PYTHON PROGRAMMING LANGUAGE

1. Python Became the Best Programming Language & fastest programming language.
2. Python is used in Machine Learning, Data Science, Big Data, Web Development, Scripting.
3. we will learn pyton from start to end || basic to expert.
4. if you are not done programm then that is totally fine.
5. I will explain from starting from scratch.
6. python software - pycharm || vs code || jupyter || spyder

PYTHON INTERPRETTER

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

PYTHON INTERPRETER --> What is Python interpreter? 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

PYTHON INTERPRETER EXAMPLE --> You write your Python code in a text file with a name like hello.py . How does that code Run? There is program installed on your computer named "python3" or "python", and its job is looking at and running your Python code. This type of program is called an "interpre

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT) =>

- using IDE - one can write code, run the code, debug the code
- IDE takes care of interpreting the Python code, running python scripts, building executables, and debugging the applications.
- An IDE enables programmers to combine the different aspects of writing a computer program.
- if you wnated to be python developer only then you need to install (IDE -- PYCHARM)ter".

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 -->

- Translates program one statement at a time
- Interpreter run every line item
- Execut the single, partial line of code

- Easy for programming

Compiler -->

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

Is Python an interpreter or compiler?

1. Python is an interpreted language, which means the source code of a Python program is converted into bytecode that is then executed by the Python virtual machine.
2. Python is different from major compiled languages, such as C and C + +, as Python code is not required to be built and linked like code for these languages.

# How to create python environment variable 1- cmd - python (if it not works) 2- find the location where the python is installed --> C:\Users\kdata\AppData\Local\Programs\Python\Python311\Scripts 3- system -- env - environment variable screen will pop up 4- select on system variable - click on path - create New 5- C:\Users\kdata\AppData\Local\Programs\Python\Python311 6- env - sys variable - path - new - C:\Users\kdata\AppData\Local\Programs\Python\Python311\Scripts 7- cmd - type python -version 8- successfully python install in cmd

ANACONDA

1. 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.

15th April 2025

Variable

Variable = Identifier = Object

1. Variables never start with digit but ends with digits
2. Variables are Case Sensitive
3. Special Characters are not allowed
4. Keywords or Reserved words never be declare as Variable

Syntax Error

The Error made by me is called as Syntax Error

In [1]:
```
va = 9
va
```

```
Out[1]:  9
```

```
In [2]:  id(va)     # Address of Memory location is called as 'id'
```

```
Out[2]:  140731177581240
```

```
In [6]:  1nit = 18   # Variables never start with digit but ends with digits
         1nit
```

```
  Cell In[6], line 1
    1nit = 18
    ^
SyntaxError: invalid decimal literal
```

```
In [5]:  nit2 = 19   # Variables are Case Sensitive
         NIT2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 2
      1 nit2 = 19
----> 2 NIT2

NameError: name 'NIT2' is not defined
```

```
In [7]:  v$ = 90    #Special Characters are not allowed
         v$
```

```
  Cell In[7], line 1
    v$ = 90
     ^
SyntaxError: invalid syntax
```

```
In [8]:  v_ = 90
         v_
```

```
Out[8]:  90
```

```
In [9]:  import keyword   # Keywords or Reserved words
         keyword.kwlist
```

```
Out[9]: ['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']
```

```
In [10]: len(keyword.kwlist)
```

```
Out[10]: 35
```

```
In [12]: for = 67
         for
```

```
  Cell In[12], line 1
    for = 67
        ^
SyntaxError: invalid syntax
```

```
In [13]: For = 97
         For
```

```
Out[13]: 97
```

Print

1. If user has only one variable 'Print not required'
2. If user have multiple variables 'Print required'

In [14]:
```
a = 5
b = 6
c = 7

print(a)
print(b)
print(c)
```

```
5
6
7
```

In [15]:
```
import sys
sys.version
```

Out[15]:
```
'3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1929 6
4 bit (AMD64)]'
```

16th April 2025

Data types

1. Integer Data type
2. Float Data type
3. Strings Data type
4. Boolean Data type
5. Complex Data type

In [3]:
```
i = 45
print(i)
print(type(i))
print(id(i))
```

```
45
<class 'int'>
140731179548472
```

In [5]:
```
petrol = 110.56
print(petrol)
print(id(petrol))
print(int(petrol))
print(type(petrol))
```

```
110.56
2136811298192
110
<class 'float'>
```

