In [1]:
```
print("Hello")
```

```
Hello
```

In [ ]:
```
# python native types
# numbers
    |-->int float complex
 |-->bool ->True False
 |-->NoneType ->None
 |-->Collection
         |
    --------------
    |             \_____________________________unordered items
    collection of orderd items(index based)           |->dict, set
    \__ str, bytes ,list, tuple

    str - collection of chars '' <or> " "
    bytes - collection of chars ASCII b'' <or> b" "
    |
     list - collection of different types of item - mutable []
     tuple - collection of different types of item - immutable ()
```

In [ ]:
```
type() - determine python type
|
type(value)
```

In [7]:
```
print(type(45))
print(type(45.0))
print(type('45'))
```

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

In [8]:
```
print("Hello")
print('Hello')
#print("Hello') # Error
```

```
Hello
Hello
```

In [9]:
```
# using print() - display following details
#                 filename   (ex: prod.csv)
#                 fileindex  (ex: 4562)
#                 fileusage  (ex: 98.34)
#                 fileopen status (ex: True)
print("About prod.csv file")
print('--------------------')
print('File name:prod.csv')
print('prod.csv index number:4562')
print('prod.csv disk utilization:98.34')
print('prod.csv open status is True')
```

```
About prod.csv file
--------------------
File name:prod.csv
prod.csv index number:4562
prod.csv disk utilization:98.34
prod.csv open status is True
```

In [10]:
```
print('Data1\nData2\nData3')
```

```
Data1
Data2
Data3
```

In [11]:
```
# multiline string <or> multiline statement
print('''Data1
Data2
Data3''')
```

```
Data1
Data2
Data3
```

In [12]:
```
print("Data1","Data2","Data3")
print("File name is:","prod.csv","File index:",456,"Utilization:",98.45)
print("File open status:",True)
```

```
Data1 Data2 Data3
File name is: prod.csv File index: 456 Utilization: 98.45
File open status: True
```

In [13]:
```
# Variable - placeholder - holding a value
# |->user defined
#    ------------
# variableName = value <=== initialization

Fname = "prod.csv"
FINDX = 456
futil = 98.42
print(Fname,FINDX,futil)
```

```
prod.csv 456 98.42
```

In [16]:
```
Fname = "emp.csv"
print(Fname)
print("Fname")
# File name is: emp.csv
# ------------- _______
#
print("File name is:",Fname)
```

```
emp.csv
Fname
File name is: emp.csv
```

In [19]:
```
Fname = "sales.xlsx"
Findex = 2345
Futil = 99.23
Fstatus = False
print("About",Fname,"file")
print('--------------------')
print('File name:',Fname)
print(Fname,"index number:",Findex)
print(Fname,"disk utilization:",Futil)
print(Fname,'open status is:',Fstatus)
```

```
About sales.xlsx file
--------------------
File name: sales.xlsx
sales.xlsx index number: 2345
sales.xlsx disk utilization: 99.23
sales.xlsx open status is: False
```

In [22]:
```
Fname = "prod.csv"
print(Fname)
print(filename)
```

```
prod.csv

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[22], line 3
      1 Fname = "prod.csv"
      2 print(Fname)
----> 3 print(filename)

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

In [24]:
```
va = 45
vb = 94.2
vc = "data"
print(type(va),type(vb),type(vc))
print("va value:",va)
print("vb value is:",vb)
print("vc value is:",vc)
```

```
<class 'int'> <class 'float'> <class 'str'>
va value: 45
vb value is: 94.2
vc value is: data
```

In [25]:
```
# In C program
# int va=10;
# printf("%d",va);
print("va value is:%d vb value is:%f vc value is:%s"%(va,vb,vc))
```

```
va value is:45 vb value is:94.200000 vc value is:data
```

In [28]:
```
n=56
print('n value is:%d'%(n))
n="Hello"
print('n value is:%d'%(n))
```

```
n value is:56

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[28], line 4
      2 print('n value is:%d'%(n))
      3 n="Hello"
----> 4 print('n value is:%d'%(n))

TypeError: %d format: a real number is required, not str
```

In [29]:
```
n=56
print("n value is:{}".format(n))
n="Hello"
print("n value is:{}".format(n))
n=4.12
print("n value is:{}".format(n))
```

