EML4930/EML6934: Lecture 02

Loops, Functions, Classes, and intro to Objects

Charles Jekel

September 7, 2017

Style guide for Python

`https://www.python.org/dev/peps/pep-0008/`

One thing I didn't know is that Python 3 disallows mixing tabs and spaces in your code.

You should use spaces to denote tabs in Python.

Code blocks in C and MATLAB

In C, code blocks are denoted with { and }. In MATLAB a code block ends with an *end* statement. However Python requires neither, just that the code block is indented!

```c
// C code
int total = 0;
for(int i=0; i<100; i++)
        {
                // curly braces indicate code block
                total += i;
        }
```

```
% MATLAB code
total = 0
for i = 1:99
% code block starts on the line after the for loop
total = total + i;
end % statment terminates the for loop
```

Python code for previous slide

```
total = 0
for i in range(100):
    # the first code line indented begins code block
    total = total + i
# when the indentation ends, so does the code block
```

- Statements clearly begin with a : in Python
- Python forces you to indent your code blocks – this makes Python highly readable
- You don't need end statements in Python, instead you end the indentation block

While loop

```
i = 0
while i < 10:
    print(i)
    i += 1 # this takes the previous i value and adds 1 to it
print('The while loop has finished')
```

Notice how the format is just like a if statment. Using the : to begin the code block. The code block is indicated by the indentation. There are no end statements in Python. Rather code blocks end when the indentation is returned to normal.

Iterating over a list

```
x = [0,1,2,3,4,5,6]
for i in x:
    print(i)
```

Iterating over two lists

This code simultaneously iterates over both lists

```
x = [0,1,2,3,4,5,6]
y = [7,8,9,10,11,12,13]
for xVal, yVal in zip(x,y):
    print(xVal, yVal)
```

Iterating over two lists - using range

This code simultaneously iterates over both lists. len() tells us the length of a list.

```
x = [0,1,2,3,4,5,6]
y = [7,8,9,10,11,12,13]
for i in range(len(x)):
    print(x[i], y[i])
```

For loop - Iterating with range()

```
total = 0
for i in range(100):
    # the first code line indented begins code block
    total = total + i
```

How range() works.

- for i in range(100): - iterates from 0 to 99 (100 times)
- range(0,100) - iterates from 0 to 99 (100 times)
- range(0,100,2) - iterates 0, 2, 4, ..., 98 (50 times)
- range(startPoint, endPoint, stepSize) where startPoint is inclusive, endPoint is exclusive

For loop range(0,9)

```
for i in range(0,9):
    '''
    this is a for loop,
    that will print 0, 1, 2, 3, 4, 5, 6, 7, 8
    '''
    print(i)
```

These are the most basic for loop. Again the : initiates the code block, and the indentation indicates the beginning and end of a code block.

Nesting a for loop

Remember your code block is denoted with indentation in Python.

```
for i in range(-10,11):
    if i < 0:
        print(i, 'is negative')
    elif i > 0:
        print(i, 'is positive')
    else:
        print(i, 'is zero')
```

This will loop for i = -10, -9, ..., 10. Then an if-else statement is used to print whether i is negative, positive,or zero.

Use break to break loop

```
for i in range(-10,11):
    if i == 0:
        break
    print(i)

i = 0
n = 0
while i == 0:
    n += 1
    if n > 100:
        break
```

Double nest loops

You can double nest loops like this.

```
for i in range(0,10):
    for j in range(0,10):
        print(i,j)
```

flat is better than nested

- Use functions with specific tasks to avoid a large number of nested layers
- It is really hard to debug heavily layered code- but easy to test functions
- Your break placement in nested code matters. Which loop are you trying to break???

Loops with the index and value - enumerate

Often I need to create a loop through an array, where I'll need the index and value. I use enumerate() to do this.

```
x = [8,181,129,10,'hi']
for index, value in enumerate(x):
    # index refers to the index on the list x
    # value is the current value of x the loop is on
    print(index, value)
```

Loops with the index and value - enumerate

If it isn't clear, index and value can be named whatever you want.

```
x = [8,181,129,10,'hi']
for i,j in enumerate(x):
    print(i,j)
```

List comprehension - basics

This is one really neat feature in Python where you can run loops and if statements while generating a list.

List comprehension creates many Pythonic code one liners.

