

Lecture 3: Functions, Lists and dictionaries

Overview

In this lecture I am going to finish the introduction to functions and examples of their use. We will then introduce concepts of storing related items together in appropriate containers. The two that you will most likely come across are 'lists' which store an ordered set of data. I will also show you 'dictionaries' which store data in terms of a 'key/data' pair. These concepts will be come clearer shortly.

Finally, I will discuss the class test which will be held during next week's lab session.

Recap from previous lecture

if/elif/else statements

Create a variable that stores a string such as Glasgow or Edinburgh.

Using this variable along with an if/elif/else statement print out a message such as `good choice` if the variable equals `Glasgow`, `relatively good choice` if the variable equals `Edinburgh` and a default message otherwise.

In [1]:

```
university = 'Glasgow'

if university == 'Glasgow':
    print('Good choice of university.')
elif university == 'Edinburgh':
    print('Why?! You should have gone to Glasgow.')
else:
    print('I don\'t know that university')
```

Good choice of university.

for loops

Write a for loop that computes the sum of all integers from 1 to 1000.

In [6]:

```
min_val = 1
max_val = 1000

my_sum = 0
for integer in range(min_val, max_val + 1):
    my_sum += integer

print('The sum of integers from {} to {} is {}'.format(min_val, max_val, my_sum
))
```

The sum of integers from 1 to 1000 is 500500

while loops

Write a while loop that forces a user to enter an input greater than 10.

In [8]:

```
while True:
    raw_string = input('Please enter a number greater than 10: ')
    number_input = float(raw_string)

    if number_input > 10:
        print('Thanks for your input')
        break
    else:
        print('NO!!! I said a number GREATER THAN 10')
```

```
Please enter a number greater than 10: 1
NO!!! I said a number GREATER THAN 10
Please enter a number greater than 10: 11
Thanks for your input
```

Functions

Write a function that computes $|x^2|$:

In [2]:

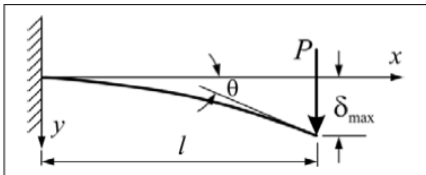
```
def squaredAbs(x):
    return abs(x**2)

print(squaredAbs(-2.0))
print(squaredAbs(2.0))
```

```
4.0
4.0
```

New Lecture Content

Functions with multiple arguments

	$\theta = \frac{Pl^2}{2EI}$	$y = \frac{Px^2}{6EI}(3l - x)$
--	-----------------------------	--------------------------------

Write a function that returns the displacement at any point along a cantilever beam

The function should take four inputs:

- P : the load
- L : the length of the beam
- x : the position along the beam where we wish to evaluate the bending moment.
- E : Young's Modulus
- I : Second moment of area

The equation is given above

In [7]:

```
def deflection(x, P, L, E, I):
    return P * x**2 / (6.0 * E * I) * (3.0 * L - x)
```

Functions can also work with strings, both as a input and output:

In [8]:

```
def addsurname(name):

    if name == 'jack':
        return name + ' bauer'
    elif name == 'robert':
        return name + ' simpson'
    else:
        return name + ' gobbledegook'
```

```
myfullname = addsurname('robert')
print(myfullname)
```

```
robert simpson
```

Problems seen in student's code during lab sessions

Be careful about how you write variables. For example, you cannot leave a space:

In []:

```
my variable = 5 # you can't leave a space in a variable name
```

Remember, everything in quotation marks will be treated as a string:

In []:

```
a = '5+4' # you've just defined a string, not an integer
print(a)
```

Remember to include the colon at the end of the first line of a function:

In []:

```
def myfunc(x)      # here we've forgotten to include the colon
    return x**2
```

New programming concepts

Lists

Lists are used to store collections of data. They can be used for any kind of data. Let's define a list of integers, floats and strings:

In [9]:

```
int_list = [2, 4, 7, 8, 9]
float_list = [3.14, 3.2345, 6.345, 7.98]
string_list = ['bricks', 'steel', 'mortar']
```

We can also mix and match:

In [24]:

```
hybrid_list = [3, 'pies', 3.45]
```

Looping over lists

We've seen this before:

In [27]:

```
mylist = [1,3,5,7,9]

for value in mylist:
    print(value)
```

1
3
5
7
9

You can also loop over multiple lists at the same time using the 'zip()' function. But you should ensure they are the same length otherwise certain values will be omitted:

In [31]:

```
first_list = [1,2,3,4,5,6]
second_list = [2,4,6,8,10,12]

for first,second in zip(first_list, second_list):
    print('First value: {} Second value: {}'.format(first, second))
```

```
First value: 1 Second value: 2
First value: 2 Second value: 4
First value: 3 Second value: 6
First value: 4 Second value: 8
First value: 5 Second value: 10
First value: 6 Second value: 12
```

Use of built-in functions with lists

range()

A very common way to create a list is to use the built-in function *range()*. It creates a list of *integers* like so:

In [11]:

```
my_list = list(range(0,10))
print(my_list)
```

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

This also has a third argument which specifies the *step* we can take:

In [12]:

```
my_jump_list = list(range(0,10,2))
print(my_jump_list)
```

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

len()

This will return the length of a list which is useful in a variety of situations

In [13]:

```
my_list = ['jack', 'jill', 'adrian', 'peter']
print('I have %s students' % len(my_list))
```

```
I have 4 students
```

sum()

As this function implies, we can sum up all the values in our list. This will not work with a list of strings.

