Introduction to Python

John Kelleher

Dublin Institute of Technology

Acknowledgements

- These notes are intended to provide an introduction to Python, they are not meant to be an exhaustive explanation of the language.
- Python is a very popular language and there are a lot of resources online for learning Python. I encourage you to use them. Indeed, I drew on several online Python tutorials when preparing these notes.
- If you are looking for a good introductory text to Python I would also recommend the book `Python Programming: An Introduction to Computer Science` 2nd `Edition` by `John Zelle`, I found it very useful in preparing these notes.

- There are two versions of Python that are currently available Python 2.x and Python 3.x. These notes assume you are using Python 3.x. But if you are using 2.x you should be fine once you keep an eye out for the following differences:
  1. division works slightly differently in the different versions of Python (in 2.x division rounds and in 3.x it doesn't).
  2. in 2.x you don't need to put brackets around parameter lists in function calls (e.g., *print x*) in 3.x you do (e.g., *print(x)*).

- Ok, lets get started by looking at some Python code!

```
for line in open("file.txt"):
        for word in line.split():
                if word.endswith('ing'):
                        print(word)
```

- **Can you figure out what the Python program listed above does?**

- Ok, lets get started by looking at some Python code!

```
for line in open("file.txt"):
        for word in line.split():
                if word.endswith('ing'):
                        print(word)
```

- Python is so easy to read I bet you can guess: it processes *file.txt* and prints all the words ending in *ing*

```
for line in open("file.txt"):
        for word in line.split():
                if word.endswith('ing'):
                        print(word)
```

Features of Python

- Python is **interpreted** - no compilation necessary
- **Whitespace** is used to nest lines of code (forces you into a clean style)
- Python is **object-oriented**; each variable is an entity that has certain defined attributes and methods.
- Methods have **arguments** expressed inside parentheses.
- Python is **highly readable**, **open source**, **extensible**, **popular**

Getting Python

- You can download Python for free from: **www.python.org**

Starting Python

The easiest way to interact with Python is through the Interactive DeveLopment Environment (IDLE) IDE. To startup IDLE:

start→All Program→Python X*→IDLE (Python GUI)

Try starting Python now. You should see something like:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18
   2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
   on darwin
Type "copyright", "credits" or "license()"
   for more information.
>>>
```

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18
   2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
   on darwin
Type "copyright", "credits" or "license()"
   for more information.
>>>
```

- Once we have IDLE started we can interact with Python in
  1. an interactive mode, typing our programs directly in at the prompt >>>
  2. or in a non-interactive mode, saving our programs before we execute them.

Let's begin by using Python as an interactive calculator:

Python 3.4.1 Shell

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more informati
on.
>>> 1 + 5 * 2 - 3
8
>>>
```

Ln: 6 Col: 4

Let's begin by using Python as an interactive calculator:

```
Python 3.4.1 Shell
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more informati
on.
>>> 1 + 5 * 2 - 3
8
>>>
Ln: 6 Col: 4
```

What is the value off: $57 - 3 * 26 + 2$?

Let's begin by using Python as an interactive calculator:

```
Python 3.4.1 Shell
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more informati
on.
>>> 1 + 5 * 2 - 3
8
>>>
Ln: 6 Col: 4
```

-19

Your Turn:

Enter a few more expressions of your own. You can use:

- asterisk (*) for multiplication
- slash (/) for division,
- parentheses for bracketing expressions.

Modules

Python is easily *extensible*. Users can easily write programs that extend the basic functionality, and these programs can be used by other programs, by loading them as a *module*

Your Turn:

1. load the math module: `import math`
2. Round 35.4 down to the nearest integer: `math.floor(35.4)`

Now lets try interacting with Python in a **non-interactive manner**.

Create the Data

1. Create a folder where you can save your work for this class.
2. Open a text editor and create a file call **file.txt**
3. Enter some random words into the file but be sure that some of the words end in **ing**
4. Save the file an close it.

Entering the Python Script

1. In the IDLE environment go to **File→New Window**
2. In the new window that opens up, enter the Python program listed below, don't forget the **:**s
3. Once you have entered the Python code go to **File→Save** and save the script with the name **ex1.py** in the same directory as **file.txt**

