

# Output: x != y is True

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

# Output: x >= y is False

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

# Output: x <= y is True

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

Run Code

**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

## Logical operators

Logical operators are the and, or, not operators.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false (complements the operand) | not x |

### Example 3: Logical Operators in Python

x = True

y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)

Run Code

**Output**

x and y is False

x or y is True

not x is False

Here is the [truth table](https://www.programiz.com/python-programming/keyword-list#and_or_not) for these operators.

## Bitwise operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

**In the table below:** Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| & | Bitwise AND | x & y = 0 (0000 0000) |
| | | Bitwise OR | x | y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x >> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x << 2 = 40 (0010 1000) |

## Assignment operators

Assignment operators are used in Python to assign values to variables.

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

|  |  |  |
| --- | --- | --- |
| Operator | Example | Equivalent to |
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| \*= | x \*= 5 | x = x \* 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| \*\*= | x \*\*= 5 | x = x \*\* 5 |
| &= | x &= 5 | x = x & 5 |
| |= | x |= 5 | x = x | 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |

## Special operators

Python language offers some special types of operators like the identity operator or the membership operator. They are described below with examples.

### Identity operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |

### Example 4: Identity operators in Python

x1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

# Output: False

print(x1 is not y1)

# Output: True

print(x2 is y2)

# Output: False

print(x3 is y3)

Run Code

**Output**

False

True

False

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

### Membership operators

in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence ([string](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set) and [dictionary](https://www.programiz.com/python-programming/dictionary)).

In a dictionary we can only test for presence of key, not the value.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| in | True if value/variable is found in the sequence | 5 in x |
| not in | True if value/variable is not found in the sequence | 5 not in x |

### Example #5: Membership operators in Python

x = 'Hello world'

y = {1:'a',2:'b'}

# Output: True

print('H' in x)

# Output: True

print('hello' not in x)

# Output: True

print(1 in y)

# Output: False

print('a' in y)

Run Code

**Output**

True

True

True

False

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

**• Bitwise operators and its working**

**• Memory allocation strategy used by Python**

**• Numbers in Python**

**• Input Output mechanisms**

**• Command line arguments**

**• Procedural programming approach in Python**

**• Function definition and function calling techniques**

**• Function arguments and its types**

**• Inner function and its calling techniques**

**• Returning multiple values from function**

**• Anonymous function and its use**

**• Default function arguments**

**• Required function argument**

**• Variable number of argument of function**