```
import sys sys.version
```

8:30 AM IST - BASIC PYTHON PROGRAMMING

Work with Numbers

```
In [3]: 2+3
Out[3]: 5

In [5]: 2
Out[5]: 2

In [9]: 3-2
Out[9]: 1

In [1]: 5*2
Out[1]: 10

In [3]: 3**2
Out[3]: 9

In [5]: 10/5
Out[5]: 2.0

In [7]: 10//5
Out[7]: 2
```

Work With STRING

```
In [11]: 'Thank You Universe'
Out[11]: 'Thank You Universe'

In [17]: "Thank You Universe"
Out[17]: 'Thank You Universe'

In [19]: '''Thank You Universe'''
Out[19]: 'Thank You Universe'
```

Variable = Object

In [44]:
```
v=5 # v=variable, 5 =value
v
```

Out[44]: 5

In [46]:
```
type(v)
```

Out[46]: int

In [31]:
```
v1=5
```

In [33]:
```
v1
```

Out[33]: 5

In [41]:
```
v1='nit'
v1 #Calling V1
```

Out[41]: 'nit'

In [45]:
```
v1='nit'
v2 #test to check how system responds
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[45], line 2
      1 v1='nit'
----> 2 v2

NameError: name 'v2' is not defined
```

In [47]:
```
v1="Thank you Kodi sir, Python seems Easy!!"
v1
```

Out[47]: 'Thank you Kodi sir, Python seems Easy!!'

Rules for Python Variables

In [52]:
```
nit=15
NIT # Python is case sensitive
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[52], line 2
      1 nit=15
----> 2 NIT

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

In [54]:
```
nit
```

Out[54]: 15

In [56]:
```
1a=67
```

```
  Cell In[56], line 1
    1a=67
    ^
SyntaxError: invalid decimal literal
```

In [60]:
```
1a=67
1a # variable never start with digits
```

```
  Cell In[60], line 1
    1a=67
    ^
SyntaxError: invalid decimal literal
```

In [62]:
```
a1=67
a1
```

Out[62]: 67

In [64]:
```
nit$=87
nit$
```

```
  Cell In[64], line 1
    nit$=87
       ^
SyntaxError: invalid syntax
```

In [66]:
```
x_train, x_test, y_train, y_test = 80, 9, 100
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[66], line 1
----> 1 x_train, x_test, y_train, y_test = 80, 9, 100

ValueError: not enough values to unpack (expected 4, got 3)
```

In [68]:
```
x_train, x_test, y_train, y_test = 80, 9, 100, 20
```

In [72]:
```
x_train
x_test
y_train
y_test # Defined 4 variables but received only 1 value
```

Out[72]: 20

In [74]:
```
print(x_train) #Print function displays all value
print(x_test)
print(y_train)
print(y_test)
```

```
80
9
100
20
```

In [1]:
```
import keyword
keyword.kwlist
```

Out[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']
```

In [3]:
```
import keyword
len(keyword.kwlist)
```

Out[3]:
```
35
```

In [9]:
```
if=90 #if is a reserve words - has special meaning
if
```

```
  Cell In[9], line 1
    if=90 #if is a reserve words - has special meaning
      ^
SyntaxError: invalid syntax
```

In [11]:
```
a10 = 78
a9 = 89
```

In [13]:
```
print(a10)
print(a9)
```

```
78
89
```

In [15]:
```
del a10
```

In [17]:
```
a10
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[17], line 1
----> 1 a10

NameError: name 'a10' is not defined
```

In [19]:
```
for = 90
```

```
  Cell In[19], line 1
    for = 90
        ^
SyntaxError: invalid syntax
```

In [23]:
```
For = 90
For
```

Out[23]:
```
90
```

In [25]:
```
a = true #This will give error as T should be capital as it is a keyword
```

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

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

In [29]:
```
b = 'true' #here true is a string value assigned to b
b
```

Out[29]:
```
'true'
```

Valid Invalid Identifier Check

In [ ]:
```

```

In [50]:
```
c = 'hi'
c #c is a variable and also an identifier that is being called
```

Out[50]:
```
'hi'
```

In [52]:
```
type(c)
```

Out[52]:
```
str
```

In [54]:
```
x@ = 3
```

```
  Cell In[54], line 1
    x@ = 3
     ^
SyntaxError: invalid syntax
```

In [64]:
```
#Integer
a = 2
type(a)
```

Out[64]: int

In [68]:
```
#Integer
a1 = 2454545
print(a1)
```

```
2454545
```

In [66]:
```
#Float
b = 2.2
type(b)
```

Out[66]: float

In [70]:
```
#Float
pi = 3.17
print(pi)
```

```
3.17
```

In [72]:
```
type(pi)
```

Out[72]: float

In [74]:
```
pi = 3.17
pi
```

Out[74]: 3.17

In [76]:
```
pi = 3.20
pi
```

Out[76]: 3.2

In [78]:
```
ABC = 50
ABC
```

Out[78]: 50

In [82]:
```
abc= 60
abc
```

Out[82]: 60

In [86]:
```
abc = 60
ABCD
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[86], line 2
      1 abc = 60
----> 2 ABCD

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

In [92]:
```
abc = 60
ABC #Output is 50 bec ABC is defined as 50 above
```

Out[92]: 50

In [90]:
```
xyz = 20000
xyz
```

Out[90]: 20000

In [94]:
```
NIT = 15000
nit1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[94], line 2
      1 NIT = 15000
----> 2 nit1

NameError: name 'nit1' is not defined
```

In [96]:
```
nIT = 20
nIT
```

Out[96]: 20

In [98]:
```
nIT = 20
nIt
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[98], line 2
      1 nIT = 20
----> 2 nIt

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

In [100…
```
cash123 = 10
cash123
```

Out[100… 10

In [102…
```
123cash = 10
123cash # Error bec it started with numbers
```

```
  Cell In[102], line 1
    123cash = 10
      ^
SyntaxError: invalid decimal literal
```

In [104…
```
1A = 5
1A
```

```
  Cell In[104], line 1
    1A = 5
    ^
SyntaxError: invalid decimal literal
```

In [106…
```
A1 = 5
A1
```

Out[106… 5

In [110…
```
ca$h = 20
ca$h
```

```
  Cell In[110], line 1
    ca$h = 20
      ^
SyntaxError: invalid syntax
```

In [112…
```
ca*h = 20
ca*h
```

```
  Cell In[112], line 1
    ca*h = 20
    ^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of
'='?
```

In [114…
```
CASH = 20
CASH
```

Out[114… 20

In [116…
```
DEF = 4
DEF
```

Out[116… 4

In [118…
```
if = 780
if #if is a keyword
```

```
  Cell In[118], line 1
    if = 780
       ^
SyntaxError: invalid syntax
```

In [121…
```
ifffffffffffffhhhhhhhhhhhlj = 56
ifffffffffffffhhhhhhhhhhhlj
```

Out[121… 56

In [125…
```
_abc_def_ghi = 20
_abc_def_ghi
```

Out[125… 20

In [131…
```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 5
aaaaaaaa #Calling but will give error as it is not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[131], line 2
      1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 5
----> 2 aaaaaaaa

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

Variable is Completed

February 27, 2025 - Data Types

INT FLOAT BOOLEAN COMPLEX STRING

In [3]:
```
i = 25 #Value without decimal
i
```

Out[3]: 25

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

Out[5]: int

In [7]:
```
print(type(i))
```

```
<class 'int'>
```

In [13]:
```
petrol = 109.50 #Value with decimal
petrol
```

Out[13]: 109.5

In [15]:
```
print(type(petrol))
```

```
<class 'float'>
```

In [17]:
```
type(petrol)
```

Out[17]: float

In [20]:
```
b = true
b # true will give error as t is small letter, it's a keyword
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 b = true
      2 b

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

In [9]:
```
b = True
b # true will not give error now
```

Out[9]: True

In [11]:
```
b1 = False
b1
```

Out[11]: False

In [28]:
```
True + False  # In memory True = 1 and False = 0 thus 0+1 = 1
```

Out[28]: 1

In [30]:
```
True - True
```

Out[30]: 0

In [32]:
```
True * False
```

Out[32]: 0

In [34]:
```
False/True
```

Out[34]: 0.0

In [36]:
```
False//True
```

Out[36]: 0

In [38]:
```
True/False
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[38], line 1
----> 1 True/False

ZeroDivisionError: division by zero
```

In [40]:
```
c1 = 10 + 20j # 10=Real part and 20=Imaginary Part
c1
```

Out[40]: (10+20j)

In [42]:
```
type(c1)
```

Out[42]: complex

In [44]:
```
c1.real # To print only real part - typec1.r(tab) real
```

Out[44]: 10.0

In [46]:
```
c1.imag # To print only imaginary part - typec1.im(tab) imag
```

Out[46]: 20.0

In [48]:
```
c1
```

Out[48]: (10+20j)

In [50]:
```
c2 = 20 +30j
```

In [7]:
```
c3 = 20 +30i #i is invalid
```

```
  Cell In[7], line 1
    c3 = 20 +30i
               ^
SyntaxError: invalid decimal literal
```

In [54]:
```
print(c1)
print(c2)
```

```
(10+20j)
(20+30j)
```

In [58]:
```
c1 + c2 #System knows that I have to add real part to real part and imaginary to
```

Out[58]: (30+50j)

In [60]:
```
c1 - c2
```

Out[60]: (-10-10j)

In [62]:
```
c2 - c1
```

Out[62]: (10+10j)

In [64]:
```
print(c1)
print(c2)
```

```
(10+20j)
(20+30j)
```

In [66]:
```
c1 * c2 #Result is -400+700j because of j which is the square root of -1
```

Out[66]: (-400+700j)

