Sequential Data Structures: List, Tuple, and Range

List

A list contains a sequence of items.

```
X = ["John", 1992, 3.14, False, [3, 4]]
```

string	integer	float	bool	list

List elements can be accessed by index.

```
Y = ["apple", "banana", "cherry", "mango"]
```

Index	0	1	2	3
Y =	"apple"	"banana"	"cherry"	"mango"

```
Y[0] → "apple"

Y[-4] → "apple"
```

List Slicing

Syntax

Exclusive

```
L[start: stop: step]
```

Inclusive

Default value: **1**

```
L = [0, 1, 2, 3, 4, 5, 6]
```

```
L[:3]   →   [0, 1, 2]
L[4:]   →   [4, 5, 6]
```

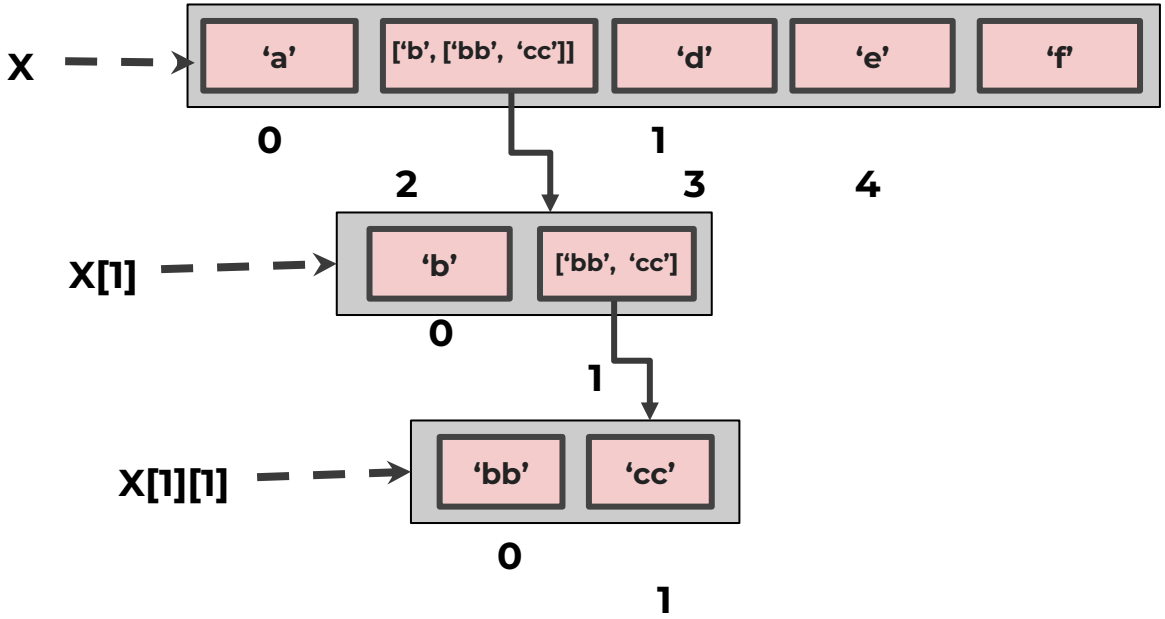
List Slicing

```
L = [0, 1, 2, 3, 4, 5, 6]
```

```
L[:]    → [0, 1, 2, 3, 4, 5, 6]
L[::2]  → [0, 2, 4, 6]
L[::-1] → [6, 5, 4, 3, 2, 1, 0]
```

Nested List: Lists can be nested to arbitrary depth.

```
X = ['a', ['b', ['bb', 'cc']], 'd', 'e', 'f']
```



Lists are mutable

```
Y = ["apple", "banana", "cherry", "mango"]
```

Index	0	1	2	3

```
Y → [10, "banana", "cherry", "mango"]
```

Index	0	1	2	3

Lists are dynamic

```
Y = ["apple", "banana", "cherry", "mango"]
```

Index	0	1	2	3

```
Y += ["peanut", "bean"]
```

addition assignment operator

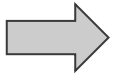
```
Y → ["apple", "banana", "cherry", "mango", "peanut", "bean"]
```

Index	0	1	2	3	4	5

Some List methods

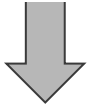
```
Y = [1, 2, 'a', 4, 5.5, 6]
```

```
Y.remove('a')
```