```
n value is:56
n value is:Hello
n value is:4.12
```

In [31]:
```
va = 45
vb = 94.2
vc = "data"
print("va value is:",va,"vb value is:",vb,"vc value is:",vc)     # (1)
print("va value is:%d vb value is:%f vc value is:%s"%(va,vb,vc)) # (2)
print("va value is:{} vb value is:{} vc value is:{}".format(va,vb,vc)) # (3)
print(f"va value is:{va} vb value is:{vb} vc value is:{vc}") # (4)
```

```
va value is: 45 vb value is: 94.2 vc value is: data
va value is:45 vb value is:94.200000 vc value is:data
va value is:45 vb value is:94.2 vc value is:data
va value is:45 vb value is:94.2 vc value is:data
```

In [39]:
```
n=67
print("n value is:%0.2f"%(n))
print(f"n value is:{n}")
```

```
n value is:67.00
n value is:67
```

In [41]:
```
Fname = "sales.xlsx"
Findex = 2345
Futil = 99.23
Fstatus = False
print(f'''About {Fname} file details
-------------------------------------
File name:{Fname}
{Fname} index number:{Findex}
{Fname} disk utilization:{Futil}
{Fname} open status is:{Fstatus}''')
print("\n") # empty line
print('''About {} file details
-------------------------------------
File name:{}
{} index number:{}
{} disk utilization:{}
{} open status is:{}'''.format(Fname,Fname,Fname,Findex,Fname,
                               Futil,Fname,Fstatus))
```

```
About sales.xlsx file details
-------------------------------------
File name:sales.xlsx
sales.xlsx index number:2345
sales.xlsx disk utilization:99.23
sales.xlsx open status is:False


About sales.xlsx file details
-------------------------------------
File name:sales.xlsx
sales.xlsx index number:2345
sales.xlsx disk utilization:99.23
sales.xlsx open status is:False
```

In [42]:
```
# typecasting
va = 45
print(type(va))
```

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

In [43]:
```
print(float(va)) # convert to float
str(va)   # convert to str
```

```
45.0
```

Out[43]:
```
'45'
```

In [45]:
```
print(va,type(va))
```

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

In [50]:
```
print(10+20)
print(float(10+20))
print(10+4.5)
print(int(10+4.5))
```

```
30
30.0
14.5
14
```

In [52]:
```
cost='4567.89'
print(type(cost))
print(cost * 0.18)
```

```
<class 'str'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[52], line 3
      1 cost='4567.89'
      2 print(type(cost))
----> 3 print(cost * 0.18)

TypeError: can't multiply sequence by non-int of type 'float'
```

In [53]:
```
print(float(cost) * 0.18)
```

```
822.2202
```

In [54]:
```
f=45.67
int(f)
```

Out[54]:
```
45
```

In [55]:
```
f='45.67'
int(f)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[55], line 2
      1 f='45.67'
----> 2 int(f)

ValueError: invalid literal for int() with base 10: '45.67'
```

In [57]:
```
int(float(f))
```

Out[57]:
```
45
```

In [58]:
```
n=56
n=float(n) # typecast to float ->then initialize to namedVariable
print(n,type(n))
```

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

In [ ]:
```
Keyboard(<STDIN>)-----<-------python---------------->--Monitor
                    input()           print()
Syntax:-
----------
variable = input('user defined prompt message')
#########       ###############################
```

In [59]:
```
n = input('Enter a n value:')
print(f'Input value is:{n}')
```

```
Enter a n value:78
Input value is:78
```

In [60]:
```
n = input('Enter a n value:')
print(f'Input value is:{n}')
```

```
Enter a n value:data1
Input value is:data1
```

In [61]:
```
n = input('Enter a n value:')
print(type(n))
n
```

```
Enter a n value:45
<class 'str'>
```

Out[61]:
```
'45'
```

In [62]:
```
n = input('Enter a n value:')
print(type(n))
n
```

```
Enter a n value:45.67
<class 'str'>
```

Out[62]:
```
'45.67'
```

In [63]:
```
float(n)+100
```

Out[63]:
```
145.67000000000002
```

In [64]:
```
n=input('Enter n value:')
print(type(n),n)
```