In [7]:
```
i1,i2 = 10,20
print(i1)
print(i2)
print(i1+i2)
```

```
10
20
30
```

In [14]:
```
s = 'Vihari Nandan'
print(s)
print(id(s))
print(type(s))
print(s.lower())
print(s.upper())
print(s.replace("N","C"))
print(s[3:])
print(s[-3:-1])
```

```
Vihari Nandan
2136818937904
<class 'str'>
vihari nandan
VIHARI NANDAN
Vihari Candan
ari Nandan
da
```

17th April 2025

In [1]:
```
# Boolean Data Types

true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 3
      1 # Boolean Data Types
----> 3 true

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

In [2]:
```
True
```

Out[2]:
```
True
```

In [3]:
```
False
```

Out[3]:
```
False
```

In [4]:
```
false
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 false

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

In [9]:
```
b = True
b1 = False
print(b + b1)
print(b-b1)
print(b*b1)
print(b1/b)   # float division
print(b1//b)  # int division
```

```
1
1
0
0.0
0
```

In [ ]:
```
# Complex datatypes - datatype we can use more math operation
a+bj

a - real part
b - imaginary part
j - square root of -1
```

In [12]:
```
c1 = 10+20j
print(c1)
print(type(c1))
```

```
(10+20j)
<class 'complex'>
```

In [13]:
```
c1.real
```

Out[13]:
```
10.0
```

In [15]:
```
c1.imag
```

Out[15]:
```
20.0
```

Type Casting or Type Conversions

In [16]:
```
int(3.4)  # float to int
```

Out[16]:
```
3
```

In [17]:
```
int(True, False)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 1
----> 1 int(True, False)

TypeError: int() can't convert non-string with explicit base
```

In [18]:
```
float(2)   # into to float
```

Out[18]:
```
2.0
```

In [19]:
```
int(3.4, 5.7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 int(3.4, 5.7)

TypeError: 'float' object cannot be interpreted as an integer
```

In [20]:
```
int(3,6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 int(3,6)

TypeError: int() can't convert non-string with explicit base
```

In [26]:
```
print(int(3.4))
print(int(True))
print(int('10'))
```

```
3
1
10
```

In [27]:
```
print(int(2+3j))
print(int('ten'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[27], line 1
----> 1 print(int(2+3j))
      2 print(int('ten'))

TypeError: int() argument must be a string, a bytes-like object or a real number, no
t 'complex'
```

18th April 2025

In [1]:
```
float(23)
```

Out[1]:
```
23.0
```

```
In [2]: float(23,56)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 float(23,56)

TypeError: float expected at most 1 argument, got 2
```

```
In [3]: float(True)
```

```
Out[3]: 1.0
```

```
In [4]: float(False)
```

```
Out[4]: 0.0
```

```
In [5]: float(1+2j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 1
----> 1 float(1+2j)

TypeError: float() argument must be a string or a real number, not 'complex'
```

```
In [6]: float('10')
```

```
Out[6]: 10.0
```

```
In [7]: float('ten')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 float('ten')

ValueError: could not convert string to float: 'ten'
```

from other datatypes to float except complex& string text

```
In [8]: complex(10)
```

```
Out[8]: (10+0j)
```

```
In [9]: complex(10,20)
```

```
Out[9]: (10+20j)
```

```
In [10]: complex(True)
```

```
Out[10]: (1+0j)
```

```
In [11]: complex(False)

Out[11]: 0j

In [12]: complex(3.2, 56)

Out[12]: (3.2+56j)

In [13]: complex(3.2, 5.6)

Out[13]: (3.2+5.6j)

In [15]: # Bool

         bool(0)

Out[15]: False

In [16]: bool(3.2)

Out[16]: True

In [17]: bool(0.0)

Out[17]: False

In [18]: bool(1+2j)

Out[18]: True

In [19]: bool(0+0j)

Out[19]: False

In [20]: bool('ten')

Out[20]: True

In [21]: bool('10')

Out[21]: True

In [22]: bool(_)        # Interview Question

Out[22]: True

In [23]: bool( )          # Non-Zero means True, Zero means False

Out[23]: False
```

