PYTHON PROGRAMMING

Python is a high level programming language developed by Guido Van Rossum in late 1980s. He named it Python, taken from his favourite British comedy Series - Monty Python's Flying Circus.

Features of Python

-> high level language
-> interpreted
-> general purpose programming language (has many applications from simple web development to Artificial Intelligence)
-> multi-paradigm (both procedural and object oriented, supports functional and aspect oriented programming)
-> easy to code and read
-> robust
-> open source and free (free to download and anyone can access and modify python's code and contribute to its development)
-> highly portable (can run on a variety of computer platforms)
-> dynamically typed language (no need to specify data types)
-> large standard library

A program, written in a programming language, is a set of instructions by which the computer comes to know what is to be done. It is a coding language used by programmers to write the instructions that a computer can understand.
There are basically three types of computer languages

● High-level Language - uses English-like language. It is converted into machine level language using a converting software such as **compiler or interpreter.**

● Assembly Language - low-level programming language. It is machine friendly. Symbols and mnemonics such as 'add', 'sub', 'mul' etc are used to represent various machine language instructions

● Machine Language - only 0's and 1's because it is made of switches, transistors, and other electronic devices which can only be in the state of either **on or off**. The off state is represented by 0 and on state by 1. It is a low-level computer programming language and is more machine friendly.

Programmers have a variety of tools available to enhance the software development process. Some common tools include:

-> Editors. An editor allows the programmer to enter the program source code and save it to files.
-> Compilers. A compiler translates the source code to target code.
-> Interpreters. An interpreter is like a compiler, in that it translates higher-level source code into machine language. Interpreter translates just one statement of the program at a time into machine code. Compiler scans the entire program and translates the whole of it into machine code at once. **Interpreter keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy. A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.**

In general, compiled programs execute more quickly than interpreted programs because the translation activity occurs only once. Interpreted programs, on the other hand, can run as is on any platform with an appropriate interpreter; they do not need to be recompiled to run on a different platform.

-> Debuggers. A debugger allows programmers to simultaneously run a program and see which source code line is currently being executed, values of variables and other program elements can be watched to see if their values change as expected. Debuggers are valuable for locating errors(called bugs).
->Profilers. A profiler is used to evaluate a program's performance. It indicates how many times a portion of a program is executed during a particular run, and how long that portion takes to execute. Used in testing.

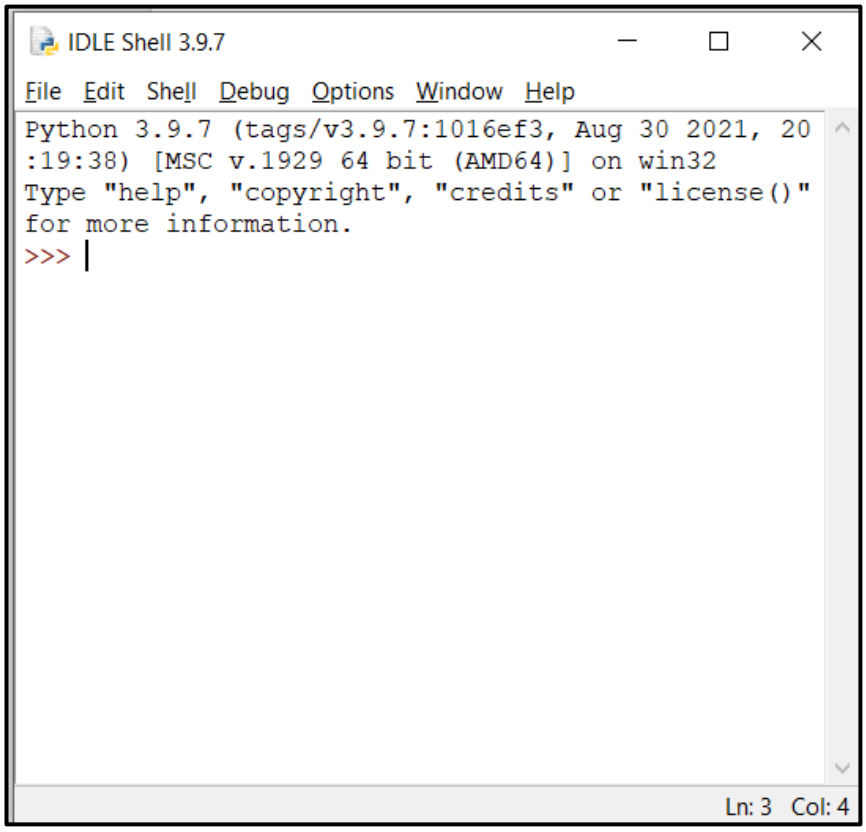
Integrated Development Environments (IDEs)
An IDE includes editors, debuggers, and other programming aids in one comprehensive program.