In [14]:

```
int_list = [2, 4, 7, 8, 9]
mysum = sum(int_list)
print(mysum)
```

30

map()

This comes in handy when you want to perform a function on the values in your list. It is called as map(function, list). For example:

In [15]:

```
def myfunc(x):
    return x + 1

my_list = [1,2,3,4,5]
new_list = list(map(myfunc, my_list))
print(new_list)
```

[2, 3, 4, 5, 6]

Member functions

The previous three functions are known as built-in functions and can be used for different variable types. For lists, there are also member functions that are called in the following way: list_name.member_function().

Let's consider some of the more common member functions that operate on list:

Access operator

This allows us to access individual elements of the list:

In [37]:

```
random_list = list(range(0,20,2))

print(random_list)

print(random_list[0]) # we can access individual values like this
print(random_list[2])
print(random_list[6])
```

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
0
4
12

The most common use of this is when we loop over a list using list indices like so:

In [38]:

```
for i in range(0, len(random_list)):
    print(random_list[i])
```

```
0
2
4
6
8
10
12
14
16
18
```

.append()

As implied, this will add a value to the end of a list:

In [17]:

```
my_list = [] # create an empty list
my_list.append(3)
my_list.append(4)
print(my_list)
```

```
[3, 4]
```

.insert()

This inserts a value at a given index. All values after the given index will move up one space in the list. For example:

In [18]:

```
my_list = [2, 4, 6]
my_list.insert(1,8)
print(my_list)
```

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

.remove()

This will remove the first item with the given value. An error will occur if the value is not in the list. For example:

In [19]:

```
my_list = [3,6,9,11]
my_list.remove(6)
print(my_list)
```

```
[3, 9, 11]
```

.sort()

As implied, this sorts the list:

In [20]:

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

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

.reverse()

We can reverse the order of our list if required:

In [21]:

```
my_list = [1,3,5,7,9,11,13]
my_list.reverse()
print(my_list)
```

```
[13, 11, 9, 7, 5, 3, 1]
```

.count()

This counts the number of times a given element appears in the list.

In [22]:

```
another_list = [3.14, 4.345, 3.14, 5.754]
pi_count = another_list.count(3.14)
print(pi_count)
```

```
2
```

Class example (5-10mins)

Create a program that allows a user to input several numbers as input. We know when input is finished when the user types 'done'. Populate a list with these numbers, sort them and then output the cube of them.

In []:

```
def cubed(x):    # our cubed function
    return x**3

my_list = []     # first create an empty list that we will populate in the loop
while True:
    input_data = input('Please enter value --> ')
    if input_data == 'done':
        break    # if the user types 'done' we exit the loop
    value = float(input_data)
    my_list.append(value) # add the value to the list

my_list.sort()
cubed_list = list(map(cubed, my_list))

print(cubed_list)
print('We\'re done')
```

Dictionaries

These are much like lists but now we can specify keys for accessing items in the list. As an initial way to think of it, imagine it like a phone book which can be constructed like so:

In []:

```
my_dict = {'james' : 3354356, 'fiona' : 3354328, 'david' : 3245438}
print(my_dict)
```

This sometimes makes it more handy for accessing items in your code:

In []:

```
my_dict = {'james' : 3354356, 'fiona' : 3354328, 'david' : 3245438}
print(my_dict['james'])
print(my_dict['david'])
```

Looping over dictionaries

We can loop over all items in a dictionary like so:

In [32]:

```
my_dict = {'james' : 3354356, 'fiona' : 3354328, 'david' : 3245438}

for key,value in my_dict.items():
    print('name: {} and phone number: {}'.format(key,value))
```

```
name: james and phone number: 3354356
name: fiona and phone number: 3354328
name: david and phone number: 3245438
```

Lists in dictionaries

In [33]:

```
river_levels = {'Kelvin' : [2.34, 2.3456, 2.567, 2.897],
                'Clyde' : [5.234, 5.234, 5.23455, 5.345],
                'White Cart' : [1.234, 1.455, 1.234, 1.4556] }

print(river_levels['Clyde'])

[5.234, 5.234, 5.23455, 5.345]
```

Adding a new entry

For example, to add some new river level data:

In [34]:

```
newriver = 'Severn'
newdata = [2.3, 3.4, 4.5, 4.4, 5.6]

river_levels[newriver] = newdata # add the new data to the dictionary. This will
overwrite any existing entry

print(river_levels)

{'Kelvin': [2.34, 2.3456, 2.567, 2.897], 'Clyde': [5.234, 5.234, 5.23455, 5.345], 'White Cart': [1.234, 1.455, 1.234, 1.4556], 'Severn': [2.3, 3.4, 4.5, 4.4, 5.6]}
```

For those interested in more mathematical functions provided by the numpy library (e.g. standard deviation, etc.) see <http://docs.scipy.org/doc/numpy/reference/routines.statistics.html> (<http://docs.scipy.org/doc/numpy/reference/routines.statistics.html>).

Last modified by RNS on 18/2/16