Python Quick Tips

Working with Lists

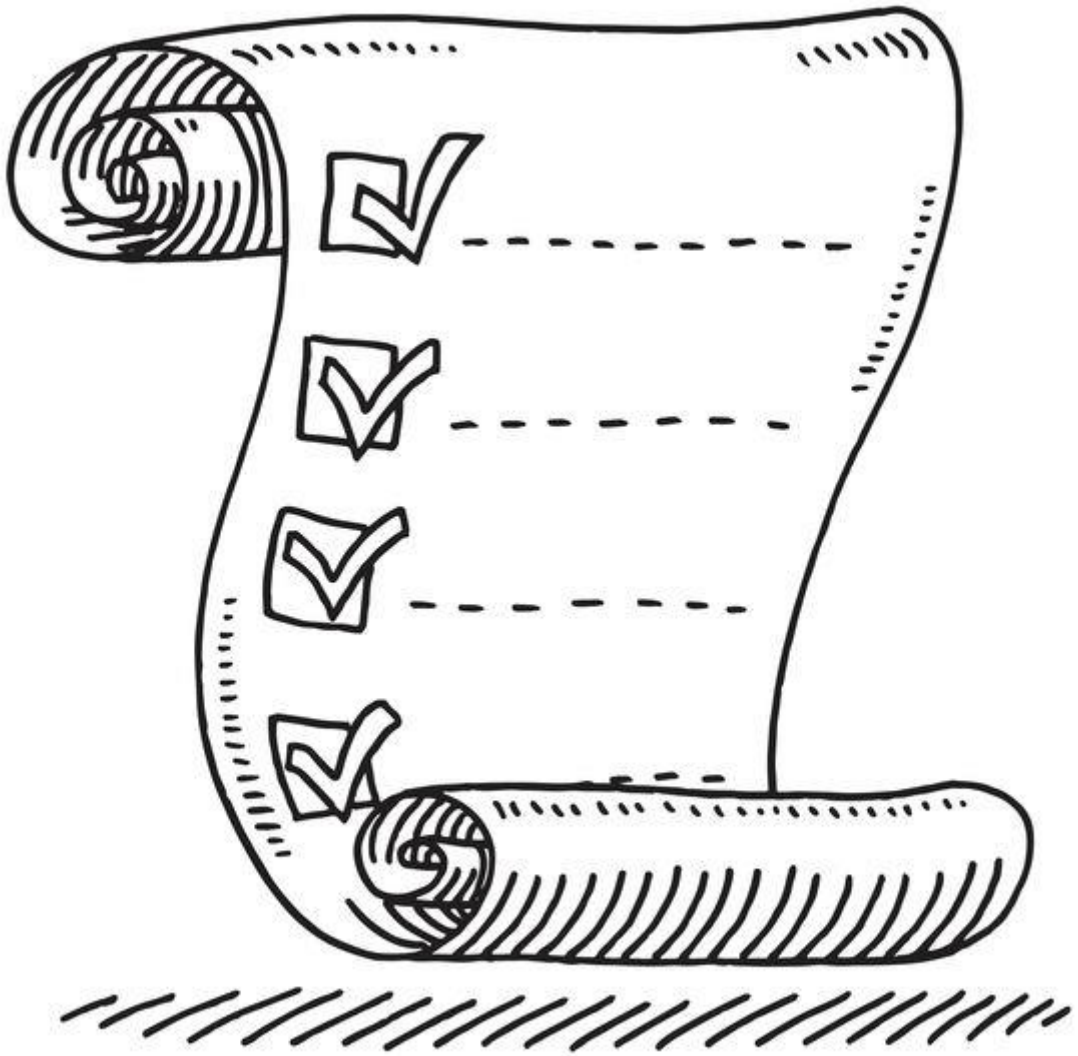
Python Quick Tips #3

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.]

test.py | Open... (Ctrl+O) | <untitled>

```
my_list = [1,2,3,4]

my_type = type(my_list)
print(my_type)
```

Shell

```
>>> %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]

test.py | <untitled>

```python
my_list = [0,1,2,3,4]
my_obj  = my_list[3]

print(my_obj)
```

Shell

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

Item at Index (Sub-lists)

Syntax

list[Index][Sub-Index]

test.py | <untitled>

```python
my_list = [[0,1,2],[3,4,5]]
my_obj  = my_list[0][2]

print(my_obj)
```

Shell

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

Replace Item at Index

Syntax (recursive)

list[Index] = NewValue

test.py <untitled>

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

print(my_list)
```