IDLE (Integrated Development and Learning Environment) is a simple Python integrated development environment available for Windows, Linux, and Mac OS X.

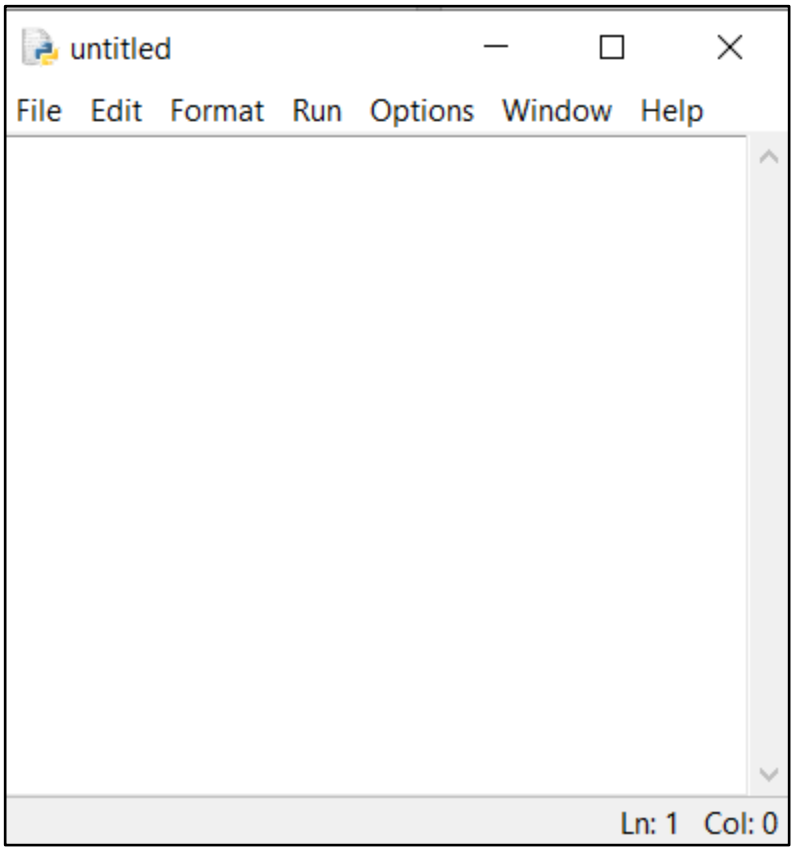
Other IDEs for pyhton include Eclipse-PyDev, PyCharm, Eric, NetBeans.

IDLE can be used to execute single statement. It also provides text editor to create, modify and execute python scripts. Text-editor has features such as syntax highlighting, auto-completion, smart indent. It also has debugger.

IDLE Menus
IDLE has two main window types : **Shell Window** and **Editor Window**