String Indexing & Slicing & Advanced Slicing

```
In [26]: s9 = 'Hello Python'
         print(s9)
```

```
Hello Python
```

```
In [27]: s9[:]      # : is called as Empty Slicing (displays entire string)
```

```
Out[27]: 'Hello Python'
```

```
In [28]: for i in s9:
             print(i)
```

```
H
e
l
l
o
 
P
y
t
h
o
n
```

```
In [29]: s9[0]
```

```
Out[29]: 'H'
```

```
In [30]: s9[1]
```

```
Out[30]: 'e'
```

```
In [32]: s9[-1]
```

```
Out[32]: 'n'
```

```
In [33]: s9[-4]
```

```
Out[33]: 't'
```

```
In [34]: s9[::-1]        # Interview Questions
```

```
Out[34]: 'nohtyP olleH'
```

```
In [35]: s9[0:5]
```

```
Out[35]: 'Hello'
```

```
In [39]: print(s9[3:9])   # from 3rd index to 9th index
         print(s9[:9])    # from 0th index to 9th index
         print(s9[3:])    # from 3rd index to 0th index
```

```
lo Pyt
Hello Pyt
lo Python
```

```
In [40]: print(s9[0:11:2])
```

```
HloPto
```

21st April 2025

Data Structures

- data type = user or developer can declare only one value (int, float, string, complex, bool)
- data structure = user or developer can declare more than one value
- matrix = collection of rows * columns
- 2 types: In-build/ Build-In/Build In data structure (list, tuple, set, dict) User-defined data structure (stack, linklist, array, tree)
- a =5 # is data type b= 3,4,5,6 # is data structure c = Table (rows & columns) # matrices

```
In [1]: i = 5
        type(i)
```

```
Out[1]: int
```

```
In [2]: i = 5.5
        type(i)
```

```
Out[2]: float
```

LIST Data Structures

- denoted by []
- List is a collection of items/elements which are Mutable (add adn modify), list is growable, Duplicate is allowed, append() add element at last
- .append: add element in list
- .copy: copy entire list
- list inside list is called as Nested List

- Multiple datatype we defined

In [3]:
```
l = []
l
```
Out[3]:
```
[]
```

In [4]:
```
type(l)
```
Out[4]:
```
list
```

In [5]:
```
l.append(10)
print(l)
```
```
[10]
```

In [7]:
```
print(l.append(20))
print(l.append(30))
print(l.append(40))
print(l)
```
```
None
None
None
[10, 20, 30, 40, 20, 30, 40]
```

In [8]:
```
l.append(70,80,90)
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 l.append(70,80,90)

TypeError: list.append() takes exactly one argument (3 given)
```

In [10]:
```
l.append(78)
print(l)
print(len(l))
```
```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
9
```

In [11]:
```
l1 = l.copy()
print(l1)
```
```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
```

In [18]:
```
l1.append(2.3)
l1.append('vihari')
l1.append(1+2j)
l1.append([1,2,3])

print(l)
print(l1)
print(len(l))
print(len(l1))
```

```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
[10, 20, 30, 40, 20, 30, 40, 78, 78, 2.3, 'nit', (1+2j), [1, 2, 3], 2.3, 'nit', (1+2
j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3],
2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari',
(1+2j), [1, 2, 3]]
9
37
```

In [19]:
```
l == l1
```
Out[19]:
```
False
```

In [21]:
```
l2 = l.copy()
print(l2)
l == l2
```
```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
```
Out[21]:
```
True
```

In [22]:
```
print(l)
print(l1)
print(l2)
```
```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
[10, 20, 30, 40, 20, 30, 40, 78, 78, 2.3, 'nit', (1+2j), [1, 2, 3], 2.3, 'nit', (1+2
j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3],
2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari', (1+2j), [1, 2, 3], 2.3, 'vihari',
(1+2j), [1, 2, 3]]
[10, 20, 30, 40, 20, 30, 40, 78, 78]
```

In [24]:
```
print(l[:])
print(l[0])
l[0] = 100
print(l)
```
```
[10, 20, 30, 40, 20, 30, 40, 78, 78]
10
[100, 20, 30, 40, 20, 30, 40, 78, 78]
```

In [25]:
```
l[-1] = 200
print(l)
```
```
[100, 20, 30, 40, 20, 30, 40, 78, 200]
```