```
Enter n value:45
<class 'str'> 45
```

In [75]:
```
n=45
print(f'n value is:{n:.2f}')
print(f'n value is:{n:.3f}')
```

```
n value is:45.00
n value is:45.000
```

In [ ]:
```
Q1. Write a python program:
    read a product name,product Cost,vendor name from <STDIN>
    calculate 18% of product Cost - initialize to new variable
    calculate Sum of product Cost + 18% Tax
    display - productName,VendorName,Cost,Tax,TotalAmount(includingTax)
    Note: display each field separated by ,
          use single print()

Enter a product name: prodA
Enter prodA vendor name:Vabc
Enter prodA cost:1000
    ...
    ...
Results:-
prodA,Vabc,1000,180,1180
```

In [74]:
```
pName = input('Enter a product name:')
pVendor = input(f'Enter {pName} vendor name:')
pCost = input('Enter {} Cost:'.format(pName))

tax = int(float(pCost)) * 0.18
gs = int(float(pCost)) + tax
print(pName,pVendor,pCost,tax,gs)
print('\n') # empty line
print(f'''prodName,Vendor,pCost,Tax,Total
----------------------------------------
{pName},{pVendor},{pCost},{tax},{gs}
----------------------------------------''')
```

```
Enter a product name:prodA
Enter prodA vendor name:Vabc
Enter prodA Cost:1000
prodA Vabc 1000 180.0 1180.0


prodName,Vendor,pCost,Tax,Total
----------------------------------------
prodA,Vabc,1000,180.0,1180.0
----------------------------------------
```

In [73]:
```
f='1005.4'
int(float(f))
```

Out[73]:
```
1005
```

In [ ]:
```
Arithmetic operators
+ - * / // ** <=== input_types(int,float) ->output_types(int,float)
```

In [76]:
```
n=5
n=n+1
print(n)
n=5
n+=1 # n = n + 1
print(n)
```

```
6
6
```

In [ ]:
```
string operators
+ *
```

In [78]:
```
print(10 + 20) # addition
print("python" + "programming") # string concat
```

```
30
pythonprogramming
```

In [79]:
```
va='50'
vb='60'
print(va+vb) ## (A)   5060
print(int(va)+int(vb)) ##(B) 110
print(type(int(va)+float(vb))) ## (C) <class 'float'>
```

```
5060
110
<class 'float'>
```

In [81]:
```
Ename='Leo'
empID=456
print('Emp Name is:'+Ename)
print('Emp id is:'+str(empID))
```

```
Emp Name is:Leo
Emp id is:456
```

In [85]:
```
print(10*3)

# input_String * int_n ->output_string
#
print("Hello"*3)
print("-"*50)
print("Test server\n"*5)
```

```
30
HelloHelloHello
--------------------------------------------------
Test server
Test server
Test server
Test server
Test server
```

In [87]:
```
server='Test Server\n' * 5
print(server)
server='Test Server\\n' * 5
print(server)
```

```
Test Server
Test Server
Test Server
Test Server
Test Server

Test Server\nTest Server\nTest Server\nTest Server\nTest Server\n
```

In [ ]:
```
Relational operators (input_types: int,float,str ->output_type:bool)
== != < <= > >=
450 > 400
450 < 400
98.56 >98.64
0.02 > 0.02
'root' == 'root' ->True
'root' == 'Root' ->False
'abc' != 'ABC' ->True

In python any expression <or> method <or> function ->bool(True/False)
-----------------------------------------------------------------------
 \___ we use conditional statement
                |
               Code block - execute only one time.
               if statement
                \__ 3 ways
                    i. if only style
                    ii. if else style
                    iii. if elif elif elif .. else style
```

In [89]:
```
print(type(True),type(False))
print(type('True'))
```

```
<class 'bool'> <class 'bool'>
<class 'str'>
```

In [91]:
```
va = 10
 vb = 20
```

```
  Cell In[91], line 2
    vb = 20
    ^
IndentationError: unexpected indent
```

In [92]:
```
# if only style
pName = 'prodA' # initialization

pName == 'prodA' # condition1
```

Out[92]:
```
True
```

In [94]:
```
pName = 'prodA'
if(pName == 'prodA'):
    print('product name is matched')
    pID = 101
    print(f'product name:{pName} pid:{pID}')
```

