In [ ]:
```
######### Python divided 3 parts #######################
----- Basics
1) Variables
2) Data types
3) Type conversions
4) Eval input
5) packages
6) Conditional statements
7) Try-execpt
8) Functions
9) For
10) While ( you did this)

------- Intermediate part
11) Strings
12) Lists
13) Dictionary
14) Tuples (You will do this)
15) Sets
16) Lambda functions
17) File handling session

--------- Advanced
18) OOPS
```

In [ ]:
```
==================== PART-1=====================================
- Intiaization

- type

- len

- max

- min

- sorted

- reveresed

- in

- for loop using in

- index

- for loop using index

- mutable

- concatenation

#================== PART-2====================
# Methods
```

Intialization

```
In [1]: str1='python'
        str1
```

```
Out[1]: 'python'
```

```
In [2]: str2="python"
        str2
```

```
Out[2]: 'python'
```

```
In [3]: print(str1) # Do not confuse Im not seeing the quotes
```

```
python
```

Note

- If we mentioned single quotes or double quotes by default python provides in single quotes only
- If we print the strings, we will answer with out quotes

Triple quotes

```
In [4]: str3="""hi how are you
                im good"""
        str3
```

```
Out[4]: 'hi how are you\n        im good'
```

```
In [5]: print(str3)
```

```
hi how are you
        im good
```

Note

- If we mentioned single quotes or double quotes by default python provides in single quotes only
- If we print the strings, we will answer with out quotes
- Triple quotes are using for doc string
- In order to convey the information about coding part we will use doc string
- anything inside the single or double quotes is considered as string in python

type

```
In [6]: type(str1) # type is string time
```

```
Out[6]: str
```

```
In [ ]: str4='10'  # integer value in the form of string
        str5='10.5' # Float value in the for of string
```

```
str6='123abc' # Alpha numeric in the form of strng
str7='True'  # Boolean in the form string
str8='sinx'

# All are in quotes, means in red color : strings
```

len

```
In [7]: str1='python'
        len(str1)
```

```
Out[7]: 6
```

max and min

```
In [8]: str1='python'
        max(str1)
```

```
Out[8]: 'y'
```

```
In [9]: ord('p'),ord('y'),ord('t'),ord('h'),ord('o'),ord('n')
```

```
Out[9]: (112, 121, 116, 104, 111, 110)
```

```
In [10]: str1='python'
         min(str1)
```

```
Out[10]: 'h'
```

Keywords vs Methods

```
In [ ]: type(<>)
        print(<>)
        len(<>)
        max(<>)
        min(<>)
        eval(<>)
        input(<>)
        sum(<>)
```

```
In [ ]: - in order to use method we need to call package

        import random
        random.randint()

        import math
        math.sqrt()

        import time
        time.sleep()
```

```
In [11]: str1='python123'
         max(str1)
```

```
Out[11]: 'y'
```

sum

```
In [12]: str1='python'
         sum(str1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 2
      1 str1='python'
----> 2 sum(str1)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
In [13]: 'p'+'y'
```

```
Out[13]: 'py'
```

```
In [14]: sum([1,2,3])
```

```
Out[14]: 6
```

sorted

```
In [ ]: len('python')
        max('python')
        print('python')
        min('python')
        max('python')
```

```
In [15]: sorted('python')
```

```
Out[15]: ['h', 'n', 'o', 'p', 't', 'y']
```

- sorted gives ascending order based on ASCII numbers
- sorted is kind of a function
- Every function we have arguments
- In that one defualt argument is **reverse=False**
- By defualt sorted output gives **ascending order**
- We can change the order, by providing **reverse =True**
- **If we do not mention, it will give the default answer**

```
In [16]: sorted('python',reverse=True)
```

```
Out[16]: ['y', 't', 'p', 'o', 'n', 'h']
```

```
In [17]: complex()   # 0+0j
```

```
Out[17]: 0j
```

```
In [18]: complex(10,20)
```

```
Out[18]:  (10+20j)
```

```
In [19]:  complex(real=10,imag=30)
```

```
Out[19]:  (10+30j)
```

```
In [20]:  import random
          random.randint()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 2
      1 import random
----> 2 random.randint()

TypeError: Random.randint() missing 2 required positional arguments: 'a' and 'b'
```

```
In [ ]:  funtion()  # Answer
         function(a,b) # this might return error, if we dont give the values of a,b
         function(a=10,b=10) # Ans
```

```
In [ ]:  funtion() #
         function(a,b)
         function(a=10,b=10)
```

```
In [21]:  import random
          random.random()  # we will get answer but answer is not in our hands
```

```
Out[21]:  0.5974424274486018
```

```
In [22]:  import random
          random.randint(10,20) # Answer we can change
```

```
Out[22]:  10
```

```
In [23]:  complex()  # By default  0,0
```

```
Out[23]:  0j
```

```
In [24]:  complex(10,20)
```

```
Out[24]:  (10+20j)
```

```
In [ ]:  sorted(iterable='python') # Error
         random.randint(a=10,b=30) # Anaswer
```

```
In [ ]:  sorted('python') # Anaswer
         sorted('python',reverse=True)
```