In [26]:
```
l[3:]
```
Out[26]:
```
[40, 20, 30, 40, 78, 200]
```

In [27]:
```
l[10:]
```
Out[27]:
```
[]
```

In [28]:
```
l[:10]
```
Out[28]:
```
[100, 20, 30, 40, 20, 30, 40, 78, 200]
```

In [29]:
```
l2.clear()
print(l2)
```
```
[]
```

In [30]:
```
del l2
print(l2)
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[30], line 2
      1 del l2
----> 2 print(l2)

NameError: name 'l2' is not defined
```

In [31]:
```
l1[0:12:5]
```
Out[31]:
```
[10, 30, 'nit']
```

In [32]:
```
l1[0:11:3]
```
Out[32]:
```
[10, 40, 40, 2.3]
```

In [37]:
```
l1.index('nit')
```
Out[37]:
```
10
```

22nd April 2025

In [8]:
```
l = [100, 20, 30, 40, 20, 30, 40, 78, 200]
print(l)
```
```
[100, 20, 30, 40, 20, 30, 40, 78, 200]
```

In [9]:
```
# Count
l.count(100)
```
Out[9]:
```
1
```

In [10]:
```
l.count(20)
```
Out[10]:
```
2
```

In [14]:
```
# Remove  - it remove first occurance
l.remove(20)
print(l)
```
```
[100, 30, 40, 20, 30, 40, 78, 200]
```

In [15]:
```
l.append("IoT")
print(l)
```

```
[100, 30, 40, 20, 30, 40, 78, 200, 'IoT']
```

In [16]:
```
# insert bydefault user need to pass 2 arguments. 1st arg is Index position & 2nd a
l.insert(2, 25)
print(l)
```

```
[100, 30, 25, 40, 20, 30, 40, 78, 200, 'IoT']
```

In [19]:
```
# POP - Remove element by indexing & default is last item will remove
l.pop()
print(l)
l.pop(2)
print(l)
```

```
[100, 25, 40, 20, 30, 40, 78]
[100, 25, 20, 30, 40, 78]
```

In [18]:
```
l.remove(30)
print(l)
```

```
[100, 25, 40, 20, 30, 40, 78, 200]
```

In [22]:
```
# Extend
l3 = [1,2]
l3.extend(l)
print(l3)
```

```
[1, 2, 1, 2]
```

In [23]:
```
# Reverse
l.reverse()
print(l)
```

```
[2, 1]
```

In [ ]:
```
# Sort
```

In [ ]:
```

```

23rd April 2025

In [3]:
```
l6 = ['a','c','b','d']
l6
```

Out[3]:
```
['a', 'c', 'b', 'd']
```

In [4]:
```
l6.sort()    # Parameter Tuning
print(l6)
```

```
['a', 'b', 'c', 'd']
```

In [5]:
```
l6.sort(reverse=True)    # default reverse is False  (Ascending)   # Hyperparameter
print(l6)      # Decending
```

```
['d', 'c', 'b', 'a']
```

```
In [6]: for i in l6:                    # i,k, j etc means iteration (next value)
            print(i)
```

```
d
c
b
a
```

```
In [7]: # List Membership
        'b' in l6
```

```
Out[7]: True
```

```
In [8]: for i in enumerate(l6):      # enumerate will give count with items
            print(i)
```

```
(0, 'd')
(1, 'c')
(2, 'b')
(3, 'a')
```

```
In [9]: all(l6)
```

```
Out[9]: True
```

```
In [10]: any(l6)
```

```
Out[10]: True
```

```
In [11]: l6.append(0)
         l6
```

```
Out[11]: ['d', 'c', 'b', 'a', 0]
```

```
In [12]: all(l6)
```

```
Out[12]: False
```

```
In [13]: any(l6)
```

```
Out[13]: True
```

Tuple

- Immutable (Not changeable)

```
In [16]: t = ()
         print(t)
         print(len(t))
         print(id(t))
```

```
()
0
140708801435368
```

In [17]:
```
t1 = (10,20,30,40,50)
t2 = ('a','z','m','n')
print(t1)
print(t2)
```

```
(10, 20, 30, 40, 50)
('a', 'z', 'm', 'n')
```

In [18]:
```
t2.count('z')
```

Out[18]:
```
1
```

In [20]:
```
t2.index('a')
```

Out[20]:
```
0
```

In [21]:
```
t1 = (10,20,30,40,50)
print(t1[0])
```

```
10
```

In [22]:
```
t3 = t1 * 3
print(t3)
```

```
(10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40, 50)
```

24th April 2025

SET Data Structure

In [25]:
```
s = {}
print(s)
print(type(s))
```

```
{}
<class 'dict'>
```

In [26]:
```
s1 = set()
print(s1)
```

```
set()
```

In [28]:
```
s1.add(10)
print(s1)
```

```
{10}
```

In [30]:
```
s1.add(20)
s1.add('nit')
s1.add(True)
```

```
s1.add(1+2j)
s1.add(3.2)
print(s1)
```

```
{True, 3.2, (1+2j), 10, 20, 'nit'}
```

In [31]:
```
s2 = set()
print(s2)
```

```
set()
```

In [32]:
```
s2.add(100)
s2.add(10)
s2.add(200)
s2.add(9)
print(s2)
```

```
{200, 9, 10, 100}
```

In [33]:
```
s3 = set()
print(s3)
```

```
set()
```

In [34]:
```
s3.add('z')
s3.add('a')
s3.add('m')
s3.add('b')
print(s3)
```

```
{'z', 'm', 'b', 'a'}
```

In [35]:
```
print(s1)
print(s2)
print(s3)
```

```
{True, 3.2, (1+2j), 10, 20, 'nit'}
{200, 9, 10, 100}
{'z', 'm', 'b', 'a'}
```

In [36]:
```
print(len(s3))
```

```
4
```

In [37]:
```
print(s3)
```

```
{'z', 'm', 'b', 'a'}
```

In [38]:
```
print(s3[0])   # Index not possible in SET
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 print(s3[0])

TypeError: 'set' object is not subscriptable
```

In [39]:
```
print(s3[:])    # Slicing not possible in SET
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 1
----> 1 print(s3[:])

TypeError: 'set' object is not subscriptable
```

In [40]:
```
s4 = s3.copy()
print(s4)
```
```
{'z', 'm', 'b', 'a'}
```

In [41]:
```
print(s3)
```
```
{'z', 'm', 'b', 'a'}
```

In [42]:
```
print(s3 == s4)
```
```
True
```

In [43]:
```
print(s4)
```
```
{'z', 'm', 'b', 'a'}
```

In [44]:
```
print(s4.clear())
```
```
None
```

In [45]:
```
print(s4)
```
```
set()
```

In [46]:
```
s4 = s2.copy()
print(s4)
```
```
{200, 9, 10, 100}
```

In [47]:
```
s4.pop()
print(s4)
```
```
{9, 10, 100}
```

In [48]:
```
print(s3)
print(s1)
```
```
{'z', 'm', 'b', 'a'}
{True, 3.2, (1+2j), 10, 20, 'nit'}
```

In [50]:
```
s1.pop(0)   # No Arguments
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[50], line 1
----> 1 s1.pop(0)   # No Arguments

TypeError: set.pop() takes no arguments (1 given)
```

In [51]:
```
print(s1)
```
```
{True, 3.2, (1+2j), 10, 20, 'nit'}
```

In [52]:
```
s1.remove((1+2j))
print(s1)
```
```
{True, 3.2, 10, 20, 'nit'}
```

In [53]:
```
for i in s1:
    print(i)
```
```
True
3.2
10
20
nit
```

In [54]:
```
for i in enumerate(s1):
    print(i)
```
```
(0, True)
(1, 3.2)
(2, 10)
(3, 20)
(4, 'nit')
```

Set Operation

In [3]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [4]:
```
A | B    # Union
```
Out[4]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [6]:
```
B.union(C)
```
Out[6]:
```
{4, 5, 6, 7, 8, 9, 10}
```

In [7]:
```
C | B | A
```
Out[7]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [8]:
```
C
```
Out[8]:
```
{8, 9, 10}
```

In [9]:
```
D = C.copy()
D
```
Out[9]:
```
{8, 9, 10}
```

In [10]:
```
print(C)
print(D)
```

```
{8, 9, 10}
{8, 9, 10}
```

In [11]:
```
C.update(D)
print(C)
```

```
{8, 9, 10}
```

In [12]:
```
C.update(B)
print(C)
```

```
{4, 5, 6, 7, 8, 9, 10}
```

In [13]:
```
print(A)
print(B)
print(C)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
{4, 5, 6, 7, 8, 9, 10}
```