In [3]:
```
s = 'nareshIT'
s
```

Out[3]: 'nareshIT'

In [5]:
```
s = "nareshIT"
s
```

Out[5]: 'nareshIT'

In [9]:
```
s = "naresh
IT"
s
```

```
  Cell In[9], line 1
    s = "naresh
        ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [13]:
```
s = '''naresh
          IT'''
s
```

Out[13]:
```
'naresh\n          IT'
```

In [25]:
```
s = 'nareshIT'
s
```

Out[25]:
```
'nareshIT'
```

String Slicing [:]

In [25]:
```
s
```

Out[25]:
```
'nareshIT'
```

In [27]:
```
s [:]
```

Out[27]:
```
'nareshIT'
```

In [29]:
```
s[3]  #Forward Index
```

Out[29]:
```
'e'
```

In [31]:
```
s[4]  #Forward Index
```

Out[31]:
```
's'
```

In [33]:
```
s[-4]  #Backward Index
```

Out[33]:
```
's'
```

In [35]:
```
s[-1]  #Backward Index
```

Out[35]:
```
'T'
```

In [13]:
```
b1
```

Out[13]:
```
False
```

In [15]:
```
int(False)
```

Out[15]:
```
0
```

In [17]:
```
True+False
```

Out[17]:
```
1
```

In [19]:
```
False+False
```

Out[19]: 0

In [21]:
```
True #When we will cast only then we will get 1 i.e int(True)=1
```

Out[21]: True

In [27]:
```
s[1:7]
```

Out[27]: 'areshI'

In [29]:
```
s[10] #Hey Python, give me 10th index but there is no 10th index, it is only upt
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[29], line 1
----> 1 s[10]

IndexError: string index out of range
```

In [32]:
```
len(s) # len() function Calculates the total length. s is an argument here
```

Out[32]: 8

Python DataTypes are Completed

February 28, 2025 - Type Casting

In [35]:
```
int(2.5) # float value 2.5 is converted into integer
```

Out[35]: 2

In [37]:
```
int(2.3,3.0) # This will give error as we can pass only 1 argument
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 int(2.3,3.0)

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

In [39]:
```
int(True)
```

Out[39]: 1

In [41]:
```
True
```

Out[41]: True

In [43]:
```
True + True
```

Out[43]: 2

In [45]:
```
int(1+2j) # this will give error as integer cannot convert complex data type
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[45], line 1
----> 1 int(1+2j)

TypeError: int() argument must be a string, a bytes-like object or a real number,
not 'complex'
```

In [47]:
```
float(2)
```

Out[47]: 2.0

In [49]:
```
int('10')
```

Out[49]: 10

In [51]:
```
int('ten')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[51], line 1
----> 1 int('ten')

ValueError: invalid literal for int() with base 10: 'ten'
```

In [53]:
```
float(10, 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[53], line 1
----> 1 float(10, 2)

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

In [55]:
```
float(True)
```

Out[55]: 1.0

In [57]:
```
float(False)
```

Out[57]: 0.0

In [59]:
```
float(1+2j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[59], line 1
----> 1 float(1+2j)

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

In [61]:
```
float(100)
```

Out[61]: 100.0

In [63]:
```
float('hundred')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[63], line 1
----> 1 float('hundred')

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

In [65]:
```
complex(10)
```

Out[65]: (10+0j)

In [67]:
```
complex(10, 20)
```

Out[67]: (10+20j)

In [69]:
```
complex(10, 20, 30)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[69], line 1
----> 1 complex(10, 20, 30)

TypeError: complex() takes at most 2 arguments (3 given)
```

In [71]:
```
complex(10, 20, 30, 40, 50) #10, 20, 30, 40, 50 are called arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[71], line 1
----> 1 complex(10, 20, 30, 40, 50)

TypeError: complex() takes at most 2 arguments (5 given)
```

In [1]:
```
complex(2.3)
```

Out[1]: (2.3+0j)

In [3]:
```
complex(2.3,4)
```

Out[3]: (2.3+4j)

In [5]:
```
complex(True, True)
```

Out[5]: (1+1j)

In [7]:
```
complex(False, False) #0+0j=0j (Multiple choice Question) - Python is a dynamic
```

Out[7]: 0j

In [9]:
```
complex(False)
```

Out[9]: 0j

In [11]:
```
complex('10')
```

Out[11]: (10+0j)

In [15]:
```
complex('10','20')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 1
----> 1 complex('10','20')

TypeError: complex() can't take second arg if first is a string
```

March 1, 2025

In [3]:
```
bool(2) #Converting integer to boolean - True because value is 2 not 0
```

Out[3]: True

In [5]:
```
bool(0) #Converting integer to boolean - False because value is 0
```

Out[5]: False

In [7]:
```
bool(2,5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 bool(2,5)

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

In [9]:
```
bool(1.5)
```

Out[9]: True

In [11]:
```
bool(1+2j) #Complex to bool is possible
```

Out[11]: True

In [13]:
```
bool(0+0j) # Answer is false bec 0 means false
```

Out[13]: False

In [15]:
```
bool('hi')
```

Out[15]: True

In [19]:
```
bool() #Space means nothing i.e 0 so false
```

Out[19]: False

In [21]:
```
bool(*) #* is invalid datatype
```

```
  Cell In[21], line 1
    bool(*)
         ^
SyntaxError: Invalid star expression
```

In [23]:
```
string(7) #int to string
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 string(7)

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

In [25]:
```
str(7) #int to string
```

Out[25]: '7'

In [27]:
```
str(3.14)  #float to string
```

Out[27]: '3.14'

In [29]:
```
str(1+4j)  #complex to string
```

Out[29]: '(1+4j)'

In [31]:
```
str(true)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[31], line 1
----> 1 str(true)

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

In [33]:
```
str(True)
```

Out[33]: 'True'

Type casting is completed

Python Operator

Arithmetc operators (+,-,,/,//,*)

In [38]:
```
x1, y1 = 10,5
```

In [40]:
```
x1+y1 #Calling with variables
```

Out[40]: 15

In [42]:
```
x1-y1
```

Out[42]: 5

In [44]:
```
x1/y1
```

Out[44]: 2.0

In [48]:
```
x1**y1 #x1 i.e 10 with 5 power = 10*10*10*10*10
```

Out[48]: 100000

Assignment Operator - (As a beginner, Execute only 1 time, 2nd time the value will change - requires restrting the kernel)

In [167…
```
x = 2
x
```

Out[167… 2

In [169…
```
x = x + 2  #Increment - single enter press
x
```

Out[169… 4

In [171…
```
x = x + 2  #Increment - 3 times press
x
```

Out[171… 6

In [173…
```
x += 2  # += is another way of writing x+2
x
```

Out[173… 8

In [175…
```
x -= 2
x      # Generates result by performing operation on the current value of x
```

Out[175… 6

In [177…
```
x -= 4
x      # Generates result by performing operation on the current value of x - in
```

Out[177… 2

In [181…
```
x
```

Out[181… 2

In [183…
```
x *= 3  # *= means multiplication i.e 3*2=6
x
```

Out[183… 6

In [185…
```
x /= 2   #Answer will come as float division
x
```

Out[185… 3.0

In [187…
```
x //= 2
x
```

Out[187… 1.0

Unary Operator

In [ ]:
```
# Unary gives minus value -> minus is called unary operator
```

In [192…
```
n = 7
n
```

Out[192… 7

In [194…
```
m = -n
m
```

Out[194… -7

In [196…
```
n
```

Out[196… 7

In [198…
```
-n
```

Out[198… -7

Relational Operator

In [1]:
```
r1 = 5
r2 = 6
```

In [3]:
```
r1 > r2
```

Out[3]: False

In [5]:
```
r1 < r2
```

Out[5]: True

In [7]:
```
r1 == r2
```

Out[7]: False

In [9]:
```
r1 != r2
```

Out[9]: True

In [11]:
```
r3 = 6
```

In [13]:
```
r1 == r3
```

Out[13]: False

In [15]:
```
r2 == r3
```

Out[15]: True

In [18]:
```
print (r1)
print (r2)
print (r3)
```

```
5
6
6
```

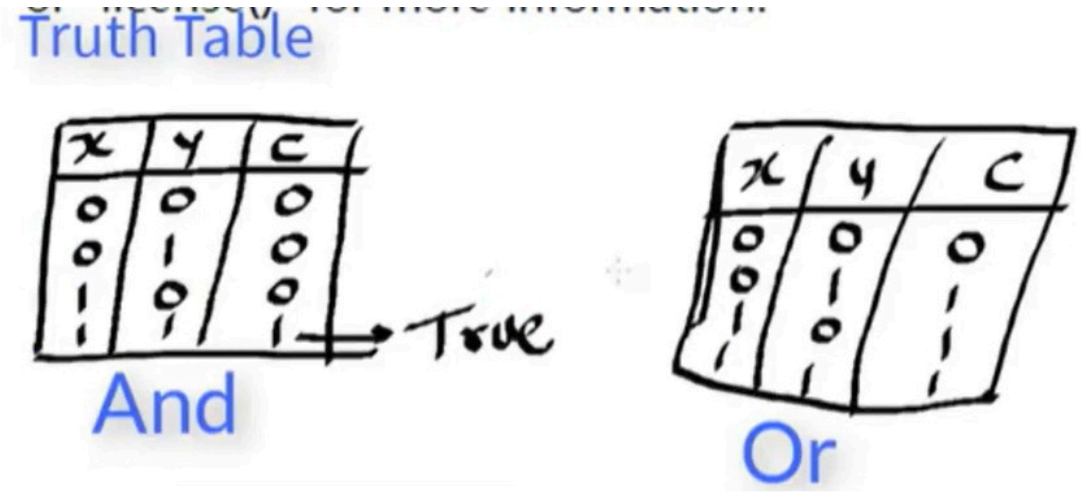
In [22]:
```
r1 >= r2
```

Out[22]: False

In [24]:
```
r1 <= r3
```

Out[24]: True

Logical Operator (And, Or, Not) - True = 1, False=0 (AND - If Both conditions are true only then answer is true, Or - If one of the condition is true answer is true)



In [27]:
```
a = 5
b = 4
```

In [29]:
```
a < 8 and b < 5  # Means 5<8 and 4<5 i.e 1 and 1 thus answer is 1 (Above image i
```

Out[29]: True

In [31]:
```
a < 8 or b < 5 # Means 5<8 or 4<5 i.e 1 or 1 thus answer is 1 (Above image is th
```

Out[31]: True

In [33]:
```
b > 5 or a < 10  # 0 OR 1 is 1
```

Out[33]: True

In [36]:
```
x = False
x
```

Out[36]: False

In [38]:
```
not x
```

Out[38]: True

In [40]:
```
y = True
y
```

Out[40]: True

In [42]:
```
not y
```

Out[42]: False

Python Operators Completed

March 3 - Python Data Structure

Data Structure - User defines more than 1 values -

- list - Always starts with [] (Square Bracket)
- tuple
- set
- **dict**

In [6]:
```
l = []  #Empty list
l
```

Out[6]: []

In [8]:
```
len(l)  #Length of l is 0 as we didn't pass any argument thus it is an empty lis
```

Out[8]: 0

l. #press tab key after l. - all built-in/Internal functions/Inbuild funtions will be displayed

In [11]:
```
l.append(10)
```

In [13]:
```
l
```

Out[13]: [10]

In [15]:
```
len(l)
```

Out[15]: 1

In [17]:
```
l.append(20)
l.append(30)
l.append(40)
l.append(50)
```

In [19]:
```
l
```

Out[19]: [10, 20, 30, 40, 50]

In [19]:
```
len(l)
```

Out[19]: 5

In [21]:
```
id(l)  #id is the address/memory location of l in the system memory (Every varia
```

Out[21]: 1805274822400

In [23]:
```
print(type(l))
```

```
<class 'list'>
```

In [25]:
```
a = 5
type(a)
```

Out[25]: int

In [27]:
```
a = 5.5
type(a)
```

Out[27]: float

In [29]:
```
a = 'happy'
type(a)
```

Out[29]: str

In [30]:
```
a = 5+5j
type(a)
```

Out[30]: complex

In [31]:
```
a = True
type(a)
```

Out[31]: bool

In [32]:
```
import keyword
keyword.kwlist
```

Out[32]:
```
['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 [33]:
```
len (keyword.kwlist) #output is 35, if we complete all these 35 keywords, our py
```

Out[33]:
```
35
```

In [34]:
```
l.append(50)
```

In [21]:
```
l
```

Out[21]:
```
[10, 20, 30, 40, 50]
```

In list duplicate numbers are allowed

List Slicing

In [38]:
```
l[:]      #l[:] - list slicing - It displays all numbers
```

Out[38]:
```
[10, 20, 30, 40, 50, 50]
```

In [39]:
```
l[1]  #Output is 20 as foward indexing starts with 0
```

Out[39]: 20

In [40]:
```
l[-1]   #Output is 50 as backward indexing starts with -1
```

Out[40]: 50

In [41]:
```
l[0]
```

Out[41]: 10

In [42]:
```
l[-3]
```

Out[42]: 40

copy()

In [23]:
```
l1 = l.copy
l1
```

Out[23]: <function list.copy()>

In [5]:
```
l1 = l.copy()
l1
```

Out[5]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [45]:
```
l == l1
```

Out[45]: True

In [46]:
```
print (len(l))
print (len(l1))
```

```
6
6
```

In [47]:
```
print(id(l))  #Address will be different of both l and l1, in datatypes it is sa
print(id(l1))
```

```
1805274822400
1805274936896
```

clear()

In [48]:
```
l1.clear()
```

In [49]:
```
l1
```

Out[49]: []

In [132…
```
l1 = l.copy()
l1
```

Out[132… [10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

append()

In [71]:
```
l1.append(2.3)
l1.append(True)
l1.append(1+2j)
```

In [72]:
```
l1
```

Out[72]: [10, 20, 30, 40, 50, 50, 2.3, True, (1+2j)]

In [73]:
```
l1.append(60) #Append/Adds object to the end of the list.
l1
```

Out[73]: [10, 20, 30, 40, 50, 50, 2.3, True, (1+2j), 60]

count()

In [74]:
```
l1.count(50) #Return number of occurrences of value. (shift+tab)
```

Out[74]: 2

In [75]:
```
l1.count(100) #Return number of occurrences of value. (shift+tab)
```

Out[75]: 0

In [76]:
```
l
```

Out[76]: [10, 20, 30, 40, 50, 50]

In [77]:
```
l1
```

Out[77]: [10, 20, 30, 40, 50, 50, 2.3, True, (1+2j), 60]

In [78]:
```
l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[78], line 1
----> 1 l2

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

In [27]:
```
l2 = l1.copy()
```

In [ ]:
```
l2
```

To remove any value use remove()

In [ ]:
```
l2.remove(True)
```

In [ ]:
```
l2
```

In [ ]:
```
l2.remove(1+2j)
```

In [ ]:
```
l2
```

clear()

In [ ]:
```
l2.clear() #deletes completes list
```

In [ ]:
```
l2
```

To delete the l2 use keyword 'del'

In [79]:
```
del l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[79], line 1
----> 1 del l2

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

In [80]:
```
l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[80], line 1
----> 1 l2

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

Variable

In [81]:
```
x = 2
```

In [82]:
```
x
```

Out[82]: 2

In [83]:
```
_ + 2
```

Out[83]: 4

Jupiter updated version thus _+2 is not working but on Google colab it will work. So, it's not a problem.

March-4 - Data Structure Continuation

- Python is an Object Oriented Programming Language, thus when file is closed all objects are cleared thus we have to run the cells again

In [86]:
```
print(l)
print(l1) # 2 variables are created in the memory i.e l and l1
```

```
[10, 20, 30, 40, 50, 50]
[10, 20, 30, 40, 50, 50, 2.3, True, (1+2j), 60]
```

In [87]:
```
print(len(l))
print(len(l1))
```

```
6
10
```

In [190…
```
l
```

Out[190…
```
[10, 20, 30, [1, 2, 3, 'hi'], [1, 2, 3, 'hi']]
```

In [138…
```
for i in l:  # for is a keyword, i is an iteration(next element) . Behind the sc
    print(i) # Thus, list is duplicated. For front end it's for keyword only.
```

```
10
20
30
40
50
50
```

In [142…
```
l.append([1,2,3,'hi']) #Nested List - Just like we have Nested for and nested if
l
```

Out[142…
```
[10, 20, 30, 40, 50, 50, [1, 2, 3, 'hi']]
```

In [144…
```
l
```

Out[144…
```
[10, 20, 30, 40, 50, 50, [1, 2, 3, 'hi']]
```

remove()

In [146…
```
l.remove(50)     # shift+tab - Remove first occurrence of value.
```

In [150…
```
l     # Output comes with first 50 removed
```

Out[150…
```
[10, 20, 30, 40, 50, [1, 2, 3, 'hi']]
```

To remove another 50, do 1 by 1. Again write l.remove(50)

In [184…
```
l  # Some digits removed because I pressed enter on l.pop() 2-3 times. Need to c
```

Out[184…
```
[10, 20, 30, [1, 2, 3, 'hi']]
```

In [170…
```
l[0]
```

Out[170…
```
10
```

In [172…
```
l[1]
```

Out[172…
```
20
```

In [174…
```
l[2]
```

Out[174…
```
30
```

In [176…
```
l[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[176], line 1
----> 1 l[3]

IndexError: list index out of range
```

In [195…
```
l  # Output is 2 list because already I ran l.append([1,2,3,'hi']) twice
```

Out[195…
```
[10, 20, 30, [1, 2, 3, 'hi'], [1, 2, 3, 'hi']]
```

In [199…
```
l[0]
```

Out[199…
```
10
```

In [201…
```
l[1]
```

Out[201…
```
20
```

In [203…
```
l[2]
```

Out[203…
```
30
```

In [205…
```
l[3]
```

Out[205…
```
[1, 2, 3, 'hi']
```

In [207…
```
l[4]
```

Out[207…
```
[1, 2, 3, 'hi']
```

In [209…
```
l
```

Out[209…
```
[10, 20, 30, [1, 2, 3, 'hi'], [1, 2, 3, 'hi']]
```

pop()

In [211…
```
l.pop()  # First it removes then prints i.e Function Removes and return item at
```

Out[211…
```
[1, 2, 3, 'hi']
```

In [213…
```
l
```

Out[213… [10, 20, 30, [1, 2, 3, 'hi']]

In [215…
```
l.pop()
```

Out[215… [1, 2, 3, 'hi']

In [217…
```
l
```

Out[217… [10, 20, 30]

INTERVIEW QUESTION - Difference between Remove() and Pop() remove()- Remove the element pop()- Remove the element Indexwise(If no argument is passed, then by default it removes the last element)

In [220…
```
l1
```

Out[220… [10, 20, 30, 40, 50, 50, 2.3, True, (1+2j), 60]

In [222…
```
l1.pop() # This will remove the last index i.e 60
```

Out[222… 60

In [224…
```
l1
```

Out[224… [10, 20, 30, 40, 50, 50, 2.3, True, (1+2j)]

In [228…
```
l1.pop() # This will remove the last index i.e (1+2j) as no argument is passed
```

Out[228… (1+2j)