```
product name is matched
product name:prodA pid:101
```

In [97]:
```
pName = 'prodB'

print(pName == 'prodA')

if(pName == 'prodA'):
    print('-'*50)
    print('product name is matched')
    pID = 101
    print(f'product name:{pName} pid:{pID}')
    print('-'*50)
```

```
False
```

In [ ]:
```
if(condition):
    True block
else:
    False block
```

In [99]:
```
pName = 'prodB'

if(pName == 'prodA'):
    print('-'*50)
    print('product name is matched')
    pID = 101
    print(f'product name:{pName} pid:{pID}')
    print('-'*50)
else:
    print('-'*50)
    print(f"Sorry product {pName} is not matched")
    print('-'*50)
```

```
--------------------------------------------------
Sorry product prodB is not matched
--------------------------------------------------
```

In [ ]:
```
Write a python program:
    read a port number from <STDIN>
    test - input port number is above 500 - initialize an app name is TestApp
                                below 500 - initialize an app name is DemoApp
    use print() - display App name and running port number to monitor.
```

In [100]:
```
port = input('Enter a port number:')
if(int(port) > 500):
    app = 'TestApp'
else:
    app = 'DemoApp'
print(f'App name is:{app} running port number:{port}')
```

```
Enter a port number:560
App name is:TestApp running port number:560
```

In [101]:
```
port = input('Enter a port number:')
if(int(port) > 500):
    app = 'TestApp'
else:
    app = 'DemoApp'
print(f'App name is:{app} running port number:{port}')
```

```
Enter a port number:450
App name is:DemoApp running port number:450
```

In [103]:
```
# len(input_string) ->output_int
print(len(''))
s='abc'
print(len(s))
```

```
0
3
```

In [104]:
```
name = input('Enter your name:')
if(len(name) == 0):
    print('Sorry your input is empty')
else:
    print(f'Hello...{name}')
```

```
Enter your name:
Sorry your input is empty
```

In [106]:
```
name = input('Enter your name:')
if(len(name) == 0):
    print('Sorry your input is empty')
else:
    print(f'Hello...{name}')
```

```
Enter your name:Abc
Hello...Abc
```

In [110]:
```
shell_var = input('Enter a shell name:')

if(shell_var == 'bash'):
    fname = 'bashrc'
elif(shell_var == 'ksh'):
    fname = 'kshrc'
elif(shell_var == 'psh'):
    fname = 'Winprofile'
else:
    shell_var = '/bin/nologin'
    fname = '/etc/profile'

print(f'Shell name:{shell_var} profile filename:{fname}')
```

```
Enter a shell name:tcsh
Shell name:/bin/nologin profile filename:/etc/profile
```

In [ ]:
```
In Single Conditional statement, test more than one condition
                                    //use logical operators(keywords: and or not)
```

In [111]:
```
shell_var = input('Enter a shell name:')

if(shell_var == 'bash' or shell_var == 'sh'):
    fname = 'bashrc'
elif(shell_var == 'ksh'):
    fname = 'kshrc'
elif(shell_var == 'psh'):
    fname = 'Winprofile'
else:
    shell_var = '/bin/nologin'
    fname = '/etc/profile'

print(f'Shell name:{shell_var} profile filename:{fname}')
```

```
Enter a shell name:sh
Shell name:sh profile filename:bashrc
```

In [113]:
```
port = input('Enter a port number:')
if(int(port) > 500 and int(port) < 600):
    app = 'TestApp'
else:
    app = 'DemoApp'
print(f'App name is:{app} running port number:{port}')
```

```
Enter a port number:650
App name is:DemoApp running port number:650
```

In [114]:
```
# Looping statements
# ----------------------
#   Code block - execute more than one time.
#   ----------
#       \____1. Conditional style - based on the condition - while
#       \____2. Collection style - based on the data - for
#