In [14]:
```
print( A & B)   # Intersection
```

```
{4, 5}
```

In [17]:
```
print(B.intersection (C))
```

```
{4, 5, 6, 7, 8}
```

In [18]:
```
print( A & B & C)
```

```
{4, 5}
```

Difference

In [3]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
print(A)
print(B)
print(C)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
{8, 9, 10}
```

In [20]:
```
print(A-B)         # difference  (prints uncommon items)
```

```
{1, 2, 3}
```

In [21]:
```
print(B-C)
```

```
set()
```

In [22]:
```
print(C-B)
```

```
{9, 10}
```

In [23]:
```
C.difference(A)
```
Out[23]:
```
{6, 7, 8, 9, 10}
```

25th April 2025

In [4]:
```
print(A)
print(B)
print(C)
```
```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
{8, 9, 10}
```

In [5]:
```
A.difference(B)
```
Out[5]:
```
{1, 2, 3}
```

In [6]:
```
A.symmetric_difference(B)
```
Out[6]:
```
{1, 2, 3, 6, 7, 8}
```

In [8]:
```
s3 = {'a', 'b', 'm', 'z'}
```

In [9]:
```
s3.remove('x')
```
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[9], line 1
----> 1 s3.remove('x')

KeyError: 'x'
```

In [11]:
```
s3.discard('z')
s3
```
Out[11]:
```
{'a', 'b', 'm'}
```

superset

subset

disjoint

In [12]:
```
a9 = {1,2,3,4,5,6,7,8,9}
b9 = {3,4,5,6,7,8}
c9 = {10,20,30,40}
```

In [13]:
```
b9.issubset(a9)
```

Out[13]:
```
True
```

In [14]:
```
a9.issuperset(b9)
```

Out[14]:
```
True
```

In [15]:
```
c9.issubset(b9)
```

Out[15]:
```
False
```

In [16]:
```
c9.isdisjoint(b9)
```

Out[16]:
```
True
```

In [17]:
```
c9.isdisjoint(a9)
```

Out[17]:
```
True
```

In [18]:
```
a8 = {1,2,3,4,5,6}
b8 = {7,8,9}
c8 = {10,20,30,40}
```

In [19]:
```
a8.issuperset(b8)
```

Out[19]:
```
False
```

In [20]:
```
b8.issubset(a8)
```

Out[20]:
```
False
```

In [21]:
```
c8.isdisjoint(b8)
```

Out[21]:
```
True
```

In [22]:
```
c8.isdisjoint(a8)
```

Out[22]:
```
True
```

Dictionary Data Structure

In [23]:
```
d = {}
type(d)
```

```
Out[23]: dict

In [24]: d = {1:'one', 2:'two', 3:'three', 4:'four'}

In [25]: d.keys()

Out[25]: dict_keys([1, 2, 3, 4])

In [26]: d.values()

Out[26]: dict_values(['one', 'two', 'three', 'four'])

In [27]: d.items()

Out[27]: dict_items([(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')])

In [28]: len(d)

Out[28]: 4

In [29]: d[4]

Out[29]: 'four'

In [30]: d.get(1)

Out[30]: 'one'

In [31]: d.get('one') # wrong

In [32]: d1 = d.copy()

In [33]: d1

Out[33]: {1: 'one', 2: 'two', 3: 'three', 4: 'four'}

In [34]: d1.pop(1)

Out[34]: 'one'

In [35]: d1.popitem()

Out[35]: (4, 'four')

In [37]: for i in d:
             print(i)

1
2
3
4
```

In [38]:
```
for i in d:
    print(i,':', d[i])
```

```
1 : one
2 : two
3 : three
4 : four
```

In [39]:
```
for i in d:
    print(i,':', d)
```

```
1 : {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
2 : {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
3 : {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
4 : {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

Range

In [40]:
```
range(10)
```

Out[40]:
```
range(0, 10)
```

In [41]:
```
r = range(10)
r
```

Out[41]:
```
range(0, 10)
```

In [42]:
```
for i in r:
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [43]:
```
list(r)
```

Out[43]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [44]:
```
r1 = range(10,20)
r1
```

Out[44]:
```
range(10, 20)
```

In [45]:
```
list(r1)
```

Out[45]:
```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [46]: list(range(0,10))
```

```
Out[46]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [47]: list(range(10,20))
```

```
Out[47]: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [48]: list(range(10,100,10))
```

```
Out[48]: [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```
In [49]: list(range(10,100,10,5))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[49], line 1
----> 1 list(range(10,100,10,5))

TypeError: range expected at most 3 arguments, got 4
```

