Slice object

Slice object allows to store or save slice values, so that it can be used one or more than time.

How to create slice object?

Syntax: slice(start,stop,step)

Example:

```
A=[10,20,30,40,50,60,70,80,90,100]
print(A)
s1=slice(0,3,1)
B=A[s1]
print(B)
C=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
D=C[s1]
print(C)
print(D)
```

Output

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[1, 2, 3]
```

What is difference between slice operator and slice object?

Slice Operator	Slice Object
The slice operator is a syntax used to extract a portion of a sequence (like lists, strings, tuples).	A slice object is an object created by the built-in slice() function or by using the slice operator directly.

It is denoted by square brackets [] with colons : separating the start, stop, and step values.	It encapsulates the start, stop, and step values for a slice operation.
This is applied directly one sequence	This can be applied to any number of sequences

for loop

"for" is keyword which represents for loop
In python for loop is used to read/iterate values from iterables/collections. For loop each time read one value from iterable/collection and execute block of statements

Syntax:

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

Example:

```
A=[10,20,30,40,50]
for x in A:
   print(x)
```

Output

```
10
20
30
40
50
```

Example:

```
# Write a program to find length of list without
# using len() function

A=[10,60,30,56,34,89,23,56,87,21,34,65,87,98,99,67]

c=0
for value in A:
    c=c+1

print(f'List is {A}')
print(f'Count is {c}')
```

Output

```
List is [10, 60, 30, 56, 34, 89, 23, 56, 87, 21, 34, 65, 87, 98, 99, 67]
Count is 16
```

Example:

```
# Write a program to count even and odd number

A=[10,60,30,56,34,89,23,56,87,21,34,65,87,98,99,67]
c1=0
c2=0
for value in A:
    if value%2==0:
        c1+=1
    else:
        c2+=1
print(f'List is {A}')
print(f'Even Count {c1}')
print(f'Odd Count {c2}')
```

Output

List is [10, 60, 30, 56, 34, 89, 23, 56, 87, 21, 34, 65, 87, 98, 99, 67]
Even Count 8
Odd Count 8

Example:

```
# Write a program to print sum of list
A=[10,60,30,56,34,89,23,56,87,21,34,65,87,98,99,67]
s=0
for x in A:
    s=s+x

print(f'List is {A}')
print(f'Sum of elements of list is {s}')
```

Output

List is [10, 60, 30, 56, 34, 89, 23, 56, 87, 21, 34, 65, 87, 98, 99, 67]
Sum of elements of list is 916

Iterator

iter() is a predefined function in python which is used to create iterator object. This object is used to read or iterate values from iterables.

next() is a predefined function in python, this function is used to read values from iterable using iterator object.

Syntax: iter(iterable)
Syntax: next(iterator)

Note: iterator object is used for non index based collections.

Example:

```
A=[10,20,30,40,50]
x=iter(A)
n1=next(x)
n2=next(x)
n3=next(x)
n4=next(x)
n5=next(x)
#n6=next(x) Error
print(n1,n2,n3,n4,n5)
```

Output

10 20 30 40 50

Enumerator

enumerate() is a predefined function in python used to create enumerate object using iterable/collection.

next() function is used by enumerate object to read each value from iterable.

Syntax: enumerate(iterable,start=0)

Start is nothing but count, default count is 0

Enumerate returns two values

1. Value read from iterable
2. Count

These 2 values are return in one tuple

Example:

```
A=[10,20,30,40,50]
e=enumerate(A)
t1=next(e)
t2=next(e)
t3=next(e)
print(t1,t2,t3,sep="\n")

sales=[1000,2000,3000,4000,5000]
e=enumerate(sales,start=1993)
s1=next(e)
print(s1)
s2=next(e)
print(s2)
s3=next(e)
print(s3)
```

Output

```
(0, 10)
(1, 20)
(2, 30)
(1993, 1000)
(1994, 2000)
(1995, 3000)
```

In application development enumerate is used for generating content for other collection like dictionary and to read values with count.

Mutable Operations of List

List is a mutable collection after creating list changes can be done. These changes are done using methods/functions provided by list.

1. append()
2. extend()
3. insert()
4. remove()
5. sort()
6. pop()
7. clear()
8. del keyword

append()

append() is predefined method of list
This method is used to add an item at the end of list.

Syntax: list-name.append(item)

Note: collections are dynamic in side. Any number of values can be added with collection.

Example:

```
A=[]
print(A)
A.append(10)
A.append(20)
A.append(30)
A.append(40)
A.append(50)
print(A)
```

Output

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

Example:

```
# Write a program to input 5 values in list during runtime

A=[]
print(f'Before Adding {A}')
for i in range(5):
    value=int(input("Enter Value "))
    A.append(value)

print(f'After Adding {A}')
```

Output

```
Before Adding []
Enter Value 1
Enter Value 2
Enter Value 3
Enter Value 4
Enter Value 5
After Adding [1, 2, 3, 4, 5]
```

Example

```
# Write a program to read scores of n players
# total score,maximum score,minimum score

players=[]
n=int(input("Enter How many Players ?"))
for i in range(n):
    s=int(input("Enter Score :"))
    players.append(s)

total=sum(players)
```

```
max_score=max(players)
min_score=min(players)
print(f'Scores are {players}')
print(f'Total Score {total}')
print(f'Maximum Score {max_score}')
print(f'Minimum Score {min_score}')
```

Output

```
Enter How many Players ?5
Enter Score :10
Enter Score :25
Enter Score :5
Enter Score :60
Enter Score :30
Scores are [10, 25, 5, 60, 30]
Total Score 130
Maximum Score 60
Minimum Score 5
```

```
>>> A=[]
>>> A.append(10,20)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    A.append(10,20)
TypeError: list.append() takes exactly one argument (2 given)
```

How to append multiple values?

Multiple values or more than one value can be append using different approaches

1. extend() method
2. using slicing operator

Syntax1: list-name.extend(iterable)

Syntax2: list-name[start-index:]=iterable
Start-index for append is last index/len of list

Example:

```
A=[10,20]
print(A)
A.extend([40,50,60,70])
print(A)
A.extend("NIT")
print(A)
A.extend(range(1,6))
print(A)
B=[10,20]
print(B)
B[2:]=[30,40,50,60]
print(B)
B[len(B):]="NIT"
print(B)
B[len(B):]=range(1,6)
print(B)
```

Output

```
[10, 20]
[10, 20, 40, 50, 60, 70]
[10, 20, 40, 50, 60, 70, 'N', 'I', 'T']
[10, 20, 40, 50, 60, 70, 'N', 'I', 'T', 1, 2, 3, 4, 5]
[10, 20]
[10, 20, 30, 40, 50, 60]
[10, 20, 30, 40, 50, 60, 'N', 'I', 'T']
[10, 20, 30, 40, 50, 60, 'N', 'I', 'T', 1, 2, 3, 4, 5]
```

Example:

```
cart1=["mouse","keyboard"]
cart2=["monitor","printer"]
print(f'Cart1 {cart1}')
print(f'Cart2 {cart2}')
cart1.extend(cart2)
print(cart1)
```

Output

```
Cart1 ['mouse', 'keyboard']
Cart2 ['monitor', 'printer']
['mouse', 'keyboard', 'monitor', 'printer']
```

Example:

```
>>> A=[10,20]
>>> A.append([50,60])
>>> A
[10, 20, [50, 60]]
>>> A.extend([40,50])
>>> A
[10, 20, [50, 60], 40, 50]
```

Replace or Update values

List is mutable collection, it allows replacing of values.
This replacing is done in 2 ways

1. using index
2. using slicing

Using index

Index is used to replace or update one value/single value

Syntax: list-name[index]=value

If index within range it replace value
If index not within range it raises error(indexerror)

Example:

```
>>> A=[10,20,30,40,50]
>>> print(A)
[10, 20, 30, 40, 50]
>>> A[0]=99
>>> print(A)
[99, 20, 30, 40, 50]
>>> A[-1]=66
>>> print(A)
[99, 20, 30, 40, 66]
>>> A[2]=55
>>> print(A)
[99, 20, 55, 40, 66]
>>> A[6]=88
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    A[6]=88
IndexError: list assignment index out of range
>>> A[-7]=99
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    A[-7]=99
IndexError: list assignment index out of range
```

Example:

```
# Python program to interchange first and last elements
# in a list

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

```
print(f'Before Swaping {A}')
A[0],A[-1]=A[-1],A[0]
print(f'After Swaping {A}')
```

Output

```
Before Swaping [10, 20, 30, 40, 50]
After Swaping [50, 20, 30, 40, 10]
```