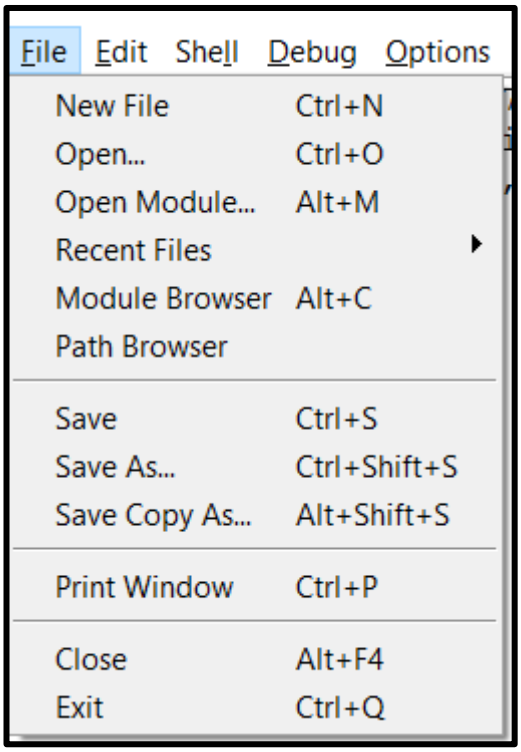


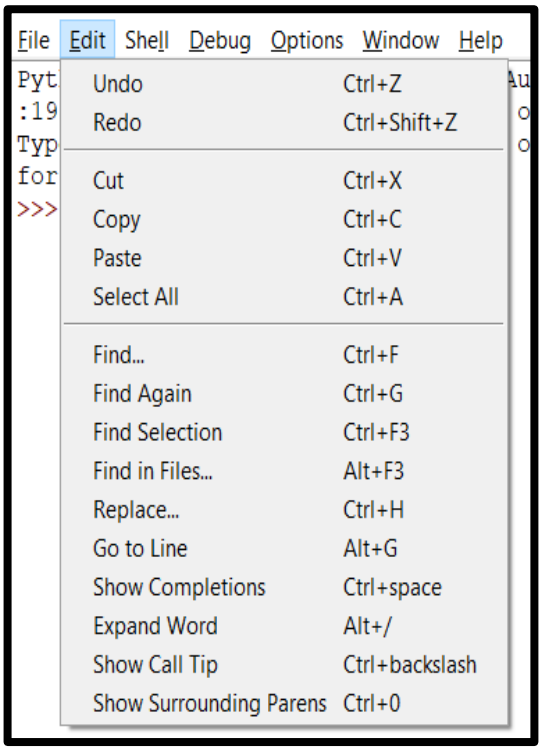
SHELL WINDOW

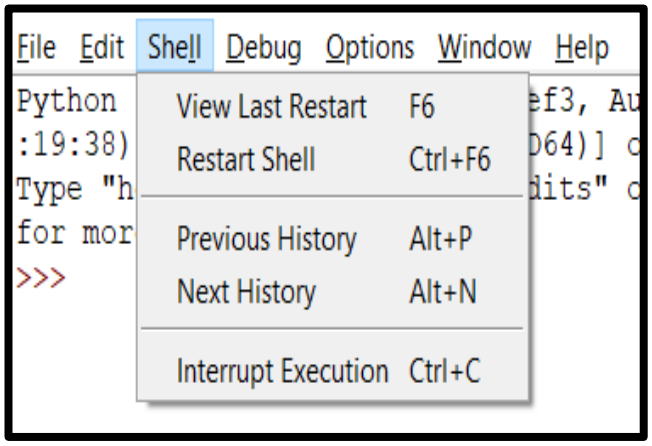


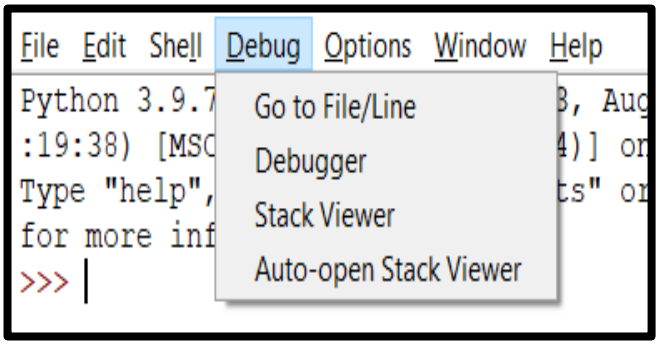
Editor Window

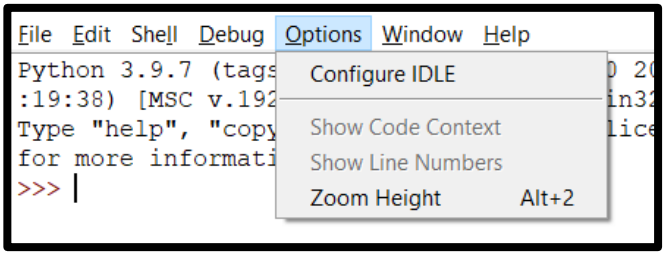
File menu : Available in both window
Edit Menu : Available in both window
Format Menu : Editor only
Run Menu : Editor only
Shell Menu : Shell only
Debug Menu : Shell only
Options : Available in both window
Help : Available in both window