In [230…
```
l1
```

Out[230… [10, 20, 30, 40, 50, 50, 2.3, True]

In [232…
```
l1.pop(-1)   # This will remove backward indexing last value i.e True
```

Out[232… True

In [234…
```
l1
```

Out[234… [10, 20, 30, 40, 50, 50, 2.3]

In [236…
```
l1.pop(3)  # This will remove the 3rd index i.e 40
```

Out[236… 40

In [238…
```
l1
```

Out[238… [10, 20, 30, 50, 50, 2.3]

In [240…
```
print(l)
print(l1)
```

```
[10, 20, 30]
[10, 20, 30, 50, 50, 2.3]
```

In [242…
```
l
```

Out[242…
```
[10, 20, 30]
```

insert()

To insert 25 between 20 and 30

In [247…
```
l.insert(25)   #"insert" function Inserts object before index and takes 2 argume
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[247], line 1
----> 1 l.insert(25)

TypeError: insert expected 2 arguments, got 1
```

In [249…
```
l.insert(2,25)   #Here 2 is the index where we want to insert the number. Syntax
```

In [251…
```
l
```

Out[251…
```
[10, 20, 25, 30]
```

In [263…
```
l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[263], line 1
----> 1 l2

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

In [9]:
```
l2 = []
```

In [267…
```
l2
```

Out[267…
```
[]
```

extend()

ex. a = 1,2 b = 3,4 a.extend(b) = 1,2,3,4

In [270…
```
l2.extend()     #Extend list by appending elements from the iterable i.e Concate
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[270], line 1
----> 1 l2.extend()

TypeError: list.extend() takes exactly one argument (0 given)
```

In [272…
```
l1
```

Out[272… [10, 20, 30, 50, 50, 2.3]

In [274…
```
l2
```

Out[274… []

In [13]:
```
l2.extend(l1)  # Asking Python to extend l1 in l2  - entered it twice thus l2 is
```

In [15]:
```
l2
```

Out[15]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [286…
```
l2.clear()    #I am recorrecting as don't know how to undo above action
```

In [288…
```
l2
```

Out[288… []

In [290…
```
l2.extend(l1)   # Asking Python to extend l1 in l2
```

In [292…
```
l2
```

Out[292… [10, 20, 30, 50, 50, 2.3]

In [294…
```
l
```

Out[294… [10, 20, 25, 30]

In [296…
```
l1
```

Out[296… [10, 20, 30, 50, 50, 2.3]

In [298…
```
l.extend(l1)  # Extending the length
```

In [300…
```
l  # Prints elements of l list and concates elements of l1 list
```

Out[300… [10, 20, 25, 30, 10, 20, 30, 50, 50, 2.3]

In [303…
```
print(l)
```

```
[10, 20, 25, 30, 10, 20, 30, 50, 50, 2.3]
```

In [305…
```
print(l1)
```

```
[10, 20, 30, 50, 50, 2.3]
```

In [307…
```
print(l2)
```

```
[10, 20, 30, 50, 50, 2.3]
```

index()

In [313…
```
l2.index(30)       #This counts the index of the element
```

Out[313… 2

In [317…
```
l2.index(50)   # By default it gives the index of first occurrence of duplicate
```
Out[317…
```
3
```
In [319…
```
l
```
Out[319…
```
[10, 20, 25, 30, 10, 20, 30, 50, 50, 2.3]
```
In [321…
```
l1
```
Out[321…
```
[10, 20, 30, 50, 50, 2.3]
```

Sort()

In [323…
```
l1.sort()  # Sorts the elements in Ascending order - Sort the list in ascending
```
In [325…
```
l1
```
Out[325…
```
[2.3, 10, 20, 30, 50, 50]
```
In [331…
```
l1.sort(reverse=True)    # Sorts the elements in Descending order - Shift+Tab
```
In [333…
```
l1
```
Out[333…
```
[50, 50, 30, 20, 10, 2.3]
```
In [339…
```
l1.sort(reverse=False) # Sorts the elements in Ascending order - Shift+Tab. Just
```
In [337…
```
l1
```
Out[337…
```
[2.3, 10, 20, 30, 50, 50]
```
In [17]:
```
l3=[3, 100, 4]
l3
```
Out[17]:
```
[3, 100, 4]
```
In [343…
```
l3.sort() # Sorts in ascending order
```
In [345…
```
l3
```
Out[345…
```
[3, 4, 100]
```
In [351…
```
l3.sort(reverse = True)  # Sorts in descending order
```
In [349…
```
l3
```
Out[349…
```
[100, 4, 3]
```
In [353…
```
l4 = [3, 5.6, 'a', 1+2j]
```
In [355…
```
l4
```

Out[355… [3, 5.6, 'a', (1+2j)]

In [359…
```
l4.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[359], line 1
----> 1 l4.sort()

TypeError: '<' not supported between instances of 'str' and 'float'
```

Error because in Sorting only 1 datatype is allowed. If int, then only int etc.

In [362…
```
l5 = ['l', 'a', 'm', 'z', 'k']
```

In [364…
```
l5
```

Out[364… ['l', 'a', 'm', 'z', 'k']

In [366…
```
l5.sort()
```

In [368…
```
l5
```

Out[368… ['a', 'k', 'l', 'm', 'z']

In [370…
```
l1
```

Out[370… [2.3, 10, 20, 30, 50, 50]

In [372…
```
l1.reverse()
```

In [374…
```
l1
```

Out[374… [50, 50, 30, 20, 10, 2.3]

In [376…
```
l1.reverse()
```

In [378…
```
l1
```

Out[378… [2.3, 10, 20, 30, 50, 50]

In [380…
```
l
```

Out[380… [10, 20, 25, 30, 10, 20, 30, 50, 50, 2.3]

In [384…
```
l[::-1] # Another method for reverse listing
```

Out[384… [2.3, 50, 50, 30, 20, 10, 30, 25, 20, 10]

March 5 - List Continuation

In [29]:
```
print(l)
print(l1)
print(l2)
```

```
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```

String List Slicing (Note - String is a Datatype)

In [34]:
```
s1 = 'nit'
```

In [36]:
```
s1[0]
```

Out[36]: 'n'

In [38]:
```
s1[1]
```

Out[38]: 'i'

In [40]:
```
s1[2]
```

Out[40]: 't'

In [42]:
```
s1[3]  # Gives error as there is no more index after 0, 1, 2
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[42], line 1
----> 1 s1[3]

IndexError: string index out of range
```

To print all 'nit' together

In [46]:
```
for i in s1:
    print(i)
```

```
n
i
t
```

In [48]:
```
s1
```

Out[48]: 'nit'

List Slicing - Useful for dealing with datasets in Machine learning

- Left Index - Starts with 0

- Right Index - Starts with n-1

In [51]:
```
print(l1)
```

```
[10, 20, 30, 40, 50]
```

In [53]:
```
print(l)
```

```
[10, 20, 30, 40, 50]
```

In [3]:
```
l=[10, 20, 30, 40, 50, 60, 100, 8, 0, 1]
```

In [57]:
```
print(l)
```

```
[10, 20, 30, 40, 50, 60, 100, 8, 0, 1]
```

In [59]:
```
l[:]
```

Out[59]:
```
[10, 20, 30, 40, 50, 60, 100, 8, 0, 1]
```

In [61]:
```
l[0:8]  # here 8 is a right index i.e n-1
```

Out[61]:
```
[10, 20, 30, 40, 50, 60, 100, 8]
```

In [65]:
```
l[3:] # here 3 is a left index i.e starting with 0 thus 3 means starting with 40
```

Out[65]:
```
[40, 50, 60, 100, 8, 0, 1]
```

In [67]:
```
l
```

Out[67]:
```
[10, 20, 30, 40, 50, 60, 100, 8, 0, 1]
```

In [71]:
```
l[:7] # list of slice 7 means n-1 i.e uptil 6th index
```

Out[71]:
```
[10, 20, 30, 40, 50, 60, 100]
```

In [73]:
```
l
```

Out[73]:
```
[10, 20, 30, 40, 50, 60, 100, 8, 0, 1]
```

In [78]:
```
l[0:5] # Means from 0th index to (n-1)th index i.e 0th to 4th Index
```

Out[78]:
```
[10, 20, 30, 40, 50]
```

In [84]:
```
#Advance Python
l[0:20:5]  # It will print 0th index, then count by 5 steps, the value will be p
```

Out[84]:
```
[10, 60]
```

In [86]:
```
# Increasing the elements in l to retest above
l=[10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]
l
```

Out[86]:
```
[10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]
```

In [88]:
```
l[0:20:5] # It will print 0th index, then count by 5 steps
```

Out[88]: [10, 60, 120, 31]

In [92]:
```
l[3:10:3] # It will print by 3rd index, then count by 3 steps. Count the indexin
```

Out[92]: [40, 100, 1]

In [94]:
```
l[::-1] #It will print backward index, elements in reverse.Ex. to print sales in
```

Out[94]: [20, 31, 71, 55, 11, 90, 120, 1, 0, 8, 100, 60, 50, 40, 30, 20, 10]

In [98]:
```
l
```

Out[98]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

In [100…
```
l[::-2] #::-2 means first it will print the last index, then -2th value from bac
```

Out[100… [20, 71, 11, 120, 0, 100, 50, 30, 10]

In [102…
```
l
```

Out[102… [10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

In [104…
```
l[::-3] #::-3 means first it will print the last index, then -3rd value from bac
```

Out[104… [20, 55, 120, 8, 50, 20]

In [106…
```
l
```

Out[106… [10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

Mutable/Hashable

In [136…
```
l1
```

Out[136… [10, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

In [114…
```
l1[0]
```

Out[114… 10

In [138…
```
l1[0]=45
l1
```

Out[138… [45, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

Thus, the 0th index value of l1 is changed, this concept is called mutable/hashable

