Intro to Python

Class 3

Review

- Functions - calls, definitions, returns, arguments
- Conditions - if, elif, else
- Loops - while, for

What we will cover today

- More Functions
- Method calls
- Lists and dictionaries
- Python builtin functions

More with Functions

Functions can also call other functions.

One can use this to break up tasks into small pieces that rely on others to do their work.

```
from math import sqrt

def absolute_difference(value_a, value_b):
    return abs(value_a - value_b)

def find_width_height(x1, y1, x2, y2):
    width = absolute_difference(x1, x2)
    height = absolute_difference(y1, y2)
    return width, height

def get_hypotenuse(a, b):
    return sqrt(a ** 2 + b ** 2)

def get_area_rectangle(width, height):
    return width * height

def print_area_and_hypotenuse(x1, y1, x2, y2):
    width, height = find_width_height(x1, y1, x2, y2)
    area = get_area_rectangle(width, height)
    hypotenuse = get_hypotenuse(width, height)
    print 'Area of the rectangle is:'
    print area
    print 'The diagonal of the rectangle is:'
    print hypotenuse
```

Function Composition

Function composition is when the output of one function acts as the input of another

```
from math import sqrt

def find_width_height(x1, y1, x2, y2):
    return abs(x1 - x2), abs(y1 - y2)

def get_hypotenuse(a, b):
    return sqrt(a ** 2 + b ** 2)

def print_hypotenuse(x1, y1, x2, y2):
    print 'The diagonal of the rectangle is:'
    print get_hypotenuse(find_width_height(x1, y1, x2, y2))


# f(g(x))
# is the same as:
#     y = g(x)
#     f(y)
```

Method Calls

A **method** is like a function but it is "bound" to an object.

For example, the integers and strings we've been using have methods attached to them

We can use the dir() function to see the methods of an object and help() to see what they do.

```
a = 4
print dir(a)

name = 'caleb'
sentence = 'the quick brown fox did the thing with the thing'
print dir(name)

print name.capitalize()
print sentence.count('the')

help(name.upper)
```

Let's Develop It

Open a Python shell and define a string varaible

Use 'dir(string_variable)' and the help() function to explore the various methods

Hint: Like functions, some methods take arguments and others don't

Hint: Use help() on a method. It will tell you the arguments to use and the expected behavior

Hint: Don't be afraid of errors. They seem to be in a foreign language but they are there to help you. Read them carefully.

Lists

A list is an ordered collection of elements

In Python, a list is defined using `[ ]` with elements separated by commas, as in the following example

```
words = ['list', 'of', 'strings']
```

A list can, but doesn't have to be of all one type. A list of one type is **homogenous** as opposed to a list of multiple types, which is **heterogeneous**.

```
# heterogenous list
words = [0, 'list', 'of', 3, 'strings', 'and', 'numbers']
```

List Methods

Lists have several methods, the most useful of which is `append`

A list can be created as an empty list and have values added to it with `append`

```
to_dos = []
to_dos.append('buy soy milk')
to_dos.append('install git')
print to_dos
```

Therefore, lists are **mutable**

This means that a list can change values during the duration of a program

Iteration

Lists and many other collections are **iterable**.

Once defined, we can iterate on them, performing an action with each element

```
shipping_cost = 2.5
prices = [3, 4, 5.25]
costs = []

for price in prices:
    costs.append(price + shipping_cost)

for cost in costs:
    print cost
```

Indexing

An element can also be obtained from a list through **indexing**

This allows us to obtain an element without iterating through the entire collection if we just want one value.

To index on a collection, follow it immediately with `[index]`. (index here is a number, variable or expression)

```
numbers = [10, 20, 30]
print numbers[0]
```

Indexing continued

Lists and other collections in Python are **zero indexed**. This means that the number 0 refers to first element in the list.

```
to_dos = [
    'install git', 'read email', 'make lunch',
]
print to_dos[0]
print to_dos[1]

print to_dos[-1]
```

Negative numbers mean "go back from the end this many". e.g. -1 is the last element, -2 is the penultimate

An IndexError results if an index exceeds the length of the list minus 1

Dictionaries

A **dictionary** (sometimes called a "hashmap") is a collection of key/value pairs, defined with '{}'

```
menu_categories = {
    'food': 'stuff you eat',
    'beverage': 'stuff you drink',
}
```

Think of words in a dictionary. The words are keys and the definitions are values.

This dictionary would be indexed with strings such as 'food' and 'beverage' instead of integers like in a list

Indexing on Dictionaries

Dictionaries aren't literally just for definitions. They represent a group of mappings. A mapping might be: menu items -> costs.

We can also index on dictionaries.

The most common indexes are strings, but they can be whatever type the keys are.

```
menu = {
    'tofu': 4,
}

tofu_cost = menu['tofu']
```

Indexing on a key that doesn't exist results in a KeyError

If you aren't certain a key is present, you can use the 'get' method

Dictionary Methods

Some of the most essential methods are 'keys', 'values' and 'items'

```
menu = {
    'tofu': 4,
    'pizza': 8,
    'baguette': 3,
}

print menu.keys()
print menu.values()
print menu.items()
print menu.get('pizza')
print menu.get('water')
print menu.get('juice', 5)
```

'get' will return None if the key isn't present or a default value if provided.

The in operator

The `in` operator is used to determine if an element is in a given collection

For dictionaries, the keys are searched for the element.

```
color = (255, 255, 0)
if 0 in color:
    print '0 is in the color'

menu = {'tofu': 4}
print 'tofu' in menu

names = ['Mary', 'Martha', 'George']
george_present = 'George' in names
```

Let's Develop It

Write a program that opens a text file and does some processing.

