Session-12

Seperating without string methods -

looping a string

```
sample = 'a b c'
sep = ' '
res = [ ]
for ch in sample:
    if ch == sep:
        continue
    res.append(ch)
print(res)
```

o/p ⇒ ['a','b','c']

with string method (split) -

```
sample = 'a b c'
print(sample.split())
```

o/p ⇒ ['a','b','c']

3) find 4) index 5) Count 6) replace

find( )

```
a = "python,java,c++,go"
print(a.find("c++"))
```

// returns the start index of the given element.

o/p ⇒ 12

```
a = "neha"
print(a.find("Neha"))
```

o/p ⇒ -1 // element is not present.

```
s = "python,c++,java,go"
ans = ""
str_search = "c++"        → sub string
str_idx = s.find(str_search)
if str_idx == -1:
    ans = "Not found"
else:
    ans = str_idx
print(f"{str_search}:", ans)
```

o/p ⇒ c++:7

- if present, prints index.
- if not, returns -1.

index( )

```
a = "python,java,c++"
print(a.index("java"))
```

o/p ⇒ 7

- if present, returns index
- if not, raises an error.

```
s = "python,c++,java,go"
ans = ""
str_search = "python"
if str_search in s:
    ans = s.index(str_search)
else:
    ans = "Not found"
print(f"{str_search}:", ans)
```

o/p ⇒ 0

count( ) ⇒ if not present, returns 0

```
a = 'a,b,c,d,e,f,a,a,a'
print(a.count('a'))
```

o/p ⇒ 4

replace( )

```
a = "a,b,c,d,e,f"
a = a.replace("a,b,c","p,q,r")
print(a)
```

o/p ⇒ p,q,r,d,e,f

7) Capitalize( )

```
s = "abcdEf"
print(s.isupper()) | print(s.islower())
```

o/p ⇒ false | o/p ⇒ false

```
a = "abcdEf"
print(s.capitalize())
```

o/p ⇒ Abcdef // first letter Capital

- strip()
- lstrip()
- rstrip()

Session - 13

swapcase()

```
s = "Neha"
output = []
for char in s:
    output.append(char.swapcase())
print(output)
```

o/p ⇒ ['n', 'E', 'H', 'A']

```
print("".join(output))
```

o/p ⇒ nEHA

Functions in Python -

→ input(), int(), float(), str(), tuple(), dict(), list(), set(), etc are predefined functions ⇒ shipped with installation.

→ Using functions, modularity in our program is achieved.

Ex -

```
def add():          # def → keyword (define), add → function name
    print(2+3)
    print(3+4)
add()               # calling function (2)
```

function definition (1)

o/p ⇒ 5
7

arguments

```
def add(a,b):
    print(a+b)
add(4,6)            # a, b
```

→ fixed by default (default arguments)

o/p ⇒ 10

```
def sub(a,b):
    print(a-b)
sub(b=10,a=5)
sub(a=10,b=5)
```

} positional arguments

o/p ⇒ -5
5

```
def add(a,b):
    ans = a+b
    return ans
res = add(a=2,b=3)
res = res+10
print(res)
```

o/p ⇒ 15

```
def add(a,b,c=10):
    return a+b+c
res = add(2,5)
print(res)
```

o/p ⇒ 17

Linear Search -

```
given_list = [1,2,3,10,20,30]
search_element = 10
def linear_search(given_list, search_element):
    for idx,ele in enumerate(given_list, search_element):
        if ele == search_element:
            return True, idx
    return False, -1.
given_list = [1,2,3,10,20,30]
search_element = 10
res = linear_search(given_list, search_element)
print(type(res))
print(res)
```

o/p ⇒ (True, 3) tuple.