Note

- we have arguments before / and after slash
- we can use the argument names after / only
- For example sorted has two arguments one is iterable and another one is reverse

- iterbale is mentioned before / symbol
- reverse is mentioned after / symbol
- so we can use only reverse argument name while we are doing the work
- we can not use iterable argument name
- Instead of using iterable name , we can provide direct value at that posistion

reversed

```
In [ ]: type()
        len()
        max()
        min()
        print()
        sorted()
```

```
In [25]: reversed('python')
```

```
Out[25]: <reversed at 0x215e603b3d0>
```

- I already done my work
- Your output is stored in that memory location
- whenever you see this kind of answer less than and greater than symbol or memory
- use for loop to see the answer

```
In [28]: str1='azycd'
         output=reversed(str1)
         for i in output:
             print(i)
```

```
d
c
y
z
a
```

```
In [29]: sorted('azycd')
```

```
Out[29]: ['a', 'c', 'd', 'y', 'z']
```

```
In [ ]: - Intialization

        - type

        - len

        - max

        - min
- sum

- sorted

- reveresd
```

In [ ]:
```
type(<value>)
eval(<value>)
input(<value>)
max(<v>)
min()
len()
sum()
sorted()
reveresd()
```

In [1]:
```
# sorted()
# It will provide the values either ascending or descending

sorted('naresh') # Ascending or Descending

# when I run directly it will give one answer
# the function has a default behaviour: ascending

# How this default mentioned
# we need to understand about arguments
```

Out[1]:
```
['a', 'e', 'h', 'n', 'r', 's']
```

In [5]:
```
sorted('naresh',reverse=True)

# iterable
# key
# reverse
```

Out[5]:
```
['s', 'r', 'n', 'h', 'e', 'a']
```

In [9]:
```
for i in reversed('naresh'):
    print(i)
```

```
h
s
e
r
a
n
```

In [10]:
```
sorted([1,10,2,20,30])
# 1,2,10,20,30
# 30,20,10,2,1
```

Out[10]:
```
[1, 2, 10, 20, 30]
```

In [11]:
```
sorted([1,10,2,20,30],reverse=True)
```

Out[11]:
```
[30, 20, 10, 2, 1]
```

in

In [12]:
```
str1='naresh'

'n' in str1
```
Out[12]:
```
True
```
In [ ]:
```
# strings
# list
# tuples
# sets
# dictionary
```
In [14]:
```
str1='naresh'

'nn' not in str1
```
Out[14]:
```
True
```
In [15]:
```
for i in str1:
    print(i)
```
```
n
a
r
e
s
h
```
Note

- In operator directly access the letters

In [17]:
```
#Q1) WAP ask the user count how many 'a' are present in a given string
# str1='hai naresh how are you'

# Idea: counter wrapper
# step-1: count=0
# step-2: using for loop in operator iterate through  given string
# step-3: apply the if condition, whenever the letter equal to 'a'
# step-4: count=count+1

str1='hai naresh how are you'
count=0
for i in str1:
    if i=='a':
        count=count+1
print(count)
# step-1:  i='h'  if 'h'=='a'  False
# step-2:  i='a'  if  'a'=='a' T      count=0+1=1
# step-3:  i='i'  if  'i'=='a'  F      count=1
#
#step-6:  i='a'   if   'a'=='a'  T    count=1+1=2
```
```
3
```
In [18]:
```
# Q2) WAP ask the find how many vowels are there in a given string
# str1='hai naresh how are you'
# ans=9
count=0
```

```
str1='hai naresh how are you'
for i in str1:
    if i=='a' or i=='e' or i=='i' or i=='o' or i=='u':
        count=count+1
print("the count is:",count)
```

```
the count is: 9
```

In [19]:
```
count=0
str1='hai naresh how are you'
for i in str1:
    if i in 'aeiou':
        count=count+1
print("the count is:",count)

# step-1: i='h'  if 'h' in 'aeiou'  F
```

```
the count is: 9
```

In [ ]:
```
count=0
str1='hai naresh how are you'
for i in str1:
    if i=='a' or i=='e' or i=='i' or i=='o' or i=='u':
        count=count+1
print("the count is:",count)


str1='hai naresh how are you'
count=0
for i in str1:
    if i=='a':
        count=count+1
print(count)

# Idea: counter wrapper
# step-1: count=0
# step-2: using for loop in operator iterate through  given string
# step-3: apply the if condition, whenever the letter equal to 'a'
# step-4: count=count+1
```

In [ ]:
```
# Q3) Home work qn
# str1='hai naresh how are you'
# Repetaed vowels are there
# We dont want repetaed vowels
# Count the vowels avoid the repeatition: Unique vowels

# Step-1: Count=0
# Step-2: take one more empty string: ex= s2=''
s2=''
# Step-3: using for loop in operator iterate through  given string
# step-4: condition-1: That letter shoud not avaialble in s2
# step-5         condition-2: vowel check condition
# step-6                      count=count+1
# step-7:                     update the s2= s2=s2+<letter>
```

How to update the empty strings

In [21]:
```
s1='python'
s2=''
```

```
for i in s1:
    s2=s2+i   # Concatenation

s2
```

Out[21]:
```
'python'
```

Concatenation

In [22]:
```
s1='hello'
s2='bye'
s1+s2
```

Out[22]:
```
'hellobye'
```

In [23]:
```
s1='hello '
s2='bye'
s1+s2
```

Out[23]:
```
'hello bye'
```

In [24]:
```
s1='hello'
s2=' bye'
s1+s2
```

Out[24]:
```
'hello bye'
```

In [26]:
```
s1='hello'
s2=' '
s3='bye'
s4=s1+s2+s3
s4
```

Out[26]:
```
'hello bye'
```

In [ ]:
```
s1='hello'
s2='bye'
s1*s2
s1/s2
s1-s2
```

In [27]:
```
s1='hello'
s2='bye'
s1*s2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[27], line 3
      1 s1='hello'
      2 s2='bye'
----> 3 s1*s2

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

In [28]:
```
s1='hello'
s2='bye'
s1-s2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[28], line 3
      1 s1='hello'
      2 s2='bye'
----> 3 s1-s2

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

In [29]:
```
s1/s2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 s1/s2

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

In [ ]:
```
# s1*s2: can't multiply sequence by non-int of type 'str'

# s1-s2: unsupported operand type(s) for -: 'str' and 'str'

# s1/s2:  unsupported operand type(s) for /: 'str' and 'str'
```

In [30]:
```
s1='hello'
s2=2
s1*s2
```

Out[30]:
```
'hellohello'
```

In [ ]:
```
s1+s1 ===> s1*2

2*3    3*2
```

Index

- Index meaning a number attached to a letter
- In python index starts with zero
- We have two directions
    - Poistive direction
        - Positive numbers starts with zero
    - Negtaive direction
        - Negative numbers starts with -1 , applicable to last letter

In [ ]:
```
s1='python'

-6  -5    -4    -3    -2   -1   === > negative
p    y     t     h     o    n
0    1     2     3     4    5   === > postive
```

In [35]:
```
s1='python'
s1[0],s1[1],s1[2],s1[3],s1[4],s1[5]
```

```
# s1[i]
```

```
Out[35]: ('p', 'y', 't', 'h', 'o', 'n')
```

```
In [36]: s1='python'
         s1[-1],s1[-2],s1[-3],s1[-4],s1[-5],s1[-6]
```

```
Out[36]: ('n', 'o', 'h', 't', 'y', 'p')
```

mutable-immutable

- mutable: can change
    - we can change using indexing
- immutable: can not change
    - we can not change using indexing

```
In [3]: s='welcome'    # welcome
        # I want to replace 'l'  with  'L'
        # Possible
        # Not possible
        s[2]='L'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 5
      1 s='welcome'    # welcome
      2 # I want to replace 'l'  with  'L'
      3 # Possible
      4 # Not possible
----> 5 s[2]='L'

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

Strings are Immutable

```
In [5]: l=[1,2,3,4]    # 2= 200
        l[1]=200
        l
```

```
Out[5]: [1, 200, 3, 4]
```

```
In [ ]: s='welcome'
        s[2]='L'
        ########################################
        l=[1,2,3,4]
        l[1]=200
        l
```

- range belongs to which family
    - Math family
- inside range bracket what we need to provide

    - number
- How many numbers we need to provide
    - Number of letter : len(string)
- How to access the letter using number
    - by using index

In [10]:
```
# Q4) print each letter using for loop range
s='welcome'
# range belongs to which family
# inside bracket we need to provide a number
# How many numbers we need to provide
#       number of letters=7
# How to access a letter using number:
s='welcome'
for i in range(7):
    print(s[i])
# i=0   s[0]=w
# i=1   s[1]=e
```

```
w
e
l
c
o
m
e
```

In [11]:
```
s='welcome to naresh it'
n=len(s)
for i in range(n):
    print(s[i])
```

```
w
e
l
c
o
m
e

t
o

n
a
r
e
s
h

i
t
```

- **for-in** directly access the letter
- **for-range** will access using index

```
In [ ]: # Q5) wap ask the user iterate using a string
        # string='welcome'
        # print the postive index of w is 0
        #       the positive index of e is 1
        #       the positive index of l is 1

        # Q6) wap ask the user iterate using a string
        # string='welcome'
        # print the negative index of w is -7
        #       the negative index of e is -6
        #       the negative index of l is -5
```

```
In [14]: # Q5) wap ask the user iterate using a string
         # string='welcome'
         # print the postive index of w is 0
         #       the positive index of e is 1
         #       the positive index of l is 1
         s='welcome'
         for i in range(len(s)):
             print(i,s[i])
             print(f"The postive index of {s[i]} is {i}")
```

```
0 w
The postive index of w is 0
1 e
The postive index of e is 1
2 l
The postive index of l is 2
3 c
The postive index of c is 3
4 o
The postive index of o is 4
5 m
The postive index of m is 5
6 e
The postive index of e is 6
```

```
In [18]: # Q6) wap ask the user iterate using a string
         # string='welcome'
         # print the negative index of w is -7
         #       the negative index of e is -6
         #       the negative index of l is -5
         # the positive index is 0 and the negtaive index -7 for w
         s='welcome'
         for i in range(len(s)):
             print(i,i-len(s))
             print(f"the negative index of {s[i]} is {i-len(s)}")
         # 0-7==== -7
         # 1-7==== -6
         # 2-7 ==== -5
```

```
0 -7
the negative index of w is -7
1 -6
the negative index of e is -6
2 -5
the negative index of l is -5
3 -4
the negative index of c is -4
4 -3
the negative index of o is -3
5 -2
the negative index of m is -2
6 -1
the negative index of e is -1
```

In [21]:
```
# Q7) wap ask the user iterate using a string
# string='welcome'
# # the positive index is 0 and the negtaive index -7 for w
s='welcome'
for i in range(len(s)):
    print(f"the positive index is {i} and the negtaive index {i-len(s)} for {s[i
```

```
the positive index is 0 and the negtaive index -7 for w
the positive index is 1 and the negtaive index -6 for e
the positive index is 2 and the negtaive index -5 for l
the positive index is 3 and the negtaive index -4 for c
the positive index is 4 and the negtaive index -3 for o
the positive index is 5 and the negtaive index -2 for m
the positive index is 6 and the negtaive index -1 for e
```

In [ ]:
```
#Q8) wap ask the user get the index of each 'a' in a given string
# s='hai how are you i am good'
# a=1  8  18

#Q9) wap ask the user to get count of number of 'a' in a given string
#     using for-range

#Q10) wap ask the user get the sum of all index numbers of 'a'
# 1+8+18= 27

#Q11) wap ask the user get the vowels from a given string using for-range
```

In [23]:
```
#Q8) wap ask the user get the index of each 'a' in a given string
# s='hai how are you i am good'
# a=1  8  18
s='hai how are you i am good'
for i in range(len(s)):
    if s[i]=='a':
        print(i)

s='hai how are you i am good'
for i in s:
    if i=='a':
        print(i)
```

```
1
8
18
a
a
a
```

In [24]:
```
#Q9) wap ask the user to get count of number of 'a' in a given string
#     using for-range

s='hai how are you i am good'
count=0
for i in range(len(s)):
    if s[i]=='a':
        count=count+1

count
```

Out[24]:
```
3
```

In [25]:
```
#Q10) wap ask the user get the sum of all index numbers of 'a'
# 1+8+18= 27
s='hai how are you i am good'
count=0
for i in range(len(s)):
    if s[i]=='a':
        summ=summ+i

summ
```

Out[25]:
```
27
```

In [27]:
```
#Q11) wap ask the user get the vowels from a given string using for-range
s='hai how are you i am good'
count=0
for i in range(len(s)):
    if s[i] in 'aeiou':
        print(s[i])
        count=count+1

print("The number of vowels are:",count)
```

```
a
i
o
a
e
o
u
i
a
o
o
The number of vowels are: 11
```

In [30]:
```
#Q12) wap ask the user get the vowels from a given string using for-range
s='hi how re you i m good'
s1=''
count=0
for i in range(len(s)):
    if s[i] not in s1:  # we are checking the letter is available in s1( It shou
        if s[i] in 'aeiou': # we are checking vowel condition
            print(s[i])
            s1=s1+s[i]      # we need to update the s1
            count=count+1

print("The number of vowels are:",count)
```

```
i
o
e
u
The number of vowels are: 4
```

In [31]:
```
#Q13)updated wap ask the user get the vowels from a given string using for-range
s='hi how re you i m good'
s1=''
count=0
for i in range(len(s)):
    if s[i] not in s1 and s[i] in 'aeiou':
        print(s[i])
        s1=s1+s[i]
        count=count+1

print("The number of vowels are:",count)
```

```
i
o
e
u
The number of vowels are: 4
```

In [ ]:
```
# Q14) string1='ola ola ola'
# Number of ola  = 3

# Q15) string1='hello hello hello how how how how are you'
# What is the most repeated word : how

# Q16) string1='hellooooo how aree u'
# what is the maximum length of word: helloooo
# what is the minimum length of word: u

# Q17) with out using sorted then sort the letters
```

In [32]:
```
sorted('hello')
```

Out[32]:
```
['e', 'h', 'l', 'l', 'o']
```

Slice

- we can cut into pieces
- Similar to range concept
- start , stop ,step here also

