Python Programming
Unit 1
(KNC-302)

Prepared By

Abhishek Kesharwani

Assistant Professor,UCER Naini,Allahabad

UNIT I

- Introduction of Python
- The Programming Cycle for Python
- Python IDE,
- Interacting with Python Programs,
- Elements of Python, Type Conversion
- Basics: Expressions,
- Assignment Statement,
- Arithmetic Operators
- Operator Precedence,
- Boolean Expression.

Python Introduction

- **Python** is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.
- Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.
- Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.
- Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

- Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.

Python Features

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5) Free and Open Source

Python language is freely available at offical web address. The source-code is also available. Therefore it is open source.

6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8) Large Standard Library

Python has a large and broad library and prvides rich set of module and functions for rapid application development.

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

Python Applications

1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing.It also provides Frameworks such as Django, Pyramid, Flask etc to design web based applications

2) Desktop GUI Applications

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

3) Software Development

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc

4) Scientific and Numeric

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications

Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application

We can use Python to develop console based applications. For example: **IPython**.

7) Audio or Video based Applications

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications

To create CAD application Fandango is a real application which provides full features of CAD.

9) Enterprise Applications

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

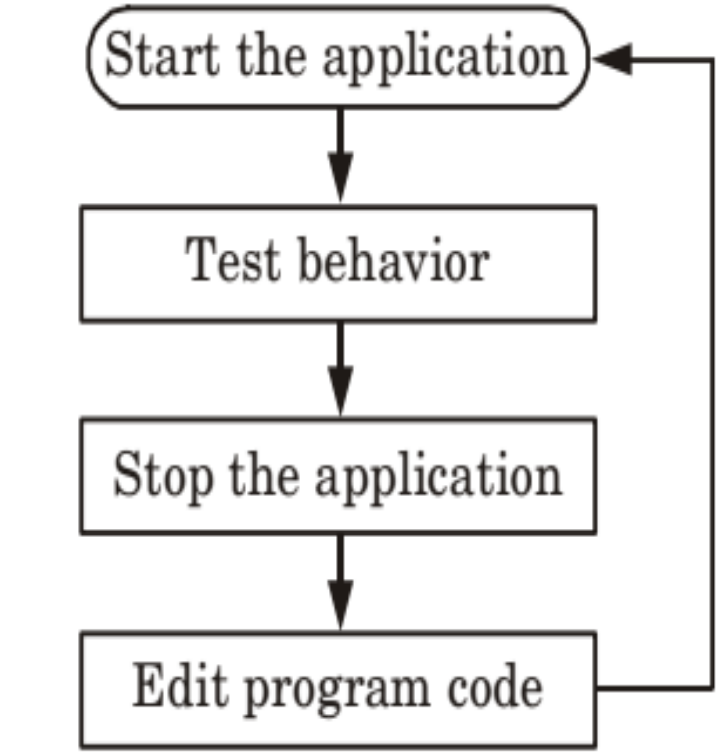
10) Applications for Images

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

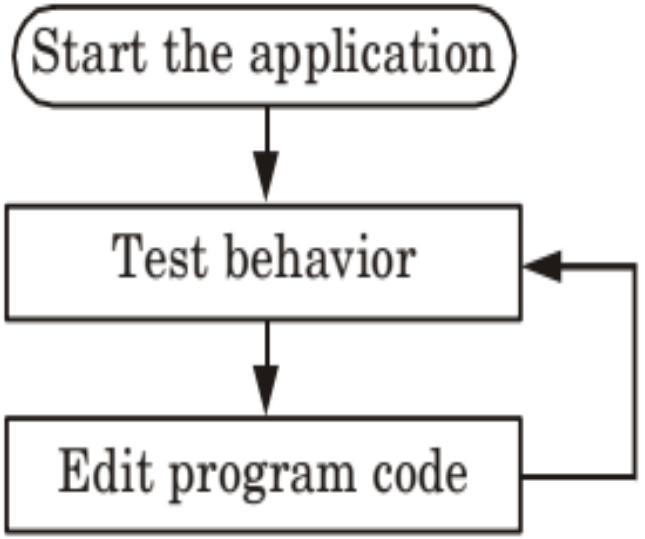
Programming cycle for Python

- Python's programming cycle is dramatically shorter than that of traditional programming cycle.
- 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.
- 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.
- Python's impact on the programming cycle is as follows:



(a) Python's programming cycle



(b) Python's programming cycle with module reloading

- Since Python is interpreted, there is a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it is easy to modify programs at runtime.

Integrated Development Environment

1. IDE is a software package that consists of several tools for developing and testing the software.
2. An IDE helps the developer by automating the process.
3. IDEs integrate many tools that are designed for SDLC.
4. IDEs were introduced to diminish the coding and typing errors.
5. Some of the Python IDEs are pycharm,

PyCharm

PyCharm assists the developers to be more productive and provides smart suggestions. It saves time by taking care of routine tasks, hence increases productivity.

Features of PyCharm

i. It has smart code navigation, good code editor, a function for quick refactoring.

ii. The integrated activities with PyCharm are profiling, testing, debugging, remote development, and deployments.

iii. PyCharm supports Python web development frameworks, Angular JS, JavaScript, CSS, HTML and live editing functions.

Spyder

Spyder is widely used for data science works. It is mostly used to create a secure and scientific environment for Python. Spyder Python uses PyQt (Python plug-in) which a developer can add as an extension.

Features of Spyder :

i. It has good syntax highlighting and auto code completion features.

ii. Spyder Python explores and edits variables directly from GUI.

iii. It performs very well in multi-language editor.

PyDev

It is an external plug-in for Eclipse and is very popular as Python interpreter.

Features of PyDev :

i. PyDev has strong parameters like refactoring, debugging, type hinting, code analysis, and code coverage function.

ii. PyDev supports tokens browser, PyLint integration, interactive console, remote debugger, Unittest integration, etc.

IDLE

IDLE is a basic IDE mainly used by beginner level developer.

i. IDLE Python is a cross-platform IDE, hence it increases the flexibility for users.

ii. It is developed only in Python in collaboration with Tkinter GUI toolkit.

iii. The feature of multi-window text editor in IDLE has some great functions like smart indentation, call tips, Python colorizing, and undo option.

Python Installation on Ubuntu

1) **Update the APT Repository**

$ apt-get update

2) **Install Python**

$ apt-get install python3.6

3) **Verify Python**

When we type **python** it shows default installed python that is 2.7.

4) Check python –version.

Type python - - version. It will show the current version of python.

Python Variables

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

Identifier Naming

- Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.
- The first character of the variable must be an alphabet or underscore (_).
- 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).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).

- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive for example my name, and MyName is not the same.

Examples of valid identifiers :

a123, _n, n_9, etc.

Examples of invalid identifiers:

1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

- Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.
- We don't need to declare explicitly variable in Python.
- When we assign any value to the variable that variable is declared automatically.

Multiple Assignment

- Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print x
print y
print z
```

2. Assigning multiple values to multiple variables

```
a,b,c=5,10,15
print a
print b
print c
```

Python Data Types

- Variables can hold values of different data types.
- Python is a dynamically typed language hence we need not define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.
- Python enables us to check the type of the variable used in the program.
- Python provides us the **type()** function which returns the type of the variable passed.

```
A=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values

Python provides various standard data types that define the storage method on each of them.

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

- Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;
- a = 3 , b = 5 #a and b are number objects

Python supports 3 types of numeric data.

- int (signed integers like 10, 2, 29, etc.)
- float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
- complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

String

- The string can be defined as the sequence of characters represented in the quotation marks.
- In python, we can use single, double, or triple quotes to define a string.
- In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.
- The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello Abhishek' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

List

- Lists are similar to arrays in C.
- However; the list can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list.
- The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Example

```
l  = [1, "hi", "python", 2]
print (l[3:])
print (l[0:2])
print (l)
print (l + l)
print (l * 3)
```

Output

- [2]
- [1, 'hi']
- [1, 'hi', 'python', 2]
- [1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
- [1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]

Tuple

A tuple is similar to the list in many ways.

- Like lists, tuples also contain the collection of the items of different data types.
- The items of the tuple are separated with a comma (,) and enclosed in parentheses ().
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Example

```
t  = ("hi", "python", 2)
print (t[1:])
print (t[0:1])
print (t)
print (t + t)
print (t * 3)
print (type(t))
t[2] = "hi"
```

output

- ('python', 2)
- ('hi',)
- ('hi', 'python', 2)
- ('hi', 'python', 2, 'hi', 'python', 2)
- ('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
- <type 'tuple'>
- TypeError: 'tuple' object does not support item assignment

Dictionary

- Dictionary is an ordered set of a key-value pair of items.
- It is like an associative array or a hash table where each key stores a specific value.
- Key can hold any primitive data type whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Example

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
print("1st name is "+d[1]);
print(“4th name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

Output

- 1st name is Jimmy
- 4th name is mike
- {1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
- [1, 2, 3, 4]
- ['Jimmy', 'Alex', 'john', 'mike']

Python Keywords

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

Python Literals

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

I. String literals

II. Numeric literals

III. Boolean literals

IV. Special literals.

I. String literals

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Eg:"Aman" , '12345'

Types of Strings:

There are two types of Strings supported in Python:

a).Single line String- Strings that are terminated within a single line are known as Single line Strings.