```
for line in open("file.txt"):
        for word in line.split():
                if word.endswith('ing'):
                        print(word)
```

Running the Script

1. To run your script: **Run→Run Module**
2. Python will print out all the words ending in *ing* in the interactive screen.

- Python variables do not have to be explicitly declared, the declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.
  - the operand to the left of the = operator is the name of the variable,
  - and the operand to the right of the = operator is the value stored in the variable.

- The Python interpreter automatically types the variable based on the value you assign it.
- You can change the variable type by changing the value stored in it.

Python 3.4.1 Shell

```
>>> myInt = 10
>>> print(myInt)
10
>>> myInt + 5
15
>>> myInt = "stringValue"
>>> print(myInt)
stringValue
>>> myInt + 5
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    myInt + 5
TypeError: Can't convert 'int' object to str implicitly
>>>
>>>
```

Ln: 36 Col: 4

Python has five standard data types:

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

Python's variable naming conventions

- should start with a letter, optionally followed by digits (0 to 9) or letters
- are case sensitive, (e.g. myVar and myvar are different variables)
- cannot contain a whitespace
- cannot be a python reserved word

Strings in Python

- Strings must be enclosed in quotes (double or single)
- concatenate using the **+** operator
- repeat using the ***** operator
- string indices start at 0
- **join** a list of strings into one string
- **split** a strings into a list

Strings in Python

Python 3.4.1 Shell

```
>>> str="Monty"
>>> str[0]
'M'
>>> str[1]
'o'
>>> str[:4]
'Mont'
>>> str[-3:]
'nty'
>>> sent="This is a sentence"
>>> words = sent.split()
>>> words
['This', 'is', 'a', 'sentence']
>>> words[0]
'This'
>>> germanword = ''.join(words)
>>> print(germanword)
Thisisasentence
>>>
```

Ln: 84 Col: 4

Your turn: join a list of strings into one string

```
','.join(['foo', 'bar'])
```

Your turn: split a strings into a list

```
mystring = 'hello world'
mylist = mystring.split()
```

- A python list can be written as a list of comma-separated values (items) between square brackets .
- List items need not all have the same type.
- Like string indices, list indices start at 0

Example

```
>>> sent1 = ["Call","me","Ishmael",".”]
>>> sent1[1]
"me"
>>>
```

Your turn: Lists

1. Create a list *foo*, with the following values: 25, 68, “bar”, 89.45, 789, “spam”, 0, “last item”
```
foo = [25, 68, 'bar', 89.45, 789,
'spam', 0,'last item']
```
2. print just the first item foo
3. print just the last item foo

Slicing

A list slice takes part of a list. A slice `list[i:j]` starts at the i_{th} index, and goes up to (but does not include) the j_{th} index.

Let: `foo = [25, 68, 'bar', 89.45, 789, 'spam', 0,'last item']`

Your Turn: Slicing Practice

1. Print the 1st to 3rd item in the list *foo*
   `print(foo[:3])`
2. Print the 3rd to last item in the list *foo*
   `print(foo[2:])`
3. Print the 2nd to the 2nd to last item in the list *foo*
   `print(foo[1:-1])`
4. Copy the entire *foo* list to a new list named *bar*

HINT: remember that indices start at `0`

Lists in Python

Operations on lists

- `val in list`
- `len(list)`
- `list.append(val)`
- `list.insert(index, val)`
- `len(list)`
- `list.index(val)`
- `last = list.pop()`

On the next slide we will carry out some operations on a list called `foo`. For the purposes of these exercises we will assume that: **`foo = [25, 68, 'bar', 89.45, 789, 'spam', 0,'last item']`**

Your turn: operations on lists

1. Change the first item in the *foo* list to 12
2. Now multiply the first item in the *foo* list by 2
3. Test whether "ham" is **in** the list *foo*
4. How many items does *foo* contain?
5. Append the value 24 to the list *foo*
6. Insert the value "twenty" to the list *foo* as the 4th item
7. Find the index of "spam" in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. Now multiply the first item in the *foo* list by 2
3. Test whether "ham" is **in** the list *foo*
4. How many items does *foo* contain?
5. Append the value 24 to the list *foo*
6. Insert the value "twenty" to the list *foo* as the 4th item
7. Find the index of "spam" in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. Test whether “ham” is **in** the list *foo*
4. How many items does *foo* contain?
5. Append the value 24 to the list *foo*
6. Insert the value “twenty” to the list *foo* as the 4th item
7. Find the index of “spam” in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. How many items does *foo* contain?
5. Append the value 24 to the list *foo*
6. Insert the value “twenty” to the list *foo* as the 4th item
7. Find the index of “spam” in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. `len(foo)`
5. Append the value 24 to the list *foo*
6. Insert the value “twenty” to the list *foo* as the 4th item
7. Find the index of “spam” in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. `len(foo)`
5. `foo.append(24)`
6. Insert the value “twenty” to the list *foo* as the 4th item
7. Find the index of “spam” in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. `len(foo)`
5. `foo.append(24)`
6. `foo.insert(3,'twenty')`
7. Find the index of “spam” in the list *foo*
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. `len(foo)`
5. `foo.append(24)`
6. `foo.insert(3,'twenty')`
7. `foo.index('spam')`
8. Remove the last item from *foo*, and store it as a new variable

Your turn: operations on lists

1. `foo[0]=12`
2. `foo[0] = foo[0]*2`
3. `'ham' in foo`
4. `len(foo)`
5. `foo.append(24)`
6. `foo.insert(3,'twenty')`
7. `foo.index('spam')`
8. `lastfoo=foo.pop()`

Definition

- A Tuple is just another kind of sequence in Python.
- Tuples are very similar to lists but:
  1. they are enclosed in round brackets **()** rather than square brackets **[]**
  2. and they are **immutable** (i.e., the items can't be changed)

- If the contents of a sequence won't be changed after it is created using a tuple is more efficient than using a list.

- Indexing and slicing work with tuples just as with lists.
- Tuples do not support methods such as sorting.
- You can create them with parentheses:

Example

