



Python Quick Tips

Working with Lists



Python Quick Tips #3

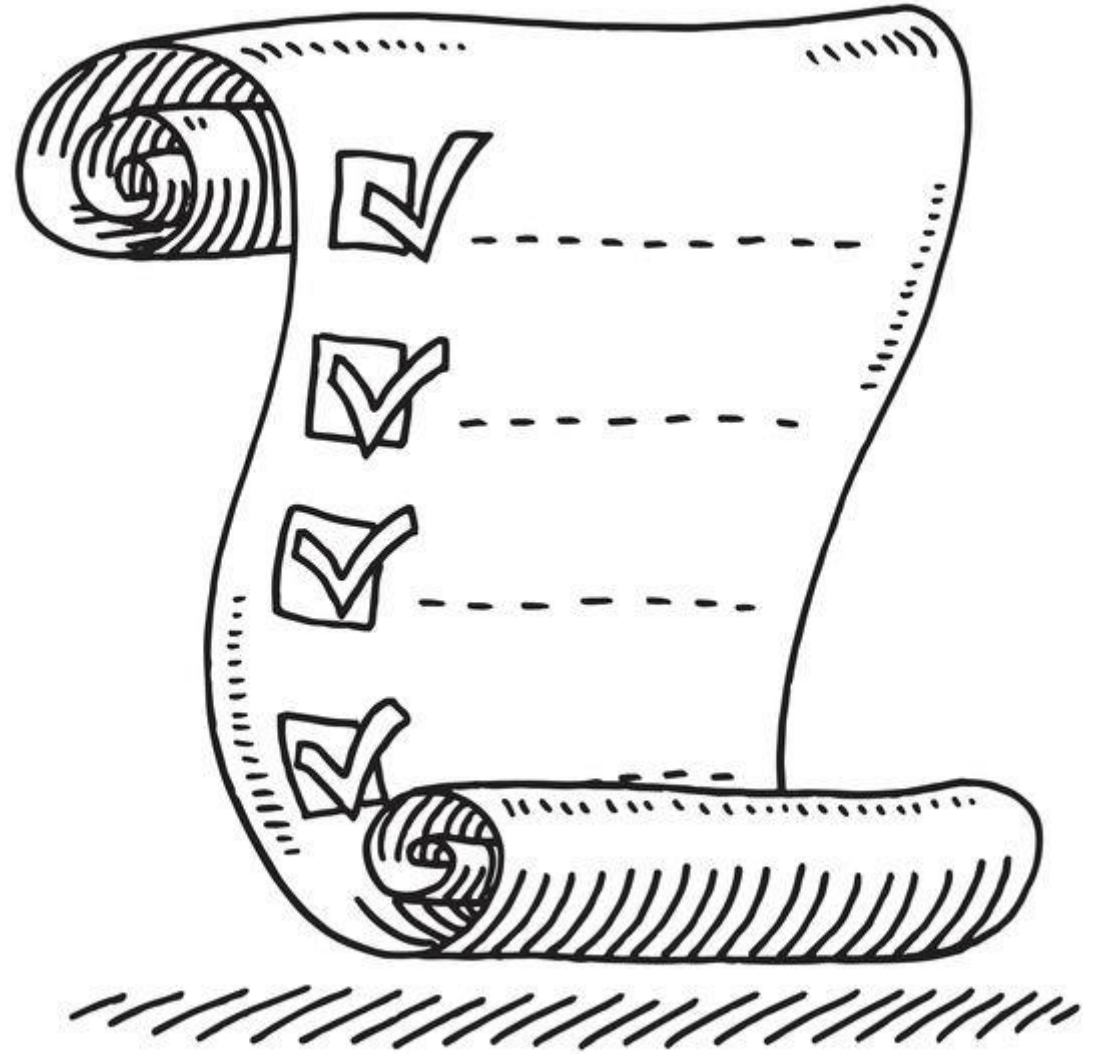
Working with Lists

Lists



In part 1, we saw that
lists are a data type

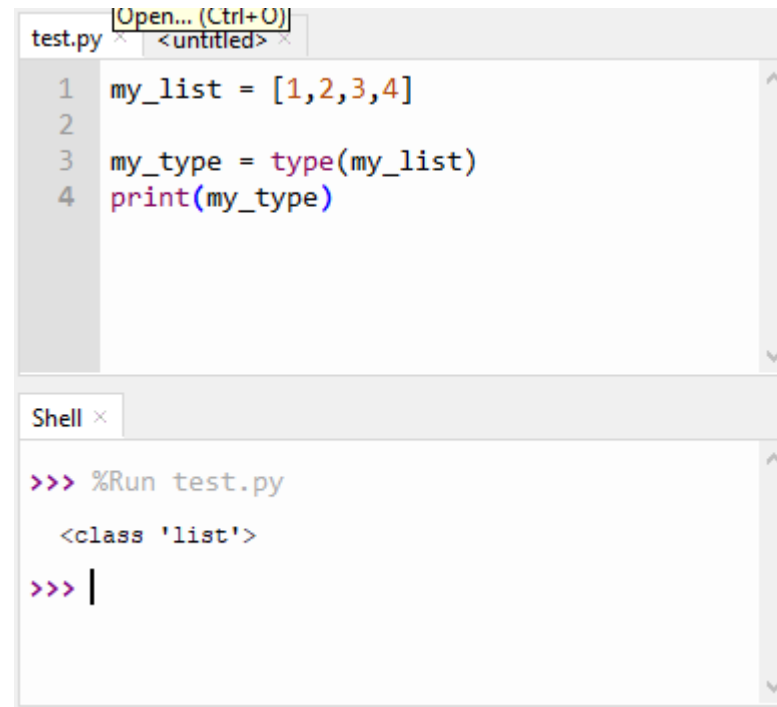
What are Lists



Building a List

Syntax

[object1, object2 ... etc.]