In [1]:
```
str1='hai how are you'
```

```
In [ ]: -15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
         h    a    i         h    o    w        a   r    e        y   o   u
         0    1    2    3    4    5    6   7    8   9   10   11  12  13  14
```

```
In [ ]: str1[start:stop:step]
```

$pattern - 1$

str1[start:]

- By default start value = start only
- Last value nothing mentioned automatically it will go till = last only
- Step nothing mentioned means , It is a postive direction and increment by 1

```
In [2]: #-15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
        # h    a    i         h    o    w        a   r    e        y   o   u
        # 0    1    2    3    4    5    6   7    8   9   10   11  12  13  14
        str1='hai how are you'
        str1[5:]

Out[2]: 'ow are you'
```

```
In [3]: str1[10:]

Out[3]: 'e you'
```

```
In [4]: str1[-5:]

Out[4]: 'e you'
```

```
In [5]: str1[5:],str1[-10:]  # Both will give same answer

Out[5]: ('ow are you', 'ow are you')
```

```
In [6]: str1[14:]

Out[6]: 'u'
```

```
In [7]: str1[-1:]

Out[7]: 'u'
```

$Pattern - 2$

str1[start:stop]

- start value means by default start only
- step is not mentioned postive direction
- last = stop-1

```
In [8]: #-15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
        # h    a     i         h    o    w        a   r    e        y   o  u
        # 0    1     2    3    4    5    6    7   8   9   10   11  12  13  14

        str1='hai how are you'
        str1[5:14]

Out[8]: 'ow are yo'

In [9]: str1[-5:14]
        # start=-5
        # last = 14-1 =13   positive

Out[9]: 'e yo'

In [11]: str1[-5:14]

Out[11]: 'how are'

In [13]: str1[-5:-14]   #  Postive

Out[13]: ''

In [ ]: #-15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
        # h    a     i         h    o    w        a   r    e        y   o  u
        # 0    1     2    3    4    5    6    7   8   9   10   11  12  13  14
        str1='hai how are you'  # step is mentioned or not mentioned
        str1[5:14],   # Answer
        str1[-5:14],  #  Answer
        str1[5:-14],  # No answer
        str1[-5:-14]  # No answer

In [14]: str1[5:500]

Out[14]: 'ow are you'
```

Truncation behaviour

- when ever string range is limited, but we provided unlimited number
- string will consider till the range we have
- In the above example our range only 15 letters
- we provided 500 , then 500 will truncate to till 15 letters only

```
In [ ]: str1[5:500] #  Answer
        str1[-500:500] # Answer
        str1[-1:500]  # Answer
        str1[500:]    # no answer
        str1[-500:-100] # No answer
```

$Pattern - 3$ **str1[start:stop:step]**

- start value means start only

- If step value positive
    - last value= stop-1
- If step value negative
    - last value=stop+1

In [15]:
```
len(str1)
```

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

In [ ]:
```
str2='yeshwanth'

y  e  s  h  w  a  n  t h
0  1  2  3  4  5  6  7 8
```

In [ ]:
```
#-15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
# h    a     i         h    o    w       a   r    e        y   o  u
# 0    1     2    3    4    5    6   7   8   9   10   11  12  13  14

str1[2:13:2]  # P
str1[2:13:-2] # NP
str1[2:-13:2] # NP (CHECK)  str1[2:-13]  str1[2:3:2]  str1[2:2]
str1[2:-13:-2] # NP (CHECK)
str1[-2:13:2]  # NP (CHECK)
str1[-2:-13:2] # NP
str1[-2:-13:-2] # P
str1[13:2:2] # NP
str1[-13:2:2] # NP (CHECK)
str1[-13:-2:2] # P
str1[-13:-2:-2] # NP
str1[-13:2:-2]  # NP(CHECK)
```

In [ ]:
```
==================== PART-1=====================================
- Intiaization

- type

- len

- max

- min

- sorted

- reveresed

- in

- for loop using in

- index

- for loop using index

- mutable
- concatenation

- slice

#================== PART-2=====================
# Methods
```

In [16]:
```
#-15  -14  -13  -12  -11  -10  -9  -8   -7  -6  -5  -4  -3  -2  -1
# h    a     i         h    o    w        a   r    e        y   o  u
# 0    1     2    3    4    5    6   7    8   9   10   11  12  13  14

str1[:5:-2]  # start value  -1
```

Out[16]:
```
'uyeaw'
```

In [ ]:
```
str1[-13:2:2]  # postive direction    start value :0
```

In [ ]:
```
str1[:]  # Complete string
str1[::] # Complete string
str1[<start>::-1] # Reveres  start value=-1
str1[<start>:5:-2]
```

Methods

- Strings lists tuple dictionary every data type has its own methods
- We already seen that packages has different method
- For example random package : randint method
- Math package : sqrt
- In the same way strings also have methods
- In order to see methods for the packages
  - we are importing the package
  - we are applying dir
- In the similar way in order to see the methods of string , we need to apply dir only