Eg: text1='hello'

b).Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

1). Adding black slash at the end of each line.

Eg:

```
>>> text1='hello\
user'
>>> text1
'hellouser'
>>>
```

2).Using triple quotation marks:-

Eg:

```
>>> str2="""welcome
to
SSSIT"""
>>> print str2
welcome
to
SSSIT
>>>
```

II. Numeric literals:

Int(signed integers)

Numbers(can be both positive and negative) with no fractional part.eg: 100

float(floating point)

Real numbers with both integer and fractional part eg: -26.2

Complex(complex)

In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j

III. Boolean literals

A Boolean literal can have any of the two values: True or False.

IV. Special literals

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

V. Literal Collections.

Collections such as tuples, lists and Dictionary are used in Python.

Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands.

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

Arithmetic operators

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y

Examples of Arithmetic Operator

```
a = 9
b = 4
add = a + b
sub = a - b
mul = a * b
div1 = a / b
div2 = a // b
mod = a % b
# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
```

Output

- 13
- 5
- 36
- 2.25
- 2
- 1

Relational Operators

OPERATOR	DESCRIPTION	SYNTAX
>	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

```
# Examples of Relational Operators
a = 13
b = 33
# a > b is False
print(a > b)
# a < b is True
print(a < b)
# a == b is False
print(a == b)
# a != b is True
print(a != b)
# a >= b is False
print(a >= b)
# a <= b is True
print(a <= b)
```

Logical operators

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

Bitwise operators

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0
c = a & b;        # 12 = 0000 1100
print ("Line 1 - Value of c is ", c)

c = a | b;        # 61 = 0011 1101
print ("Line 2 - Value of c is ", c)

c = a ^ b;        # 49 = 0011 0001
print ("Line 3 - Value of c is ", c)

c = ~a;           # -61 = 1100 0011   -(n+1)
print ("Line 4 - Value of c is ", c)

c = a << 2;       # 240 = 1111 0000
print ("Line 5 - Value of c is ", c)

c = a >> 2;       # 15 = 0000 1111
print ("Line 6 - Value of c is ", c)
```

Assignment operators

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b

Special operators

There are some special type of operators like

- **Identity operators**
- **Membership operators**

Identity operators-

is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory.

```
a1 = 3
b1 = 3
a2 = ‘Hello'
b2 = ‘Hello'
a3 = [1,2,3]
b3 = [1,2,3]
print(a1 is not b1)  # False
print(a2 is b2)  # True
print(a3 is b3) # Output is False, since lists are mutable.
```

Membership operators-

in and **not in** are the membership operators used to test whether a value or variable is in a sequence.

in True if value is found in the sequence

not in True if value is not found in the sequence

```
x = 'Geeks for Geeks'
y = {3:'a',4:'b'}


print('G' in x)    # True

print('geeks' not in x) # True

print('Geeks' not in x) # False

print(3 in y) # True

print('b' in y)  # False
```

Python Comments

Comments in Python can be used to explain any program code. It can also be used to hide the code as well.

1) Single Line Comment:

In case user wants to specify a single line comment, then comment must start with #

Eg:

```
# This is single line comment.
```

2) Multi Line Comment:

Multi lined comment can be given inside triple quotes.

eg:

''' This

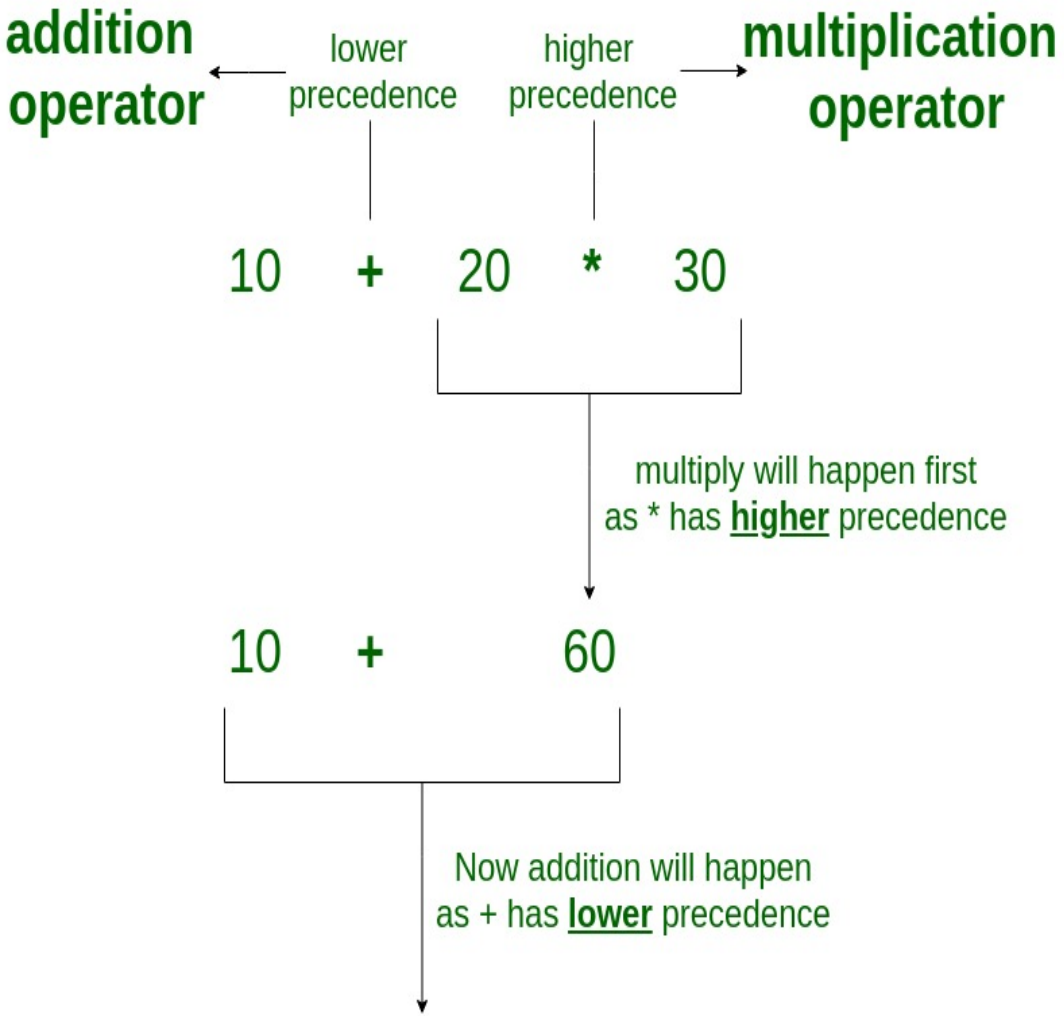
Is

Multipline comment'''

Operator Precedence

This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

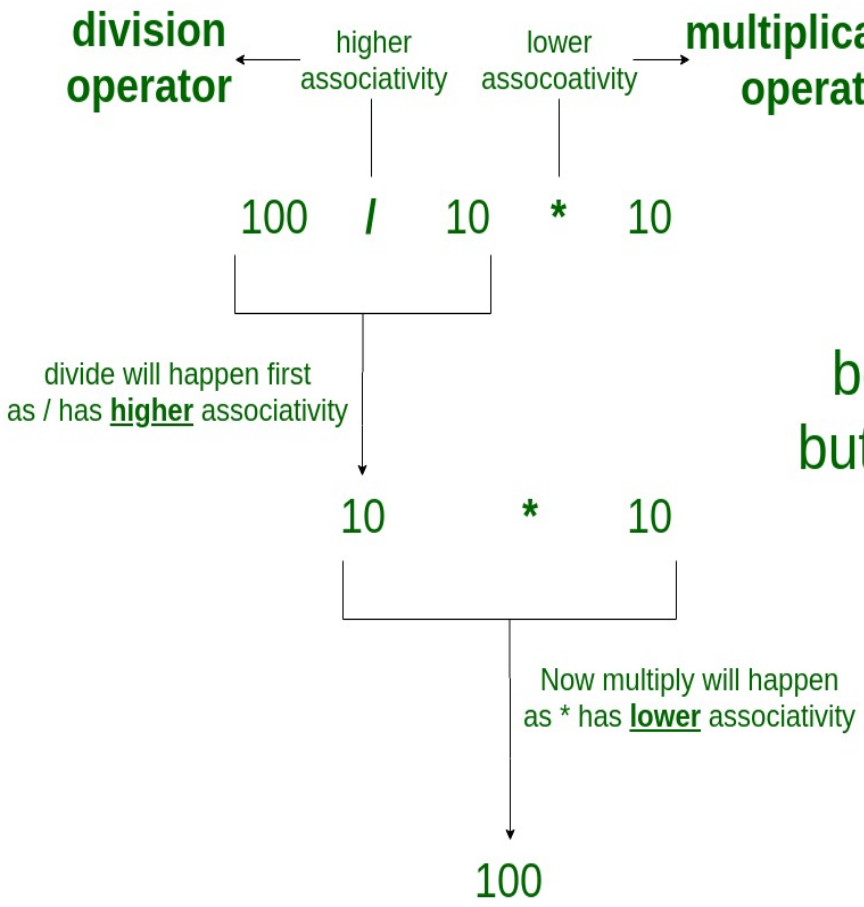
Operator Precedence



Operator Associativity

If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

Operator Associativity



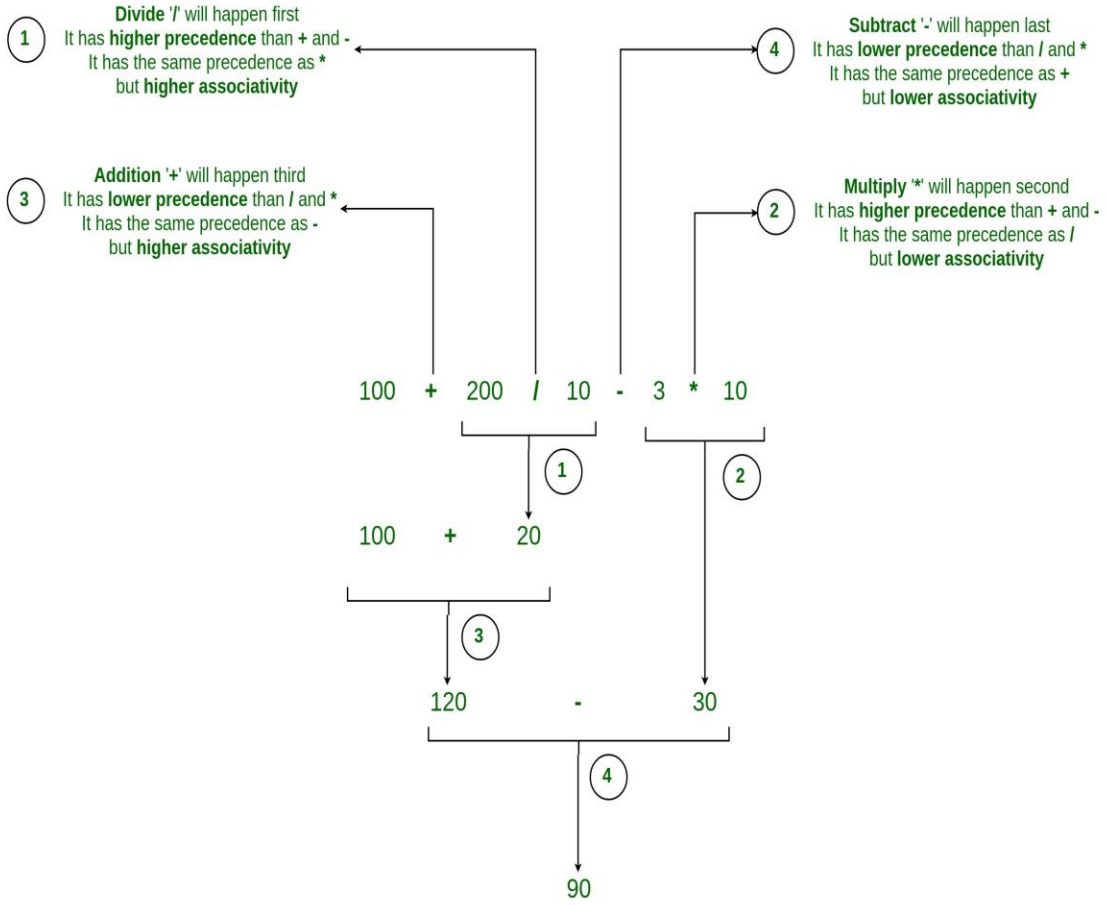
/ and ***
both have the same precedence
but Left to Right (**LTR**) associativity

Operators Precedence and Associativity are two main characteristics of operators that determine the evaluation order of sub-expressions in absence of brackets.

Evaluate

100 + 200 / 10 - 3 * 10

Operator Precedence and Associativity



/ and ***** both have the same precedence but Left to Right (**LTR**) associativity

+ and **-** both have the same precedence but Left to Right (**LTR**) associativity

/ and ***** have the higher precedence than **+** and **-**

Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right

== !=	Relational is equal to/is not equal to	left-to-right
is, is not in, not in	Identity Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
not	Logical NOT	right-to-left
and	Logical AND	left-to-right
or	Logical OR	left-to-right

= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left