- The program should take a word as input and determine if the word appears in the file
- The program should use at least one function to do its work and you should be able to import this function in a Python shell and call it with a word and filename
- Use the functions from **helpers.py** to help with reading in the lines and/or words of the file
- Download a book in plain text from **Project Gutenburg** and put it into the same directory as your python file.

The next slide has some code and other resources that should help you get started

Let's Develop It- Example Code

```
from helpers import generate_cleaned_lines

def is_word_in_file(word, filename):
    for line in generate_cleaned_lines(filename):
        # line will be a string of each line of the file in order
        # Your code goes here.
        # Your code should do something with the word and line variables and assign the
value to a variable for returning

input_word = raw_input("Enter a word to search for:")
answer = is_word_in_file(input_word, 'pride.txt')
# Display the answer in some meaningful way
```

I have used **Pride and Prejudice** from Project Gutenburg with my example code. You can click this link and copy/paste the text into a new text file called 'pride.txt' and save it in the same folder as your code

Lists of ... Lists

Lists can contain not only numbers or strings, but also other lists.

Such a structure is said to be **nested**

The following is a list of lists:

```
game_board = [
    ['O', 'X', ' '],
    [' ', 'X', ' '],
    [' ', ' ', ' '],
]
```

This can be indexed successively with `game_board[0][1]`

Nested Lists continued

A list can be appended to a list as well

```
mary_to_dos = ['eat', 'work', 'pick up laundry', 'care for baby', 'sleep']
fran_to_dos = ['eat', 'work', 'call plumber', 'sleep']
baby_to_dos = ['eat', 'sleep']

all_to_dos = []
all_to_dos.append(mary_to_dos)
all_to_dos.append(fran_to_dos)
all_to_dos.append(baby_to_dos)

print all_to_dos

for to_dos in all_to_dos:
    for to_do in to_dos:
        print to_do
```

What if we want to **flatten** the to do's?

What if we want the to do's to be unique?

Lists of Dictionaries

One of the most common and useful nested data structures, is a list of dictionaries

```
card_a = {
    'suit': 'spades',
    'number': 4,
}

card_b = {
    'suit': 'hearts',
    'number': 8,
}

hand = [card_a, card_b]

print 'The hand contains:'
for card in hand:
    print 'A', card['number'], 'of', card['suit']
```

Dictionary of Lists

A dictionary can also contain values that are themselves other collections, such as lists.

Let's revisit the group of to do lists and find a better representation:

```
mary_to_dos = ['eat', 'work', 'pick up laundry', 'care for baby', 'sleep']
fran_to_dos = ['eat', 'work', 'call plumber', 'sleep']
baby_to_dos = ['eat', 'sleep']

all_to_dos = {
    'mary': mary_to_dos,
    'fran': fran_to_dos,
    'baby': baby_to_dos,
}

for name, to_dos in all_to_dos.items():
    print name, 'needs to: ', to_dos

# Changing this later can be accomplished with
all_to_dos['baby'].append('cry')
```

Now the to do lists can be indexed or modified by name

Means of Combination

Lists, dictionaries, and other collections are all a means of combination.

They can be freely combined to create the data structure needed for a particular problem

Eg. A list of dictionaries with lists

```
all_tweets = [
    {
        'author': 'mary',
        'handle': '@hadalittlelamb',
        'date': '2013-01-22',
        'tweets': [
            'at Loco Pops enjoying a Raspberry Sage popsicle',
            'Learning Python is so much fun',
        ],
    },
]
```

Builtins for collections

Python provides several functions that help us work with these collections.

len()	Given a collection, return its length
range()	Create a list of integers in the range provided.
sorted()	Given a collection, returns a sorted copy of that collection
enumerate()	Returns a list of (index, element) from the list
zip()	Given one or more iterables, returns a list

of tuples with an element from each iterable

Examples of using Builtins

```
# Using len() - Determines length
print len([1, 2])

# range() - Quickly creates a list of integers
print range(5)
print range(5, 10)
print range(0, 10, 2)
print range(9, -1, -1)

# sorted() - Sort a given list
grades = [93, 100, 60]
grades = sorted(grades)
print grades
```

Builtins Examples continued

```
# enumerate() - Obtain the index of the element in the loop

print 'To Do:'
to_dos = ['work', 'sleep', 'work']
for index, item in enumerate(to_dos):
    print '{0}. {1}'.format(index + 1, item)

print list(enumerate(to_dos))

# zip()
widths = [10, 15, 20]
heights = [5, 8, 12]
for width, height in zip(widths, heights):
    print 'Area is {0}'.format(width * height)
```

Let's Develop It

Write a program that expands on your previous one. If it is unfinished, feel free to finish the original exercise first. To expand on it, choose one of the following:

1. Determine how many times the user provided word appears in the file and/or what lines it appears on
2. Change the program so that it counts the number of times each word occurs. E.g. A dictionary of all words in the file, whose values are a count of their occurrences
3. Use **boilerplate.py** to help you improve the reusability of the program. (The comments in that file should explain the how and why)

Resources for this and the previous exercise are provided on the next slide for convenience

Resources

- Helper functions are in **helpers.py**
- Download a book in plain text from **Project Gutenburg** and put it into the same directory as your python file.
- You can use this link for **Pride and Prejudice**. Click this link and copy/paste the text into a new text file called 'pride.txt' and save it in the same folder as your code

```
from helpers import generate_cleaned_lines

def is_word_in_file(word, filename):
    for line in generate_cleaned_lines(filename):
        # line will be a string of each line of the file in order
        # Your code goes here. Do something with the word and line variables
    return result

input_word = raw_input("Enter a word to search for:")
answer = is_word_in_file(input_word, 'pride.txt')
# Display the answer in some meaningful way
```

Questions?