# break - exit from loop ; continue - won't exit from loop
# -----                    --------

# conditional style code
#  rule 1: start - initialization
#  rule 2: stop  - condition
#  rule 3: step  - arithmetic

i=0
while(i <5):
    print('i value is:',i)
    i=i+1
```

```
i value is: 0
i value is: 1
i value is: 2
i value is: 3
i value is: 4
```

```
Write a python program:
    initialize a pin Number: (Ex: pin=4567)
    use while loop - limit is 3
          -> read a pinNumber from <STDIN>
          -> test - inputpin with existing pin
                      ->display message - pin is matched - count( 1 2 3)
   pin is blocked if all 3 inputs are failed.
```

In [119]:
```
pin = 4567
count = 0
while(count < 3):
    p = input('Enter a pin Number:')
    count = count + 1
    if(int(p) == pin):
        print(f'Pin is matched - at {count}')
        break

if(int(p) != pin):
    print('Sorry pin is blocked')
```

```
Enter a pin Number:4567
Pin is matched - at 1
```

In [120]:
```
while(True):
    print('Always active')
    break
```

```
Always active
```

In [ ]:
```
for variable in <Collection>:
    <Code block>  \__ str,bytes,list,tuple,dict,set

for in - keywords
--------------------
```

In [121]:
```
s='python'     # s | p | y | t | h | o | n |
for var in s:  #   | 0 | 1 | 2 | 3 | 4 | 5 | <== index
    print('var value is:',var)
    print('-'*45)
```

```
var value is: p
---------------------------------------------
var value is: y
---------------------------------------------
var value is: t
---------------------------------------------
var value is: h
---------------------------------------------
var value is: o
---------------------------------------------
var value is: n
---------------------------------------------
```

In [122]:
```
s='python'
for var in s:
    print('var value is:',var)
```

```
var value is: p
var value is: y
var value is: t
var value is: h
var value is: o
var value is: n
```

In [124]:
```
s='python'

# s -> | p | y | t | h | o | n |
#      | 0 | 1 | 2 | 3 | 4 | 5 | <== index
#       -6  -5  -4  -3  -2  -1   <== index
# How to fetch single chars ?
#  stringName[index]
print(s[0])
print(s[1])
print(s[5])
print(s[6])
```

```
p
y
n

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[124], line 11
      9 print(s[1])
     10 print(s[5])
---> 11 print(s[6])

IndexError: string index out of range
```

In [125]:
```
s='python'
c=0
while(c <len(s)):
    print('var value is:',s[c])
    c=c+1
```

```
var value is: p
var value is: y
var value is: t
var value is: h
var value is: o
var value is: n
```

In [126]:
```
for var in 'a:b-c':           # |a|:|b|-|c|
    print('var value:',var)   #  0 1 2 3 4 <== index
    print('') # empty line
```

```
var value: a

var value: :

var value: b

var value: -

var value: c
```

In [ ]:
```
for var in '121':          Vs  for var in 'AB':
    print('Test')                      print('Hello')

# '121' ->|1|2|1|              |A|B|
#

for var in str(45):
    print("OK")
```

In [127]:
```
for var in 45:
    print('OK')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[127], line 1
----> 1 for var in 45:
      2     print('OK')

TypeError: 'int' object is not iterable
```

In [128]:
```
for var in str(45):
    print('OK')
```

```
OK
OK
```

In [131]:
```
msg='test-python3.12 code in dev machine'
print(type(msg),len(msg))
print(msg[0])
print(msg[1])
print(msg[2])
print(msg[-1])
```

```
<class 'str'> 35
t
e
s
e
```

In [ ]:
```
# slicing group of chars
s[n] ->nth index

s[n:m] ->from nth index to m-1 index
s[4:9] ->from 4th index to 8th index(9-1)

s[n:] ->from nth index to ALL
s[:n] ->from 0th index to n-1

print(s[n]) ->nth index
print(s[n:]) ->from nth index to all
```

In [134]:
```
print(msg)
print(msg[0:5]) # from 0th index to 4th index(5-1)
print(msg[15:20]) # from 15th index to 19th index(20-1)
```

```
test-python3.12 code in dev machine
test-
 code
```

In [136]:
```
print(msg[15:]) # from 15th index to list of all
print(msg[:15]) # from 0th index to 14th
```

```
 code in dev machine
test-python3.12
```

In [155]:
```
             #    0   1   2   3   4   5   6
s='abcdefg'  #  | a | b | c | d | e | f | g
             #   -7  -6  -5  -4   -3 -2  -1
print(s[-4])
print(s[-4:]) # last 4chars
print(s[-5:-2])
print(s[-1])
```