28th April 2025

# Number System 1. Binary - base value is 2 (0b) (bin) 2. Octal - base value is 8 (0o) (oct) 3. Decimal - base value is 10 (0x) (Dec) 4. HexaDecimal - base value is 0,1,2,3,4,5,6,7,8,9,a=10,b=11,c=12,d=13,e=14,f=15 (0x) (Hex) binary : base (0-1) --> please divide 15/2 & count in reverse order octal : base (0-7) hexadecimal : base (0-9 & then a-f) when you check ipaddress you will these format --> cmd - ipconfig

```
In [1]: 25
        bin(25)
```

```
Out[1]: '0b11001'
```

```
In [2]: int(0b11001)
```

```
Out[2]: 25
```

```
In [3]: bin(35)
```

```
Out[3]: '0b100011'
```

```
In [4]: bin(20)
```

```
Out[4]: '0b10100'
```

```
In [5]: int(0b10100)
```

```
Out[5]: 20
```

```
In [6]: 0b1111
```

```
Out[6]: 15
```

```
In [7]: oct(15)

Out[7]: '0o17'

In [8]: hex(9)

Out[8]: '0x9'

In [9]: 0xf

Out[9]: 15

In [10]: hex(10)

Out[10]: '0xa'

In [11]: hex(25)

Out[11]: '0x19'

In [12]: 0x15

Out[12]: 21
```

Swap variable in python

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

```
In [13]: a = 5
         b = 6

In [14]: a = b
         b = a

In [15]: a,b = b,a

In [16]: print(a)
         print(b)

6
6

In [17]: # in above scenario we lost the value 5
         a1 = 7
         b1 = 8

In [18]: temp = a1
         a1 = b1
         b1 = temp
```

In [19]:
```
print(a1)
print(b1)
```

```
8
7
```

In [22]:
```
#swap variable formulas

a2 = 5
b2 = 6

a2 = a2 + b2
b2 = a2 - b2
a2 = a2 - b2
```

In [23]:
```
print(a2)
print(b2)
```

```
6
5
```

In [24]:
```
print(0b101) # 101 is 3 bit
print(0b110) # 110 also 3bit
```

```
5
6
```

In [25]:
```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1 bit
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

In [26]:
```
#there is other way to work using swap variable also which is XOR because it will n
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [27]:
```
print(a2)
print(b2)
```

```
5
6
```

In [28]:
```
a2 , b2 = b2, a2

print(a2)
print(b2)
```

```
6
5
```

Operators

1. Arithematic 2. Assignment 3. Relational 4. Logical 5. Unary 1- ARITHMETIC OPERATOR (+ , -, *, /, %, %%, **, ^ 2- ASSIGNMEN OPERATOR (=) 3- RELATIONAL OPERATOR 4- LOGICAL OPERATOR 5- UNARY OPERATOR 6. BITWISE OPERATOR

1. Arithematic

In [30]:
```
x1, y1 = 10, 5

print(x1  + y1)
print(x1 - y1)
print(x1 * y1)
print(x1 / y1)
print(x1 // y1)
print(x1 % y1)
print(x1 ** y1)
```

```
15
5
50
2.0
2
0
100000
```

2. Assignment operator

In [31]:
```
x = 2
```

In [32]:
```
x = x + 2
x
```

Out[32]:
```
4
```

In [33]:
```
x += 2
x
```

Out[33]:
```
6
```

In [34]:
```
x += 2
x
```

Out[34]:
```
8
```

In [35]:
```
x *= 2
x
```

Out[35]:
```
16
```

In [36]:
```
x -= 2
x
```

```
Out[36]: 14
```

```
In [37]: x /= 2
         x
```

```
Out[37]: 7.0
```

```
In [38]: x
```

```
Out[38]: 7.0
```

5. Unary operator

Here we are applying unary minus operator(-) on the operand n; the value of m becomes -7, which indicates it as a negative value.

```
In [40]: n = 7 #negattion
         m = -(n)
         m
```

```
Out[40]: -7
```