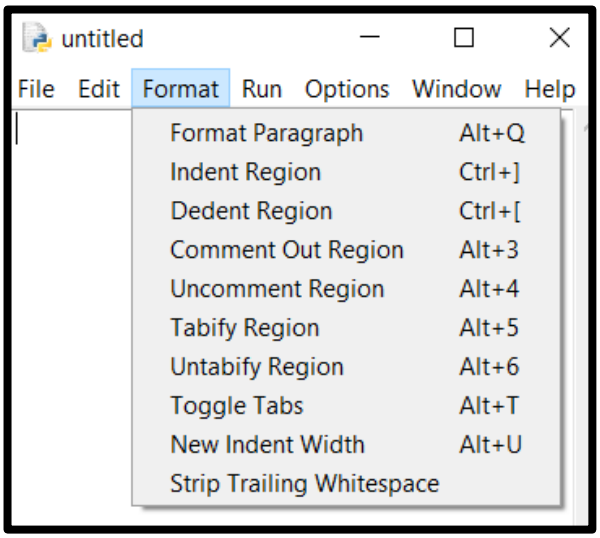


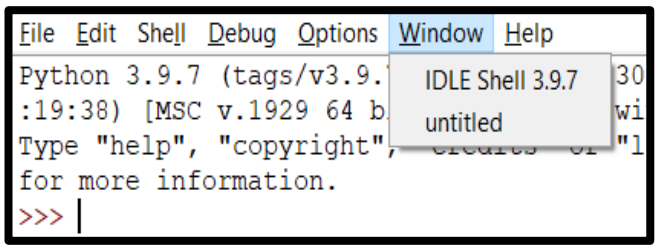


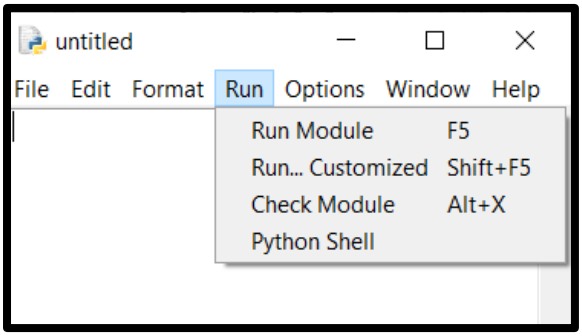

File Edit Shell Debug Options Window Help
View Last Restart F6
Restart Shell Ctrl+F6
Previous History Alt+P
Next History Alt+N
Interrupt Execution Ctrl+C
>>>

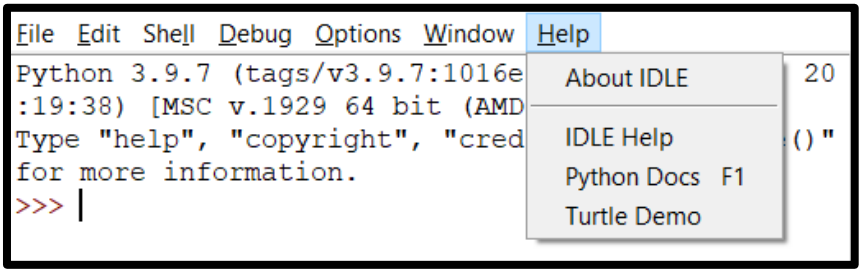

File Edit Shell Debug Options Window Help
Go to File/Line
Debugger
Stack Viewer
Auto-open Stack Viewer
>>>


File Edit Shell Debug Options Window Help
Configure IDLE
Show Code Context
Show Line Numbers
Zoom Height Alt+2
>>>


untitled
File Edit Format Run Options Window Help
Format Paragraph Alt+Q
Indent Region Ctrl+]
Dedent Region Ctrl+[
Comment Out Region Alt+3
Uncomment Region Alt+4
Tabify Region Alt+5
Untabify Region Alt+6
Toggle Tabs Alt+T
New Indent Width Alt+U
Strip Trailing Whitespace


File Edit Shell Debug Options Window Help
IDLE Shell 3.9.7
untitled
for more information.
>>>


untitled
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell


File Edit Shell Debug Options Window Help
About IDLE
IDLE Help
Python Docs F1
Turtle Demo
for more information.
>>>

PYTHON INTERPRETER

It is a software that converts source code written by the developer into intermediate language which is again translated into machine language.

Translates just one statement of the program at a time.

Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.

Can run as is on any platform with an appropriate interpreter; they do not need to be recompiled to run on a different platform.

A simple Python program

```
def sum(a,b):
    print("sum :",a+b)
sel='yes'
while sel=='yes':
    a=int(input("enter value for a"))
    b=int(input("enter value for b"))
    sum(a,b)
    sel=input("Do you want to continue?")
```

OUTPUT

enter value for a **2**
enter value for b **3**
sum : **5**
Do you want to continue? **yes**
enter value for a **5**
enter value for b **6**
sum : **11**
Do you want to continue? **no**

Keywords : **def, print, while, int, input**
Identifiers : sum, sel, a, b
Variables : sel, a, b

User defined Function :

```
def sum(a,b):
    print("sum : ",a+b)
```

Loop in the program :

```
while sel=='yes':
    a=int(input("enter value for a "))
    b=int(input("enter value for b "))
    sum(a,b)
    sel=input("Do you want to continue? ")
```

WRITING AND EXECUTING A PYTHON PROGRAM

We can write and execute python program in two ways :
1. enter the program directly into IDLE's interactive shell and
2. enter the program into IDLE's editor, save it, and run it.

