PYTHON PROGRAMMING LANGUAGE

Fastest programming language. Python is used in Machine Learning, Data Science, Big Data,

Web Development, Scripting

PYTHON INTERPRETTER

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

A python interpreter is a computer program that converts each high-level program

statement into machine code. An interpreter translates the command that you write out into

code that the computer can understand

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT) =>

1. using IDE - one can write code, run the code, debug the code

2. IDE takes care of interpreting the Python code, running python scripts, building

executables, and debugging the applications.

3. An IDE enables programmers to combine the different aspects of writing a computer program

4. if you wnated to be python developer only then you need to install (IDE -- PYCHARM)

PYTHON INTERPRETER & COMPILER

Both compilers and interpreters are used to convert a program written in a

high-level language into machine code understood by computers.

Interpreter -->

```
1. Translates program one statement at a time
2. Interpreter run every line item
3. Execut the single, partial line of code
4. Easy for programming
```

Compiler -->

```
1. Scans the entire program and translates it as a
whole into
machine code.
2. No execution if an error occurs
3. you can not fix the bug (debug) line by line
```

ANACONDA

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In [26]:
```
1 + 1 #ADDITION
```

Out[26]: 2

In [28]:
```
2-1
```

Out[28]: 1

In [30]:
```
3*4
```

Out[30]: 12

In [32]:
```
8/4 #Division
```

Out[32]: 2.0

In [34]:
```
8/5 # float Division
```

Out[34]: 1.6

In [38]:
```
8 // 4 #intergeer division
```

Out[38]: 2

In [40]:
```
8 + 9 - 7
```

Out[40]: 10

In [42]:
```
(5 + 5) * 5 # BODMAS (Bracket || Oders || Divide || Multiply || Add || Substact)
```

Out[42]: 50

In [44]:
```
2*2*2*2*2
```

Out[44]: 32

In [46]:
```
2 ** 5 #exponentaion
```

Out[46]: 32

In [48]:
```
14 % 2 #Modulus [Remainder]
```

Out[48]: 0

In [7]:
```
a,b,c,d,e = 15, 7.8, 'Mandeep', 8+9j, True
print(a)
print(b)
print(c)
```

```
print(d)
print(e)
```

```
15
7.8
Mandeep
(8+9j)
True
```

In [9]:
```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'complex'>
<class 'bool'>
```

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Lets see print()

In [13]:
```
print('Hello World')
```

```
Hello World
```

In [15]:
```
a = 2
b = 3
a+b
```

Out[15]: 5

In [17]:
```
c = a + b
c
```

Out[17]: 5

In [19]:
```
print('Mandeep\'s "Technology"') #\ has some special meaning to ignore the error
```

```
Mandeep's "Technology"
```

In [25]:
```
#print the nit 2 times
'red' + ' red'
```

Out[25]: 'red red'

In [29]:
```
#5 time print
5* ' five'
```

Out[29]: ' five five five five five'

\n ---New line

In [32]:
```
print('Hello\nYoo') #\n --- new line
```

```
Hello
Yoo
```

Print result with string

In [35]:
```
num1 = 20
num2 = 30
add=num1+num2
print('The addition of',num1,'and',num2,'is =',add)
```

```
The addition of 20 and 30 is = 50
```

.format( ) string print method

In [40]:
```
num1 = 20
num2 = 30
add = num1 + num2
print(f'The addition of{num1} and {num2} is = {add}')
```

```
The addition of20 and 30 is = 50
```

round ( ) : Rounds a number to a given precision in decimal digits.

In [45]:
```
num1 = 100
num2 = 25
num3 = 333
avg = round((num1+num2+num3)/3)
print(f'The avrage of {num1}, {num2} and {num3} is = {avg}')
```

```
The avrage of 100, 25 and 333 is = 153
```

In [49]:
```
# let's combine all
num1 = 10
num2 = 20
add = num1 + num2
print('The addition of', num1,num2,'is=',add)
print('The addition of {} and {} is = {}'.format(num1,num2,add))
print(f'The addition of {num1} and {num2} is= {add}')
```

```
The addition of 10 20 is= 30
The addition of 10 and 20 is = 30
The addition of 10 and 20 is= 30
```

End Statement: Starts next string at end of earlier string

In [54]:
```
print('hello') #1st statement
print('Good morinnning') #2 statemet
# i want print like:-hello good morning
```

```
hello
Good morinnning
```

In [71]:
```
print('hello') #1st statement
print('Good morinnning', end='_') #2 statemet
# i want print like:-hello good morning
```

```
hello
Good morinnning_
```

Seprator

1. here one print statement only we use

2. insisde one print statement we have multipal value.

