Python Basic

Prepared by Professor Yuefeng Li for CAB431

School of Computer Science

Queensland University of Technology

Learning Objectives

1. *Installing Python*
2. *Python commands*
3. *Strings and Lists*
4. *Program and functions*
5. Dictionaries, Structuring Data and Classes
6. *Reading and Writing Files*
7. *Web Scraping*

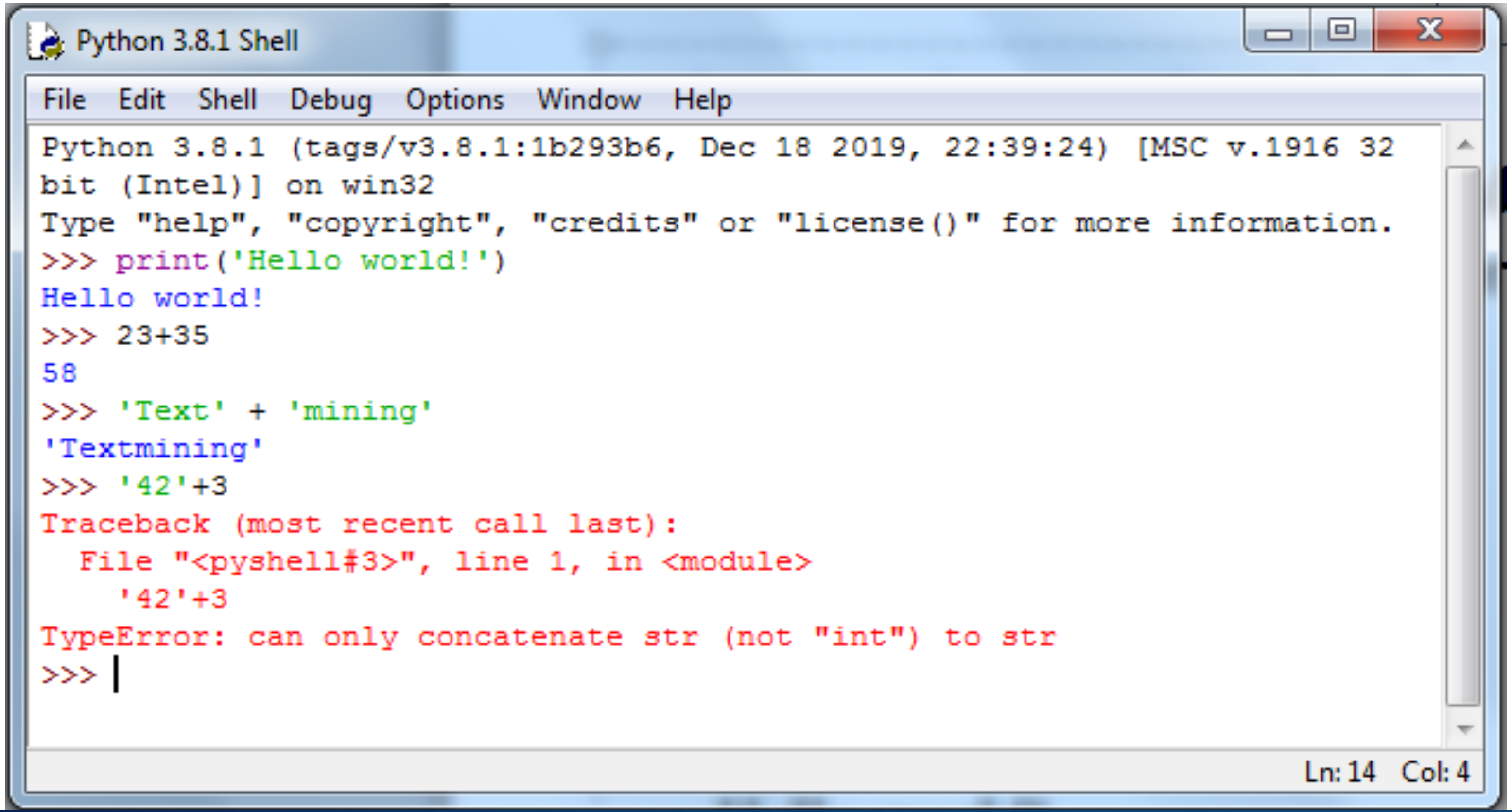
Why Python?