IDLE's interactive shell : We can type one line Python program directly into IDLE and press enter to execute the program. Since it does not provide a way to save the code you enter, the interactive shell is not the best tool for writing larger programs. The IDLE interactive shell is useful for experimenting with small snippets of Python code.

IDLE's editor : IDLE has a built in editor. From the IDLE menu, select "New File", type the python program in the resulting editor window. We can save the program using the "Save" option in the File menu. Give a name to the program, the
extension ".py" is the extension used for Python source code. We can run the program from within the IDLE editor by pressing the F5 function key or from the editor's "Run menu: Run -> Run Module". The output appears in the IDLE interactive shell window.

COMMENTS

Comments are remarks that explain the purpose of a section of code or why the programmer chose to write a section of code the way he/she did. These notes are meant for human readers, not the interpreter.

A different programmer may be required to finish or fix a part of the program written by someone else. Well-written comments can help others understand new code quicker. Even the same programmer working on her own code months later can have a difficult time remembering what various parts do. Comments can help greatly.

The comment begins with the # symbol and continues until the end of that line. Any text contained within comments is ignored by the Python interpreter.

```
# Compute the average of the values
avg = sum / number
```

Multiline comments are written by inserting # at the beginning for each line. OR
SInce Python ignores strings that are not assigned to a variable, multiline strings can be used as multiline comments.

```
# Compute the
#average of the
#values
avg = sum / number
```

OR

```
" " "Compute the
average of the
values" " "
avg = sum / number
```

IDENTIFIERS

A Python identifier is a name given to program entities like variables, class, functions, etc. Names given to identifiers should be related to its purpose within the program. Good identifiers make programs more readable by humans.

- Identifiers must contain at least one character.
- The first character must be an alphabetic letter (upper or lower case) or the underscore
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_
- The remaining characters (if any) may be alphabetic characters (upper or lower case), the underscore, or a digit
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789
- No other characters (including spaces) are permitted in identifiers.
- A reserved word(keyword) cannot be used as an identifier

Python is a case-sensitive language. Identifiers also are case sensitive; the variable called "Name" is different from the variable called "name".

- All of the following words are valid identifiers and so can be used as variable names :
  x, x2, total, port_22, and FLAG.
- None of the following words are valid identifiers:
  sub-total (dash is not a legal symbol in an identifier),
  first entry (space is not a legal symbol in an identifier),
  4all (begins with a digit),
  #2 (pound sign is not a legal symbol in an identifier),
  class (class is a reserved word).

KEYWORDS

Python reserves a number of words for special use called "reserved words" or "keywords".

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

VARIABLES

A variable is an identifier that is associated with a value. A variable may be assigned and reassigned values as often as necessary. A "variable" is named so since its value can change often.

A variable is a way of referring to a memory location. It is a symbolic name for this physical location.

Declaration of a variable with its type is not required in Python. The declaration happens when a value is assigned to a variable. Equal sign (=) is used for assignment. Operand to the left of = is name of the variable, the one to the right is the value stored in the variable.

Example : x = 4

name = "John"

DATA TYPES

Data stored in memory can be of different types.

CATEGORY	DATA TYPES / CLASS NAMES
Text/String	str
Numeric	int, float, complex
List / Sequence	list, tuple, range
Map	dict
Set	set, frozenset
Boolean	bool
Binary	bytes, bytearray, memoryview

String :

- Strings are contiguous set of characters in quotation marks.

```
x = 'A'
y = "B"
z = """this
    is a multiline
    string"""
```

Numeric :

There are three numeric data types in Python :

- **int** – Integer whole number, positive or negative, without decimals
- **float** – floating point number, positive or negative, have one or more decimals
- **complex** – numbers having real and imaginary part(imaginary represented by **j**

```
x = 2
x = 2.5
x = 100+3j
```

Sequence :

- list is a collection , which is ordered and changeable.
  Example : randomList = [1, "one", 2, "two"]
- tuple is a collection which is ordered and unchangeable
  randomTuple = (1, "one", 2, "two")

Range in python is another in-built python datatype which is mainly used with loops in python. It returns a sequence of numbers specified in the function arguments.
x = range(6)

Mapping type :