An interested read `http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/`

```
x = [0,1,2,3,4,5]
y = [i**2 for i in x]
# y is a list where each item in x is squared
```

List comprehension - more advanced

```
x = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
y = [i for i in x if abs(i) < 3]
# y is a list containing each item in x
# such that the absolute value of each item in x is less than 3

x = [0,1,2,3,4,5]
y = [i**2 for i in x if i%2 ==0]
# y is a list where each item in x is squared
# if the item in x is an even number

x = [0,1,2,3,4,5]
y = [i**2  for i in x if i**2%2 == 0]
# y is a list where each item in x is squared
# if the item squared is an even number
```

list comprehension - multiple loops

```
in : [[i,j] for i in range(2) for j in range(2)]
out: [[0, 0], [0, 1], [1, 0], [1, 1]]

in : # this is a triple nested for loop creating a list!
[[i,j,k] for i in range(2) for j in range(2) for k in range(2)]
out: [[0, 0, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 1, 1],
 [1, 0, 0],
 [1, 0, 1],
 [1, 1, 0],
 [1, 1, 1]]
```

You know list comprehension

List comprehension - Eye of Thundera, give me your power

Functions - some basics

Functions are simple in Python.

Let's create a simple hello world function.

```
def helloWorld():
    print('Hello world function :-)')
```

Let's create a function to take the cube root of a number.

```
def cubeRoot(x):
    return x**(1.0/3.0)
```

- The def statement is used to define a function.
- The name of the function here is cubeRoot.
- Functions used () to pass input.
- Input for cubeRoot is x.
- Input is optional, see helloWorld().
- Again the : is used at the end of the statement.
- Indentation denotes the code block of the function.

Functions - optional arguments and input - output

```
def myRoot(x,c=3.0):
    '''
    myRoot(x) returns the cube root of x by default

    myRoot(x,c) returns the c root of x
    '''
    return x**(1.0/c)
```

- To view the docstring run help(myRoot) or print(myRoot.__doc__)
- c is an optional argument, that we set to 3 on default
- return is used to pass output
- x is the input
- in general the input can be anything, objects, lists, strings, floats, etc.
- the output can also be anything, objects, lists, strings, floats, etc

Functions - multiple returns

```
def rootz(x):
    '''
    a,b,c,d = rootz(x) for some float or integer x
    a = square root of x; b = cube root of x
    c = x**0.25;          d = x**0.2
    '''
    return x**0.5, x**(1.0/3.0), x**0.25, x**0.2

in : y = rootz(4.0)
print(y)
out: (2.0, 1.587401051968199, 1.4142135623730951, 1.31950791077

in : i,j,k,l = rootz(99.0)
print(i,j,k,l)
out: 9.949874371066 4.626065009182741 3.1543421455299043 2.5068
```

- Return can pass multiple output
- Separate output with a comma

Functions - return terminates a function

```
def someFun():
    print("Let us have some fun")
    return None
    print('This will never get printed')

in : someFun()
out: Let us have some fun
```

Functions - Flexible arguments

Sometimes you might want to create a function where you pass flexible arguments

```
def catchAll(*args, **kwargs):
    '''
    A catch all function to demonstrate
    arguments and keywords

    args is a tuple of the arguments passed to the function

    kwargs is a dictionary of the keywords passed
    '''
    print('args', args)
    print('kwargs', kwargs)
```

```
in : catchAll(1,2,3,4,a=7.0,b=3.7)
out: args (1, 2, 3, 4)
kwargs {'b': 3.7, 'a': 7.0}
```

lambda functions

Use lambda to create quick one line functions

```
square = lambda x, y, z: x**2 + y**2 + z**2
```

this creates a function square that we can call

```
in : square(1,2,3)
out: 14
```

Objects - everything in Python is an object– Moved to lecture 03

In this example x isn't really a string. It is an object containing a collection of functions

```
in : x = 'hello world'
```

```
in : dir(x)
```

dir(object) returns an alphabetized list of the attributes of the object

Use class to create a new form of object

Some reading:
`https://docs.python.org/3/tutorial/classes.html`
`http://anandology.com/python-practice-book/object_oriented_programming.html`

I like this idea of creating a bank account to explain how to use class.