- Python is an easy to learn, powerful programming language for text process.
- It has efficient high-level data structures and a simple but effective approach to object-oriented programming.
- Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Professor Yuefeng Li
School of Computer Science

1. Installing Python

- http://python.org/downloads/
- Python installers for 64-bit or 32-bit computers (see system in the Control Panel, or hardware in system reports of OS X)
- Starting IDLE
  - Python GUI



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32
bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello world!')
Hello world!
>>> 23+35
58
>>> 'Text' + 'mining'
'Textmining'
>>> '42'+3
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    '42'+3
TypeError: can only concatenate str (not "int") to str
>>> 
```

2. Python commands

Start the interpreter and wait for the primary prompt, >>>

- Expression syntax is straightforward: the operators +, -, *, /, % (remainder) and ** (calculate powers) work just like in other languages; parentheses (()) can be used for grouping.
- Strings –
  - They can be enclosed in single quotes ('...') or double quotes ("...") with the same result, where special characters are escaped with backslashes \.
  - The string is enclosed in double quotes if the string contains a single quote and no double quotes, otherwise it is enclosed in single quotes.
  - The print() function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters.

Escape character	Prints as
\’	single quote
\”	double quote
\t	tab
\n	newline
\\	backslash

```
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

3. Strings and Lists

- **Strings** can be concatenated (glued together) with the + operator, and repeated with *
- Two or more *string literals* (i.e. the ones enclosed between quotes) next to each other are automatically concatenated.
    - Please note can't concatenate a variable and a string literal. Use + is OK.
- Strings can be indexed (subscripted), with the first character having index 0. Indices may also be negative numbers, to start counting from the right.
    - There is no separate character type; **a character** is simply a string of size one
- **Slicing** is used to obtain substring.
- PS: Python strings cannot be changed — they are **immutable**. Therefore, **assigning to an indexed position in the string results in an error**:

  TypeError: 'str' object does not support item assignment

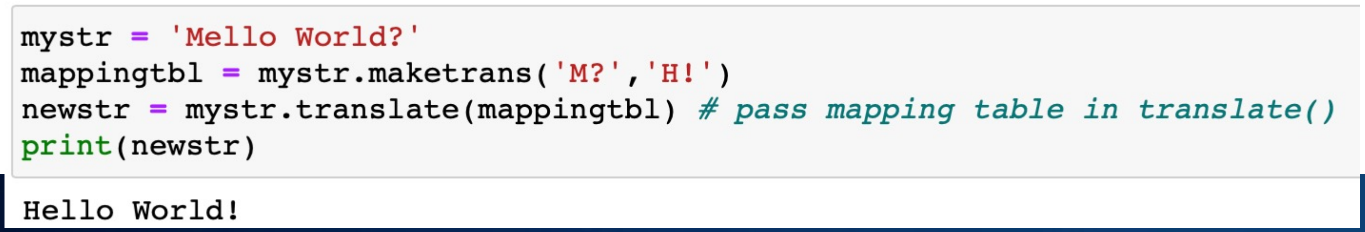
```
>>> 3 * 'un' + 'ium'
'unununium'
>>> 'Py' 'thon'
'Python'


>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[-1]  # last character
'n'
>>> word[2:5]  # from position 2 to
'tho'
>>> word[:2]   # from beginning to 2
'Py'
>>> len(word)
6
```

String Methods

Try the following methods:

- upper(), lower(), isupper() and islower()
- isalpha(), isalnum(), isdecimal(), isspace() and istitle()
- startswith() and endswith()
- join()
- split(sep=None, maxsplit=-1)