- **dict** : dictionary
  It is a collection of unordered key-value pairs
  charsMap = {1:'a', 2:'b', 3:'c', 4:'d'}

Set :

- It includes **set** and **frozenset**
- **set** is a collection of changeable, unordered and unindexed items.
  digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- **frozenset** is a set which is unchangeable.
  frozenSetOfDigits = frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})

Boolean :

- Boolean represents two values **True**, **False**. The datatype is given as **bool**.
  x = True
  y = False

Binary:

- It includes **bytes, bytearray, memoryview**
- **bytes** hold array of bytes and is mutable(unchangeable)

```
x= bytes(4)
    x=b'\x00\x00\x00\x00'

   l=[1,2,3,4]
   x=bytes(l)
   x=b'\x01\x02\x03\x04'
```

- **bytearray** hold array of bytes and is immutable(changeable)

```
x= bytearray(4)
    x=b'\x00\x00\x00\x00'

   l=[1,2,3,4]
   x=bytearray(l)
   x=b'\x01\x02\x03\x04'
```

- The memoryview() method returns a memory view object of the given object.
  >>>list=[11,12]
  >>>bytearray(list)
  bytearray(b'\x0b\x0c')
  >>>memoryview(bytearray(list))
  <memory at 0x000001C8308A0280>

Operators in Python

Operators are symbols used to perform operations on **operands**. Operands are the variables and values on which operators are applied. Example : a+2, **a** and **2** are operands, **+** is the operator.

The operators available in Python are :

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

Arithmetic Operators

Arithmetic Operators are the type of operators which take numerical values as their operands and return a single numerical value.

Let's take two variables a and b having values 3 and 2 respectively.

Operator	Description	Example
+	Adds operands	a + b = 5
-	Subtracts right operand from left operand	a - b = 1
*	Multiplies both operands	a * b = 6
/	Quotient of division of left operand by right operand	a / b = 1.5 (float)
//	Quotient of division of left operand by right	a // b = 1 (int)
%	Remainder of division of left operand by right operand	a % b = 1
**	Left operand raised to the power of right operand	a ** b = 9

Both / and // operators divide the operands and return the quotient. The difference is that / returns the quotient as it is while // returns the quotient by truncating its fractional part and returning only the integer part.

Relational Operators

Relational Operators check the relationship between two operands. They return True if the relationship is true and False if it is false.

Again, take two variables a and b having values 3 and 2 respectively.

continued……

Operator	Description	Example
==	Equal to	(a == b) is False
!=	Not equal to	(a != b) is True
>	Greater than	(a > b) is True
<	Less than	(a < b) is False
>=	Greater than or equal to	(a >= b) is True
<=	Less than or equal to	(a <= b) is False

Although = and == seem to be the same, they are quite different from each other. = is the **assignment operator** while **==** is the **equality operator**.
= assigns values from its right side operands to its left side operands whereas == compares values.

x = 10
print(x == 10)

By writing x = 10, we assigned a value *10* to x, whereas by writing x == 10, we checked if the value of x is equal to *10* or not.

Logical Operators

Operator	Description	Example
and	If both the operands are True, then the condition becomes True	(a and b) is False
or	If any one or both the operands are True, then the condition becomes True	(a or b) is True
not	It is used to reverse the condition. So, if a condition is True, not makes it False and vice versa.	not(a) is False, not(b) is True

Look at the following table in which Exp1 and Exp2 are the two operands.

Exp1	Operator	Exp1	Output (Boolean)
True	and	True	True
True	and	False	False
False	and	False	False
True	or	True	True
True	or	false	True
False	or	False	False

Example:

x = 10
y = 20
print(x == 10 and y == 20)
print(x == 3 or y == 20)

Output
True
True

Assignment Operators

Assignment Operators are used to assign values from its **right side operands to its left side operands**.

Operator	Description	Example
=	Assigns value of right operand to left operand	C = A + B
+=	Adds the value of right operand to left operand and assigns the final value to the left operand	C += A is same as C = C + A
-=	Subtracts the value of right operand from left operand and assigns the final value to left operand	C -= A is same as C = C - A
*=	Multiplies the value of right operand to left operand and assigns the final value to left operand	C *= A is same as C = C * A
/=	Divides the value of left operand by right operand and assigns the quotient to left operand	C /= A is same as C = C / A
//=	Divides the value of left operand by right operand and assigns the quotient (integer) to left operand	C //= A is same as C = C // A
%=	Takes modulus using two operands and assigns the result to left operand	C %= A is same as C = C % A
**=	Calculates left operand raised to the power right operand and assigns the result to the left operand	C **= A is same as C = C ** A

Consider the value of a variable **n** as 5. Now if we write n += 2, the expression gets evaluated as **n = n + 2** thus making the value of n as 7 (n = 5 + 2).