In [17]:
```
dir('')
# dir('hai')
# str1='apple'
# dir(str1)
```

```
Out[17]: ['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

- lower
- upper
- capitalize
- Title
- casefold
- Center

upper

In [18]:
```
str1='hai how are you'
```

In [19]:
```
type(str1)
help(str1.upper)
```

```
Help on built-in function upper:

upper() method of builtins.str instance
    Return a copy of the string converted to uppercase.
```

In [20]:
```
str1.upper()
```

Out[20]:
```
'HAI HOW ARE YOU'
```

In [21]:
```
str1.upper()
```

Out[21]:
```
<function str.upper()>
```

In [22]:
```
str1='hai how are you'
str1.upper()
```

```
Out[22]:  'HAI HOW ARE YOU'
```

lower

```
In [23]: help(str1.lower)
```

```
Help on built-in function lower:

lower() method of builtins.str instance
    Return a copy of the string converted to lowercase.
```

```
In [24]: str1.lower()
```

```
Out[24]:  'hai how are you'
```

capitalize

```
In [26]: str1.capitalize()
```

```
Out[26]:  'Hai how are you'
```

casefold

```
In [27]: str1.casefold()
```

```
Out[27]:  'hai how are you'
```

Title

```
In [28]: str1.title()
```

```
Out[28]:  'Hai How Are You'
```

```
In [ ]: # str1='hai how are you'
        # o/p = 'Hai How Are You'
        # with out using any method
        # One more level : iterate each letter apply capitalize
        # scratch : ord char
```

center

```
In [30]: str1.center(20)
```

```
Out[30]:  '  hai how are you   '
```

```
In [31]: len(str1.center(20))
```

```
Out[31]:  20
```

- center has two arguments
    - width
    - fill char

- original string alread : 15 lettters
- we want to create a new string with 20 letters
- Remaining 5 letters by default empty
- we can fill with charcters also

```
In [32]: str1.center(26,'*')
```

```
Out[32]: '*****hai how are you******'
```

```
In [34]: str1.center()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 1
----> 1 str1.center()

TypeError: center expected at least 1 argument, got 0
```

- upper
- lower
- casefold
- Title
- Capitalize
- Center

Count

```
In [1]: str1='hai how are you'
        # How many 'a's are there
```

```
In [2]: count=0
        for i in str1:
            if i=='a':
                count=count+1
        count=0
```

```
In [7]: str1='hai how are you'
        str1.count('a')
```

```
Out[7]: 2
```

```
In [8]: str1='hai hai hai'
        str1.count('a')
```

```
Out[8]: 3
```

- We want number of a from a specific index
- we want number of a between two indexes

In [12]:
```
#h  a   i    h   a   i    h   a   i
#0  1   2 3  4   5   6  7 8   9   10
str1.count('a')  # All the 'a'
str1.count('a',5)
```

Out[12]:
```
2
```

In [ ]:
```
str1.count('a',3)
# we are searching number of 'a'  from index=3
```

In [13]:
```
#h  a   i    h   a   i    h   a   i
#0  1   2 3  4   5   6  7 8   9   10
str1.count('a')  # All the 'a'
str1.count('a',5) # we are searching number of 'a'  from index=3
str1.count('a',4,8) #we are searching number of 'a'  from index=4 to index=8
```

Out[13]:
```
1
```

In [ ]:
```
# Reverse check karo
```

In [14]:
```
str1='ola ola ola'
str1.count('ola')
```

Out[14]:
```
3
```

In [ ]:
```
# In the interviwes he will ask with out using method
# After you got job
```

In [15]:
```
str1='ola ola ola'
for i in str1:
    if i=='ola':   # 'o'  == 'ola'
        print(i)
```

Window method

In [ ]:
```
str1[i:i+3]
i=0   str1[0:3]  === > ola
i=1   str1[1:4]  ==== > la
i=2
```

In [ ]:
```
str1='ola ola ola'
count=0
for i in range(len(str1)):
    if str1[i:i+3]=='ola':
        count=count+1

# step-1: count=0   i =0  str1[0:3]=='ola'  True  count=1
# step-2:           i=1   str1[1:4]=='ola'  F
```

In [20]:
```
str1='ola ola ola'
```

```
str1.count('ola ola ola')
```

```
Out[20]: 1
```

```
In [21]: str1='ola ola ola'
         str1.count(str1)
```

```
Out[21]: 1
```

```
In [22]: str1.count('z')
         # No answer ''
         # Error
         # 0
         # NT
```

```
Out[22]: 0
```

Replace

```
In [23]: str1='welcome'
         # I want to replace 'l'  with  'L'
```

```
In [ ]: # we know that strings are immutable
        # we can not use index operations also
        # slice and concatenation
        # Divide welcome :  we    come
        # s1='we'
        # s2='come'
        # s1+'L'+s2

        sir, you said string is immutable
        it won't change but if we are doing concatenation
        s1=s1+'anything' total value of s1 is changing write sir how
```