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

```
1 my_list = [1,2,3,4]
2
3 my_type = type(my_list)
4 print(my_type)
```

The bottom panel, titled 'Shell', shows the output of running the script:

```
>>> %Run test.py
      <class 'list'>
>>> |
```

Lists of lists

Nesting lists

```
[[1,2,3], [4,5,6]]
```

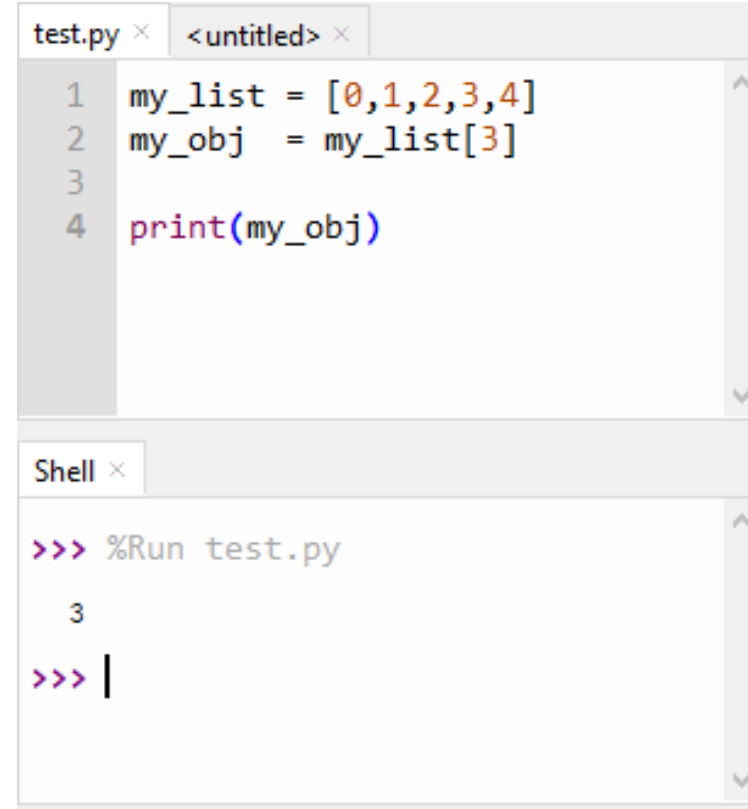
Later on we'll cover how you can work across lists using **Loops (Iteration)**

List Indices

Items begin at Index 0!
[index **0**, index **1**... etc.]

Item at Index

Syntax
list[Index]



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

```
1 my_list = [0,1,2,3,4]
2 my_obj  = my_list[3]
3
4 print(my_obj)
```

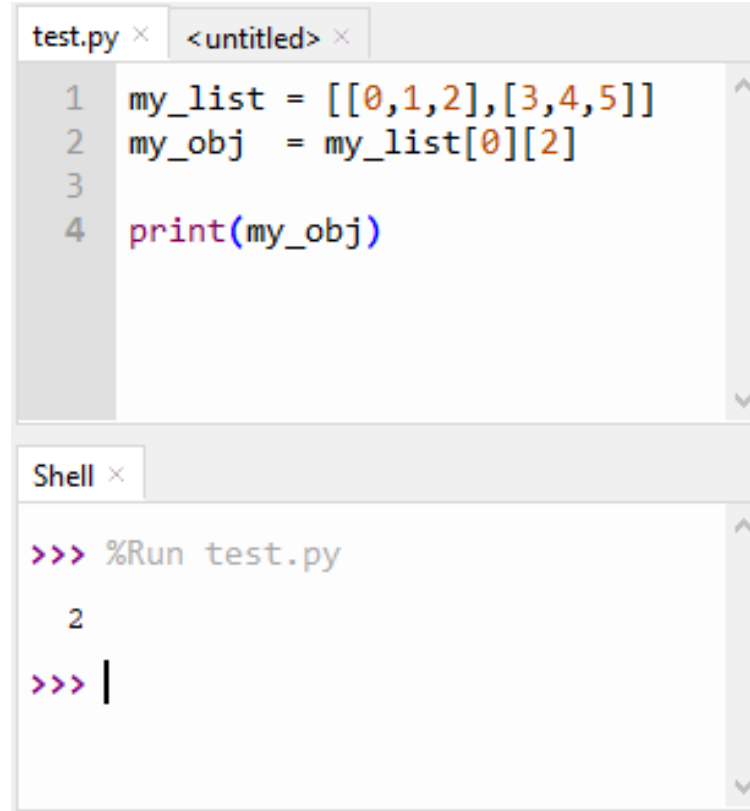
The bottom panel, titled 'Shell', shows the execution output:

```
>>> %Run test.py
3
>>> |
```


Item at Index (Sub-lists)

Syntax

list[Index][Sub-Index]



The screenshot shows a Python IDE with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following code:

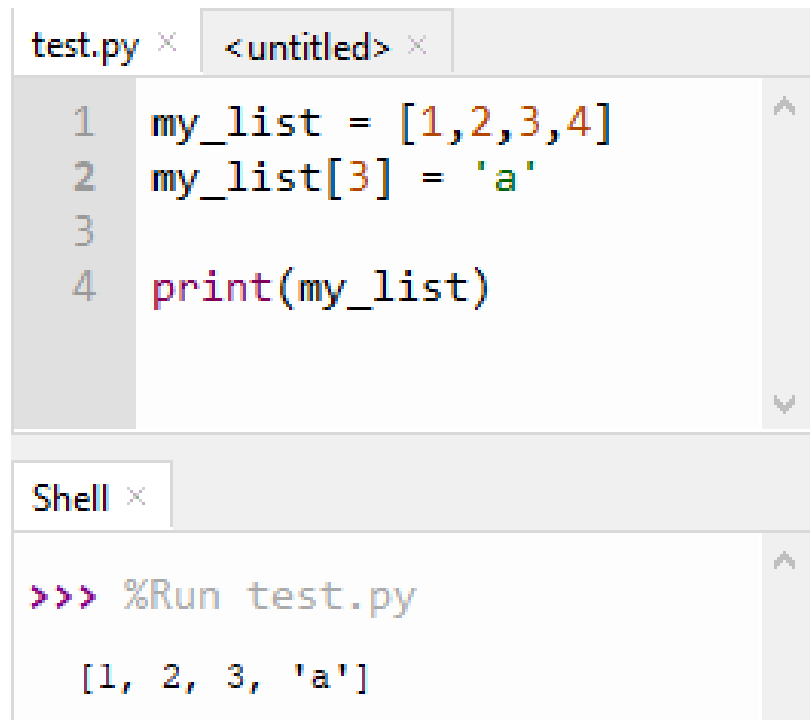
```
1 my_list = [[0,1,2],[3,4,5]]
2 my_obj  = my_list[0][2]
3
4 print(my_obj)
```

Below the code editor is a 'Shell' tab. It shows the command prompt output for running the script:

```
>>> %Run test.py
2
>>> |
```

Replace Item at Index

Syntax (recursive)
`list[Index] = NewValue`



The screenshot shows a code editor with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following Python code:

```
1 my_list = [1,2,3,4]
2 my_list[3] = 'a'
3
4 print(my_list)
```

Below the code editor is a 'Shell' tab. It shows the command prompt output for running the script:

```
>>> %Run test.py
[1, 2, 3, 'a']
```

Remove Item at Index

Function (recursive)
del(list[Index])



The screenshot shows a code editor with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following Python code:

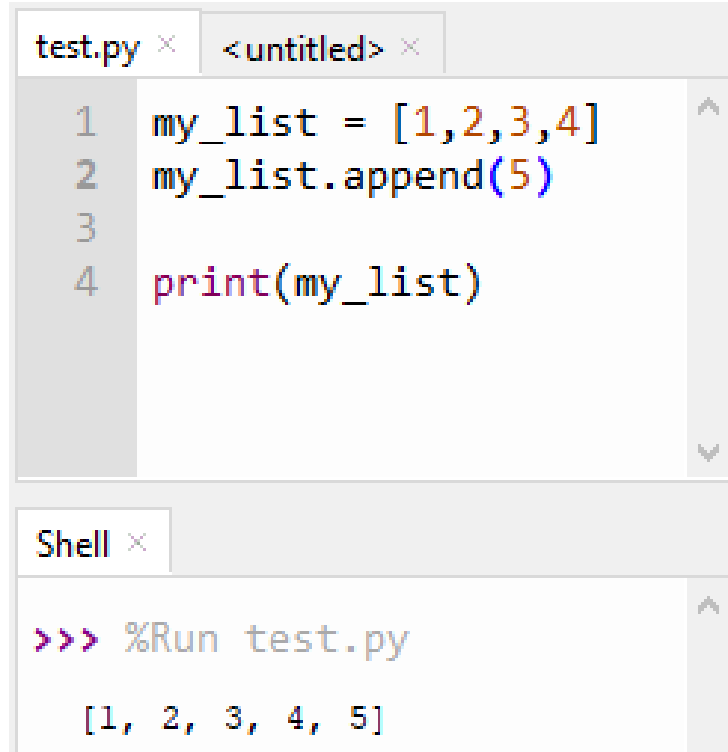
```
1 my_list = [1,2,3,4]
2 del(my_list[2])
3
4 print(my_list)
```

Below the code editor is a 'Shell' tab. It contains the command prompt output for running the script:

```
>>> %Run test.py
[1, 2, 4]
```

Append

Method (recursive)
list.append(object)



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

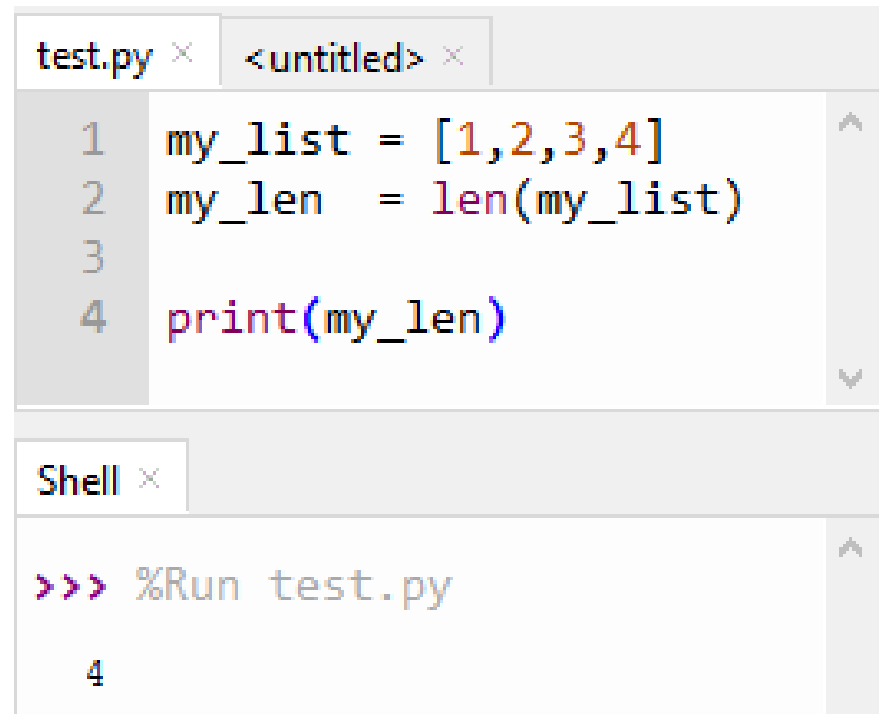
```
1 my_list = [1,2,3,4]
2 my_list.append(5)
3
4 print(my_list)
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
[1, 2, 3, 4, 5]
```

Length

Function `len(list)`



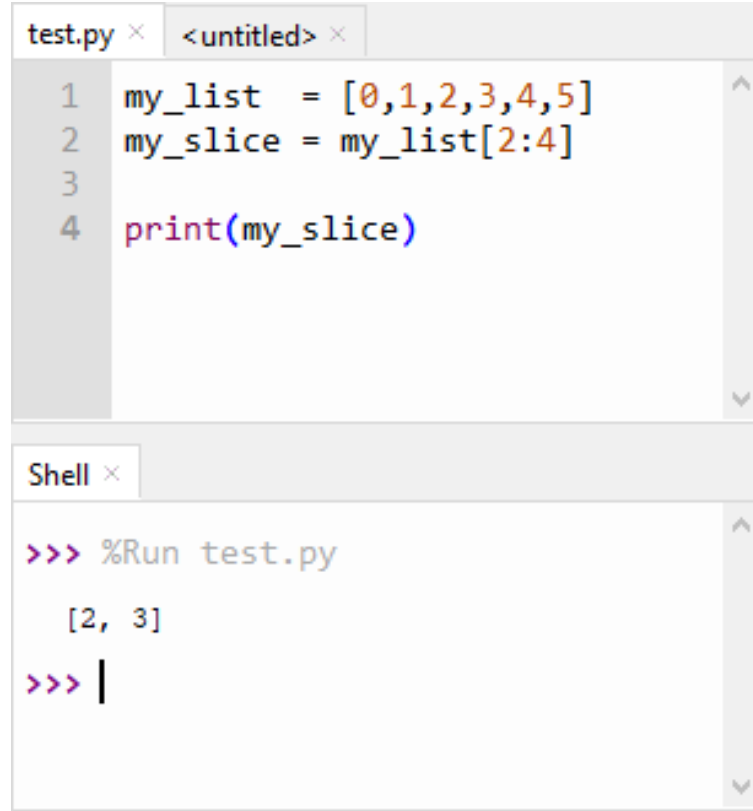
The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains a Python script with four lines of code. The bottom panel, titled 'Shell', shows the output of running the script.

```
test.py × <untitled> ×  
1 my_list = [1,2,3,4]  
2 my_len = len(my_list)  
3  
4 print(my_len)  
  
Shell ×  
>>> %Run test.py  
4
```

Slicing

Syntax

list[start:end(+1)]



