Built-in Data Structures, Functions,

Functions

Functions are the primary and most important method of code organization and reuse in Python.

In [1]:
```
def my_function(x, y, z=1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

In [2]:
```
my_function(5, 6, z=0.7)
my_function(3.14, 7, 3.5)
my_function(10, 20)
```

Out[2]:
```
45.0
```

Namespaces, Scope, and Local Functions

Functions can access variables in two different scopes: global and local. An alternative and more descriptive name describing a variable scope in Python is a namespace.

In [46]:
```
def func():
    aa = []
    for i in range(5):
        aa.append(i)
print(aa)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-b94bbfc2819f> in <module>
      3     for i in range(5):
      4         aa.append(i)
----> 5 print(aa)

NameError: name 'aa' is not defined
```

In [4]:
```
a = []
def func():
    for i in range(5):
        a.append(i)
```

Returning Multiple Values

In [5]:
```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c

a, b, c = f()
```

In [6]:
```
return_value = f()
return_value
```

Out[6]:
```
(5, 6, 7)
```

In [7]:
```
def f():
    a = 5
    b = 6
    c = 7
    return {'a' : a, 'b' : b, 'c' : c}
```

In [8]:
```
return_value = f()
return_value
```

Out[8]:
```
{'a': 5, 'b': 6, 'c': 7}
```

Functions Are Objects

Since Python functions are objects, many constructs can be easily expressed that are difficult to do in other languages.

In [9]:
```
states = ['   Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
          'south   carolina##', 'West virginia?']
```

To make this list of strings uniform and ready for analysis we can do the followings: stripping whitespace, removing punctuation symbols, and stand- ardizing on proper capitalization.

In [10]:
```
import re

def clean_strings(strings):
    result = []
    for value in strings:
#         print(value)
        value = value.strip()
        value = re.sub('[!#?]', '', value)
        value = value.title()
        result.append(value)
    return result
```

In [11]:
```
clean_strings(states)
```

Out[11]:
```
['Alabama',
 'Georgia',
 'Georgia',
 'Georgia',
 'Florida',
 'South   Carolina',
 'West Virginia']
```

In [12]:
```
def remove_punctuation(value):
    return re.sub('[!#?]', '', value)

for x in map(remove_punctuation, states):
    print(x)
```

```
   Alabama
Georgia
Georgia
georgia
FlOrIda
south   carolina
West virginia
```

Anonymous (Lambda) Functions

Python has support for so-called anonymous or lambda functions, which are a way of writing functions consisting of a single statement, the result of which is the return value. They are defined with the lambda keyword, which has no meaning other than “we are declaring an anonymous function”:

In [13]:
```
def short_function(x):
    return x * 2

equiv_fn = lambda x: x * 2
```

In [14]:
```
equiv_fn(8)
```

Out[14]:
```
16
```

In [15]:
```
short_function(8)
```

Out[15]:
```
16
```

In [16]:
```
def apply_to_list(some_list, f):
    return [f(x) for x in some_list]

ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)
```

Out[16]:
```
[8, 0, 2, 10, 12]
```

Sort a collection of strings by the number of distinct letters in each string:

In [17]:
```
strings = ['foo', 'card', 'bar', 'aaaa', 'abab']
```

In [18]:
```
strings.sort(key=lambda x: len(set(list(x))))
strings
```

Out[18]:
```
['aaaa', 'foo', 'abab', 'bar', 'card']
```

In [19]:
```
fn2=lambda x: len(set(list(x)))
```

In [20]:
```
fn2('bbarr')
```

Out[20]:
```
3
```

In [21]:
```
len(set(list('bbarr'))) # Comment
```

Out[21]:
```
3
```

Hello

Generators

Having a consistent way to iterate over sequences, like objects in a list or lines in a file, is an important Python feature. This is accomplished by means of the iterator protocol, a generic way to make objects iterable. For example, iterating over a dict yields the dict keys:

In [22]:
```
some_dict = {'a': 1, 'b': 2, 'c': 3}
for key in some_dict:
    print(key)
```

```
a
b
c
```

A generator is a concise way to construct a new iterable object. Whereas normal functions execute and return a single result at a time, generators return a sequence of multiple results lazily, pausing after each one until the next one is requested. To create a generator, use the yield keyword instead of return in a function:

In [23]:
```
def squares(n=10):
    print('Generating squares from 1 to {0}'.format(n ** 2))
    for i in range(1, n + 1):
        yield i ** 2
```

In [24]:
```
gen = squares()
gen
```

Out[24]:
```
<generator object squares at 0x000002604F682F90>
```

In [25]:
```
for x in gen:
    print(x, end=' ')
```

```
Generating squares from 1 to 100
1 4 9 16 25 36 49 64 81 100
```

Errors and Exception Handling

In [45]:
```
float('1.2345')
float('something')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-6d335c618d25> in <module>
      1 float('1.2345')
----> 2 float('something')

ValueError: could not convert string to float: 'something'
```

In [27]:
```
def attempt_float(x):
    try:
        return float(x)
    except:
        return x
```

In [37]:
```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

```
Please enter a number: UNO
Oops!  That was no valid number.  Try again...
Please enter a number: 4.3
Oops!  That was no valid number.  Try again...
Please enter a number: 65
```

Files and the Operating System

To open a file for reading or writing, use the built-in open function with either a rela- tive or absolute file path:

In [38]:
```
path = 'input.txt'
f = open(path)
```

In [39]:
```
for line in f:
    print(line)
```

```
USA

Canada
```

In [40]:
```
lines = [x.rstrip() for x in open(path)]
lines
```

Out[40]:
```
['USA', 'Canada']
```

In [41]:
```
f.close()
```

In [42]:
```
with open(path) as f:
    lines = [x.rstrip() for x in f]
print(lines)
```

```
['USA', 'Canada']
```

In [43]:
```
path="Output.txt"
file2 = open(path,"w")

file2.write("USA\n")
file2.write("UNO\n")
file2.close()
```

In [44]:
```
with open(path) as f:
    lines = [x.rstrip() for x in f]
print(lines)
```

```
['USA', 'UNO']
```

In [36]:
```
import os
os.remove('Output.txt')
```

In [ ]:
```

```

In [ ]: