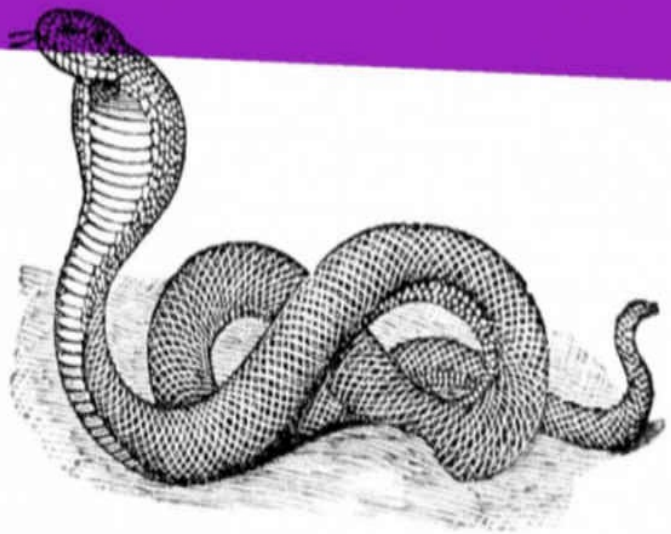
FOR BE

JAVASCR
CODIN
BASIC

FOR ABSOLUTE BEGINNERS

PYTHON BASICS



BY
TAM SEL

PYTHON AND JAVASCRIPT CODING BASICS FOR ABSOLUTE BEGINNERS

BY

TAM SEL

LOGICAL OPERATORS ON STRING

IF ELSE CONDITIONAL OPERATOR

JAVASCRIPT

JAVASCRIPT EXAMPLE

JAVASCRIPT COMMENT

JAVASCRIPT VARIABLE

JAVASCRIPT DATA TYPES

JAVASCRIPT IF ELSE

JAVASCRIPT SWITCH STATEMENT

JAVASCRIPT FUNCTION

JAVASCRIPT OBJECT

JAVASCRIPT ARRAY

JAVASCRIPT STRING

JAVASCRIPT DATE

JAVASCRIPT MATH

JAVASCRIPT NUMBER

PYTHON

CODING BASICS

FOR ABSOLUTE BEGINNERS

BY

TAM SEL

Introduction to Python

sailent features of Python

It is a programming language of general intent that can be used for both scientific and non-scientific programming.

For beginners, it is excellent as the language is translated, thereby producing immediate results.

It incorporates better than the other languages of its time the principle of exception handling and dynamic binding.

Easily readable and understandable are the programmes written in Python.

It is a programming language independent of a platform.

For customizable applications, it is suitable as an extension language.

Learning and using it is simple.

Major Applications

Google search engine , Twitter, etc.

Using Python, Bit Torrent peer to peer file sharing is written.

Python for hardware testing is used by Intel, Cisco, HP , IBM, etc.

A Python scripting API is provided by Maya.

For the creation of industrial robots, Python Used

For their scientific programming activities, NASA and others use Python.

Setting up Python Interpreter

In order to write and run a Python programme, we need to instal a Python interpreter on our computer.

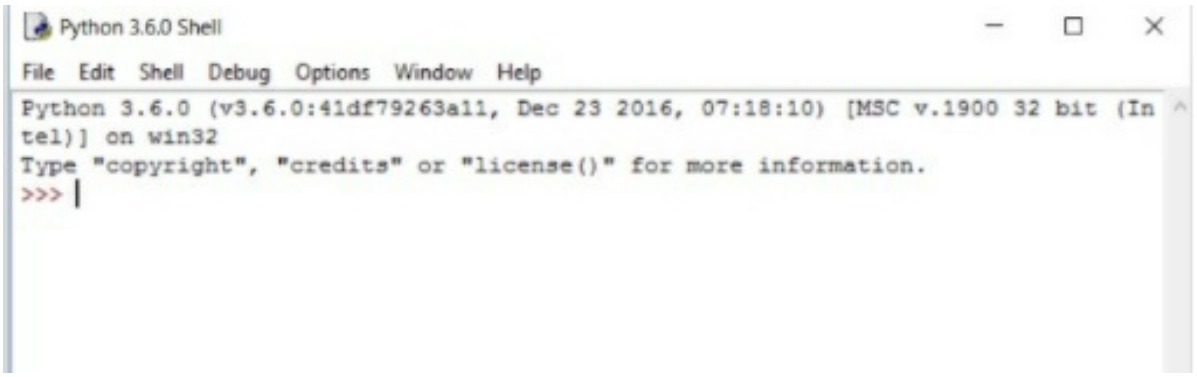
The standard and most common Python development environment is IDLE (GUI integrated).

IDLE refers to the world of Integrated Growth.

This allows a single gui to modify, run, search and debug Python programmes. This atmosphere makes writing programmes simple.

NOTE: In these tutorials, we'll be using version 3.6 of Python IDLE to develop and run Python code. It is available for free at www.python.org.

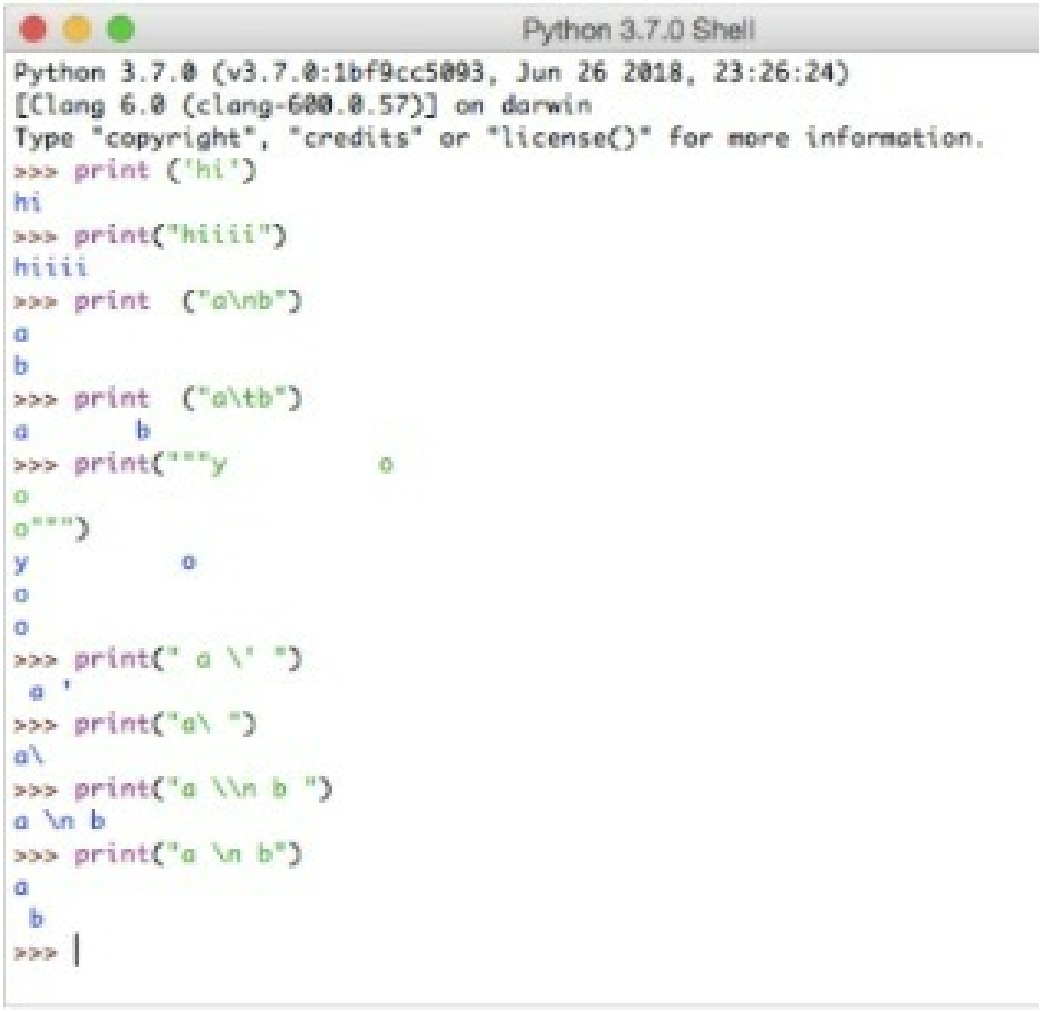
start up the IDLE



Features of python

- Object oriented
- Strong exception handling
- Automatic memory management
- Dynamic
- Strong data types
- Easy syntax
- Indentation based
- Forgiving in nature

Printing "Hello World" in Python


Python 3.7.0 Shell
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print ('hi')
hi
>>> print("hiiii")
hiiii
>>> print ("a\nb")
a
b
>>> print ("a\tb")
a b
>>> print("""y o
o
o""")
y o
o
o
>>> print(" a \' ")
a '
>>> print("a\ ")
a\
>>> print("a \\n b ")
a \n b
>>> print("a \n b")
a
b
>>>

Install pip

Pip is a platform that enables the installation and management of python packages.