Creating a bank account object– Moved to lecture 03

```
class bank_account:
    def __init__(self, initial_balance=0.0):
        self.balance = initial_balance
john = bank_account()
```

- use class to create an object named bank_account
- def __init__() is an initialization function that is run automatically on a new instance
- you can add required and optional variables to the __init__() function
- self is the naming convention in python of objects own instance (it's self)
- self is the first variable in your functions of your object
- pass self to your object's functions if you need access to your object's attributes
- balance is an attribute of the object
- a new instance of the object is created by calling bank_account()

adding a withdrawn and deposit function– Moved to lecture 03

```
class bank_account:
    def __init__(self, initial_balance=0.0):
        self.balance = initial_balance

    def withdraw(self, ammount):
        self.balance -= ammount
        print('Your new account balance is', self.balance)

    def deposit(self, ammount):
        self.balance += ammount
        print('Your new account balance is', self.balance)

in : john = bank_account(100.0)
in : john.withdraw(2.77)
out: Your new account balance is 97.23
in : john.deposit(10.0)
out: Your new account balance is 107.23
```

Adding extra attribute - liability and loan– Moved to lecture 03

```
class bank_account:
    def __init__(self, initial_balance=0.0, initial_debt=0.0):
        self.balance = initial_balance; self.debt = initial_debt
    def withdraw(self, ammount):
        self.balance -= ammount; self.print_balance()
    def deposit(self, ammount):
        self.balance += ammount; self.print_balance()
    def get_loan(self, ammount):
        self.balance += ammount; self.debt += ammount
        self.print_balance()
    def pay_debt(self, ammount):
        self.balance -= ammount; self.debt -= ammount
        self.print_balance()
    def print_balance(self):
        print('Your account balance is', self.balance,
         '\n You own the bank', self.debt)
```

Playing arround with the newly created object– Moved to lecture 03

```
in : john = bank_account(100.0,10)

in : john.withdraw(2.77)
out: Your account balance is 97.23
 You own the bank 10

in : john.deposit(10.00)
out: Your account balance is 107.23
 You own the bank 10

in : john.get_loan(1000.0)
out: Your account balance is 1107.23
 You own the bank 1010.0

in : john.pay_debt(723.0)
out: Your account balance is 384.23
```

Objects are incredibly useful – Moved to lecture 03

- I have no idea how people created large programs without object oriented programming
- You can use objects to organize a collection of functions
- Use dir() to see all of the attributes of an object
- Python naming convention object.attribute
- attributes can be new data types, data structures, and even new objects

HW 02 - turn in one week from today in Canvas

Turn in the 5 questions as a single .py file onto canvas. Use comments to clearly indicate which question you are working on. Your filename should end as _py2.py if you use Python2 and _py3.py if you use Python3.

1. The Fibonacci sequence is defined as

$$F_n = F_{n-1} + F_{n-2} \tag{1}$$

   where n denotes the n^{th} item of the Fibonacci sequence. You are given the first three numbers of the Fibonacci sequence as F = [0, 1, 1]. Create a for loop to determine the next 20 numbers of the Fibonacci sequence. Print F with the final 23 numbers. Hint: use F.append() to add a new Fibonacci value to the end of the list F.

HW 02 - turn in one week from today in Canvas

2. Parentheses are used to preserve order of operations in Python. The following code will add x+y first, then raise to the power z. This value is assigned g.

   ```
   g = (x+y)**z
   ```

   Given the list x = [2.0,3.0,5.0,7.0,9.0], create a list $Y(x)$ for each float in x. Print the list Y.

$$Y(x) = \frac{(3.0x)^2}{(99x - x^3)} - \frac{1}{x} \tag{2}$$

HW 02 - turn in one week from today in Canvas

3. The general equation for the quadratic equation is

$$ax^2 + bx + c = 0 \tag{3}$$

where the solution is

$$x_0 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{4}$$

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{5}$$

Create a function to solve the quadratic formula given a, b, c. Return x_0, x_1 with your function. Use your function to print the solution when $a = 3.3, b = 1.7, c = -9.4$.

HW 02 - turn in one week from today in Canvas

4. Use a loop to find the largest integer that when squared is less than 2000. Print the integer.
5. Create three separate functions. One function should calculate the volume (v), another to calculate the surface area (A), and another function to calculate the density (ρ) of a sphere. The input variable for these functions should be the radius r. With the density function, allow the mass m to be an optional variable that defaults to $m = 0.35$. Print the volume of a sphere with radius $r = 0.69$. Print the surface area of a sphere with radius $r = 0.4$. Print the density of a sphere with $r = 0.3$ and the default mass. Print the density of a sphere with $r = 0.25$ and $m = 2.0$.

$$v = \frac{4}{3}\pi r^3 \tag{6}$$

$$A = 4\pi r^2 \tag{7}$$

$$\rho = \frac{m}{v} \tag{8}$$