In [140…
```
l1
```

Out[140… [45, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 20]

In [142…
```
l1[-1]='nit'  #This changes the last index to string
l1
```

Out[142… [45, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 'nit']

In [144…
```
l1[-1][0]  #This is called Nested String Slicing.
```

Out[144… 'n'

In [148…
```
print(l1[-1][0])
print(l1[-1][1])
print(l1[-1][2])
```

```
n
i
t
```

Concatenation

In [1]:
```
l
```

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

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

In [152…
```
l1
```

Out[152… [45, 20, 30, 40, 50, 60, 100, 8, 0, 1, 120, 90, 11, 55, 71, 31, 'nit']

In [154…
```
l2
```

Out[154… [10, 20, 30, 40, 50]

In [158…
```
l3 = l1+l2  # This is called Concatenation. List is Mutable(changeable) thus we
```

In [160…
```
l3
```

Out[160…
```
[45,
 20,
 30,
 40,
 50,
 60,
 100,
 8,
 0,
 1,
 120,
 90,
 11,
 55,
 71,
 31,
 'nit',
 10,
 20,
 30,
 40,
 50]
```

In [162…
```
len(l1)
```

Out[162… 17

In [164…
```
len(l2)
```

Out[164… 5

In [166…
```
len(l3)
```

Out[166… 22

In [19]:
```
l
```

Out[19]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [21]:
```
l1
```

Out[21]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [23]:
```
l2
```

Out[23]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [25]:
```
l3
```

Out[25]: [3, 100, 4]

List Membership

In [27]:
```
10 in l1 # We are asking Python if 10 exists in l1 list i.e Is 10 a family membe
```

Out[27]: True

In [29]:
```
400 in l1 # We are asking Python if 400 is a part of l1 family?
```

Out[29]: False

Answer is false as in Memory location l1 does not have 400

ENumerate

In [38]:
```
l1
```

Out[38]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [54]:
```
for i in l1:
    print (i)
```

```
10
20
30
40
50
60
100
8
0
1
```

In [56]:
```
for i in enumerate(l1):
    print (i)
```

```
(0, 10)
(1, 20)
(2, 30)
(3, 40)
(4, 50)
(5, 60)
(6, 100)
(7, 8)
(8, 0)
(9, 1)
```

Shift+Tab by clicking on ENumerate -

Return an enumerate object.

iterable an object supporting iteration

The enumerate object yields pairs containing a count (from start, which defaults to zero) and a value yielded by the iterable argument.

enumerate is useful for obtaining an indexed list: (0, seq[0]), (1, seq[1]), (2, seq[2]), ...

In [59]:
```
l1
```

Out[59]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [61]:
```
l2
```

Out[61]: [10, 20, 30, 40, 50, 60, 100, 8, 0, 1]

In [63]:
```
l3
```

Out[63]: [3, 100, 4]

In [67]:
```
all(l3) #If we don't have 0 then all() function prints True
```

Out[67]: True

In [71]:
```
all(l2) #If we have 0 in the list then all() function prints False
```

Out[71]: False

In [73]:
```
any(l3)
```

Out[73]: True

In [75]:
```
any(l2)  #If we have 0 in the list then also any() function prints True, (Rememb
```

Out[75]: True

List Data structure is completed

March 6 - Tuple

In [1]:
```
t = () # Tuple always starts with ()
```

In [3]:
```
t
```

Out[3]: ()

In [4]:
```
print(type(t))
```

```
<class 'tuple'>
```

In [1]:
```
t1=(10, 20,30,40,40)
```

In [6]:
```
t1
```

Out[6]: (10, 20, 30, 40, 40)

Thus, 2 variables created - t and t1 and 5 values passed in t1

In [19]:
```
len(t1)
```

Out[19]: 5

To Know Internal Functions of t1 press .tab

In [22]:
```
t1.count() # Tuple has only 2 functions
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 t1.count()

TypeError: tuple.count() takes exactly one argument (0 given)
```

In [24]:
```
t1.count(10) # Count tells how many times the argument passed is repeated
```

Out[24]: 1

In [26]:
```
t1.count(40) # Count tells how many times the argument passed is repeated
```

Out[26]: 2

In [28]:
```
t1.index(20) # Index tells the index of the argument passed
```

Out[28]: 1

In [30]:
```
l5 = ['a','b','c','d']
l5
```

Out[30]: ['a', 'b', 'c', 'd']

In [32]:
```
l5[1]=10  #Index 1 will be replaced by 10
```

In [34]:
```
l5  # List is mutable/hashable/changeable
```

Out[34]: ['a', 10, 'c', 'd']

In [36]:
```
t2 = (100, 3.4, 'nit', True, 1+2j, [1,2,3], (5,6,7))
t2
```

Out[36]: (100, 3.4, 'nit', True, (1+2j), [1, 2, 3], (5, 6, 7))

In [38]:
```
print(t)
print(t1)
print(t2)
```

```
()
(10, 20, 30, 40, 40)
(100, 3.4, 'nit', True, (1+2j), [1, 2, 3], (5, 6, 7))
```

In [40]:
```
t1
```

Out[40]: (10, 20, 30, 40, 40)

In [42]:
```
t1[0]
```

Out[42]: 10

In [44]:
```
t1[0]=1000
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 t1[0]=1000

TypeError: 'tuple' object does not support item assignment
```

Tuple is Immutable

In [47]:
```
icici = (5467889, 'EKXS1234', 05042025, 8800116778) #Adhaar, PAN number, Date, p
icici
```

```
  Cell In[47], line 1
    icici = (5467889, 'EKXS1234', 05042025, 8800116778) #Adhaar, PAN number, Dat
e, phone number
                                        ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an
0o prefix for octal integers
```

Decimal Integer can't start with 0

In [60]:
```
icici = (5467889, 'EKXS1234', 5042025, 8800116778) #Adhaar, PAN number, Date, ph
icici
```

Out[60]: (5467889, 'EKXS1234', 5042025, 8800116778)

In [67]:
```
icici[0]=12357  # Someone else's Adhaar number cannot be exchanged in your tuple
icici
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[67], line 1
----> 1 icici[0]=12357  # Someone else's Adhaar number cannot be exchanged in you
r tuple
      2 icici

TypeError: 'tuple' object does not support item assignment
```

As Tuple is Immutable, thus we cannot change it's Value. In Banks, our detail is stored in the form of Tuple data structure

In [70]:
```
t1
```

Out[70]: (10, 20, 30, 40, 40)

In [77]:
```
t4 = t1*3 #Tuple's value can be duplicated, but not changed. t1*3 prints t1 tupl
```

In [79]:
```
t4
```

Out[79]: (10, 20, 30, 40, 40, 10, 20, 30, 40, 40, 10, 20, 30, 40, 40)

Tuple is Completed

In [81]:
```
t4[:]
```

Out[81]: (10, 20, 30, 40, 40, 10, 20, 30, 40, 40, 10, 20, 30, 40, 40)

In [85]:
```
t1[:7]   #In t1 there are only 5 values and not 7 still slicing gives all values
```

Out[85]: (10, 20, 30, 40, 40)

In [87]:
```
t1[2:]
```

Out[87]: (30, 40, 40)

In [89]:
```
t1[0:10:2] #gives right index value then skips by 2 elements
```

Out[89]: (10, 30, 40)

In [91]:
```
t1.remove(30)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[91], line 1
----> 1 t1.remove(30)

AttributeError: 'tuple' object has no attribute 'remove'
```

Thus, we cannot remove element from Tuple

In [94]:
```
t1.add(30)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[94], line 1
----> 1 t1.add(30)

AttributeError: 'tuple' object has no attribute 'add'
```

Thus, we cannot add element in Tuple

In [97]:
```
t2
```

Out[97]: (100, 3.4, 'nit', True, (1+2j), [1, 2, 3], (5, 6, 7))

In [99]:
```
t2.index(1) #Index gives int value
```

Out[99]: 3

In [101…
```
t2.index(100)
```

Out[101… 0

In [103…
```
t2.index('nit')
```

Out[103… 2

In [5]:
```
for i in t1:
    print(i)
```

```
10
20
30
40
40
```

In [12]:
```
for i in enumerate(t1):
    print(i)
```

```
(0, 10)
(1, 20)
(2, 30)
(3, 40)
(4, 40)
```

March 7 - Set Introduce

Set

In [20]:
```
s={}
s
```

Out[20]: {}

In [22]:
```
type(s)
```

Out[22]: dict

This means that both set & dict data structure are defined with {}

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

Out[26]: set

In [28]:
```
s1
```

Out[28]: set()

In [32]:
```
s2 = {20, 100, 3, 45}
s2                    # Set gives output in ordered way if similar data types ar
```

Out[32]: {3, 20, 45, 100}

In [34]:
```
s3 = {'z', 'l', 'c', 'e', 'f'}
s3                    # Set gives output in ordered way if similar data types ar
```

Out[34]: {'c', 'e', 'f', 'l', 'z'}

In [36]:
```
s4 = {1, 2.3, 'nit', 1+2j, [1,2,3], (4,5,6), True}
s4  # All diff datatypes added - int, floar, string, complex, list, tuple and bo
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[36], line 1
----> 1 s4 = {1, 2.3, 'nit', 1+2j, [1,2,3], (4,5,6), True}
      2 s4

TypeError: unhashable type: 'list'
```

In [38]:
```
s5 = {2, 3.4, 'nit', 1+2j, False}
s5    #Mix datatype - We can't say it is ordered or inordered
```

Out[38]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

In [48]:
```
print(s1)
print(s2)
print(s3)
print(s5)
```

```
set()
{45, 3, 100, 20}
{'c', 'f', 'l', 'z', 'e'}
{False, 'nit', 2, 3.4, (1+2j)}
```

print(s4) gives error as it was not defined correctly - s4 = {1, 2.3, 'nit', 1+2j, [1,2,3], (4,5,6), True} s4 # All diff datatypes added - int, floar, string, complex, list, tuple and bool.