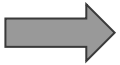
```
[1, 2, 4, 5.5, 6]
```

```
Y.extend( ["peanut", "bean"] )
```



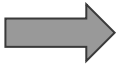
```
Y → [1, 2, 4, 5.5, 6, "peanut", "bean"]
```

```
Y.reverse()
```



```
["bean", "peanut", 6, 5.5, 4, 2, 1 ]
```

```
Y.pop(1)
```



```
["bean", 6, 5.5, 4, 2, 1 ]
```

List: Code Demo

Tuple

Tuple

```
X = ('mango', 1992, 3.14, True, [1,2], (1, 1,(3,2)))
```

Index	0	1	2	3	4	5

Negative index -4 -3 -2

Slicing **X[2:5]** →

3.14	True	[1,2]

X[-4:-1]

Index 2 3 4

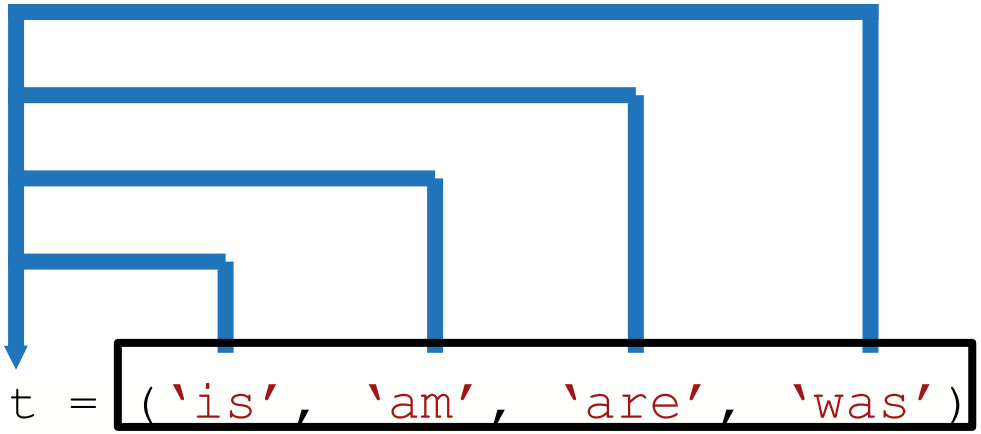
NOT allowed

X[1] = "apple"

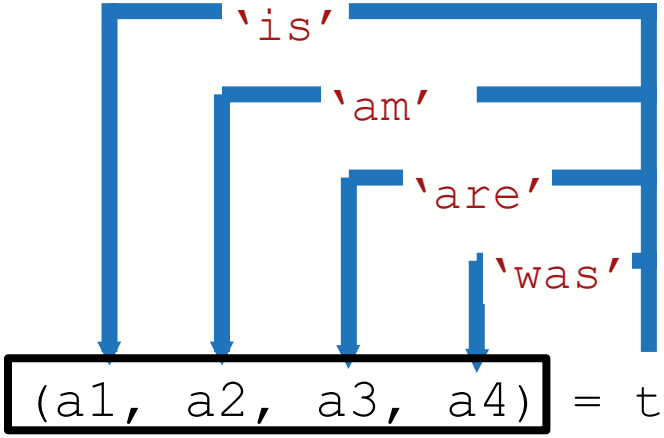
Single element in Tuple

```
X = (4, )

X = 4, #without round brackets

X = (4) #int type object

Y = () #empty tuple
```

Tuple Packing and Unpacking



Packing



Unpacking

Methods in Tuple

count()

Syntax

```
Tuple.count(value)
```

```
T = (1, 7, 8, 7, 5, 8, 5)
```

```
T.count(7)
```

2

index()

Syntax

```
Tuple.index(value)
```

```
T.index(5)
```

4

Tuple: Code Demo

Range

Range

1. **range**(stop)

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

1. **range**(start, stop)

```
>>> list(range(2, 8))
[ 2, 3, 4, 5, 6, 7]
```

1. **range**(start, stop, step)

```
>>> list(range(1, 11, 2)) #Increment
[ 1, 3, 5, 7, 9]
>>> list(range(1, -11, -2)) #Decrement
[ 1,-1, -3, -5, -7, -9]
```

Range: Code Demo