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

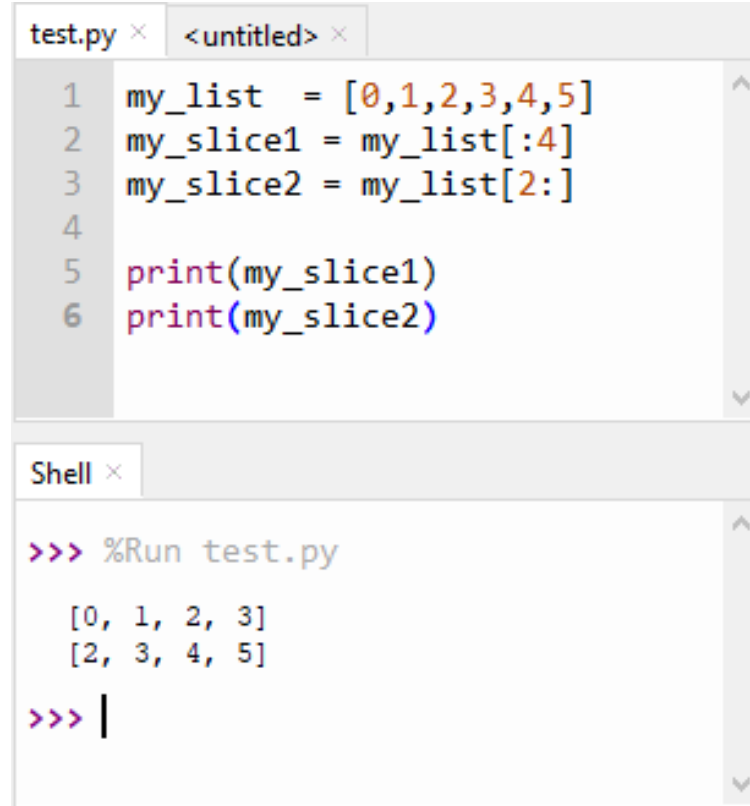
```
1 my_list = [0,1,2,3,4,5]
2 my_slice = my_list[2:4]
3
4 print(my_slice)
```

The bottom panel, titled 'Shell', shows the execution output:

```
>>> %Run test.py
[2, 3]
>>> |
```

Slicing

Syntax (to/from index)
`list[start:], list[:end(+1)]`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

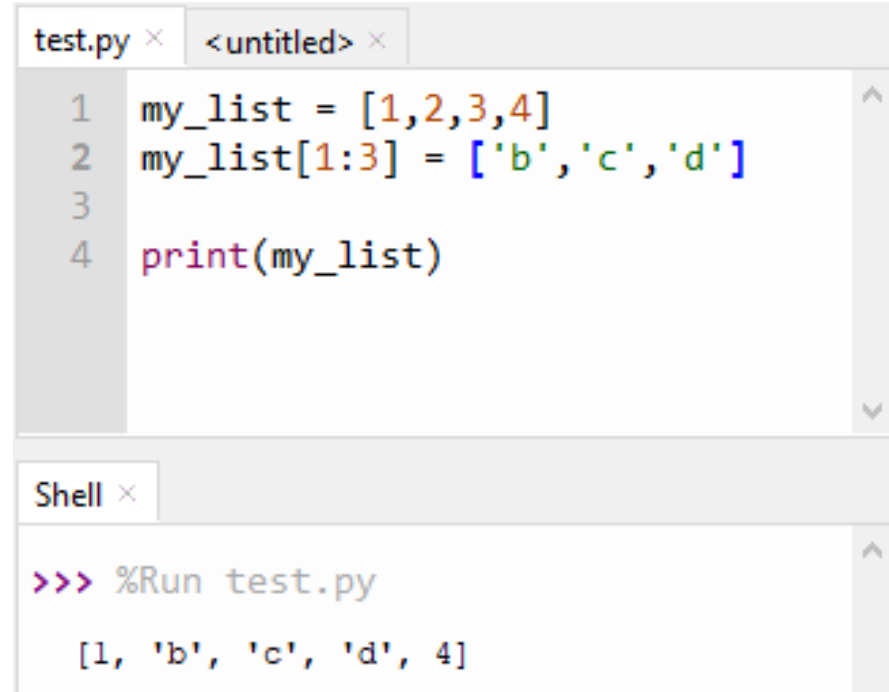
```
1 my_list = [0,1,2,3,4,5]
2 my_slice1 = my_list[:4]
3 my_slice2 = my_list[2:]
4
5 print(my_slice1)
6 print(my_slice2)
```

The bottom panel, titled 'Shell', shows the execution output:

```
>>> %Run test.py
[0, 1, 2, 3]
[2, 3, 4, 5]
>>> |
```

Replace Slice

Syntax (recursive)
`list[Slice] = [NewSlice]`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

```
1 my_list = [1,2,3,4]
2 my_list[1:3] = ['b','c','d']
3
4 print(my_list)
```

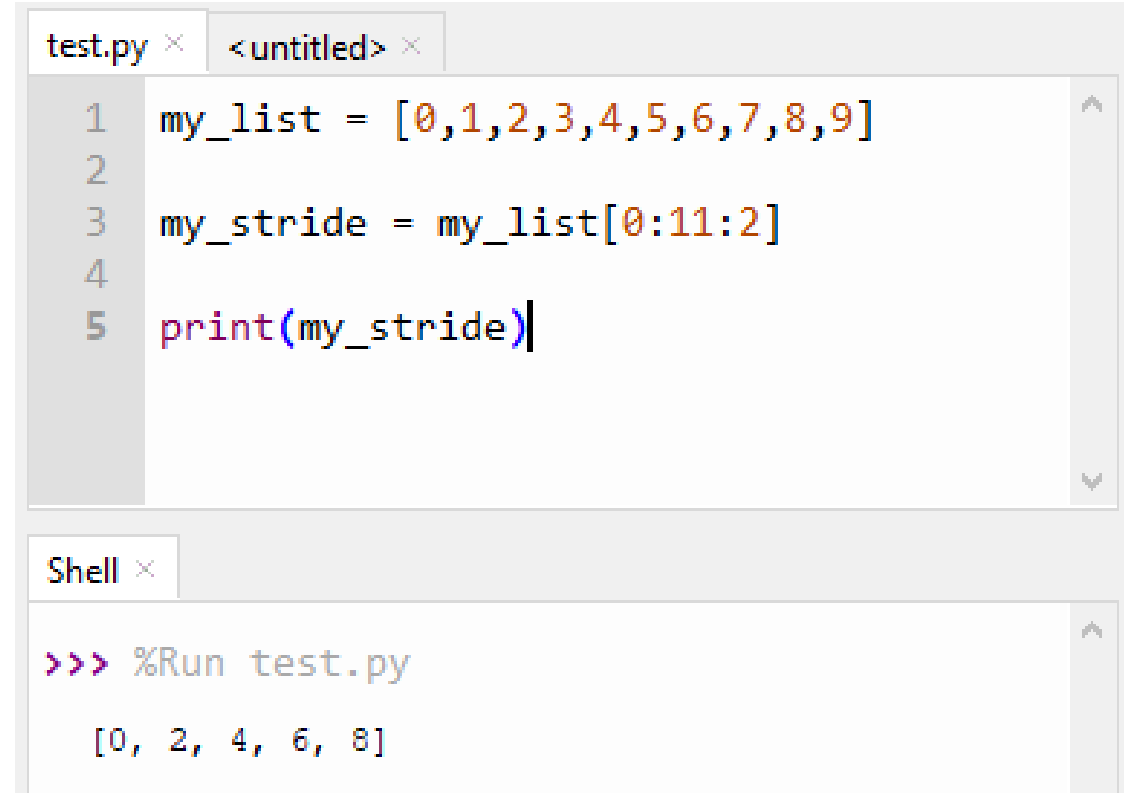
The bottom panel, titled 'Shell', shows the output of running the script:

```
>>> %Run test.py
[1, 'b', 'c', 'd', 4]
```