```
d
defg
cde
g
```

In [156]:
```
s='python'
s.upper()
```

Out[156]:
```
'PYTHON'
```

In [157]:
```
print(s)
```

```
python
```

In [159]:
```
print('ABC'.lower())
```

```
abc
```

In [160]:
```
s='python'
s.isupper()
```

Out[160]: False

In [161]:
```
s.islower()
```

Out[161]: True

In [162]:
```
if(s.islower()):
    print('Yes Given string is lowercase chars')
else:
    print('No given string is not a lowercase chars')
```

```
Yes Given string is lowercase chars
```

In [ ]:
```
s.title()
| -------
|__1st step: determine the type => type(s) -><class 'str'>
  |
  |->2nd step: understand method definition
               ============================
                  help(str.title)
```

In [165]:
```
# help(str)
#  Vs
help(str.title)
```

```
Help on method_descriptor:

title(self, /)
    Return a version of the string where each word is titlecased.

    More specifically, words start with uppercased characters and all remaini
ng
    cased characters have lower case.
```

In [166]:
```
'abc'.title()
```

Out[166]: 'Abc'

In [170]:
```
s1="data1\n"
s2="data2\t"
s3="data3 "
s4="data:"
print(s1.strip())
print(s2.strip())
print(s3.strip())
print(s4.strip())
print("-->",s4.strip(':')) # remove ':'
```

```
data1
data2
data3
data:
--> data
```

In [174]:
```
s="0.05 0.01 0.02LB"
print(s)
print(s.strip('LB')) # remove LB
print(s)
```

```
0.05 0.01 0.02LB
0.05 0.01 0.02
0.05 0.01 0.02LB
```

In [175]:
```
msg="hello test user"
msg.upper()
```

Out[175]:
```
'HELLO TEST USER'
```

In [176]:
```
print(msg)
```

```
hello test user
```

In [177]:
```
msg=msg.upper() # convert to uppercase then initialize to variable
print(msg)
```

```
HELLO TEST USER
```

In [183]:
```
s1 = '45'

s2 = 'ab'
print(type(s1),type(s2))

print(s1.isdigit(),s2.isdigit())
```

```
<class 'str'> <class 'str'>
True False
```

In [ ]:
```
Write a python program:
    read n value from <STDIN>
    \__ test - input n is digit or not
                                |       ===>we can't typecast to int
                               calculate n + 100 ->display updated n value
```

In [185]:
```
n = input('Enter a n value:')
if(n.isdigit()):
    n=int(n)+100
    print('updated n value is:',n)
else:
    print("Sorry we can't type cast to int")
```

```
Enter a n value:459ABC
Sorry we can't type cast to int
```

In [187]:
```
n = input('Enter a n value:')
if(n.isdigit()):
    n=int(n)+100
    print('updated n value is:',n)
else:
    print("Sorry we can't type cast to int")
```

```
Enter a n value:500
updated n value is: 600
```

In [188]:
```
# int float bool str
empID = 456
empName = "Leo"
ecost = 242434.23
elogin = True

emp_details=[empID,empName,ecost,elogin] # list

# list - Collection of different types of values
# ----                  ---------------
#  \______ index based
# \_______ supports - index,slicing

print(type(emp_details))
print(len(emp_details))
print(emp_details)
```

```
<class 'list'>
4
[456, 'Leo', 242434.23, True]
```

In [189]:
```
for var in emp_details:
    print("var value:",var)
```

```
var value: 456
var value: Leo
var value: 242434.23
var value: True
```

In [190]:
```
for var in 'abcdef':
    print("Hello")

print('')
for var in ['abcdef']:
    print("OK")
```

```
Hello
Hello
Hello
Hello
Hello
Hello

OK
```

In [194]:
```
fnames = ['p1.log',50,6.32,True]
print(type(fnames))
print(type(fnames[0]))
print(type(fnames[1]))
print(type(fnames[2]))
print(type(fnames[-1]))
```

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

In [ ]:
```
str - Collection of chars - index based - immutable - ''

list - Collection of items - index based - mutable - []
```

In [195]:
```
s='abc'
s[1]
```

Out[195]:
```
'b'
```

In [196]:
```
s[1]='x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[196], line 1
----> 1 s[1]='x'