```
In [26]: str1='welcome'
         s1=str1[0:2]
         s2=str1[3:]
         s1+'L'+s2
```

```
Out[26]: 'weLcome'
```

```
In [29]: chr(ord('l')-32)  #Try this
```

```
Out[29]: 'L'
```

```
In [30]: str1='welcome'
         str1.replace('l','L')
```

```
Out[30]: 'weLcome'
```

```
In [31]: str1='wellcome'
         str1.replace('l','L')
```

```
Out[31]: 'weLLcome'
```

- By defualt replace will change all the occurences

- count= -1 is responsible for that
- whcih means we can provide some count also, which how many I want to change
- Give welllcome and change count=1 and count=2 explore it

```
In [33]: str1='welllcome'
         str1.replace('l','L',1)  # old='l', new ='L'  count=1

         # when we write count=1
         # It will change only one letter
         # that to first occurence only

Out[33]: 'weLllcome'
```

```
In [35]: str1='welllcome'
         str1.replace('l','L',2)

Out[35]: 'weLLlcome'
```

```
In [40]: # Str1='restart'
         # i/p= 'resta$t'
         str1='restart'
         s1=str1[:1]
         s2=str1[1:]
         s3=s2.replace('r','$')
         s1+s3

Out[40]: 'resta$t'
```

```
In [ ]: Q. Why is replacing -1 is replaced at all places. str1='welllcome'
        str1.replace('l','L',-1)
        # Flag=-1  True Flase ALL N
```

```
In [44]: str1='restart'  # This case
         str1[::-1].replace('r','$',1)[::-1]

Out[44]: 'resta$t'
```

index-find

```
In [ ]: - Upper

        - lower

        - casefold

        - title

        - capitalize

        - count

        - replace

        - index-find
        - split

        - strip

        - startswith
```

In [45]:
```
dir('')
```

```
Out[45]:  ['__add__',
           '__class__',
           '__contains__',
           '__delattr__',
           '__dir__',
           '__doc__',
           '__eq__',
           '__format__',
           '__ge__',
           '__getattribute__',
           '__getitem__',
           '__getnewargs__',
           '__getstate__',
           '__gt__',
           '__hash__',
           '__init__',
           '__init_subclass__',
           '__iter__',
           '__le__',
           '__len__',
           '__lt__',
           '__mod__',
           '__mul__',
           '__ne__',
           '__new__',
           '__reduce__',
           '__reduce_ex__',
           '__repr__',
           '__rmod__',
           '__rmul__',
           '__setattr__',
           '__sizeof__',
           '__str__',
           '__subclasshook__',
           'capitalize',
           'casefold',
           'center',
           'count',
           'encode',
           'endswith',
           'expandtabs',
           'find',
           'format',
           'format_map',
           'index',
           'isalnum',
           'isalpha',
           'isascii',
           'isdecimal',
           'isdigit',
           'isidentifier',
           'islower',
           'isnumeric',
           'isprintable',
           'isspace',
           'istitle',
           'isupper',
           'join',
           'ljust',
           'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

In [ ]:
```
'isalnum',
'isalpha',
'isascii',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
```

In [47]:
```
str1='HELLO'
str1.isupper()
```

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

In [2]:
```
str1='hello'
str1.istitle()
```

Out[2]:
```
False
```

In [3]:
```
str1='hello'
str1.isupper()
```

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

Index

- index says that it will give the index of any letter in a given string
- we already seen about count: Count will give how many letters are there in a string
    - at what index you want to count

        - between indexes also we can count the letters
    - index meaning it will provide the index
    - imagine that there same letters repeated
    - how can we find the next index

```
In [4]: #h  a   i     h   a  i    h  a  i
        #0  1   2  3  4   5  6  7 8 9 10
        str1= 'hai hai hai'
        str1.index('a')
        # Return the lowest index in S where substring sub is found,

Out[4]: 1

In [5]: # I want to know next 'a' index after 3rd index
        str1= 'hai hai hai'
        str1.index('a',3)

Out[5]: 5

In [7]: # I want to know 'a' index between 3rd and 7th index
        #h  a   i     h   a  i    h  a  i
        #0  1   2  3  4   5  6  7 8 9 10
        str1= 'hai hai hai'
        str1.index('a',3,7)

Out[7]: 5

In [8]: # I always a Return a lowest index only
        str1= 'hai hai hai'
        str1.index('a')      # among all the indexes the lowset index is '1'

        str1= 'hai hai hai'
        str1.index('a',3)   # after 3rd index the lowset index of a is '5'

        str1= 'hai hai hai'
        str1.index('a',3,7)  # Between 3-7 index the lowset index is '5' only

Out[8]: 5

In [13]: str1= 'hai hai hai'
         str1.index('a',-3,-1)      # among all the indexes the lowset index is '1'

Out[13]: 9

In [14]: # -11  -10   -9  -8  -7  -6  -5  -4  -3  -2  -1
         # h     a     i       h   a  i       h   a   i
         # 0     1     2  3    4   5  6   7     8  9 10

         str1= 'hai hai hai'
         str1.index('a',-9,-4)      # among all the indexes the lowset index is '1'

Out[14]: 5

In [15]: str='hai hai hai'
         print(str.index('a',-9,-6))
```

```
# Postive direction
# start= start = -9
# last =end-1 = -6-1= -7
# -9 -8 -7
#  i     h
# there is no 'a'
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 2
      1 str='hai hai hai'
----> 2 print(str.index('a',-9,-6))

ValueError: substring not found
```