3. we want to seperate these multipal values with anythingsng

In [63]:
```
print('Hello','hey','how are you',sep='---->')
```

```
Hello---->hey---->how are you
```

In [73]:
```
print('One','Two','Three',sep="♥")
```

```
One♥Two♥Three
```

In [75]:
```
print(3,    ".")
```

```
3 .
```

Operators

1. Arithmetic Operators:

+, -, , /, %, *

2. Assignment Operators:

=, +=, -=, &=, *=

3. Relational Operators:

<, <=, >, >=, ==, !=

4. Logical Operators:

AND(&), OR(|), NOT(~), XOR(^), right shift(>>), left shift(<<)

5. Unary Operators

-, eg:-5

Arithmetic Operators:

In [93]:
```
x1, y1 = 10, 5
x1 + y1
```

Out[93]: 15

In [95]:
```
x1 - y1
```

Out[95]: 5

In [97]:
```
x1 * y1
```

Out[97]: 50

In [99]:
```
x1 / y1
```

Out[99]: 2.0

In [101…
```
x1 // y1
```

Out[101… 2

In [103…
```
x1 % y1
```

Out[103… 0

In [105…
```
x1 ** y1
```

Out[105… 100000

In [107…
```
2 ** 3
```

Out[107… 8

Assignment operator

In [134…
```
x = 2
```

In [136…
```
x = x + 2
x
```

Out[136… 4

In [138…
```
x += 2
x
```

Out[138… 6

In [140…
```
x += 2
x
```

Out[140… 8

In [142…
```
x *= 2
x
```

Out[142… 16

In [144…
```
x -= 2
x
```

Out[144… 14

In [146…
```
x /= 2
x
```

Out[146… 7.0

Unary Operator

In Python, a unary operator is an operator that takes only one operand (a single value or variable) and performs an operation. Common unary operators in Python include:

1. Unary Plus (+): This operator is used to indicate a positive value. It doesn't change the value of its operand.

In [159…
```
a = 5
b = +a  # b is 5
```

In [161…
```
b
```

Out[161… 5

2. Unary Minus (-): This operator is used to negate the value of its operand (i.e., it changes the sign).

In [164…
```
a = 5
b = -a  # b is -5
```

In [166…
```
b
```

Out[166… -5

3. Logical NOT (not): This operator is used to invert the Boolean value of its operand.

In [179…
```
a = True
b = not a  # b is False
```

In [181…
```
b
```

Out[181… False

Relational operator

We use this operator for comparision

In [187…
```
a = 5
b = 7
```

In [190…
```
a == b
```

Out[190… False

In [194…
```
a<b
```
Out[194… True

In [196…
```
a>b
```
Out[196… False

a = b; we cannot use = operator; because that means it is assigning

In [199…
```
a>=b
```
Out[199… False

In [201…
```
a<=b
```
Out[201… True

In [203…
```
a != b
```
Out[203… True

LOGICAL OPERATOR

AND, OR, NOT(~)

AND --> &; and

x, y --> xy

0, 0 --> 0

0, 1 --> 0

1, 0 --> 0

1, 1 --> 1

OR --> |; or

x, y --> xy 0, 0 --> 0 0, 1 --> 1 1, 0 --> 1 1, 1 --> 1

XOR --> ^

x,y --> xy

0,0 --> 0

0,1 --> 1

1,0 --> 1

1,1 --> 0

In [231…
```
a = 5
b = 4
```

In [233…
```
a < 8 and b < 5
```

Out[233… True

In [237…
```
a < 8 or b < 2
```

Out[237… True

In [239…
```
x = False
x
```

Out[239… False

In [241…
```
not x #you can reverse the operation
```

Out[241… True

In [243…
```
a < 8 ^ b < 5
```

Out[243… False

Number system conversion :

Binary --> 0&1 ; eg 10:1010

64 32 16 8 2 1 <-----

Octal--> 0 -7

Decimal --> 0-9

Hexadecimal--> 0-9, A=10,B=11, C=12,...

Binary

In [257…
```
25
```

Out[257… 25

In [259…
```
bin(25)
```

Out[259… '0b11001'

25: factorisation by 2: for BINARY

In [262…
```
2 | 25
```

Out[262… 27

2|12--> remainder 1

2 | 6 --> remainder 0

2 | 3 --> remainder 0

| 1 --> remainder 1

therefore binary of 25 = 1001

In [269…
```
int(0b11001)
```

Out[269… 25

Octal

In [272…
```
oct(15)
```

Out[272… '0o17'

Hex

In [275…
```
hex(25)
```

Out[275… '0x19'

In [277…
```
hex(10)
```

Out[277… '0xa'

In [279…
```
decimal(10)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[279], line 1
----> 1 decimal(10)

NameError: name 'decimal' is not defined
```

Swap variable in python

(a,b = 5,6) After swap we should get ==> (a, b = 6,5)

In [358…
```
a = 5
b = 6
```

In [354…
```
a = b
b = a
print(a)
print(b)
```

```
6
6
```

In [360…
```
a,b = b,a # Swapped
```

In [362…
```
print(a)
print(b)
```

```
6
5
```

In [372…
```
x = 10
y = 20
```

Swap Variable Formulas

In [364…
```
x = x + y
y = x - y
x = x - y
print(x,y)
```

```
20 10
```

there is other way to work using swap variable also which is XOR

In [374…
```
x = x^y
y = x^y
x = x^y
print(x,y)
```

```
20 10
```

complement (~)

NOT, Operator

The bitwise complement operator in Python is represented by the tilde (~). It is a unary operator that flips the bits of its operand.

1's Compliment --> reverse of binary format

12 ==> 00001100

~12 ==> 11110011 ==> -13

2's complemenet --> 1's complement + 1

In [389…
```
#COMPLEMENT (~) (TILDE OR TILD)
~12 # why we get -13 ; To remember: ~x = -(x+1)
```

Out[389… -13

In [391…
```
~-2
```

Out[391… 1

In [393…
```
~-13
```
Out[393… 12

In [395…
```
12 & 13
```
Out[395… 12

In [397…
```
13 | 12
```
Out[397… 13

In [399…
```
13 ^ 12
```
Out[399… 1

In XOR i the both number are different then we will get 1 or else we will get 0

In [402…
```
12 ^ 13
```
Out[402… 1

In [404…
```
5 ^ 5
```
Out[404… 0

LEFT Shift : here we gain bits

In [407…
```
10<<3
```
Out[407… 80

In [409…
```
# 10 -> 1010
# 10 <<2= 3 --> 1010000 --> Here 3 bits are gained
```

In [411…
```
10 << 2
```
Out[411… 40

In [413…
```
# 10 -> 1010
# 10 <<2 --> 101000 --> Here 2 bits are gained
```

Right Shift : here we loose the bit

In [416…
```
10>>1
```
Out[416… 5

In [418…
```
# 10 -> 1010
# 10>>1 -> 101 Here 1 bit is lost
```

In [420…
```
10>>5
```

Out[420… 0

help() --> to get help in jupyer notebok

type q to exit

In [ ]:
```
help()
```

Welcome to Python 3.11's help utility! If this is your first time using Python, you should definitely check out the tutorial at https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To get a list of available modules, keywords, symbols, or topics, enter "modules", "keywords", "symbols", or "topics".

Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", enter "modules spam".

To quit this help utility and return to the interpreter, enter "q" or "quit".

You are now leaving help and returning to the Python interpreter. If you want to ask for help on a particular object directly from the interpreter, you can type "help(object)". Executing "help('string')" has the same effect as typing a particular string at the help> prompt.

In [432…
```
x = sqrt(25)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[432], line 1
----> 1 x = sqrt(25)

NameError: name 'sqrt' is not defined
```

In [518…
```
import math #as #m #math is module
```

In [520…
```
x = m.sqrt(25)
x
```

Out[520… 5.0

floor - minimum or least value

In [523…
```
print(m.floor(25.628292))
```

```
25
```

ceil - maximum or highest value

In [526…
```
print(m.ceil(25.6259))
```

```
26
```

In [528…
```
print(m.pow(3,2)) #power, 3^2
```

```
9.0
```

In [530…
```
print(m.pi)
```

```
3.141592653589793
```

In [514…
```
print(math.e) #e is epsilon value
```

```
2.718281828459045
```

In [532…
```
from math import pow,sqrt # if you only want ot import pow, sqrt function
```

In [536…
```
print(m.sqrt(64))
print(m.pow(10,2))
```

```
8.0
100.0
```

In [538…
```
from math import * #all function from math
```