Python: install pip

In order to instal the new version of pip, run command in the MacOS

```
$ sudo easy_install pip
$ sudo pip install --upgrade pip
```

install the pip in the Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install python-pip
$ sudo pip install --upgrade pip
```

pip in Centos

```
$ sudo yum update
$ sudo yum install epel-release
$ sudo yum install python-pip
# version>= Centos7
$ sudo pip install --upgrade pip
```

Executing first program

We will launch Python on our device to function in the interactive mode.

After the prompt (> > >), we type the Python expression / statement / command and Python automatically responds with the output of the prompt.

First program on Python

Let 's begin by typing "Hello Python!" after the prompt.

>>>print("Hello Python!")

Try the following

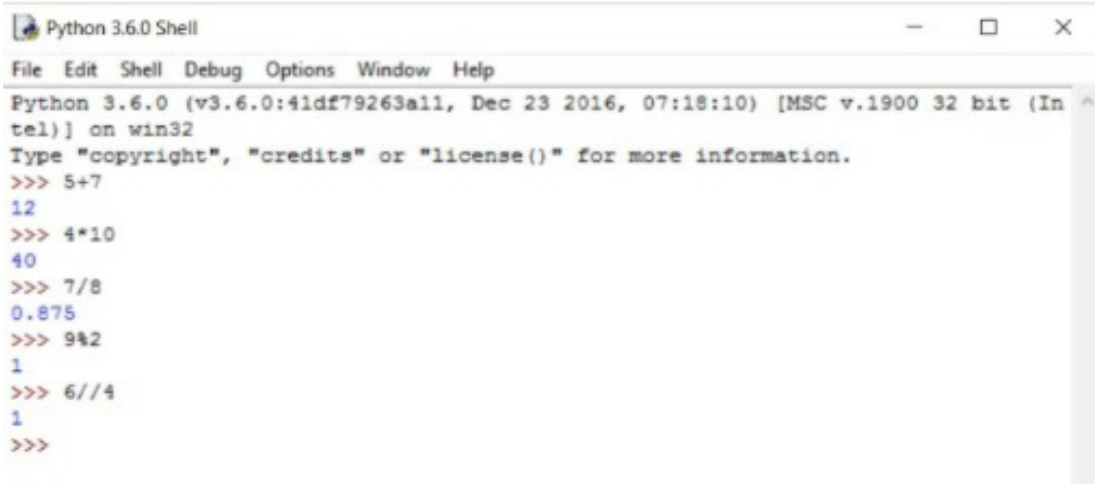
a) 5+7

b) 4*10

c) 7/8

d) 9%2

e) 6//4



Also try this

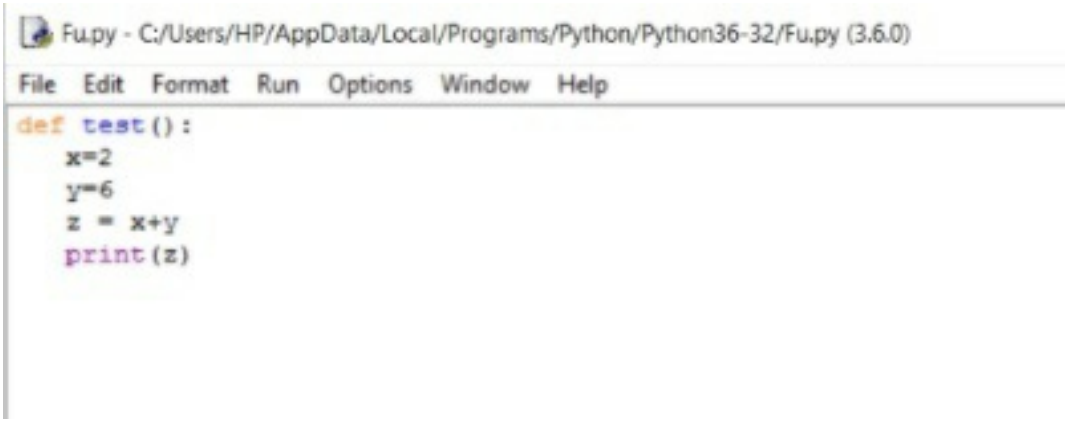
```
>>> x=1

>>> y=5

>>> z = x+y

>>> print (z)
```

```
Output: 6
```

Other operation:

```
>>> a=3
```

>>>a+1, a-1


Fu.py - C:/Users/HP/AppData/Local/Programs/Python/Python36-32/Fu.py (3.6.0)
File Edit Format Run Options Window Help
def test():
x=2
y=6
z = x+y
print(z)

2) Script Mode

We type the Python program into a file in script mode and then use the interpreter to execute the material from the file.

It is useful for beginners and for testing small pieces of code to function in interactive mode, as we can test them immediately.

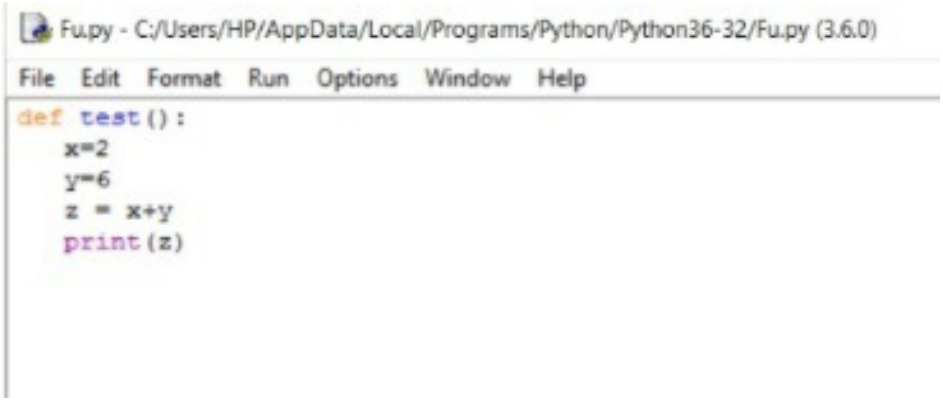
But we should still save our code so that we can change and reuse the code to encode more than a few lines.

We'll use the following steps in IDLE to build and run a Python script if the script mode is not made accessible by default in the IDLE environment.

1. File>New File
2. Write the Python code.
3. Save it **(^S)**.
4. Execute it interactive mode- by **(^F5)** - using RUN option

Step 1: File> New File
Step 2: Write:

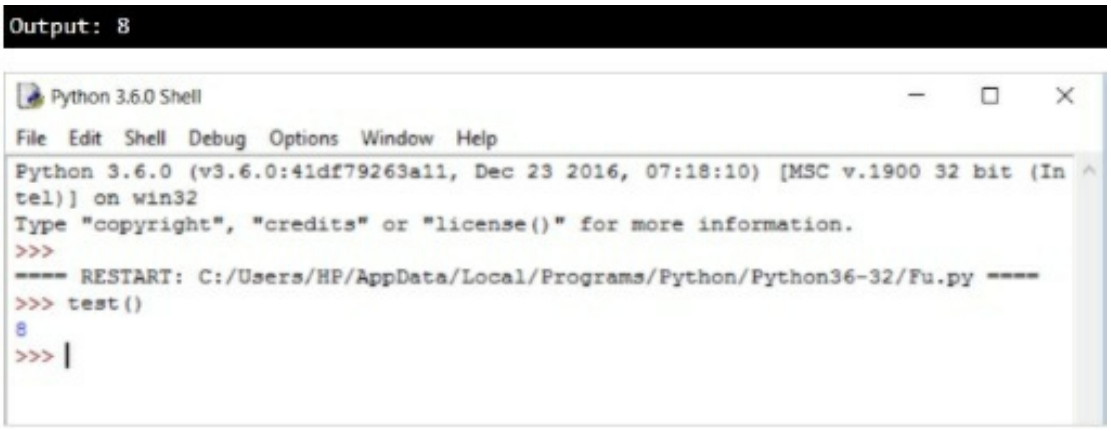
```
def test():