3. Relational

- we are using this operator for comparing

```
In [41]: a = 5
         b = 7

         a == b
```

```
Out[41]: False
```

```
In [42]: a<b
```

```
Out[42]: True
```

```
In [43]: a>b
```

```
Out[43]: False
```

```
In [45]: a = 10
         a != b
```

```
Out[45]: True
```

```
In [46]: b = 10
         a == b
```

```
Out[46]: True
```

```
In [47]: a >= b
```

```
Out[47]: True
```

```
In [48]: a <= b
```

```
Out[48]: True
```

```
In [49]: a < b
```

```
Out[49]: False
```

```
In [50]: a>b
```

```
Out[50]: False
```

```
In [51]: b = 7

         a != b
```

```
Out[51]: True
```

4. Logical

- AND, OR, NOT

```
In [52]: a = 5
         b = 4
```

```
In [53]: a < 8 and b < 5 #refer to the truth table
```

```
Out[53]: True
```

```
In [54]: a < 8 and b < 2
```

```
Out[54]: False
```

```
In [55]: a < 8 or b < 2
```

```
Out[55]: True
```

```
In [56]: a>8 or b<2
```

```
Out[56]: False
```

```
In [57]: not x  # you can reverse the operation
```

```
Out[57]: False
```

```
In [58]: x = not x
         x
```

```
Out[58]: False
```

```
In [59]: x
```

```
Out[59]: False
```

```
In [60]: not x
```

```
Out[60]: True
```

6. Bitwise Operator

- WE HAVE 6 OPERATORS COMPLEMENT (~) || AND (&) || OR (|) || XOR (^) || LEFT SHIFT (<<) || RIGHT SHIFT (>>)

```
In [61]: print(bin(12))
         print(bin(13))
```

```
0b1100
0b1101
```

complement --> you will get this key below esc character

12 ==> 1100 || first thing we need to understand what is mean by complement. complement means it will do reverse of the binary format i.e. - ~0 it will give you 1 ~1 it will give 0 12 binary format is 00001100 (complement of ~00001100 reverse the number - 11110011 which is (-13)

but the question is why we got -13 to understand this concept (we have concept of 2's complement 2's complement mean (1's complement + 1) in the system we can store +Ve number but how to store -ve number

lets understand binary form of 13 - 00001101 + 1

```
In [62]: # COMPLEMENT (~) (TILDE  OR TILD)
         ~12 # why we get -13 . first we understand what is complment means (reversr of bina
```

```
Out[62]: -13
```

```
In [63]: ~45
```

```
Out[63]: -46
```

```
In [64]: ~6
```

```
Out[64]: -7
```

```
In [65]: ~-6
```

```
Out[65]: 5
```

```
In [66]: ~-1
```

```
Out[66]: 0
```

bit wise and operator

AND - LOGICAL OPERATOR ||| & - BITWISE AND OPERATOR
(we know that 1 & 1 is 1) 12 - 00001100 13 - 00001101 when we are add both then then outut we will get as 12

```
In [67]: 12 & 13
```

```
Out[67]: 12
```

```
In [68]: 1 & 1
```

```
Out[68]: 1
```

```
In [69]: 1 | 0
```

```
Out[69]: 1
```

```
In [70]: 1 & 0
```

```
Out[70]: 0
```

```
In [71]: 12 | 13
```

```
Out[71]: 13
```

```
In [72]: 35 & 40 #please do the homework conververt 35,40 to binary format
```

```
Out[72]: 32
```

```
In [73]: 35 | 40
```

```
Out[73]: 43
```

In [74]:
```
# in XOR if the both number are different then we will get 1 or else we will get 0

12 ^ 13
```
Out[74]:
```
1
```
In [75]:
```
25 ^ 30
```
Out[75]:
```
7
```
In [76]:
```
bin(25)
```
Out[76]:
```
'0b11001'
```
In [77]:
```
bin(30)
```
Out[77]:
```
'0b11110'
```
In [78]:
```
int(0b000111)
```
Out[78]:
```
7
```

BIT WISE LEFT OPERATOR

bit wise left operator bydefault you will take 2 zeros ( )

10 binary operator is 1010 | also i can say 1010

10<<2

In [79]:
```
20<<4 #can we do this
```
Out[79]:
```
320
```

BITWISE RIGHTSHIFT OPERATOR

In [80]:
```
10>>2
```
Out[80]:
```
2
```
In [81]:
```
bin(20)
```

```
Out[81]: '0b10100'
```

```
In [82]: 20>>4
```

```
Out[82]: 1
```

29th April 2025

# print(), Input(), Interview Questions

Basic Phython Completed