In [19]:
```
# -11  -10   -9  -8  -7  -6  -5  -4  -3  -2  -1
# h      a     i      h   a  i        h   a   i
# 0      1     2  3    4   5  6   7     8  9 10
str1= 'hai hai hai'
print(str1.count('a'))
print(str1.count('a',3))
print(str1.count('a',3,7))
```

```
3
2
1
```

In [20]:
```
# -11  -10   -9  -8  -7  -6  -5  -4  -3  -2  -1
# h      a     i      h   a  i        h   a   i
# 0      1     2  3    4   5  6   7     8  9 10
str1= 'hai hai hai'
print(str1.index('a'))
print(str1.index('a',3))
print(str1.index('a',3,7))
```

```
1
5
5
```

In [ ]:
```
Sir, yesterday assignment methods are string methods right?
Why those are not visible in string package? Those are visible with dir('') not

import string
name='praveen'
dir(name)
dir(string)
```

In [21]:
```
str1='hai hai hai'
# I want to know all the indexes of 'a'
for i in range(len(str1)):
    if str1[i]=='a':
        print(i)
```

```
1
5
9
```

In [26]:
```
# h      a     i      h   a  i        h   a   i  hai
# 0      1     2  3    4   5  6   7     8  9 10
str1='hai hai hai hai'
i1=str1.index('a')
```

```
print(i1)
i2=str1.index('a',i1+1)
print(i2)
i3=str1.index('a',i2+1)
print(i3)
i4=str1.index('a',i3+1)
print(i4)
```

```
1
5
9
13
```

In [35]:
```
# Home work Implement above logic to get all the indexes
# h     a     i       h    a  i        h   a   i  hai
# 0     1     2  3    4    5  6   7     8  9 10
str1='hai hai hai hai'
i1=str1.index('a')
print(i1)  # 1
i2= str1.index('a',i1+1)
print(i2)  #  5
i3= str1.index('a',i2+1)
print(i3)
i4= str1.index('a',i3+1)
print(i4)
i5= str1.index('a',i4+1)
print(i5)
```

```
1
5
9
13
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 13
     11 i4= str1.index('a',i3+1)
     12 print(i4)
---> 13 i5= str1.index('a',i4+1)
     14 print(i5)

ValueError: substring not found
```

Find

In [33]:
```
str1='hai hai hai hai'
i1=str1.find('a')
print(i1)  # 1
i2= str1.find('a',i1+1)
print(i2)  #  5
i3= str1.find('a',i2+1)
print(i3)
i4= str1.find('a',i3+1)
print(i4)
i5= str1.find('a',i4+1)
print(i5)
```

```
1
5
9
13
-1
```

In [ ]:
```python
str1.index()  # Raises ValueError when the substring is not found.
str1.find()  # Return -1 on failure.
```

In [36]:
```python
str1='hai hai hai'
str1.count('z')
```

Out[36]:
```
0
```

In [37]:
```python
str1='hai hai hai'
str1.replace('z','Z')
```

Out[37]:
```
'hai hai hai'
```

- If substring not found
- Count method will give zero
- Replace method will give original string
- Index will give **sub string not found error**
- Find will give -1

In [ ]:
```python
str1='omkar.nallagoni@cognizant.com'
# Fisrt name= omkar
# second name= nallagoni
# company name= cognizant
str2='virat.kohli@rcb.com'
str3='rohit.sharma@mi.com'
str4='a.b@c.com'

# Idea : find the triggers
# For first name  .
# second name    .  and @
# Compnay name    @ and second .
# i1= first dot index
# i2= @ index
# i3= second index
```

In [38]:
```python
str1='omkar.nallagoni@cognizant.com'
i1=str1.index('.')
i2=str1.index('@')
i3=str1.index('.',i1+1)
first_name=str1[:i1]
second_name=str1[i1+1:i2]
cname=str1[i2+1:i3]
first_name,second_name,cname
```

Out[38]:
```
('omkar', 'nallagoni', 'cognizant')
```

- count
- replace
- find
- index

- Upper/lower/casefold
- Capitalize/Title
- Center
- Count
- Replace
- index/find
- start with is

In [ ]:
```
- split

- strip/lstrip/rstrip

- startswith/endswith
```