Slicing
Slicing is an operation of reading multiple values from sequence
Slicing is supported by only sequence data types

What is difference between indexing and slicing?
Indexing and slicing are both methods used to access elements within sequences like strings, lists, and tuples in Python, but they serve different purposes. Indexing retrieves a single element, while slicing extracts a subsequence.

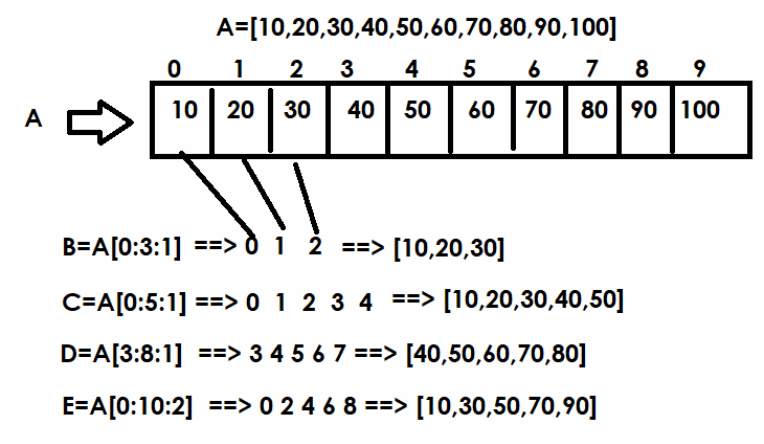
Slicing is done in two ways
1. slice operator
2. slice object

slicing generates multiple indexes for reading multiple values.

Slice operator
Slice operator required 3 inputs
1. start index (Included)
2. stop index (Excluded)
3. step

Note: internally slice operator or slice object uses range for generating multiple indexes

Syntax: sequence-name[start:stop:step]




```
>>> A=[10,20,30,40,50,60,70,80,90,100]
>>> print(A)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> B=A[0:3:1]
>>> print(B)
[10, 20, 30]
>>> C=A[0:5:1]
>>> print(C)
[10, 20, 30, 40, 50]
>>> D=A[4:9:1]
>>> print(D)
[50, 60, 70, 80, 90]
>>> E=A[0:10:2]
>>> print(E)
[10, 30, 50, 70, 90]
>>> F=A[-5:-1:1]
>>> print(F)
[60, 70, 80, 90]
>>> G=A[-5:-11:-1]
>>> print(G)
[60, 50, 40, 30, 20, 10]
>>> H=A[9:0:-3]
>>> print(H)
[100, 70, 40]
```

Syntax1: sequence-name[::] (OR) sequence-name[:]
In this syntax default start=0,stop=len(sequence),step=1
This syntax creates copy of the list

```
>>> A=[10,20,30,40,50]
>>> print(A)
[10, 20, 30, 40, 50]
>>> B=A[::]
>>> print(B)
[10, 20, 30, 40, 50]
>>> C=A[:]
>>> print(C)
[10, 20, 30, 40, 50]
```

Syntax2: sequence-name[::step]
In this syntax start and stop values are taken based step value
If step +ve, start=0,stop=len(sequence) ➔ L-R

If step –ve, start=-1,stop=-(len(sequence)+1) ➔ R-L

```
>>> A=[10,20,30,40,50]
>>> print(A)
[10, 20, 30, 40, 50]
>>> B=A[::1]
>>> print(B)
[10, 20, 30, 40, 50]
>>> C=A[::2]
>>> print(C)
[10, 30, 50]
>>> D=A[::-1]
>>> print(D)
[50, 40, 30, 20, 10]
>>> E=A[::-2]
>>> print(E)
[50, 30, 10]
```

Syntax3: sequence-name[start::] (OR) sequence-name[start:]
In this syntax default stop=len(sequence) and step=1

```
>>> A=list(range(10,110,10))
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> B=A[5:]
>>> print(B)
[60, 70, 80, 90, 100]
>>> C=A[-5:]
>>> print(C)
[60, 70, 80, 90, 100]
>>> D=A[-2:]
>>> print(D)
[90, 100]
```

Syntax4: sequence-name[:stop:] (OR) sequence-name[:stop]
Default start=0,step=1

```
>>> A=list("PROGRAMMING")
>>> print(A)
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'I', 'N', 'G']
```

```
>>> B=A[:5]
>>> print(B)
['P', 'R', 'O', 'G', 'R']
>>> C=A[:-5]
>>> print(C)
['P', 'R', 'O', 'G', 'R', 'A']
>>> D=A[:-2]
>>> print(D)
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'I']
>>> E=A[:3]
>>> print(E)
['P', 'R', 'O']
>>> A=[10,20,30,40,50,60,70,80,90,100]
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> B=A[3:-3]
>>> print(B)
[40, 50, 60, 70]
```

Slice object

Slice object persist or save slice parameters or values

1. start
2. stop
3. step

Syntax: slice(start,stop,[step]) ➔ default step=1
Syntax: slice(stop) ➔ default start=0,step=1

Example:

```
slice1=slice(0,5,1)
slice2=slice(0,5,2)

A=[10,20,30,40,50,60]
B=[100,200,300,400,500,600]

D=A[slice1]
E=B[slice1]

F=A[slice2]
```

```
G=B[slice2]

print(A,B,D,E,F,G,sep="\n")
```

Output

```
[10, 20, 30, 40, 50, 60]
[100, 200, 300, 400, 500, 600]
[10, 20, 30, 40, 50]
[100, 200, 300, 400, 500]
[10, 30, 50]
[100, 300, 500]
```

for loop

for loop iterate/read values from any iterable, it read all the values one by one.

Syntax:

```
for variable-name in iterable:
      statement-1
      statement-2
```

Python executes statement-1,statement-2 until read all the values from iterable (sequences, sets, mapping or any collection).

Example:

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

for x in A:
   print(x)
```

Output

```
10
20
30
40
50
```

Example:

```
# Write code to find length list without
# using predefined function (len)

A=[10,20,30,40,50,60]
c=0
for x in A:
    c=c+1

print(f'List {A}')
print(f'Length {c}')
```

Output
List [10, 20, 30, 40, 50, 60]
Length 6

Example

```
# Find Output

A=[10,20,30,40,50,60,70,80,90,100]

for x in A[:5]:
    print(x)

for x in A[-5:]:
    print(x)

for x in A[::-2]:
    print(x)
```

Output

```
10
20
30
40
50
60
70
80
90
100
```

100
80
60
40
20