In [52]:
```
s2
```

Out[52]:
```
{3, 20, 45, 100}
```

In [54]:
```
s2.add(30)    # click .tab for functions to display
```

In [56]:
```
s2  # add function added the value 30 at the exact position in the order
```

Out[56]:
```
{3, 20, 30, 45, 100}
```

In [58]:
```
s2.add(200)  #Add an element to a set.(shift+tab)
```

In [60]:
```
s2
```

Out[60]:
```
{3, 20, 30, 45, 100, 200}
```

Thus, add function adds the number randomly in the right sequence.

In [63]:
```
s2[:]  #s2 of empty slice
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[63], line 1
----> 1 s2[:]

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

In [65]:
```
s2[1:5]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[65], line 1
----> 1 s2[1:5]

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

In [67]:
```
s5
```

Out[67]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

In [69]:
```
s4=s5.copy()   #Do not execute this code twice
s4
```

Out[69]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

To add 2, two times

In [74]:
```
s4.add(2)
s4
```

Out[74]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

In [76]:
```
s5
```

Out[76]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

In [78]:
```
s5.clear()  #Shift+tab -> Docstring: Remove all elements from this set. (Docstri
```

In [80]:
```
s5
```

Out[80]:
```
set()
```

In [86]:
```
del s5 #Deleting s5
```

In [88]:
```
s5  #This will give error as s5 is deleted
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[88], line 1
----> 1 s5

NameError: name 's5' is not defined
```

In [90]:
```
s4
```

Out[90]:
```
{(1+2j), 2, 3.4, False, 'nit'}
```

In [92]:
```
s4.remove((1+2j))  #Remove an element from a set; it must be a member.
```

In [94]:
```
s4
```

Out[94]:
```
{2, 3.4, False, 'nit'}
```

Deleting 2 arguments

In [97]:
```
s4.remove(False, 'nit') #set.remove() takes exactly one argument. This will give
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[97], line 1
----> 1 s4.remove(False, 'nit')

TypeError: set.remove() takes exactly one argument (2 given)
```

In [99]:
```
s3
```

Out[99]:
```
{'c', 'e', 'f', 'l', 'z'}
```

In [101…
```
s3.discard('m')  #There is no 'm' in s3. 'm' is not a family. Thus, discard func
```

In [103…
```
s3
```

Out[103…
```
{'c', 'e', 'f', 'l', 'z'}
```

In [107…
```
s3.remove('m')  #There is no 'm' in s3. 'm' is not a family of s3. Thus, remove
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[107], line 1
----> 1 s3.remove('m')

KeyError: 'm'
```

In [109…
```
s3.discard('f')  #'f' is a family of s3. Thus, discard function removes it in th
```

In [111…
```
s3
```

Out[111…
```
{'c', 'e', 'l', 'z'}
```

s3.pop() # pop() function in Set picks random elements and delete it. Remove and return an arbitrary set element. It is not like pop() in list that removes last elements.

In [115…
```
s3
```

Out[115…
```
{'e', 'l', 'z'}
```

In [117…
```
s2
```

Out[117…
```
{3, 20, 30, 45, 100, 200}
```

In [119…
```
s2.pop(3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[119], line 1
----> 1 s2.pop(3)

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

Indexing is not allowed in Set, thus error displayed when 3 is passed as argument.

In [122…
```
s2.pop()
```

Out[122…
```
3
```

In [128…
```
s2
```

Out[128…
```
{20, 30, 45, 100, 200}
```

In [126…
```
for i in s2:
    print(i)
```

```
100
200
45
20
30
```

In [130…
```
for i in enumerate(s2):
    print(i)
```

```
(0, 100)
(1, 200)
(2, 45)
(3, 20)
(4, 30)
```

Set Membership

In [133…
```
s2
```

Out[133…
```
{20, 30, 45, 100, 200}
```

In [141…
```
5 in s2  #False as 5 is not present in s2
```

Out[141…
```
False
```

In [143…
```
45 in s2  #True as 45 is present in s2
```

Out[143…
```
True
```

In [145…
```
s2
```

Out[145…
```
{20, 30, 45, 100, 200}
```

In [147…
```
s2.update(s3)   #Update a set with the union of itself and others. Just like ext
```

In [149…
```
s2  # s2 is updated with s3 contents
```

Out[149…
```
{100, 20, 200, 30, 45, 'e', 'l', 'z'}
```

In [151…
```
s3
```

Out[151…
```
{'e', 'l', 'z'}
```

Set Operations

In [154…
```
s6 = {1,2,3,4,5}
s7 = {4,5,6,7,8}
s8 = {8,9,10}
```

In [156…
```
s6.union(s7)  #Union() -> Return the union of sets as a new set. Union() doesn't
```

Out[156… {1, 2, 3, 4, 5, 6, 7, 8}

In [158…
```
s6.union(s7, s8)  #We can pass 2 arguments in Union()
```

Out[158… {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In [161…
```
s6 | s7 #Another method to print Union. We write s6 (pipe sign |) s7
```

Out[161… {1, 2, 3, 4, 5, 6, 7, 8}

In [163…
```
s6 | s7 | s8  #Union is just like concatenation
```

Out[163… {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In [165…
```
print(s6)
print(s7)
print(s8)
```

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

In [167…
```
s6.intersection(s7) #intersection prints only common element
```

Out[167… {4, 5}

In [169…
```
s6.intersection(s8) # answer will be empty set as nothing in common
```

Out[169… set()

In [171…
```
s7.intersection(s8)
```

Out[171… {8}

In [173…
```
s6 & s7  #Another method to print intersection. We write s6 (Ampersand sign &) s
```

Out[173… {4, 5}

In [175…
```
print(s6)
print(s7)
print(s8)
```

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

In [181…
```
s6.difference(s7) #Removes common element, print rest of s6
```

Out[181… {1, 2, 3}

In [183…
```
s7.difference(s6) #Removes common element, print rest of s7
```

Out[183… {6, 7, 8}

In [185…
```
s6 - s7  #Another method to print difference. We write s6 (minus sign -) s7
```

Out[185… {1, 2, 3}

In [187…
```
s7 - s8
```

Out[187… {4, 5, 6, 7}

In [189…
```
s8 - s7
```

Out[189… {9, 10}

In [191…
```
print(s6)
print(s7)
print(s8)
```

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

In [195…
```
s6.symmetric_difference(s7)  #Return the symmetric difference of two sets as a n
```

Out[195… {1, 2, 3, 6, 7, 8}

In [197…
```
s10 = {50, 4, 3, 10}
s10  # When we write s10 and execute, only then set prints in ordered format
```

Out[197… {3, 4, 10, 50}

In [199…
```
print(s10)  # When we write print(s10), then set is not printed in ordered forma
```

```
{10, 3, 50, 4}
```

In [201…
```
print(s10)
```

```
{10, 3, 50, 4}
```

March 10, 2025

- Superset (Ex. Dad)
- Subset (Ex. Child)
- Disjoint (Ex. Neighbour)

In Python, there is an OOPs concept called Inheritance. In Inheritance we have -

- Superclass (Parent class)
- Subclass (Child class)

In [3]:
```
s11 = {1,2,3,4,5,6,7,8,9}  #superset
s12 = {3,4,5,6,7,8}        #subset
s13 = {10,20,30,40}        #neighbour
```

In [5]:
```
s12.issubset(s11)  #s12 is a child, subset
```

Out[5]: True

In [7]:
```
s11.issubset(s12)  #s11 is a parent, superset
```

Out[7]: False

In [9]:
```
s11.issuperset(s12)  #s11 is a parent, superset
```

Out[9]: True

In [11]:
```
s13.isdisjoint(s12)
```

Out[11]: True

In [13]:
```
s13.isdisjoint(s11)
```

Out[13]: True

In [15]:
```
s12 = {1,2,3,4,5}
s13 = {10, 20, 30}
s14 = {15, 25, 35}
```

In [19]:
```
s13.issubset(s12) #False bec no element of s12 is found in s13
```

Out[19]: False

In [21]:
```
s12.issuperset(s13) #False bec no element of s13 is found in s12
```

Out[21]: False

In [25]:
```
s14.isdisjoint(s12) #True bec elements are not common
```

Out[25]: True

In [1]:
```
s15 = {1, 2, 3, 4, 5}
s16 = {4, 5, 6}
s17 = {10, 20}
```

In [3]:
```
s16.issubset(s15)  #Answer is false bec 6 is absent in s15.
```

Out[3]: False

If all elements of superset are present in subset, only then it prints True

In [5]:
```
s17.isdisjoint(s15)
```

Out[5]: True

In [8]:
```
s17.isdisjoint(s16)
```

Out[8]:
```
True
```

In [10]:
```
s15
```

Out[10]:
```
{1, 2, 3, 4, 5}
```

Enumerate

In [21]:
```
for i in s15:
    print(i)
```