```
mytuple=(10,50,'foo')
```

- Example code illustrating how to define a list of tuples and how to iterate across a list of tuples.

```
>>> listoftuples = [('t1a','t1b','t1c'), (
   't2a','t2b','t2c'),('t3a','t3b','t3c')]
>>> for (a, b, c) in listoftuples:
          print(c,b,a)

t1c t1b t1a
t2c t2b t2a
t3c t3b t3a
>>>
```

In addition to lists, Python also has a built-in *set* type

Definition

A *set* is an unordered collection of unique elements

- You can't index into a set
- You can iterate over a set

Sets in Python

```
>>> mylist=['foo','bar','foo']
>>> myset=set(mylist)
>>> myset
{'foo', 'bar'}
>>> myset[0]
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    myset[0]
TypeError: 'set' object does not support
   indexing
>>> for i in myset:
print(i)

foo
bar
>>>
```

Your turn:

1. Define a string variable `sent1` that contains the following sentence `The quick brown fox jumps over the lazy dog`
2. Use `split()` to split the string `sent1` into the list form and store the result in the variable `list1`.
3. Use `' '.join(list1)` to convert this list back into a string and store this string in the variable `sent2`
4. Convert the variable `list1` into a set and output the result?
5. Convert the variable `sent2` into a set and output the result?
6. What is the different between these two sets? Do you understand why they are different?
7. What is the value of `list2[2][2]` do? Why?

Sets in Python

Python Shell

```
>>> sent1 = "The quick brown fox jumps over the lazy dog"
>>> list1 = sent1.split()
>>> list1
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
>>> set1 = set(list1)
>>> set1
set(['brown', 'lazy', 'over', 'fox', 'dog', 'the', 'quick', 'The', 'jumps'])
>>> sent2 = ' '.join(list1)
>>> set2 = set(sent2)
>>> set2
set([' ', 'T', 'a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l'
, 'o', 'n', 'q', 'p', 's', 'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])
>>> list1[2][2]
'o'
>>>
```

Ln: 128 Col: 4

Dictionaries: Non-sequential Data Representation

- Lists and sets allow us to store and retrieve items from sequential collections. When we want to access an item in the collection, we look them up by index.
- Many applications require a more flexible way to look up information. For example, we might want to retrieve information about students or employees based on their social security numbers.
- In programming terminology, this is a `key-value pair`: we access the value (e.g. student info) associated with a particular key (e.g. social security number). A collection that allows us to look up information associated with arbitrary keys is called a 'mapping'. Python **dictionaries** are mappings. Some other programming languages provide similar structures called 'hashes' or 'associative-arrays'

Dictionaries: Non-sequential Data Representation

- A dictionary can be created in Python by listing key-value pairs inside of curly braces.
- Here is an example dictionary that stores some fictional usernames and passwords

Python Shell