x=2

y=6

z = x+y

print (z)
```



Step 3: Save or File > Save As

Step 4: For execution, press ^F5

> >>>test()



Python Keywords

Keywords are the Python programming language's reserved words (and any other programming languages such as C , C++ , Java, etc.) whose definitions are defined and whose definitions can not be modified.

The keywords in python programming languages are case-sensitive.

Python keyword list

In Python 2.5

and del from not while

as elif global or with

assert else if pass yield

break except import print

class exec in raise

continue finally is return

def for lambda try

In Python 3.8.1

False None True and as

assert async await break class

continue def del elif else

except finally for from global

if import in is lambda

nonlocal not or pass raise

return try while with yield

Python built-in keywords

There are built-in python keywords in the python programming language, which are used to perform different tasks for different purposes.

All python keywords – List

Sr.	Keyword	Description
1	and	It is used for logical operations, it is a logical AND operator. It returns "True" if both operands are True.
2	as	It is used to create an alias of a module.
3	assert	It is used for debugging purpose, it returns an "AssertionError" if the given test condition is False.
4	break	break the loop's execution. It is used to transfer program's control from a scope of loop body to the next statement written after the loop body.
5	class	It is used to define/create a class.
6	continue	It leaves the rest of the statements in the loop body and continues the next iteration of the loop.
7	def	It is used to define/create a function.
8	del	It is used to delete an object.
9	elif	It is used with "if...elif...else" statement to check next condition. It is the same as "else if" of C, C++ language.
10	else	It is used with "if...elif...else" statement to define an else block – if any condition is not True, else block executes.

11	except	Just like "catch" block in the other programming languages, it is used to define an "except" block that executes if any exception occurs in the try block.
12	False	It is used to define a Boolean value False, it may also the result of a comparisons expression.
13	finally	It is used with the exceptions block (try...except...finally), it defines a code block that executes always – not matter "try" block has an exception or not.
14	for	It is used to create a for loop.
15	from	It is used to import a specific section (like functions, classes, etc) from a module.
16	if	It is used to create a conditional statement i.e. to check the condition.
17	import	It is used to import a module in the program.
18	in	It is used to check whether an element present in a sequence like string, list, tuple, etc.
19	is	It is used to check whether two objects are the same objects or not.
20	lambda	It is used to create an anonymous function that can have multiple arguments but a single statement/expression.
21	None	It is similar to the null, it is used to define a null value.
22	nonlocal	It is used to declare a non-local variable, by using this keyword, we can instruct the compiler that the variable is not a local variable.

23	not	It is a logical operator (logical NOT) and returns "True" if the operand is "False" or returns "False" if the operand is "True".
24	or	It is a logical operator (logical OR), it is used to combine two conditions and returns "True" if any condition is "True", else it returns "False".
25	pass	It is used as a null statement, it is very useful to define a blank function or a blank body of a conditional statement.
26	raise	It is used to raise an error and displays a customized message.
27	return	To return a value or/and returns the program's control from calling a function to called function i.e. it is used to exit from the function.
28	True	It is used to define a Boolean value False, it may also the result of a comparisons expression.
29	try	It is used with "try...except...finally" block.
30	while	It is used to create a while loop.
31	with	It is similar to "using" in C#.Net and VB.Net, it is used to simplify the exception handling.
32	yield	It is used to end a function and returns a generator.

Comments in Python

Comments in Python are used to increase the code's readability.

For a better understanding of code and logic that they have used to solve the given issue to the reader, it is valuable information provided by the programmer in source code.

During compilation, comments are not executed and are not shown on the output either.

Types of comments in Python

1. Single line comment (#)
2. Multi-line string as comment (''')

1) Single line comments

Single-line comments are used for one-line statements in Python, such as explanations of various variables , functions, expressions, etc.

A hash (#) symbol is used to render single-line comments without whitespace when the comments go to the next line, then at the beginning of the next line they must position each other's hashtag (#).

Let's see an instance and try to understand how in the software we apply the single-line comments.

EXAMPLE

```
# Single line comments example
# a program to print a given string and addition.

print('Welcome @ Learnpython')
a=2
b=5
print(a+b)

# addition of both numbers by using plus(+) sign.
```

2) Multi-line string comments

As we've seen in the above example, we have to put a hash (#) symbol in each line for multi-line comments.

In Python, multi-line string comments are given using the delimiter When it does not fit into one line, it is helpful.

We have to enclose the string with a delimiter at both ends for multiline string comments.

EXAMPLE

```
'''
   Here we will check a given number n is even or odd
   with multi-line comments in Python.
'''

n=6768

if n%2==0:
      print("Even number.")
else:
     print("Odd number.")
```

Block Comments in Python

Comments

A comment is a piece of text in a computer program that is intended to be an interpretation or annotation.

It readable by the author in the source code and that is ignored by the compiler / interpreter.

Type of Python Comments

Block Comments and **Inline Comments**,

1) Block Comments

For the piece of code that follows, Block comments apply. It may refer to a part of the code or the code as a whole.

They have the same degree of indentation as that code.

Each comment line begins with a #.

EXAMPLE

```
# Python program to print
# Hello World

print("Hello World")
```

Output:

```
Hello World
```

It is also possible to make block comments using ''' ''' Something written in these quotes is known as Comment.

EXAMPLE

```
'''
Python program to print
Hello World
'''


print("Hello World")
```

EXAMPLE 2

```
def hello():
    '''
    Here, this is docstring
    since it is written right after
    the function definition.

    below given print statement will print
    Hello World
    '''
    print("Hello World")

if __name__ == "__main__":
  # calling hello function
  hello()
```

2) Inline Comments

A comment on the same line as a declaration is an inline comment.

Inline remarks should be separated from the argument by at least two spaces.

They're supposed to begin with a # and a single room.

EXAMPLE

```
print("Hello World")  # This is an inline comment
```

OUTPUT

```
Hello World
```

Create multiline comments

There are two ways to do this when we need to comment on several lines / statements, either comment on each line or build multiline comments (or block comment).

The first approach is to comment on every section,

This can be regarded as a single line comment in Python-at the beginning of each line to be commented, we use the hash character (#).

 # This is line 1

 # This is line 2

 # This is line 3

And, the second way to comment on the segment,

This can be considered in Python as a multiline comment-we use triple single quotes (''') at the beginning of the segment to be commented and triple single quotes (''') at the end.

'''

This is line 1

This is line 2

This is line 3

'''

EXAMPLE

'''

```
Function name: print_text
Parameters: None
Return type: None
Description: This function will print
some text on the screen
'''

def print_text():
  print("Hello, world! How are you?")

if __name__ == "__main__":
  # Here, we will call the print_text function
  # that will print some text on the screen
  print_text()
```

Python Numeric Types

Data types are an important concept in programming.

Data of different kinds can be stored in variables according to the role that we want the variables to perform.

In Python, the built-in data types are

- Text Type
- Numeric Type
- Mapping Type
- Sequence Type
- Set Type
- Boolean Type
- Binary Type

3 numeric data types in Python:

1. int
2. float
3. comple

1) int

The Integer numeric form is used without decimal points to store signed integers, such as -5, 2, 78, etc.

EXAMPLE

```
# Assigning integer values to Variables
x = 8
y = -9
```

```
# Manipulating the value of x
x = x + y

# Prints the updated value of x
print("x= ", x)

# Prints the Data Type of x
print("Data type of x: ", type(x))
```

Output

```
x=  -1

Data type of x:  <class 'int'>
```

2) float

The numerical form of float is used to store floating-point values such as 6.66, 58.9, 3.14, etc.

Floats can be in scientific notation as well, with a power of 10 (3.6e3 = 3.6x 103 = 3600) indicated by E or e.

EXAMPLE

```
# Assigning float values to Variables
x = 8.9
y = -9.1

# Manipulating the value of x
x = x - y
```

```
# Prints the updated value of x
print("x= ", x)

# Prints the Data Type of x
print("Data type of x: ", type(x))
```

Output

```
x=  18.0

Data type of x:  <class 'float'>
```

3) complex

To store complex numbers such as 3.14j, 8.0 + 5.3j, 2 + 6j, etc., the complex numeric form is used.

Format: Real component + Imaginary component j

Note: J must denote the imaginary part of a complex number. It is known to be an invalid syntax in Python if we denote it with i.

EXAMPLE

```
# Assigning complex values to Variables
x = 1+8.5j
y = 4+9j

# Manipulating the value of x
x = x + y

# Prints the updated value of x
print("x= ", x)
```

```
# Prints the Data Type of x
print("Data type of x: ", type(x))
```

Output

```
x=  (5+17.5j)

Data type of x:  <class 'complex'>
```

Variables and DataTypes

1. Number

Data type numbers store numerical values. This type of data is eternal, i.e. the meaning of its object can not be modified (this aspect will be addressed later).

They come in three different kinds:

- **Integer & Long**
- **Float/floating point**
- **Complex**

i) Integers:

Integers are whole numbers with decimal digits such as 100000, -99, 0, 17 consisting of + or-sign.

L is appended to the value when we want a value to be viewed as a very long integer value. Python considers those values as long integers.