```
1
2
3
4
5
```

In [27]:
```
for i in enumerate(s15):
    print(i)   # Here (0, 1) is an OBJECT and NOT INDEX
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
```

In [29]:
```
s15
```

Out[29]:
```
{1, 2, 3, 4, 5}
```

In [31]:
```
sum(s15)
```

Out[31]:
```
15
```

In [33]:
```
min(s15)
```

Out[33]:
```
1
```

In [35]:
```
max(s15)
```

Out[35]:
```
5
```

In [37]:
```
len(s15)
```

Out[37]:
```
5
```

In [43]:
```
sorted(s15)  #Ascending order
```

Out[43]:
```
[1, 2, 3, 4, 5]
```

In [45]:
```
sorted(s15, reverse = True)  #Descending order
```

Out[45]:
```
[5, 4, 3, 2, 1]
```

Set is Completed

Dictionary

- Dictionary is a mutable data type in Python.
- A python dictionary is a collection of key and value pairs separated by a colon (:) & enclosed in curly braces {}.
- Keys must be unique in a dictionary, duplicate values are allowed.

In [49]:
```
d={}
```

In [ ]:
```
d
```

In [51]:
```
type(d)
```

Out[51]:
```
dict
```

In [67]:
```
mydict = dict() # empty dictionary
mydict
```

Out[67]:
```
{}
```

In [69]:
```
mydict = {} # empty dictionary
mydict
```

Out[69]:
```
{}
```

In [57]:
```
d1 = {1 : "one", 2 : "two", 3 : "three"}  # dictionary with integer keys
d1
```

Out[57]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [59]:
```
d1.keys
```

Out[59]:
```
<function dict.keys>
```

In [61]:
```
d1.keys()  #Calling d1 keys
```

Out[61]:
```
dict_keys([1, 2, 3])
```

In [63]:
```
d1.values()  #Calling d1 values
```

Out[63]:
```
dict_values(['one', 'two', 'three'])
```

In [65]:
```
d2 = d1.copy()
d2
```

Out[65]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [71]:
```
d1.items()
```

Out[71]:
```
dict_items([(1, 'one'), (2, 'two'), (3, 'three')])
```

In [73]:
```
d1[1]
```

Out[73]:
```
'one'
```

In [77]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[77]:
```
{'d': [10, 20, 30], 'b': [10, 20, 30], 'c': [10, 20, 30], 'a': [10, 20, 30]}
```

In [79]:
```
keys = {'kirti' , 'b' , 'c' , 'd'}
value = [10000,20000,30000]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[79]:
```
{'d': [10000, 20000, 30000],
 'b': [10000, 20000, 30000],
 'kirti': [10000, 20000, 30000],
 'c': [10000, 20000, 30000]}
```

In [79]:
```
keys = {'kirti' , 'b' , 'c' , 'd'}
value = [10000,20000,30000]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[79]:
```
{'d': [10000, 20000, 30000],
 'b': [10000, 20000, 30000],
 'kirti': [10000, 20000, 30000],
 'c': [10000, 20000, 30000]}
```

In [81]:
```
value.append(40000)
mydict3
```

Out[81]:
```
{'d': [10000, 20000, 30000, 40000],
 'b': [10000, 20000, 30000, 40000],
 'kirti': [10000, 20000, 30000, 40000],
 'c': [10000, 20000, 30000, 40000]}
```

Accessing Items

In [86]:
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[86]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [88]:
```
mydict[1] # Access item using key
```

Out[88]:
```
'one'
```

In [90]:
```
mydict[2] # Access item using key
```

Out[90]:
```
'two'
```

Add, Remove & Change Items

In [99]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki', '
mydict1
```

Out[99]:
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1991,
 'Address': 'Hilsinki',
 'Job': 'Developer'}
```

In [101…
```
mydict1['DOB'] = 1992 # Changing Dictionary Items - As Dict is Mutable
mydict1['Address'] = 'Delhi'
mydict1
```

Out[101…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1992,
 'Address': 'Delhi',
 'Job': 'Developer'}
```

In [103…
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

Out[103…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```

Loop through a Dictionary

In [106…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict1
```

Out[106…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [110…
```
for i in mydict1:
    print(i , ':' , mydict1[i]) # Key & value pair
```

```
Name : Asif
ID : 12345
DOB : 1991
Address : Hilsinki
```

In [112…
```
for i in mydict1:
    print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
```

Dictionary Membership

In [115…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[115… {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}

In [117…
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[117… True

In [119…
```
'Asif' in mydict1 # Membership test can be only done for keys.
```

Out[119… False

Dictionary Membership

In [3]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[3]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}

In [5]:
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[5]: True

In [7]:
```
'Address' in mydict1
```

Out[7]: False

All / Any

- The all() method returns:
- True - If all all keys of the dictionary are true
- False - If any key of the dictionary is false
- The any() function returns True if any key of the dictionary is True. If not, any() returns False.

Range

- We can pass 1 or maximum 3 arguments in range function

In [2]:
```
range(10)
```

Out[2]: range(0, 10)

In [4]:
```
list(range(0,10))  # Output is 0 to 9 as range follows n-1 indexing
```

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

In [6]:
```
list(range(10,20))
```

Out[6]: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [10]:
```
list(range(10,20, 3))
```

Out[10]:
```
[10, 13, 16, 19]
```

In [18]:
```
list(range(10,20, 3, 4))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 1
----> 1 list(range(10,20, 3, 4))

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

In [22]:
```
r = range(1, 10)
r
```

Out[22]:
```
range(1, 10)
```

In [24]:
```
for i in r:
    print(i)
```

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

Python - Topics Completed

- Operators
- Variables
- Data Types
- Type casting or type conversion
- Inbuilt datastructure

March 22'2025 - Advanced Python

In [3]:
```
if True :  #Indentation is always 4 spaces
    print('Data Science')
```

```
Data Science
```

In [5]:
```
if True :  #Indentation is always 4 spaces
    print('Data Science')
    print('need to spend everyday 3-4 hours')
```

```
Data Science
need to spend everyday 3-4 hours
```

In [15]:
```
if False :  #Indentation is always 4 spaces
    print('Data Science')

print('Goodbye')

# Output is just goodbye, bec if False+print('Data Science') is considered as 1
# and print('Goodbye') as another block. That's why we give 4 spaces.
```

```
Goodbye
```

In [19]:
```
if True :  #Indentation is always 4 spaces
    print('Data Science')

print('Goodbye')

# Output is Data Science and goodbye, thus it's an error as only one should be p
```

```
Data Science
Goodbye
```

In [21]:
```
if True :  #Indentation is always 4 spaces
    print('Data Science')
else :
    print('Goodbye')
```

```
Data Science
```

In [25]:
```
if False :  #Indentation is always 4 spaces
    print('Data Science')
else :
    print('Goodbye')
```

```
Goodbye
```

Print only EVEN Number

In [33]:
```
# Numbers that are divisible by 2 are called Even numbers

x = 4                    # x and r are python object/variable
r = x % 2                #r means remainder

if (r == 0) :
    print('Even Number')
```

```
Even Number
```

In [45]:
```
x = 5                    # x and r are python object/variable
r = x % 2                #r means remainder

if (r == 0) :
    print('Even Number')

#No Output as remainder is 1
```

In [55]:
```
x = 5                    # x and r are python object/variable
r = x % 2                #r means remainder

if (r == 0) :
    print('Even Number')
```

```
if (r == 1) :
    print('Odd Number')

#Ideally in company we can't write multiple if statements like this
# ***DO NOT WRITE THIS CODE**
```

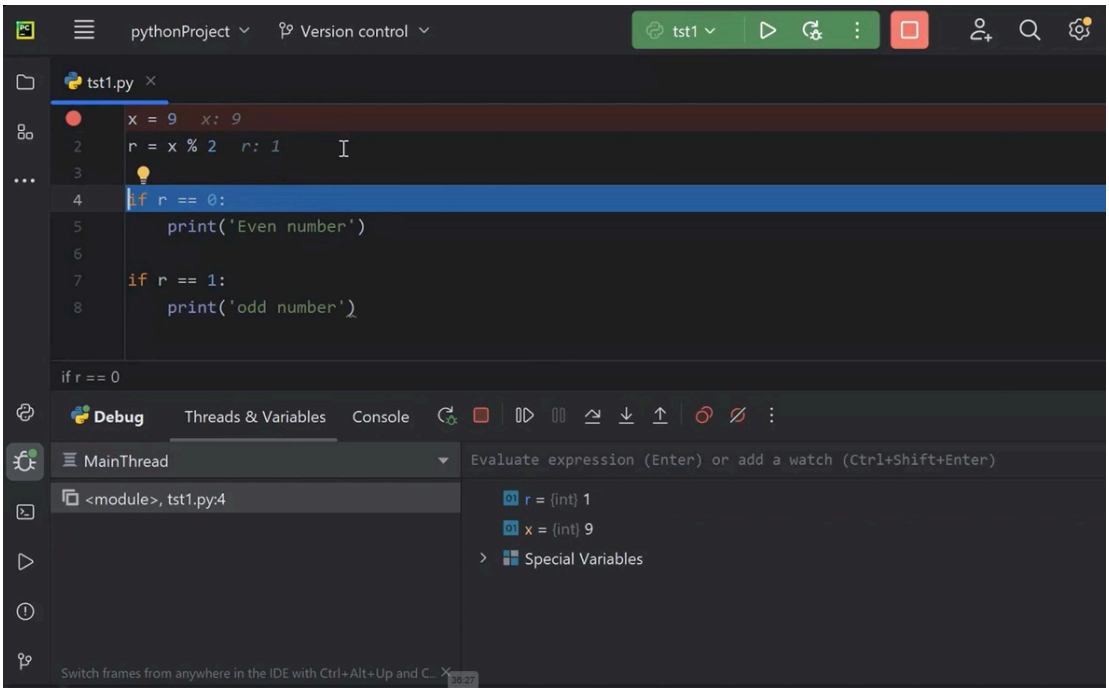
```
Odd Number
```

In [61]:
```
x = 5                    # x and r are python object/variable
r = x % 2                #r means remainder

if (r == 0) :
    print('Even Number')

else :
    print('Odd Number')

#Write this Code
```