Identity Operator

These are used to check if two operands (values) are located in the same memory. Note that two variables having the same value does not necessarily mean that their values are stored in the same memory location.

Operator	Description
is	Returns True if the operands refer to the same object. Otherwise returns False.
is not	Returns True if the operands do not refer to the same object. Otherwise returns False.

```
a = [1, 2, 3]
b = [1, 2, 3]
c = 10
d = 10
print(a is b)
print(c is d)
```

Output
False
True

Since, **c** and **d** have same value 10, only a memory location is reserved to store 10. Both **c** and **d** are tagged to this value only. Therefore, (**c is d)** gives True. But it is not the case of lists, they have different memory locations.

Membership Operators

These are used to check if a value is present in a sequence like string, list, tuple, etc.

Operator	Description
in	Returns True if the value is present in the sequence. Otherwise returns False.
not in	Returns True if the value is not present in the sequence. Otherwise returns False.

```
a = "Python programming"
print('on' in a)
print('p' not in a)
```

Output
True
False

Bitwise Operators

Bitwise Operators are used to performing operations on binary patterns *(1s and 0s).*

OperatorName		Description
`a & b`	Bitwise AND	Bits defined in both `a` and `b`
`a | b`	Bitwise OR	Bits defined in `a` or `b` or both
`a ^ b`	Bitwise XOR	Bits defined in `a` or `b` but not both
`a << b`	Bit shift left	Shift bits of `a` left by `b` units
`a >> b`	Bit shift right	Shift bits of `a` right by `b` units
`~a`	Bitwise NOT	Bitwise negation of `a`

The operands are given in decimal form even though we are performing bitwise operation. The result is also shown in decimal form. To denote binary in input and to get the binary output, we need to mention the ***"bin "*** keyword.

For example, **bin(2)** gives **'0b10'**

To represent a binary number, we prefix it with '0b'.

```
print( bin(0b10 & 0b100) )
```

output : **'0b0'**

which is equal to :

```
print(2 & 4)
```

output : **0**

2 -> 10
4 -> 100
Then, 010 & 100 gives :
0&1 = 0
1&0 = 0
0&0 = 0

Therefore, 010 & 100 = 000
That is, 2 & 4 = 0

Other examples :

print(2|4)

6

print(2 ^ 4)

6

The *<< (Bitwise left shift)* operator, as its name suggests, shifts the bits towards the left to a number represented to the right side of this operator.

print(2<<1)

4

2<<1 in binary format :
10 << 1
100
(each bit is moved by one place to left, hence the space vacated is

The *>> (right-shift)* operator, as its name suggests, shift the bits towards the right to a number represented to the right side of the operator.

print(2 >> 1)

1

Binary of 2 is 10,
10 >> 1 gives 01
That is, 2>>1 gives 1
(0 is shifted to right and hence is lost. 1 is shifted to right and occupies position of 0. The place vacated by 1 is filled with 0.)

The *~ (NOT)* operator is a very simple operator and works just as its name suggests. Additionally, it flips the bit from 0 to 1 and from 1 to 0. But *when used in programming like Python, this operator is used for returning the complement of the number.*

Example : **print(~2)** gives **-3**.

2 -> 01
Complement of **01** gives **10.**
Now, **10** is the representation of **-3** in 2's complement form.

To understand the concept easily, you should know that python returns the 2's complement of the number. 2's complement of the number is **–(num + 1)**.

That is,

~2 is –(2+1) which is -3.

~5 is -6

~8 is -9

Operator Precedence and Associativity

Precedence of operators decides the order in which operators in an expression are evaluated. For example, since multiplication have precedence over addition, the expression **a+b*c** is evaluated in the order **b*c** first, and then result is added to **a**.

If you want to add **a+b** first, and then multiply the result by **c**, we can do so by specifying the expression as **(a+b)*c**. This is because parenthesis() has the highest priority.