```
>>> a = 10
>>> b = 5192L #example of supplying a very long value to a variable
>>> c= 4298114
>>> type(c) # type ( ) is used to check data type of value
```

ii) Floating Point:

Floating-point numbers are called numbers of fractions or decimal points.

A floating point number consists of a decimal digit series of signs (+,-) and a dot such as 0.0, -21.9, 0.98333328, 15.2963.

iii) Complex:

The python complex number consists of two floating point values, each one for real and imaginary sections.

We'll use x.real and x.image to access various parts of the vector (object) x.

The imaginary part of the number is defined by j instead of I so the imaginary part of 1 + 0j denotes zero.

EXAMPLE

```
>>> x = 1+0j
>>> print (x.real,x.imag)
1.0 0.0
```

2. None

This is a particular category of data with a single value. It is used in a scenario to indicate the absence of value / false. It is pictured by None.

3. Sequence

An ordered set of objects, indexed by positive integers, is a sequence.

It is a mixture of data types that are mutable and non-mutable.

Lines, Lists & Tuples are three sequence data form types available in Python.

3.1) String:

It is an ordered sequence of characters / letters.

They are found in single quotes (') or double quotes (").

The quotes do not form part of the string.

They just tell the machine where it starts and ends with the string constant.

```
>>>type ('Good Morning')
```

<class'str'>
>>> type ('3.2')
<class'str'>

3.2) Lists:

The list is also a sequence of some sort of importance.

The values are called elements / objects in the list. Mutable and indexed / ordered are these.

In square brackets, the list is enclosed.

3.3) Tuples:

Tuples are a set of values and are indexed by integers of some kind. They're unchangeable.

4. Sets

Set is an unordered array of values, without duplicate entry, of any kind. Sets will be timeless.

EXAMPLE

>>>s = set ([1,2,3,4])
>>>print(s)

{1, 2, 3, 4 }

Determine a Python variable's type

1) type() method

If type() is passed to a single argument, it returns the type of the given object.

EXAMPLE

```
>>> test_string = "yes"
>>> test_number = 1
>>> print(type(test_string))
<class 'str'>
>>> print(type(test_number))
<class 'int'>
```

2) isinstance() method

The isinstance() function tests whether an object (the first argument) is a classinfo class instance or subclass (the second argument)

EXAMPLE

```
>>> class Example:
...     name = 'include_help'
...
>>> ExampleInstance = Example()
>>> print(isinstance(ExampleInstance, Example))
True
>>> print(isinstance(ExampleInstance, (list, set)))
False
>>> print(isinstance(ExampleInstance, (list, set, Example)))
True
```

>>>

Comparison between type() and isinstance()

type()	isinstance()
It returns the type object of an object and comparing what it returns to another type object will only return True when the same type object are on both sides.	In order to see if an object has a certain type, use **isinstance()** as it checks to see if the object passed in the first argument is of the type of any of the type objects passed in the second argument. Hence, it works as expected with subclassing and old-style classes, all of which have the legacy type object instance.

Python Arithmetic Operators

Arithmetic operators are used to perform different arithmetic / mathematical operations, and they are binary operators, meaning that calculations require two operands.

Operator	Name	Descriptions
+	Addition Operator	It returns the addition of two expressions.
-	Subtraction Operator	It returns the subtraction of two expressions.
*	Multiplication Operator	It returns the product of two expressions.
**	Power Operator	It returns the value of the expression raised to the given power i.e. it returns the expression1 raised to the power of expression2.
/	Division Operator	It returns the quotient of two expressions.
//	Floor Division Operator	It returns the integral part of the quotient of two expressions.
%	Modulus Operator	It returns the decimal part of the quotient i.e. remainder of the division of two expressions.

Syntax:

```
exp1 + exp2
exp1 - exp2
exp1 * exp2
exp1 ** exp2
exp1 / exp2
exp1 // exp2
exp1 % exp2
```

PROGRAM

```
# Python program to demonstrate
# the example of arithmetic operators

a = 10
b = 3

# printing the values
print("a:", a)
print("b:", b)

# operations
print("a + b :", a + b)
print("a - b :", a - b)
print("a * b :", a * b)
print("a ** b:", a ** b)
print("a / b :", a / b)
print("a // b:", a // b)
print("a % b :", a % b)
```

Output:

a: 10

b: 3

a + b : 13

a - b : 7

a * b : 30

a ** b: 1000

a / b : 3.3333333333333335

a // b: 3

a % b : 1

EXAMPLE 2

```
# Python program to demonstrate
# the example of arithmetic operators

a = 10.10
b = -2.5

# printing the values
print("a:", a)
print("b:", b)

# operations
print("a + b :", a + b)
print("a - b :", a - b)
print("a * b :", a * b)
print("a ** b:", a ** b)
print("a / b :", a / b)

# // operator rounds the result
# down to the nearest whole number
print("a // b:", a // b)

print("a % b :", a % b)
```

Output:

a: 10.1

b: -2.5

a + b : 7.6

a - b : 12.6

a * b : -25.25

a ** b: 0.0030845837443758094

a / b : -4.04

a // b: -5.0

a % b : -2.4000000000000004

EXAMPLE 3

```
# Python program to demonstrate
# the example of arithmetic operators
# Operations on lists, strings, etc

x = "Hello"
y = "World"

# '+', '*' with strings
print("x + y:", x + y)
print("x + y + x:", x + y + x)
print("x * 2:", x * 2)
print("x * 10:", x * 10)
print()

# '+', '*' with lists
```

```
x = [10, 20, 30, 40]
y = [40, 30, 20, 10]

print("x + y:", x + y)
print("x + y + x:", x + y + x)
print("x * 2:", x * 2)
print("x * 3:", x * 3)
print()
```

Output:

x + y: HelloWorld

x + y + x: HelloWorldHello

x * 2: HelloHello

x * 10: HelloHelloHelloHelloHelloHelloHelloHelloHelloHello

x + y: [10, 20, 30, 40, 40, 30, 20, 10]

x + y + x: [10, 20, 30, 40, 40, 30, 20, 10, 10, 20, 30, 40]

x * 2: [10, 20, 30, 40, 10, 20, 30, 40]

x * 3: [10, 20, 30, 40, 10, 20, 30, 40, 10, 20, 30, 40]

Python Identity Operators

Identity operators are used to perform an object comparison operation, i.e., these operators check whether or not both operands apply to the same objects (with the same memory location).

The Identity Operators are follows,

Operator	Descriptions	Example
is	It returns True if both operands refer to the same objects; False, otherwise.	`x is y`
is not	It returns True if both operands do not refer to the same objects; False, otherwise.	`x is not y`

Syntax:

operand1 is operand2

operand1 is not operand2

'is' operator example

```
# Python program to demonstrate the
# example of identity operators

x = [10, 20, 30]
y = [10, 20, 30]
z = x

# Comparing the values using == operator
print("x == y: ", x == y)
```

```
print("y == z: ", y == z)
print("z == x: ", z == x)
print()

# Comparing the objects
# whether they are referring
# the same objects or not
print("x is y: ", x is y)
print("y is z: ", y is z)
print("z is x: ", z is x)
print()
```

Output:

```
x == y:  True

y == z:  True

z == x:  True


x is y:  False

y is z:  False

z is x:  True
```

'is not' operator example

```
# Python program to demonstrate the
# example of identity operators
```

```
x = [10, 20, 30]
y = [10, 20, 30]
z = x

# Comparing the values
# using != operator
print("x != y: ", x != y)
print("y != z: ", y != z)
print("z != x: ", z != x)
print()

# Comparing the objects
# whether they are referring
# the same objects or not
print("x is not y: ", x is not y)
print("y is not z: ", y is not z)
print("z is not x: ", z is not x)
print()
```

Output:

```
x != y:  False

y != z:  False

z != x:  False


x is not y:  True

y is not z:  True
```

z is not x: False

Membership Operators in Python

Membership Operators are the operators used to check if a value / variable, such as string, list, tuples, sets, dictionary or not, exists in the sequence.

Python Membership Operators

Operator	Description	Example
`in`	It returns `True`, if a variable/value found in the sequence.	`10 in list1`
`not in`	It returns `True`, if a variable/value does not found in the sequence.	`10 not in list1`

EXAMPLE

```
# Python example of "in" and "not in" Operators

# declare a list and a string
str1 = "Hello world"
list1 = [10, 20, 30, 40, 50]

# Check 'w' (capital exists in the str1 or not
if 'w' in str1:
        print "Yes! w found in ", str1
else:
        print "No! w does not  found in " , str1

# check 'X' (capital) exists in the str1 or not
if 'X' not in str1:
        print "yes! X does not exist in ", str1
else:
        print "No!  X exists in ", str1
```

```
# check 30 exists in the list1 or not
if 30 in list1:
        print "Yes! 30 found in ", list1
else:
        print "No! 30 does not found in ", list1

# check 90 exists in the list1 or not
if 90 not in list1:
        print "Yes! 90 does not exist in ", list1
else:
        print "No! 90 exists in ", list1
```

Output

```
Yes! w found in  Hello world
yes! X does not exist in  Hello world
Yes! 30 found in  [10, 20, 30, 40, 50]
Yes! 90 does not exist in  [10, 20, 30, 40, 50]
```

Behavior of increment operators in Python

and decrement operators in Python

A coherent and readable language is known to be Python. Unlike in Java, the increment (+ +) and decrement (—) operators do not support python, both in precedence and in return value.

For example, x++ and + + x, or x— or —x, are not valid in Python.