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

In [197]:
```
L=['D1','D2',10,2.45]
L[1]
```

Out[197]:
```
'D2'
```

In [198]:
```
L[1]='Data-2' # we can modify an existing value - mutable
L
```

Out[198]:
```
['D1', 'Data-2', 10, 2.45]
```

In [ ]:
```
List is mutable
          \__ we can add  new item to an existing list
                    ======
                      \__ Listname.append(Value) ->None
                      \__ Listname.insert(index,Value) ->None

         \___ we can delete nth item from existing list
                      \___ Listname.pop() ->return removed_value (last index)
                      \___ Listname.pop(index) ->removed nth index value

         \___ we can modify an existing item
                      |
                      \___ Listname[oldIndex] = updatedValue
```

In [199]:
```
config_files=['network.cfg','users.cfg']

print(config_files)
print(len(config_files))
```

```
['network.cfg', 'users.cfg']
2
```

In [200]:
```
config_files.append('process.cfg')
config_files.append('sys.confg')
print(len(config_files),config_files)
```

```
4 ['network.cfg', 'users.cfg', 'process.cfg', 'sys.confg']
```

In [201]:
```
config_files.insert(1,"device.cfg")
```

In [202]:
```
print(len(config_files),config_files)
```

```
5 ['network.cfg', 'device.cfg', 'users.cfg', 'process.cfg', 'sys.confg']
```

In [204]:
```
print(len(config_files),config_files)
r = config_files.pop()
print(r)
```

```
5 ['network.cfg', 'device.cfg', 'users.cfg', 'process.cfg', 'sys.confg']
sys.confg
```

In [205]:
```
print(len(config_files),config_files)
```

```
4 ['network.cfg', 'device.cfg', 'users.cfg', 'process.cfg']
```

In [206]:
```
r = config_files.pop(2)
print('removed value:',r)
```

```
removed value: users.cfg
```

In [207]:
```
print(len(config_files),config_files)
```

```
3 ['network.cfg', 'device.cfg', 'process.cfg']
```

In [208]:
```
hosts=[] # empty list
print("No.of hosts:{}".format(len(hosts)))
```

```
No.of hosts:0
```

In [209]:
```
hosts=[] # empty list
print("No.of hosts:{}".format(len(hosts)))
c=0
while(c <5):
    h = input('Enter a hostname:')
    hosts.append(h) # adding new data to an existing list
    c=c+1

print('No.of hosts:{}'.format(len(hosts)))
print('') # empty line
print('List of hostnames:')
for var in hosts:
    print(var)
print('-'*50)
```

```
No.of hosts:0
Enter a hostname:host01
Enter a hostname:host02
Enter a hostname:host03
Enter a hostname:host04
Enter a hostname:host05
No.of hosts:5

List of hostnames:
host01
host02
host03
host04
host05
--------------------------------------------------
```

In [210]:
```
hosts
```

Out[210]:
```
['host01', 'host02', 'host03', 'host04', 'host05']
```

In [ ]:
```
list - collection of different types of items - mutable - index/slicing - []

tuple - collection of different types of items - immutable - index/slicing - ()
```

In [211]:
```
print(type([]),type(()))
```

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

In [216]:
```
L=[10,1.5,"Data"] # list
T=(10,1.5,"Data") # tuple

print(L[0],T[0])

print(L[-1],T[-1])

print(L[-2:],T[-2:]) # last 2 items

print(len(L),len(T))

for var in L:
    print(var)

print('')
for var in T:
    print(var)
```

```
10 10
Data Data
[1.5, 'Data'] (1.5, 'Data')
3 3
10
1.5
Data

10
1.5
Data
```

In [217]:
```
print(L)
L[1]=456 # mutable
L
```

```
[10, 1.5, 'Data']
```

Out[217]:
```
[10, 456, 'Data']
```

In [219]:
```
print(T)
T[1]=456 # Error - tuple is immutable
```

```
(10, 1.5, 'Data')

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[219], line 2
      1 print(T)
----> 2 T[1]=456

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