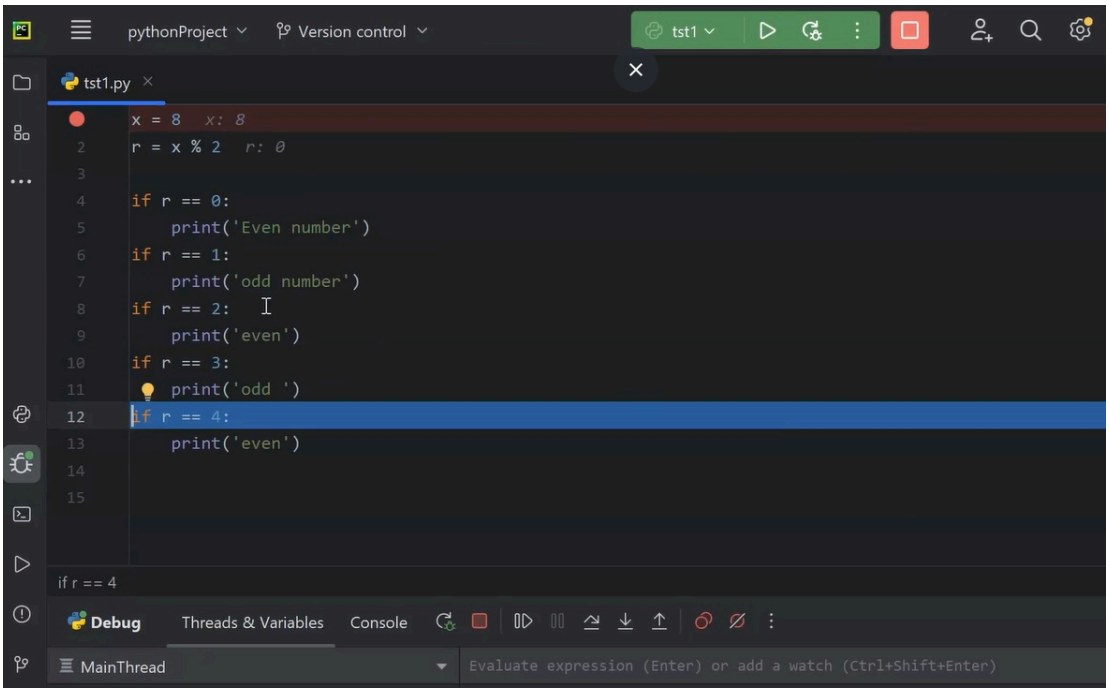
```
Odd Number
```

Above Code debugged in PyCharm

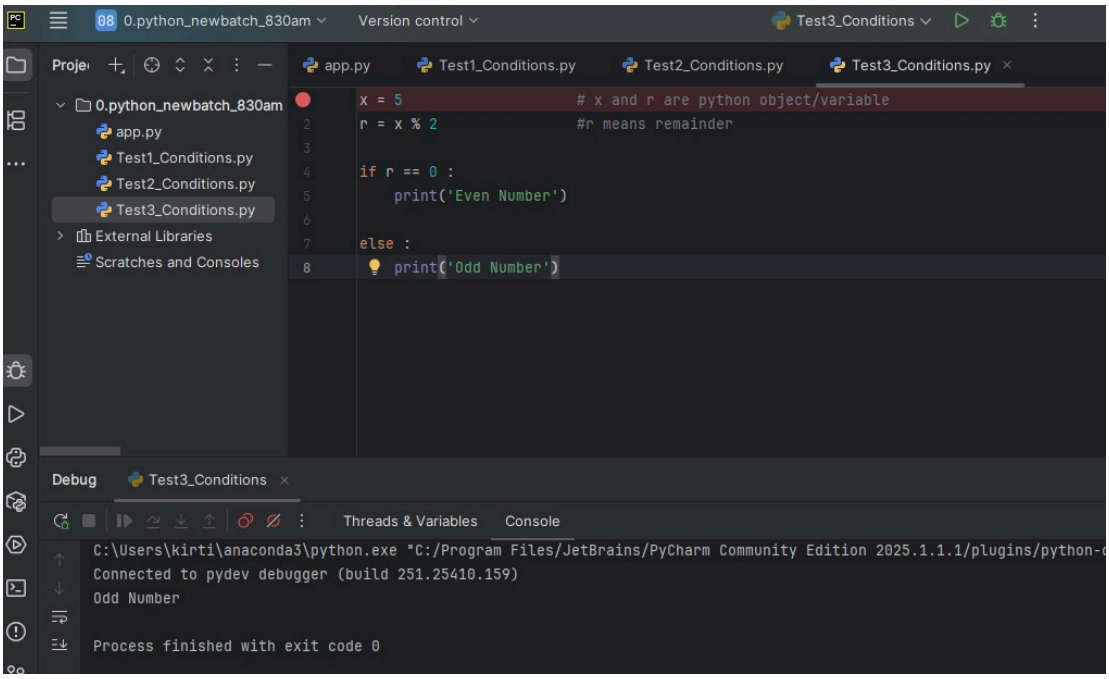
1. Open PyCharm
2. Put a debugger on 1st line by clicking on left most margin
3. Right click -> select Debug test1 (Debug means how the program is understood by memory)
4. Press F8 to debug



Code Debug in PyCharm



Multiple if consume lot of space, thus Never ever use multiple if statement



Using else statement, debugger doesn't execute if condition once else condition is met and vice-versa, consuming less memory

In [10]:
```
x = 5
r = x % 2

if (r == 0) :
    print('Even Number')

print('Odd Number')
```

```
Odd Number
```

In [14]:
```
x = 4
r = x % 2

if (r == 0) :
    print('Even Number')

print('Odd Number')

#Output will be Even number and Odd number as there is no else condition and
#all statements are executed, once condition is met
```

```
Even Number
Odd Number
```

In [18]:
```
x = 5                   # x and r are python object/variable
r = x % 2               #r means remainder

if (r == 0) :
    print('Even Number')

else :
    print('Odd Number')

#Write this Code
```

```
Odd Number
```

No Multiple If and it consumes more space

In [22]:
```
# Multiple if example

x = 5                   # x and r are python object/variable
r = x % 2               #r means remainder

if r == 0 :
    print('Even Number')

if r != 0 :             #r !=0 is same as r==1, we can write either of the two
    print('Odd Number')

#Write this Code
```

```
Odd Number
```

Nested if

- if inside if is called nested if
- Used when we have more than 1 conditions

In [27]:
```
x = 3
r = x % 2

if r == 0 :                         # Condition - 1
    print('Even Number')
    if x>5 :                        # Condition - 2
        print('Greater Number')
```

```
if r != 0 :
    print('Odd Number')

#Write this Code
```

```
Odd Number
```

In [29]:
```
x = 4
r = x % 2

if r == 0 :                         # Condition - 1
    print('Even Number')
    if x>5 :                        # Condition - 2
        print('Greater Number')

if r != 0 :
    print('Odd Number')

#Write this Code
```

```
Even Number
```

In [33]:
```
x = 8
r = x % 2

if r == 0 :                         # Condition - 1
    print('Even Number')
    if x>5 :                        # Condition - 2
        print('Greater Number')

if r != 0 :
    print('Odd Number')

#Write this Code
```

```
Even Number
Greater Number
```

In [35]:
```
x = 2
r = x % 2

if r == 0 :                         # Condition - 1
    print('Even Number')

    if x>5 :                        # Condition - 2
        print('Greater Number')
    else :
        print('Not Greater')

if r != 0 :
    print('Odd Number')

#Write this Code
```

```
Even Number
Not Greater
```

if - elif - else

In [1]:
```
x = 4

if x == 1:
    print('One')
elif x == 2:
    print('Two')
elif x == 3:
    print('Three')
elif x == 4:
    print('Four')
```

```
Four
```

In [3]:
```
x = 10

if x == 1:
    print('One')
elif x == 2:
    print('Two')
elif x == 3:
    print('Three')
elif x == 4:
    print('Four')

#No output(As there is no condition for x = 10) and no error
```

In [9]:
```
x = 10

if x == 1:
    print('One')
elif x == 2:
    print('Two')
elif x == 3:
    print('Three')
elif x == 4:
    print('Four')

else :
    print('No output')

#This prints the output

#This concept is called if - elif - else
```

```
No output
```

Conditional statement is completed

- Interview Questions To do :-
- 50_code on conditional statment.pdf
- 50_code Q & A conditional statment.pdf

50 Questions on Conditional Statement - Practice as per above concept

1. Easy level
2. Medium level
3. Hard level

Build logic by practicing!

Use int input function in every answer

In [16]:
```
#Determine if a given year is a leap year or not

year = int(input("Enter a year: "))
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print("Leap Year")
else:
    print("Not a Leap Year")
```

```
Not a Leap Year
```

In [20]:
```
#Check if a number is positive, negative, or zero

num = int(input("Enter a number: "))
if num > 0:
    print("Positive")
elif num < 0:
    print("Negative")
else:
    print("Zero")
```

```
Negative
```

- Sir, shared 50 questions
- Take 1 question (try to solve yourself with the logic), then watch answer the 1st time
- 2nd time when you code, please don't watch and fix it yourself
- 3rd time don't practice just watch the code
- 4th time - after 1 month (view the code)
- Now, code will store in your brain (memory)
- In Interview -- Memory will pass the answer
- Finally, the interview will be cleared
- To understand the 50the code - you need to practise 1-49th code
- To get 11th interview clear - you will have to fail 1-10 interviews

- Maintain LinkedIn
- Post on linkedin - create video - windows+g
- Check Kodi sir's tagged post for reference and create posts
- Career gap - speak to sir