```
x=1
x++
```

Output

```
  File "main.py", line 2
    x++
      ^
SyntaxError: invalid syntax
```

EXAMPLE

```
x=1


print(--x)
# 1
print(++x)
```

```
# 1

print(x--)
'''
  File "main.py", line 8
    print(x--)
            ^
SyntaxError: invalid syntax
'''
```

The reasoning for this is that python integers are immutable and can therefore not be modified. So, in order to raise, we would have to do the following.

```
x = 1

x=x+1
print(x)

x=x-1
print(x)

x +=2
print(x)

x -= 1
print(x)
```

Output

2

1

3

2

Logical and Bitwise NOT Operators

on Boolean in Python

Not is used for Logical NOT operator in python, and ~ is used for Bitwise NOT. Here, in Python, we'll see their uses and execution.

To reverse the result, the Logical NOT (not) operator returns "False" if the result is "True"; otherwise, "True."

The operator Bitwise NOT (~) is used to invert all the bits, i.e. it returns the number's complement.

Logical NOT (not) operator

PROGRAM

```
# Logical NOT (not) operator

x = True
y = False

# printing the values
print("x: ", x)
print("y: ", y)

# 'not' operations
print("not x: ", not x)
print("not y: ", not y)
```

Output:

```
x:  True
```

y: False

not x: False

not y: True

Bitwise NOT (~) operator

PROGRAM

```
# Bitwise NOT (~) operator

x = True
y = False

# printing the values
print("x: ", x)
print("y: ", y)

# '~' operations
print("~ x: ", ~ x)
print("~ y: ", ~ y)

# assigning numbers
x = 123
y = 128

# printing the values
print("x: ", x)
print("y: ", y)
```

```
# '~' operations
print("~ x: ", ~ x)
print("~ y: ", ~ y)
```

Output:

```
x:  True
y:  False
~ x:  -2
~ y:  -1
x:  123
y:  128
~ x:  -124
~ y:  -129
```

Logical Operators on String

Logical Operators

- Logical AND (`and`)
- Logical OR (`or`)
- Logical NOT (`not`)

With Strings

An empty string means False as a Boolean value, while True as a Boolean value is a non-empty string.

For "and" operator: If the first operand is true, the second operand is checked and the second operand is returned.

For 'or' operator: If False is the first operand, the second operand is checked and the second operand is returned.

For 'and' operator: If the operand is an empty string, True; False will be returned, otherwise.

EAMPLE 1

```
# Logical Operators on String in Python

string1 = "Hello"
string2 = "World"

# and operator on string
print("string1 and string2: ", string1 and string2)
print("string2 and string1: ", string2 and string1)
print()
```

```
# or operator on string
print("string1 or string2: ", string1 or string2)
print("string2 or string1: ", string2 or string1)
print()

# not operator on string
print("not string1: ", not string1)
print("not string2: ", not string2)
print()
```

Output:

string1 and string2: World

string2 and string1: Hello

string1 or string2: Hello

string2 or string1: World

not string1: False

not string2: False

EXAMPLE 2

```
# Logical Operators on String in Python

string1 = "" # empty string
string2 = "World" # non-empty string
```

```
# Note: 'repr()' function prints the string with
# single quotes
# and operator on string
print("string1 and string2: ", repr(string1 and string2))
print("string2 and string1: ", repr(string2 and string1))
print()

# or operator on string
print("string1 or string2: ", repr(string1 or string2))
print("string2 or string1: ", repr(string2 or string1))
print()

# not operator on string
print("not string1: ", not string1)
print("not string2: ", not string2)
print()
```

Output:

```
string1 and string2:  ''
string2 and string1:  ''


string1 or string2:  'World'
string2 or string1:  'World'


not string1:  True
not string2:  False
```

if else Conditional Operator

Python also includes the functionality to test conditional statements using the conditional operator, much like other programming languages.

If the other conditional operator is used, depending on the outcome of the given Boolean expression, to evaluate / get either value / statement (from the two values / statements given).

Syntax:

```
value1 if expression else value2
```

Here

Value1-represents the conditional expression value if it is valid.

Expression-represents the condition that a Boolean (i.e. we may assume it is a condition) to evaluate.

Value2-represents the conditional expression value if it is false.

EXAMPLE 1

```
# find the largest Value

x = 20
y = 10

# if else conditional operator
largest = x if x>y else y
# printing the values
print("x: ", x)
print("y: ", y)
```

```
print("largest: ", largest)
print()

x = 10
y = 20

# if else conditional operator
largest = x if x>y else y
# printing the values
print("x: ", x)
print("y: ", y)
print("largest: ", largest)
print()

x = 10
y = 10

# if else conditional operator
largest = x if x>y else y
# printing the values
print("x: ", x)
print("y: ", y)
print("largest: ", largest)
print()
```

Output:

```
x:  20
y:  10
```

largest: 20

x: 10

y: 20

largest: 20

x: 10

y: 10

largest: 10

EXAMPLE 2

```
# find the largest Value

x = 10
y = 20
z = 30

# if else conditional operator
largest = x if (x>y and x>z) else (y if(y>x and x>z) else z)
# printing the values
print("x: ", x)
print("y: ", y)
print("z: ", z)
print("largest: ", largest)
print()
```

```
x = 10
y = 30
z = 20

# if else conditional operator
largest = x if (x>y and x>z) else (y if(y>x and y>z) else z)
# printing the values
print("x: ", x)
print("y: ", y)
print("z: ", z)
print("largest: ", largest)
print()

x = 30
y = 20
z = 10

# if else conditional operator
largest = x if (x>y and x>z) else (y if(y>x and y>z) else z)
# printing the values
print("x: ", x)
print("y: ", y)
print("z: ", z)
print("largest: ", largest)
print()

x = 10
y = 10
```

```
z = 10

# if else conditional operator
largest = x if (x>y and x>z) else (y if(y>x and y>z) else z)
# printing the values
print("x: ", x)
print("y: ", y)
print("z: ", z)
print("largest: ", largest)
print()
```

Output:

```
x:  10
y:  20
z:  30
largest:  30

x:  10
y:  30
z:  20
largest:  30

x:  30
y:  20
z:  10
largest:  30

x:  10
y:  10
```

z: 10
largest: 10

JAVASCRIPT

CODING BASICS

FOR ABSOLUTE BEGINNERS

BY

TAM SEL

JAVASCRIPT

This JavaScript tutorial makes it easy for both beginners and experts to learn JavaScript.

To learn JavaScript, you only need a basic understanding of computers.

JAVASCRIPT EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<h2>Welcome to JavaScript</h2>
<script>
document.write("Hello Friends");
</script>
</body>
</html>
```

JavaScript can be implemented by adding JavaScript statements in the code.

< **script** >…< **/script** >.

```
<!DOCTYPE html>
<html>
<body>
<script type= "text/javascript">
document.write("Hello World");
</script>
</body>
</html>
```

JavaScript code can be written in 3 places in

JavaScript

1. Between the <body>……</body> tag of HTML.
2. Between the <head>……</head> tag of HTML.
3. In .jsfile(external JavaScript).

Example (code between the body tags)

In the example below, the body tag contains JavaScript code.

```
<!DOCTYPE   html>
<html>
<body>
<script   type="text/javascript">
alert("Hello world");
</script>
</body>
</html>
```

Example2 (code between the head tag)

We'll create the function msg() in the example below . In JavaScript, you must write function in function name to construct a function. To call a method, we must first create a case. We're using the one-click event to call the msg() function in this case.

```
<!DOCTYPE   html>
<html>
<head>
<script   type="text/javascript">
```

```
function msg(){
alert("Hello world");
}
</script>
</head>
<body>
<p>Welcome to Javascript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

External JavaScript file

We can build a JavaScript file that we can use in HTML files.

Since a single JavaScript file can be used in several HTML pages, it allows for code reuse.

It improves the web page's efficiency.

The file must have a **.js** extension.

Example

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="message.js"></script>
</head>
```

```
<body>
<p>Hello world</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Javascript comment

Comments in JavaScript are used to describe the code. It's used to add code-related content, such as warnings or recommendations, so that end users can understand it.

Types of JavaScript comments

There are two kinds of JavaScript comments:

1. Single-line Comment
2. Multi-line Comment

Single-line Comment

Double forward slashes (//) are used to represent single-line comments. While the program is running, any text between the double forward slashes (//) and the end of the line will be ignored.

eg. Comment added before the statement.