Associativity is the order in which Python evaluates an expression containing multiple operators of same precedence in an expression. For example, in the expression **4/2*2**, since both operators have the same precedence, it is evaluated from left to right.
4/2*2 gives 4
2*2/4 gives 1

The precedence(highest to lowest) and associativity of all operators are listed below :

Operator Precedence	Description	Associativity
()	Parenthesis	Left to right
**	Exponent	Right to left
~	Bitwise NOT	Left to right
*, /, %, //	Multiplication, Division, Modulo, Floor Division	Left to right
+, -	Addition, Subtraction	Left to right
>>, <<	Bitwise right and left shift	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
==, !=, >, <, >=, <=	Comparison	Left to right
=, +=, -=, *=, /=, %=, **=, //=	Assignment	Right to left
is, is not	Identity	Left to right
in, not in	Membership	Left to right
and, or, not	Logical	Left to right

Statement

A **statement** is a unit of code that the Python interpreter can execute. Different statements in Python include :
Assignment statement
return statement
break statement
for statement
while statement
if statement
continue statement etc.

A **Python Script** is a file containing statements organized in such a manner to be executed like a program.

- Statements can be extended to multiple lines using parentheses (), braces{ }, square bracket [] or continuation character slash (\).
  Example : var = 1+2+ \
  3+4+ \
  5+6
- Multiple statements can be incorporated in a single line using semi-colon(;).
  Example : a=3; print(a); c=a+1; print(c)

A set of statements grouped together to serve a purpose is called **block.** In Python, **indentation** is used to specify any block. All statements with same distance from left belong to the same block.

Expression

An **expression** can be defined as a combination of values, variables, operators and even call to functions that eventually evaluates to a value. A single value or variable is the simplest form of expression.
Example : a+3
4+3*5
a
3
are all expressions since they represent a value.

- **Arithmetic Expressions :** Expressions that evaluate to a numeric type. [3+4*2]
- **Boolean Expression :** Expressions that evaluate to Boolean value, True or False. [var = (a==c)]

User Inputs

The **input()** function is used to get an input value from the user which will be assigned to any variable.
Syntax : **input([prompt])**

- *prompt* is an optional string which represents a message that will be printed before user can give an input.
- The typing for input is terminated by **enter key**.
- The function converts all input values into a **string**.

Examples :

a = input('' Enter a value for the variable a '')

b = int(input (''Enter an integer number''))

In the second example, we used typecasting to get an integer value so that it can be used in arithmetic operations. This is done because input() function can only store all kind of values as strings.

type()

Syntax : **type(object)**

type(object, base, dict)

- If the first format is used, the function returns the type of given object.
  Example : **a=10**
  print(type(a))

  Output : **<class 'int'>**
- If second format is used, the function returns a new type object. The arguments ***object*** will be the class name, ***bases*** will be the tuple that defines the base class, ***dict*** contains definition for the body of class which is represented in a dictionary format.

eval()

Syntax : **eval(expression, globals, locals)**

This function is used to evaluate a given expression which is specified inside double quotes. Hence, **it represents an expression in terms of a string, evaluates the expression and returns the result. The type of return value is based on the result of the expression.**

Example : **c=eval("2*3.5")**
d=eval("2*5")

Here, the type of **c** will be **float** whereas type of **d** will be **int**.

- **globals** and **locals** arguments are optional. If not given, the current global and local namespaces are used.
- When custom value is give, **globals** represent a dictionary that provides a global namespace to eval() and **locals** represent dictionary that contains the variables that eval() uses as local names when evaluating expression.

print()

Syntax : **print(objects, sep=' ', end='\n', file=sys.stdout, flush=False)**

The print function prints the given message inside parenthesis into any standard output device such as monitor screen.

- The ***object*** argument specifies any objects such as values, strings etc., which is converted to string and is printed in the screen. As many objects as we like may be given.
- ***sep*** specifies the separator that is used between objects to separate them. Default value is a space ' '.
- ***end*** specifies what to print at the end of a print statement. Default value is '\n' which is a new line character.

- ***file*** specifies where to write the printing statement, default is sys.stdout which is the standard output. We can use this argument to indicate a file that was open in write or append mode, so that messages go straight to it.
- ***flush*** represents a Boolean value. If it is True, the output is flushed and if it is False, the output is buffered.

Example :

print("This","is","a","sample",sep='--',end=';',flush=False)
print("second print function")

OUTPUT

This--is--a--sample;second print function

Escape Sequences

Escape sequences allow you to include special characters in strings. To do this, simply add a backslash (\) before the character you want to escape.

For example ;

s = 'Hey, what's up?
print(s)

gives error since the encounter of second single quote is considered as the end of string.

To print the same we use the escape sequence as :

s = 'Hey, what\'s up?'
print(s)

OUTPUT

Hey, what's up?

To add newlines to your string, use \n:

print("Multiline strings \n can be created \n using escape sequences.")

OUTPUT

Multiline strings
can be created
using escape sequences.