```
>>>
>>> passwd = {"guido":"superprogrammer", "turing":"genius", "bill":"monopoly"}
>>>
```

Ln: 149 Col: 4

Dictionaries: Non-sequential Data Representation

- Notice that keys and values are joined with a ":", and commas are used to separate the pairs.
- The main use for a dictionary is to look up the value associated with a particular key.
- This is done through indexing notation:

Python Shell

```
>>>
>>> passwd = {"guido":"superprogrammer", "turing":"genius", "bill":"monopoly"}
>>> passwd["guido"]
'superprogrammer'
>>> passwd["bill"]
'monopoly'
>>>
```

Ln: 153 Col: 4

- In general, `<dictionary>[<key>]` returns the object associated with the given key.

Dictionaries: Non-sequential Data Representation

- Dictionaries are mutable; the value associated with a key can be changed through assignment.

Python Shell

```
>>> passwd = {"guido":"superprogrammer", "turing":"genius", "bill":"monopoly"}
>>> passwd["guido"]
'superprogrammer'
>>> passwd["bill"]
'monopoly'
>>> passwd["bill"]="philanthropist"
>>> passwd
{'turing': 'genius', 'bill': 'philanthropist', 'guido': 'superprogrammer'}
>>>
```

Ln: 156 Col: 4

- Note that the dictionary prints out in a different order from how it was originally created. This is not a mistake. Mappings are inherently unordered. Python stores dictionaries in a way that makes key lookup efficient. If you want to keep a collection of items in a certain order, you need a sequence, not a mapping.

Dictionaries: Non-sequential Data Representation

- It is very easy to extend a Python dictionary by adding new entries:

Python Shell

```
>>> passwd = {"guido":"superprogrammer", "turing":"genius", "bill":"monopoly"}
>>> passwd["guido"]
'superprogrammer'
>>> passwd["bill"]
'monopoly'
>>> passwd["bill"]="philanthropist"
>>> passwd
{'turing': 'genius', 'bill': 'philanthropist', 'guido': 'superprogrammer'}
>>> passwd["newuser"]="newbie"
>>> passwd
{'turing': 'genius', 'bill': 'philanthropist', 'newuser': 'newbie', 'guido': '
superprogrammer'}
>>>
```

Ln: 159 Col: 4

Dictionaries: Non-sequential Data Representation

- In fact, a common method for building dictionaries is to start with an empty collection and add the key-value paries one at a time.
- For example, suppose that usernames and passwords were stored in a file called passwords, where each line of the file contains a username and password with a space between them we could create a new dictionary from the file.

Python Shell

```
>>> passwd = {}
>>> for line in open('/Users/jkelleher/passwords.txt', 'r'):
        user, pw = line.split()
        passwd[user]=pw


>>> passwd
{'jane': 'champagne', 'john': 'icecream', 'mary': 'chocolate', 'tom': 'toffee'}
>>>
```

Ln: 163 Col: 0

passwords.txt – /Users/jkelleher

Open Save Close

```
john icecream
mary chocolate
tom toffee
jane champagne
```

y 2012 20:10 · Length: 54 · Encoding: Unicode

- **Your turn:** Create a text file of user names and passwords and use the code from the previous slide to read the contents of your file into a dictionary.

To manipulate the contents of a dictionary, Python provides the following methods:

- `key in dict` - returns true if dict contains key else returns false
- `dict.keys()` - returns a sequence of keys
- `dict.values()` - returns a sequence of values
- `dict.items()` - returns a sequence of tuples (key, value) representing the key value pairs
- `dict.get(key,default)` - If dict has a key returns its value else returns default
- `dict.clear()` - deletes all entries
- `for var in dict:` - loop over the keys

- **Your turn:** Try out the methods listed on the previous slide on the dictionary containing the usernames and passwords that you read in from the file.