```
<!DOCTYPE   html>
<html>
<body>
<script>
// single-line comment //
document.write("hello world");
</script>
</body>
</html>
```

eg. Comment added after the statement.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   a=10;
var   b=20;
var   c=a+b;   //It add values of a and b variable
document.write(c);   // print sum of 10 and 20
</script>
</body>
</html>
```

Output

30

Multi-line comment

Single and multi-line comments can also be made for multi-line comments. Comments that span multi lines begin with /* and end with */. JavaScript will ignore the text in between them.

/* Write comment here */

Example

```
<!DOCTYPE   html>
<html>
<body>
<script>
/* It is multi-line comment.
It will not be displayed */
document.write   ("Example of JavaScript multi line comment");
</script>
</body>
</html
```

Javascript Variable

Values (name=”Ram”) and expressions (Sum=x+y) are stored in variables.

We must first declare a variable before using it. To declare a variable, we use the keyword **var** as follows:

```
var name;
```

There are two types of variables:

1. Local Variable
2. Global Variable

Local Variable – It's declared within a function or a block.

Example

```
<script>
functionabc(){
var   x=10;   //local variable
</script>
```

Global Variable - It has a global scope, meaning that it can be specified anywhere in JavaScript code.

Outside of the function, a variable is declared.

Example

```
<!DOCTYPE   html>
<html>
```

```
<body>
<script>
var   data   =200;   //global variable
function   a(){
document.writeln(data);
}
function   b(){
document.writeln(data);
}
a();//calling javascript function
b();
</script>
</body>
</html>
```

Declaring JavaScript global variable within function

We need to use the **window object** to declare a JavaScript global variable within a function.

Example

```
window.value=90;
```

It can now be declared within any function and accessed from within any function.

Example

```
<!DOCTYPE   html>
```

```
<html>
<body>
<script>
function   a(){
window.value=50;   //declare global variable by use of window object
}
function   b(){
alert(window.value);//access global variable from other function
}
a();
b();
</script>
</body>
</html>
```

Javascript Data Types

It has a variety of data types to store various types of values.

In Javascript, there are two kinds of data types.

1. Primitive data types.
2. Non-primitive data types.

Since JavaScript is a dynamic style language, we don't need to define the type of variable because the JavaScript engine uses it dynamically. The data type is defined using var in this case. It can store any sort of value, including numbers, strings, and objects.

Example

```
var a=Ram; //String
var b=20; //Number
var x= {FirstName:"Ram", lastName:"Doe"}; //Object
```

JavaScript primitive data types

There are 5 types of primitive data types in JavaScript.

Data Type	Description
String	It represents sequence of characters e.g. "Hello"
Number	It represents numeric values e.g. 1,2,3,4,5,6.
Boolean	It represents Boolean value either Right or Wrong.
Undefined	It represents undefined values.
Null	It means **no values** at all.

JavaScript non-primitive data types

There is 3 non-primitive data types are as follows:

Data type	Description
Object	It represents instance through which we can access members.
Array	It represents group of similar values.
RegExp	It represents regular expression.

Operators are symbols in JavaScript that are used to execute operations on operands. 3+2 equals 5 in plain terms. The operands here are 3 and 2, and the operator is +.

Following are the operators used in JavaScript:

1. Arithmetic Operators.
2. Comparison (relational) Operators.
3. Bitwise Operators.
4. Logical Operators.
5. Assignment Operators.
6. Special Operators.

Arithmetic Operators

We use arithmetic operations on the operands in arithmetic operators.

Operators	Description
+(Addition)	Add two operands Example: A+B
-(subtraction)	Subtraction from one operands to another one Example: A-B
*(Multiplication)	Multiply both the operands Example: A*B
/(Division)	Divide the numerator by the denominator Example: B/A
%(Modulus)	Remainder of an integer division is 0 Example: B%A will give 0
++(Increment)	Increase an integer value by one Example: If value A is 9, then A++ will give 11
–(Decrement)	decrease an integer value by one Example: If value A is 11, then A– will give 7

Comparison Operators

In the Comparison Operator, two operands A and B are compared.

Operators	Description
==(Equal)	It checks the value of two operands is equal or not, if yes then the condition becomes true. **E.g.- A=1, B=2. (A==B)** condition is not true
!=(Not Equal)	It checks the value of two operands is equal or not, if the values are not equal, then the condition becomes true. **E.g.- A=1, B=2. (A!=B)** is true.
>(Greater than)	It checks the value between two operands which one is greater, if condition is right then, it display true. **E.g.- (A>B)** is not true.
<(Less than)	It checks the value between two operands which one is less, if condition is right then, it display true. **E.g.- (A<B)** is true.
>=(Greater than or Equal to)	It checks the value of the first operand is greater than or equal to the value of second operand, if condition is right then it displays true. **E.g. – (A>=B)** not true.

Assignment operator

The following assign operators are supported by JavaScript.

Operator	Description
=(Simple Assignment)	It is used to assigns a value to a variable. **E.g.-** var **x = 10;**
+=(Add and Assignment)	It is used to add the value of right operand to a variable and assigns the result to the variable. **E.g.- Operator:** x+=y **Meaning:** x =x + y
– =(Subtract and Assignment)	It is used tosubtract the value of right operand to a variable and assigns the result to the variable. **E.g.- Operator:** x – = y **Meaning:** x = x â€" y
* = (Multiply and Assignment)	It is used to multiply the variable by the value of the right operand and assigns the result to the variable. **E.g.- Operator:** x * = y **Meaning:** x = x * y
/ = (Divide and Assignment)	It is used to divide the variable by the value of the right operand and assign the result to the variable. **E.g.- Operator:** x / = y **Meaning:** x = x / y

Bitwise Operators

The following bitwise operators are provided by JavaScript:

Operators	Description
& (Bitwise AND)	It performs the AND operation on each pair of bits. **E.g.- (A & B) is 2.**
| (Bitwise OR)	It performs the OR operation on each pair of bits. **E.g. – (A | B) is 3.**
^ (Bitwise XOR)	It performs the OR operation on each pair of bits. **E.g. – (A ^ B) is 1.**
~ (Bitwise Not)	It performs NOT operation on each pair of bits. **E.g. – (~B) is -4.**
<< (Left Shift)	This operator shifts the first operand the specified no of bits to the left. **E.g. – (A << 1) is 4.**
>> (Right Shift)	The left operand's value is moved right by the number ofbits specified by the right operand. **E.g. – (A >>1) is 1.**
>>> (Right shift with zero)	This operator is just like the >> (Right operator), except that the bits shifted in on the left are always zero. **E.g. – (A >>>1) is 1.**

Logical Operators

Boolean values are commonly used for logical operators.

The logical operators are as follows:

Operator	Description
&&(Logical AND)	If both the operands are non-zero, then condition is true. **E.g. – (A && B) is true.**
|| (Logical) OR	If any of the two operands are non-zero, then the condition is true. **E.g. – (A || B) is true.**
! (Logical NOT)	Reverses the logical state of its operand. If the condition is true, then the Logical NOToperator will make it false. **E.g.: ! (A && B) is false.**

Javascript If Else

A conditional statement in JavaScript is used to perform various tasks depending on the conditions.

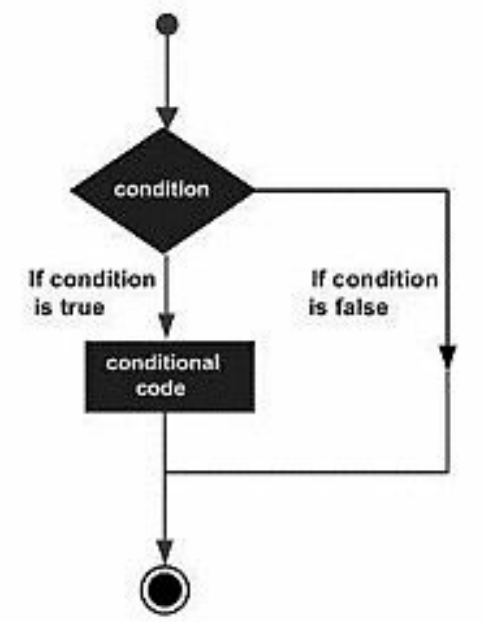
In JavaScript, there are three types of statements.