Shell

```
>>> %Run test.py

  [1, 2, 3, 'a']
```

Remove Item at Index

Function (recursive)

del(list[Index]**)**

test.py | <untitled>

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

print(my_list)
```

Shell

```
>>> %Run test.py

  [1, 2, 4]
```

Append

Method (recursive)

list.**append(**object**)**

test.py | <untitled>

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

print(my_list)
```

Shell

```
>>> %Run test.py

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

Length

Function

len(list)

test.py | <untitled>

```python
my_list = [1,2,3,4]
my_len  = len(my_list)

print(my_len)
```

Shell

```
>>> %Run test.py

  4
```

Slicing

Syntax

list[start:end(+1)]

test.py | <untitled>

```python
my_list  = [0,1,2,3,4,5]
my_slice = my_list[2:4]

print(my_slice)
```

Shell

```
>>> %Run test.py

  [2, 3]

>>> |
```

Slicing

Syntax (to/from index)

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

test.py | <untitled>

```python
my_list  = [0,1,2,3,4,5]
my_slice1 = my_list[:4]
my_slice2 = my_list[2:]

print(my_slice1)
print(my_slice2)
```

Shell

```
>>> %Run test.py

  [0, 1, 2, 3]
  [2, 3, 4, 5]

>>> |
```

Replace Slice

Syntax (recursive)

list[Slice] = [NewSlice]

test.py | <untitled>

```python
my_list = [1,2,3,4]
my_list[1:3] = ['b','c','d']

print(my_list)
```

Shell

```
>>> %Run test.py

  [1, 'b', 'c', 'd', 4]
```

Every Nth item (Striding)

Syntax

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

test.py | <untitled>

```python
my_list = [0,1,2,3,4,5,6,7,8,9]

my_stride = my_list[0:11:2]

print(my_stride)
```

Shell

```
>>> %Run test.py

  [0, 2, 4, 6, 8]
```

Ranges

Function

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

test.py | <untitled>

```python
my_range = range(0,10,1)

print(my_range)

for i in my_range:
    print(i)
```

Shell

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

Min/Max

Functions

min(list), **max**(list)

test.py | <untitled>

```python
my_list = [0,1,2,3,4,5,6,7,8,9]

my_min = min(my_list)
my_max = max(my_list)

print([my_min, my_max])
```

Shell

```
>>> %Run test.py

  [0, 9]
```

Sorted

Functions

sorted(list, reverse = bool)

test.py | <untitled>

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

print(my_sort)
print(my_sortr)
```

Shell

```
>>> %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 | <untitled>

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

print(my_tsil)
```

Shell

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

(first)

Index of

Method

list.**index(**object**)**

test.py | <untitled>

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

print(my_ind)
```

Shell

```
>>> %Run test.py

  1
```

Count occurences

Method

list.**count(**object**)**

test.py | <untitled>

```python
my_list   = [1,2,3,4,5,1,2,3,4,5]
my_count  = my_list.count(5)

print(my_count)
```

Shell

```
>>> %Run test.py

  2
```

Recursive Lists

Be careful when setting lists to new variables

test.py | <untitled>

```python
my_list   = [1,2,3,4]
your_list = my_list

my_list.append(5)

print(your_list)
```

Shell

```
>>> %Run test.py

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

Dictionaries

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

test.py <untitled>

```
my_dict = {1:['a','b'], 2:['c','d']}
my_type = type(my_dict)

print(my_type)
```

Shell

```
>>> %Run test.py

  <class 'dict'>
```

Lists and Arithmetic

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

test.py | <untitled>

```python
list1 = [1,2,3]
list2 = [4,5,6]

list3 = list1 + list2

print(list3)
```

Shell

```
>>> %Run test.py

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

Numpy Arrays

List can be converted to **Arrays** for parallel functions

test.py | <untitled>

```python
import numpy as np

list1 = [1,2,3,4]
list2 = [5,6,7,8]

print(list1*list2)
```

Shell

```
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 | <untitled>

```python
import numpy as np

list1 = [1,2,3,4]
list2 = [5,6,7,8]

array1 = np.array(list1)
array2 = np.array(list2)

print(array1*array2)
```

Shell

```
>>> %Run test.py

  [ 5 12 21 32]

>>>
```

Next on #4

Working with Strings

Python Quick Tips

Working with Lists