Every Nth item (Striding)

Syntax

list[start:end(+1):stride]



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

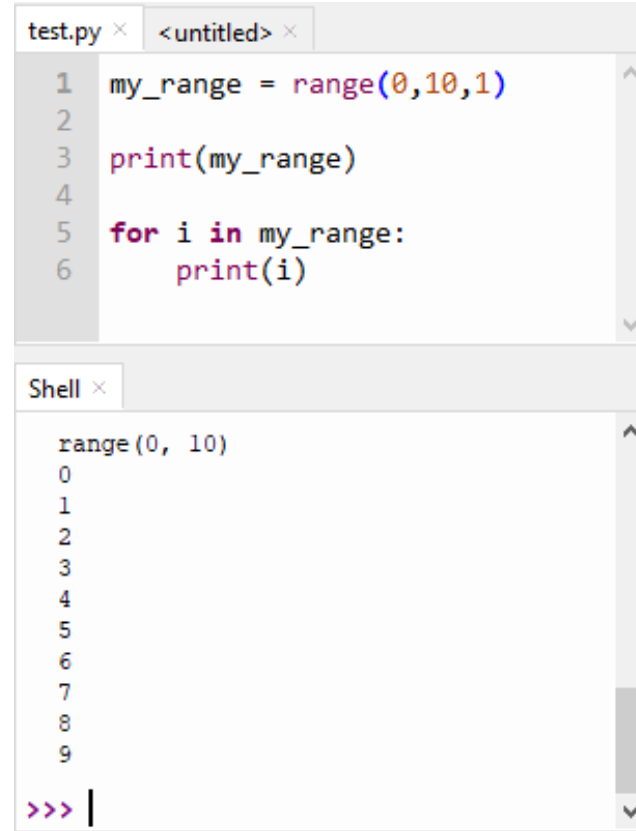
```
1 my_list = [0,1,2,3,4,5,6,7,8,9]
2
3 my_stride = my_list[0:11:2]
4
5 print(my_stride)
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
[0, 2, 4, 6, 8]
```

Ranges

Function
`range(start, end(+1), step)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

```
1 my_range = range(0,10,1)
2
3 print(my_range)
4
5 for i in my_range:
6     print(i)
```

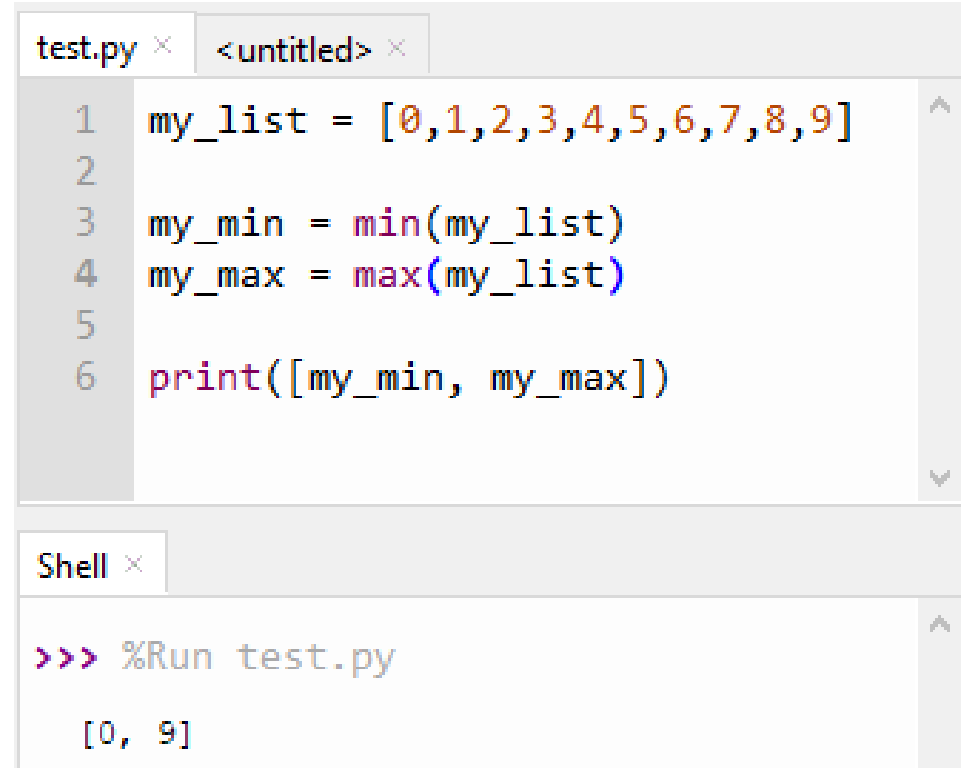
The bottom panel, titled 'Shell', shows the output of the script:

```
range(0, 10)
0
1
2
3
4
5
6
7
8
9
>>> |
```

Min/Max

Functions

`min(list)`, `max(list)`



The screenshot shows a Python IDE with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following code:

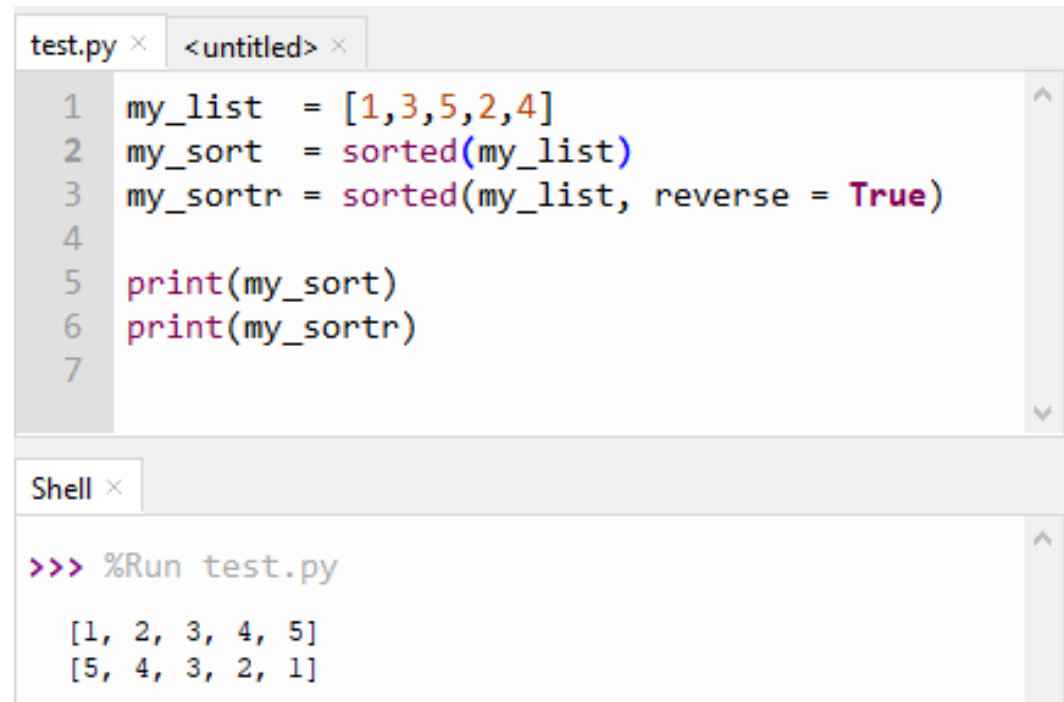
```
1 my_list = [0,1,2,3,4,5,6,7,8,9]
2
3 my_min = min(my_list)
4 my_max = max(my_list)
5
6 print([my_min, my_max])
```

Below the code editor is a 'Shell' tab. It shows the command prompt with the command '%Run test.py' and the output '[0, 9]'.

Sorted

Functions

`sorted(list, reverse = bool)`



The screenshot shows a code editor with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following Python code:

```
1 my_list = [1,3,5,2,4]
2 my_sort = sorted(my_list)
3 my_sortr = sorted(my_list, reverse = True)
4
5 print(my_sort)
6 print(my_sortr)
7
```

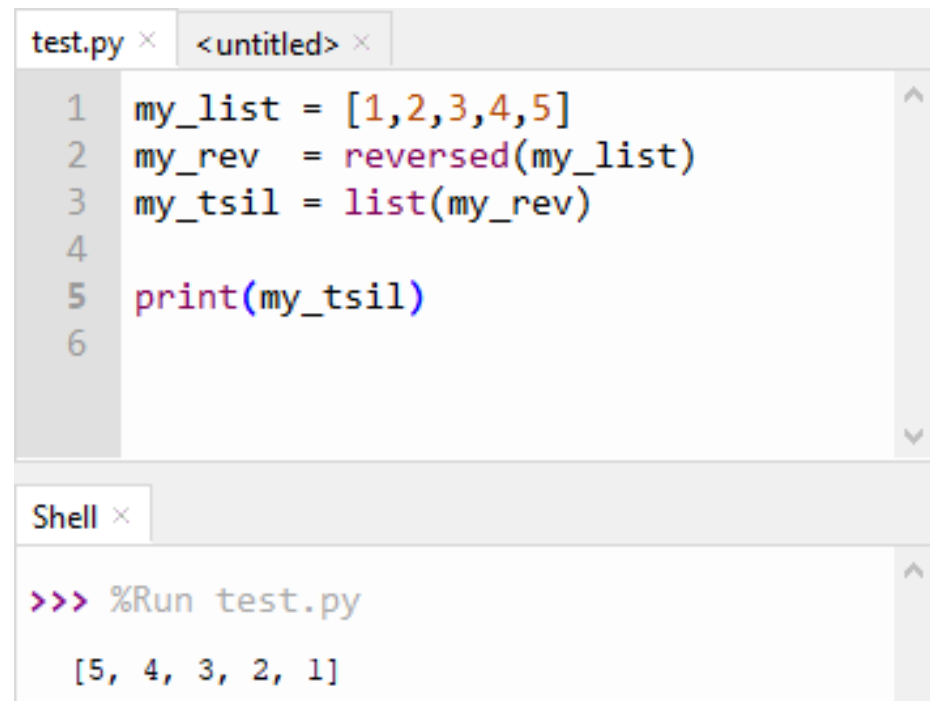
Below the code editor is a 'Shell' tab. It shows the command prompt output for running 'test.py':

```
>>> %Run test.py
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

Reversed

NB: Must be
converted to
list after

Function `reversed(list)`



```
test.py x <untitled> x
1 my_list = [1,2,3,4,5]
2 my_rev = reversed(my_list)
3 my_tsil = list(my_rev)
4
5 print(my_tsil)
6

Shell x
>>> %Run test.py
[5, 4, 3, 2, 1]
```

The screenshot shows a Python IDE with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following code:

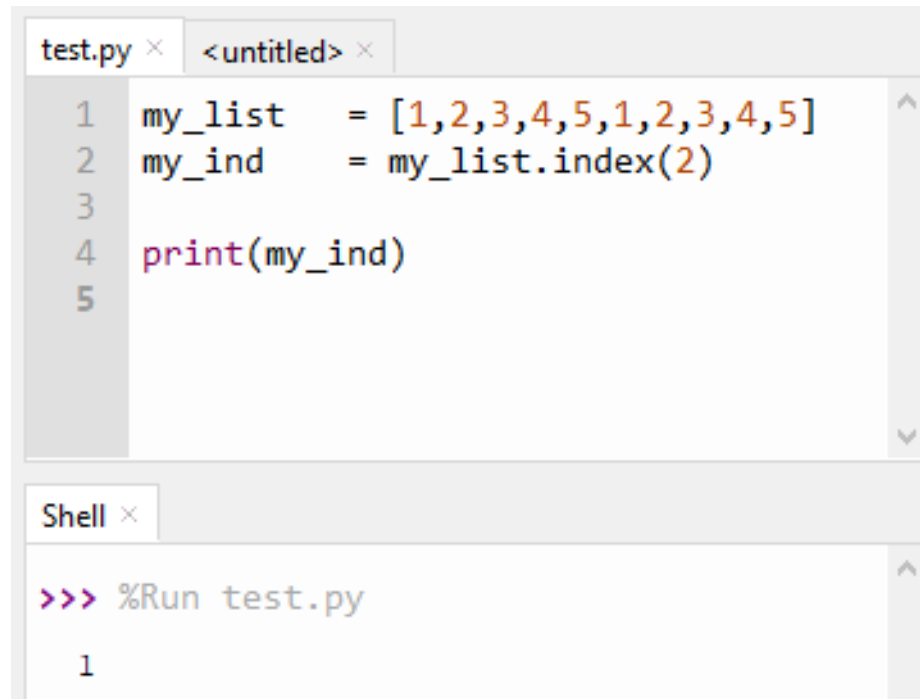
```
1 my_list = [1,2,3,4,5]
2 my_rev = reversed(my_list)
3 my_tsil = list(my_rev)
4
5 print(my_tsil)
6
```

Below the code editor is a 'Shell' tab. It shows the command prompt output for running the script:

```
>>> %Run test.py
[5, 4, 3, 2, 1]
```

(first)
Index of

Method
`list.index(object)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

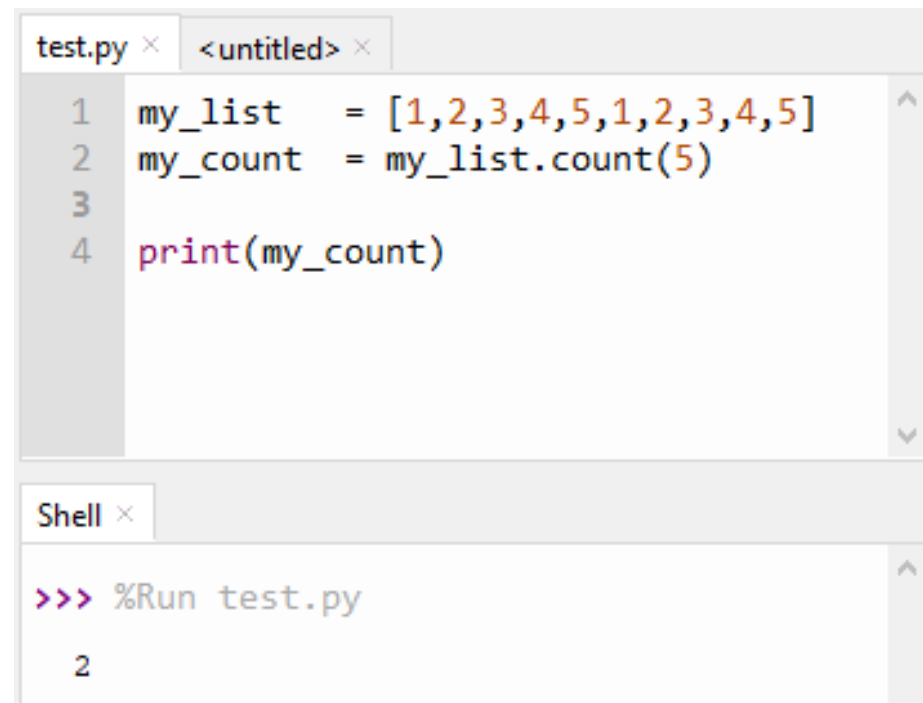
```
1 my_list = [1,2,3,4,5,1,2,3,4,5]
2 my_ind  = my_list.index(2)
3
4 print(my_ind)
5
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
1
```

Count occurences

Method
`list.count(object)`



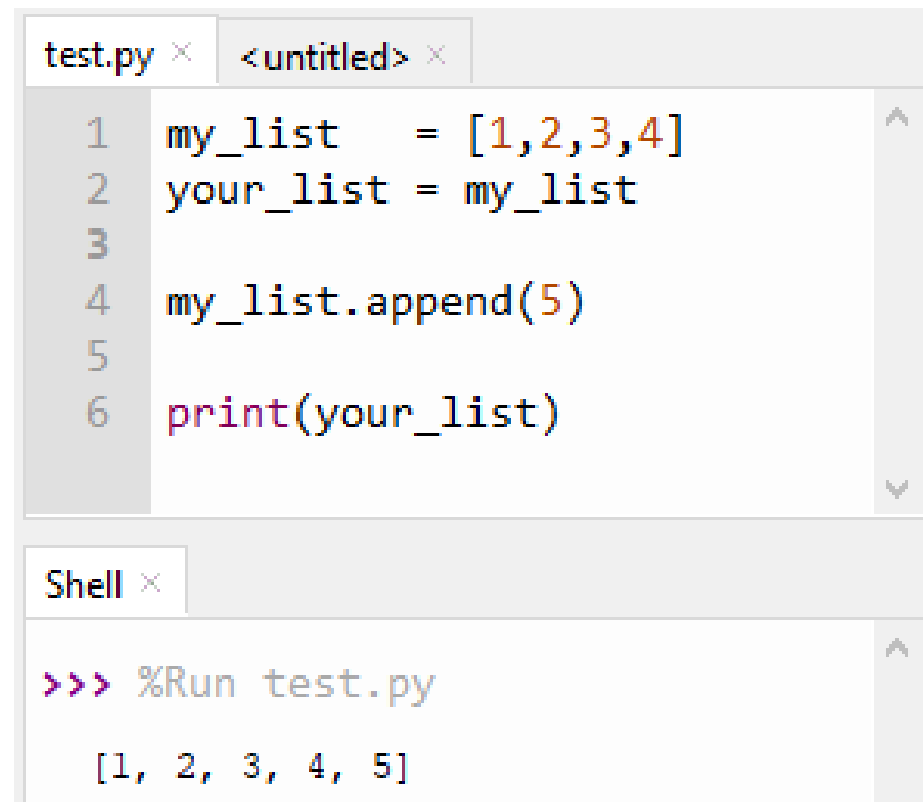
```
test.py x <untitled> x
1 my_list = [1,2,3,4,5,1,2,3,4,5]
2 my_count = my_list.count(5)
3
4 print(my_count)

Shell x
>>> %Run test.py
2
```

The screenshot shows a Python IDE with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains four lines of code: line 1 defines 'my_list' as a list of ten elements [1,2,3,4,5,1,2,3,4,5]; line 2 assigns 'my_count' the value of 'my_list.count(5)'; line 3 is empty; and line 4 prints 'my_count'. Below the code editor is a 'Shell' tab containing the command prompt output: '>>> %Run test.py' followed by the result '2'.

Recursive Lists

Be careful when setting
lists to new variables



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py' and '<untitled>', contains the following code:

```
1 my_list = [1,2,3,4]
2 your_list = my_list
3
4 my_list.append(5)
5
6 print(your_list)
```

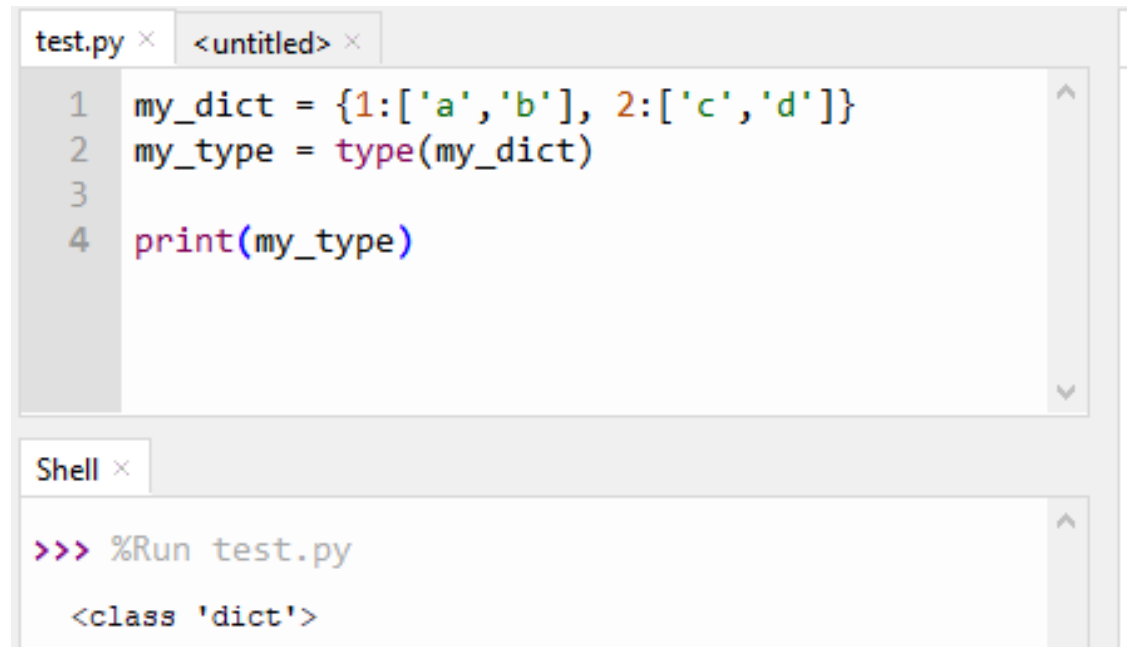
The bottom panel, titled 'Shell', shows the output of running the script:

```
>>> %Run test.py
[1, 2, 3, 4, 5]
```

This demonstrates that both variables point to the same list object in memory, so modifications to one are reflected in the other.

Dictionaries

A bit like lists, but they store values using a **key**. Represented by curly braces { }



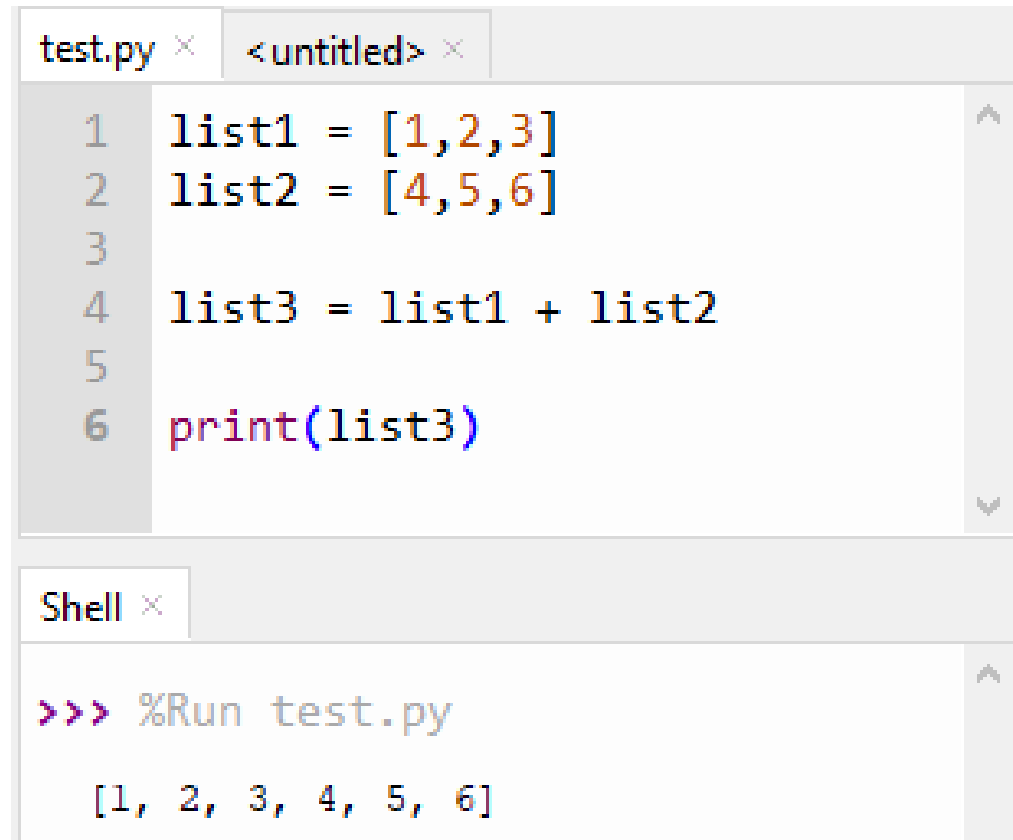
```
test.py x <untitled> x
1 my_dict = {1:['a','b'], 2:['c','d']}
2 my_type = type(my_dict)
3
4 print(my_type)

Shell x
>>> %Run test.py
<class 'dict'>
```

The screenshot shows a code editor with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains four lines of Python code. Line 1 creates a dictionary 'my_dict' with two key-value pairs: 1:['a','b'] and 2:['c','d']. Line 2 assigns the type of 'my_dict' to 'my_type'. Line 3 is empty. Line 4 prints 'my_type'. Below the code editor is a 'Shell' tab. The shell shows the command '%Run test.py' being executed, followed by the output '<class 'dict''>', which is the Python type for dictionaries.

Lists and Arithmetic

List probably don't behave as you expect when added!



The screenshot shows a Python IDE with two tabs: 'test.py' and '<untitled>'. The 'test.py' tab is active and contains the following code:

```
1 list1 = [1,2,3]
2 list2 = [4,5,6]
3
4 list3 = list1 + list2
5
6 print(list3)
```

Below the code editor is a 'Shell' tab. It shows the command prompt output after running the script:

```
>>> %Run test.py
[1, 2, 3, 4, 5, 6]
```

Numpy Arrays

List can be converted to
Arrays for parallel functions

```
test.py x <untitled> x
1 import numpy as np
2
3 list1 = [1,2,3,4]
4 list2 = [5,6,7,8]
5
6 print(list1*list2)
```

```
Shell x
Traceback (most recent call last):
  File "C:\Users\Gavin\Desktop\test.py", line 6, in <module>
    print(list1*list2)
TypeError: can't multiply sequence by non-int of type 'list'
>>> |
```

```
test.py x <untitled> x
1 import numpy as np
2
3 list1 = [1,2,3,4]
4 list2 = [5,6,7,8]
5
6 array1 = np.array(list1)
7 array2 = np.array(list2)
8
9 print(array1*array2)
```

```
Shell x
>>> %Run test.py
[ 5 12 21 32]
>>> |
```



Next on #4
Working with Strings



Python Quick Tips

Working with Lists