1. if Statement.
2. if else Statement.
3. if else if Statement.

if statement

If the condition is true or false, the **if statement** is used in JavaScript to execute the code.

```
if   (condition){   //
    //content to be evaluated
}
```



EXAMPLE

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x=50;
if(x>30){
```

```
document.write("value of x is greater than 30");
}
</script   >
</body>
</html>
```

OUTPUT

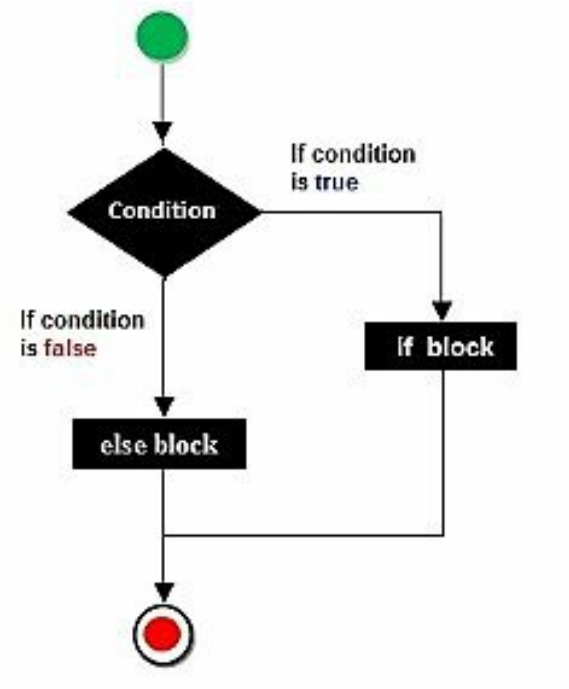
Value of x is greater than 30

if else statement

If either condition is true or false, the **if else** statement is used.

Syntax

```
if(condition)
{
// set of statements
}
else
{
// set of statements
}
```



EXAMPLE

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x=20;
if(x%2==0){
document.write("x is even number");
}
else{
document.write("a is odd number")
}
</script>
</body>
</html>
```

OUTPUT

X is even number

If else if

It just looks at the material to see if the expression is valid through several expressions. **If else if** is a more advanced version of the if else statement.

Syntax

```
If (condition1)
{
Statement(s) to be executed if condition 1 is true
}
else if (condition 2)
{
Statement(s) to be executed if condition 2 is true
}
else if (condition 3)
{
Statement(s) to be executed if condition 3 is true
}
else
{
Statement(s) to be executed if no condition is true
}
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<script>
```

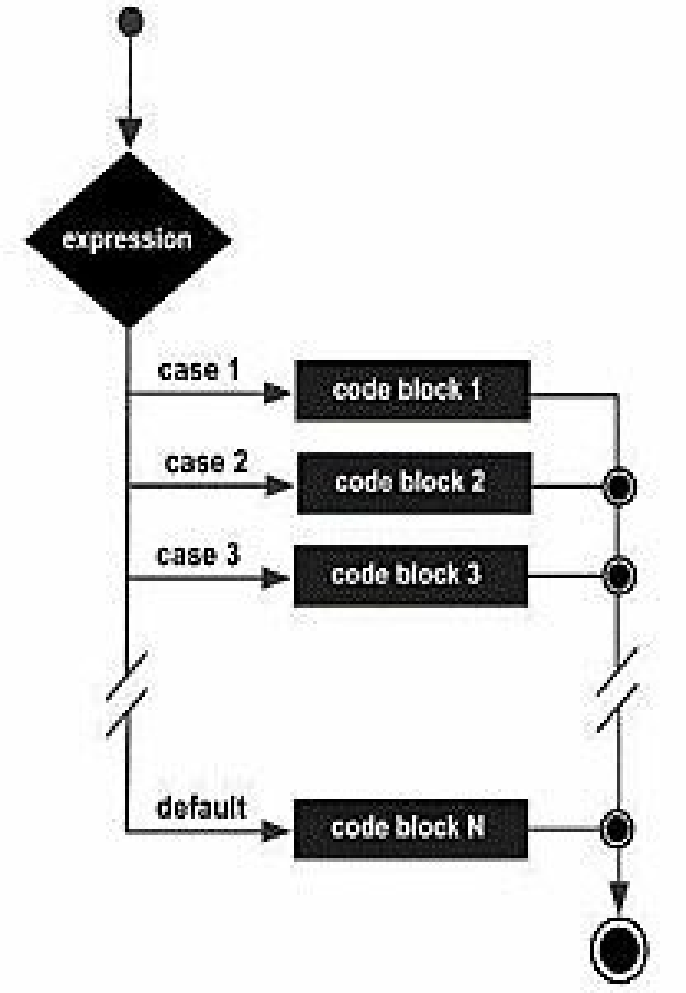
```
var   x=50;
if   (x==10){
document.write("x is equal to 10");
}
else   if(x==50){
document.write("x is equal to 50");
}
else   if(x==30){
document.write("x is equal to 30");
}
else{
document.write("a is not equal to 10, 50 or 30");
}
</script>
</body>
</html>
```

OUTPUT

x is equal to 50

Javascript Switch Statement

In JavaScript, the switch statement is used to execute a single code under different conditions. It's the same as the else if statement.



SYNTAX

```
switch   (expression)
{
case   1:   statement(s)
break;
case   2:   statement(s)
```

```
break;
â€¦ ..
case   n:   statement(s)
break;
default:   statements(s)
}
```

EXAMPLE

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   grade='B';
var   result;
switch(grade){
case   'A':
result="A Grade";
break;
case   'B':
result="B Grade";
break;
case   'C':
result="C Grade";
break;
default:
result="No Grade";
}
```

```
document.write(result);
</script>
</body>
</html>
```

OUTPUT

B Grade

JavaScript Function

The ability to build new functions inside the **<Script>.....</script>** tag is an essential feature of JavaScript. The function keyword is used to declare a function in JavaScript.

To reuse the file, we call the JavaScript function multiple times.

Advantages - JavaScript functions

1. Code reusability: We use the same feature several times to save time and code.

2. Less coding: Our software is compressed as a result of this. When we perform a routine mission, we don't write a lot of code.

Syntax

```
function functionName(parameter or not)
{
statements
}
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<script>
function msg()
{
alert("Hello! world");
```

```
}
</script>
<input   type="button"   onclick="msg()"   value="click here"/>
</body>
</html>
```

OUTPUT

Hello world!

Function with arguments

Call function by passing arguments.

```
<!DOCTYPE   html>
<html>
<body>
<script>
function   getcube(number)
{
alert(number*   number*   number);
}
</script>
<form>
<input   type="button"   value="click"   onclick="getcube(3)"/>
</form>
</body>
</html>
```

OUTPUT

27

EXAMPLE

```
<!DOCTYPE   html>
<html>
<body>
<script>
function   getname()
{
```

```
name=prompt("Enter the Name");
alert("Welcome Mr/Mrs "   +   name);
}
</script>
</body>
<form>
<input   type="button"   value="Click"   onclick="getname()"/>
</form>
</html>
```

Function with return value

Call function that return value and use it in program.

```
<!DOCTYPE   html>
<html>
<body>
<script>
function   getInfo()
{
return   "Hello Raj! How are you?";
}
</script>
<script>
document.write(getInfo());
</script>
</body>
</html>
```

OUTPUT

Hello Raj! How are you?

The Function() Constructor

A function statement isn't only for defining new functions; it can also be used to define functions dynamically using the Function() constructor and the new operator.

Syntax

```
<script>
   var variablename = new Function(Arg1, Arg2..., "Function Body"
</script>
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
<script>
var func = new Function("a", "b", "return a+b;");
function secondFunction(){
var result;
result = func(50,50);
document.write ( result );
}
</script>
</head>
<body>
<p>Click button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
```

```
</form>
<p>Use different parameters inside the function and try yourself...
</p>
</body>
</html>
```

Function Literals

The definition of function literals, which is another way to describe functions, is introduced in JavaScript 1.2. It's an expression that describes a function that doesn't have a name.

The syntax for a function literal is similar to that of a function statement, except that it is used as an expression rather than a statement, and there is no need for a function name.

Syntax

```
<script>
var variablename = function (Argument List){
        Function Body
};
</script>
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
<script>
var func = function(a,b){ return a+b };
function secondFunction(){
var result;
result = func(10,20);
document.write ( result );
}
</script>
```

```
</head>
<body>
<p>Click the button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and try yourself...
</p>
</body>
</html>
```

Javascript Object

An object is a state-and-behavioral entity. JavaScript is a scripting language that focuses on objects. While JavaScript is template-based rather than class-based, it allows us to build objects directly.

SYNTAX

```
objectName.objectProperty = propertyValue;
```

To write some content on the text, we use the write() method on document properties.

```
Document.write("Hello world")
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
var book = new Object(); // Object created
book.subject = "5 points someone"; // Properties assign to the object
book.author = "ChetanBhagat";
</script>
</head>
<body>
<script type="text/javascript">
document.write("Book Name is : " + book.subject + "<br>");
document.write("Book Author is : " + book.author + "<br>");
</script>
```

```
</body>
</html>
```

OUTPUT

Book Name is: 5 points someone

Book Author is: Chetan Bhagat

Methods of Object

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">// Define a function which will work as a metl
function addPrice(amount){
this.price = amount;
}
function book(title, author){
this.itle = title;
this.author = author;
this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
varmyBook = new book("5 Points Someone", "ChetanBhagat");
myBook.addPrice(150);
```

```
document.write("Book Title is : "  +  myBook.title  +  "<br>");
document.write("Book Author is : "  +  myBook.author  +  "<br>");
document.write("Book Price is : "  +  myBook.price  +  "<br>");
</script>
</body>
</html>
```

OUTPUT

Book Author is :ChetanBhagat

Book Price is : 150

Javascript array

Arrays are used to describe a set of elements in a single unit or memory place.

Any element that enters the array will be stored in the array with a unique index starting at zero. We can store data with the aid of indexes.

In JavaScript, we must use the new keyword to declare an array. We use new Array to generate an array (n).

The number of slots in the sequence is denoted by n.

SYNTAX

```
myarray = new array(n);
```

NOTE: If we generate an array in JavaScript without specifying the size, the array object will have a size of zero.

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<script>
var i;
var emp = new Array();
emp[0] = "Ajay";
emp[1] = "Abhay";
emp[2] = "Arun";
emp[3] = "Shipra";
for (i=0; i<emp.length; i++)
{
```

```
document.write(emp[i] + "<br>");
}
</script>
</body>
</html>
```

OUTPUT

Ajay
Abhay
Arun
Shipra

EXAMPLE 2

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function array()
{
num = new Array(5)
num[0] = 10
num[1] = 20
num[2] = 30
num[3] = 40
num[4] = 50
sum = 0;
for (i=0; i<num.length; i++)
{
sumsum = sum + num[i];
}
```

```
alert(sum)
}
</script>
</head>
<body>
<input type="button" onclick="array()" value="click here">
</body>
</html>
```

FUNCTIONS IN ARRAY

Functions	Description
concat()	To concat the element of one array at the end of another array and returns array.
sort()	Sort all elements of an array.
reverse()	Reverse all the elements.
slice()	To extract specified number of elements starting from specified index without deleting them from array.
splice()	To extract specified number of elements starting from specified index and delete them from array.
push()	Push all the elements in array to top.
pop()	Pop the top elements from array.

Javascript String

The String object is used to interact with a string of characters. It's a tool for storing and manipulating text.

In JavaScript, there are two ways to make a string:

1. String literal.
2. Using new keyword.

String literals are formed by using double quotes.

SYNTAX

```
var stringname="string value";
```

EXAMPLE

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   str="Hello String Literal";
document.write(str);
</script>
</body>
</html>
```

OUTPUT

Hello String Literal

Using new keyword

SYNTAX

var stringname=new String("new keyword")

OUTPUT

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   stringname=new   String("Hello String");
document.write(stringname);
</script>
</body>
</html>
```

OUTPUT

Hello String

String properties

There is a list of string object properties as well as their descriptions.

1. constructor.
2. length.
3. prototype.

Constructor– This method returns a reference to the object's string function.

SYNTAX

string.constructor

Length - The number of characters in a string is returned.

SYNTAX

string.length

Prototype - It allowed any object to have properties and methods applied to it (Number, Boolean, string, and data etc.).

SYNTAX

object.prototype.name = value

METHODS

Methods	Description
charAt(index)	It returns the character at the given index.
concat(str)	It merges the text of two string and returns new string.
indexOf(str)	It returns the index position of the given string.
lastindexOf(str)	It returns the last index position of the given string.
match()	It is used to match regular expression against a string.
tolowercase()	It is used to return given string value converted to lower case.
touppercase()	It is used to return the given string value converted to uppercase.
valueOf()	Returns the primitive value of the specified object.

charAt()

It's a method for getting the character from a given index.

```
<!DOCTYPE html>
<html>
<body>
<script>
var str = new String( "Hello" );
document.writeln("str.charAt(0) is: " + str.charAt(0));
document.writeln("<br />str.charAt(1) is: " + str.charAt(1));
document.writeln("<br />str.charAt(2) is: " + str.charAt(2));
document.writeln("<br />str.charAt(3) is: " + str.charAt(3));
document.writeln("<br />str.charAt(4) is: " + str.charAt(4));
</script>
</body>
</html>
```

concat(str)

This method returns a new string after adding two or more strings.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x="Hello";
var   y="World";
var   z=x+y;
document.write(z);
</script>
</body>
</html>
```

indexOf(str)

index of a string The index position of the given string is returned by the Of(str) procedure.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x="Tutorial provide by tutorialandexample";
var   y=x.indexOf("by");
document.write(y);
</script>
</body>
</html>
```

lastindexOf(str)

The string lastIndexOf(str) method returns the string's last index position.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x="Tutorial provide by tutorialandExamole";
var   y=x.lastIndexOf("by");
document.write(n);
</script>
</body>
</html>
```

match()

```
<!DOCTYPE html>
<html>
<body>
<script>
var str = "To more detail, see Chapter 2.5.4.1";
var re = /(chapter \d+(\.\d)*)/i;
var find = str.match( re );
document.write(find );
</script>
</body>
</html>
```

toLowerCase()

It returns a lowercase version of the given string.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x="HELLO WORLD";
var   y=x.toLowerCase();
document.write(y);
</script>
</body>
</html>
```

toUpperCase()

The provided string is returned in uppercase letters.

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x="hello world";
var   y=x.toUpperCase();
document.write(y);
</script>
</body>
</html>
```

valueOf()

It returns the string object's primitive value.

```
<!DOCTYPE  html>
<html>
<body>
<script>
var  str  =  new  String("Hello world");
document.write(str.valueOf(  ));
</script>
</body>
</html>
```

Javascript Date

Date Object is a data form that is built into the JavaScript language. The new Date object is used to construct data objects (). A single moment in time is represented by a JavaScript Date case. The only way to create a JavaScript date object is to use JavaScript as a constructor.

SYNTAX

The Date() constructor can be used to generate a Date object using the following syntaxes.

```
new Date();
new Date(value);
new Date(dateString);
new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

NOTE: Bracketed parameters are always optional.

Descriptions

year

The year is represented by an integer number. Map of values ranging from 0 to 99 for the years 2000 to 2099.

Month

Beginning with 0 for January and ending with 11 for December, this integer value represents the month.

Date

It's an option. The month's day is represented by an integer value.

Hours

It's an option. The hours of the day are represented by an integer value.

Minutes

It's an option. The minute section of time is represented by an integer value.

Seconds

It's an option. The second time segment is represented by an integer value.

Milliseconds

It's an option. A millisecond segment of time is represented by an integer value.

Example of Date

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   date=new   Date();
var   day=date.getDate();
var   month=date.getMonth()+1;
var   year=date.getFullYear();
document.write("<br>Date: "+day+"/"+month+"/"+year);
```

```
</script>
</body>
</html
```

Current Time - Example

```
<!DOCTYPE  html>
<html>
<body>
Current  Time:  <span  id="txt"></span>
<script>
var  today=new  Date();
var  h=today.getHours();
var  m=today.getMinutes();
var  s=today.getSeconds();
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
</script>
</body>
</html>
```

JavaScript Math

Math is a built-in object with properties and methods for mathematical functions and constants.

It enables you to carry out mathematical operations on numbers.

Syntax

1. varpi_val = Math.PI;
2. varsin_val = Math.sin(30);

Examples

Math.pow()

The value of x to the power of y is returned by Math.pow(x,y):

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Math.pow()</h2>
<p>Math.pow(x,y) returns the value of x to the power of y:
</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.pow(6,2);
</script>
</body>
</html>
```

Math.sqrt()

The square root of x is returned by Math.sqrt(x).

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Math.sqrt()</h2>
<p>Math.sqrt(x) returns the square root of x:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.sqrt(81);
</script>
</body>
</html>
```

Math.ceil()

The value of x rounded up to the nearest integer is returned by Math.ceil(x):

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Math.ceil()</h2>
<p>Math.ceil() rounds a number <strong>up</strong> to its nearest
</p>
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML = Math.ceil(6.8);
</script>
</body>
</html>
```

Javascript Number

The Number object is a wrapper object that lets you manipulate numerical values.

The Number() constructor is used to build the number object.

It could be either an integer or a floating-point number.

SYNTAX

```
var n=new Number(value);
```

EXAMPLES

```
<!DOCTYPE   html>
<html>
<body>
<script>
var   x=100;//integer value
var   y=100.7;//floating point value
var   z=12e5;//exponent value, output: 1200000
var   n=new   Number(20);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
</script>
</body>
</html>
```

OUTPUT

100 100.7 1200000 20

Description

The number object is most commonly used for the following purposes:

1. It returns NaN if the claim cannot be converted to a number (Not a Number).
2. Number can be used to perform a type conversion in a non-constructor context (i.e., without the new operator).

JavaScript Number Constants

Constant	Description
MIN_VALUE	It returns the largest minimum value.
MAX_VALUE	It returns the largest maximum value.
POSITIVE_INFINITY	It returns the positive infinity, overflow value.
NEGATIVE_INFINITY	It returns the negative infinity, overflow value.
NaN	It represent "Not a Number" value.