  # sep as the delimiter. If maxsplit is given, at most maxsplit splits are done.
- strip() #leading and trailing characters (e.g., removes # at the beginning or the end, strip('#'))
- replace(old, new[, count])
  - #substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced, e.g., spam.replace('o', 'O', 1)
- maketrans(x[, y[, z]])
  - x - can be a string or dictionary, y – optinal, len(y) len(x), and Each character in x is a replacement to its corresponding index in y. z - optional, it specifies the characters to remove in the string.
  - #returns a translation table usable for str.translate().



```
mystr = 'Mello World?'
mappingtbl = mystr.maketrans('M?','H!')
newstr = mystr.translate(mappingtbl) # pass mapping table in translate()
print(newstr)

Hello World!
```

```
>>> spam = 'Hello world!'
>>> print(spam.upper())
HELLO WORLD!
>>> spam.lower()
'hello world!'
>>> spam.isupper()
False
>>> spam.isalpha()
False
>>> spam=spam.upper()
>>> spam.istitle()
False
>>> spam = 'Hello World!'
>>> spam.istitle()
True
>>> spam.startswith('He')
True
>>> '-'.join(['My', 'Name', 'is', 'John'])
'My-Name-is-John'
>>> 'My-Name-is-John'.split('-')
['My', 'Name', 'is', 'John']
```

Professor Yuefeng Li
School of Computer Science

List

- Python knows a number of *compound* data types, used to group together other values.
- The most versatile is the **list**, which can be written as a list of comma-separated values (items) between square brackets.
- Lists might contain items of different types, but usually the items all have the same type.
- lists can be indexed and sliced.
- Lists also support operations like concatenation +
- Unlike strings, which are immutable, lists are a **mutable** type, i.e. it is possible to change their content
- You can also add new items at the end of the list, by using the append() method.
- Assignment to slices is also possible, and this can even change the size of the list or clear it entirely.
- It is possible to nest lists (create lists containing other lists).

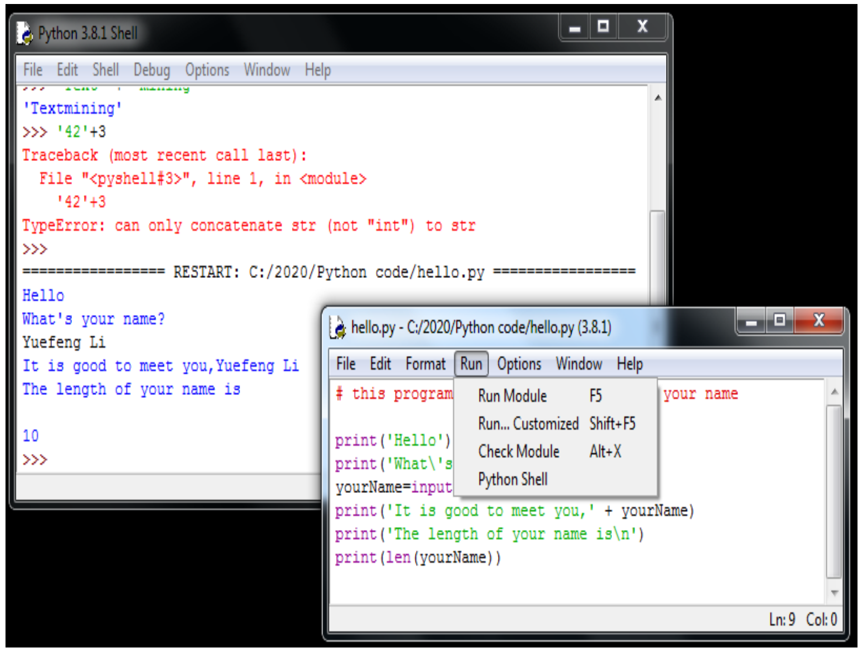
```
>>> squares = [1, 4, 9, 16, 25]
>>> squares[0]
1
>>> squares[-1]
25
>>> squares[-3:]  # get a new list
[9, 16, 25]
>>> cubes = [1, 8, 27, 65, 125]
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
>>> cubes.append(216)  # add the cube of 6
>>> cubes
[1, 8, 27, 64, 125, 216]
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> len(letters)
4
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
```

4. Program and functions: The First Program

- Type the instructions into a file editor OR in IDLE, select File -> New File

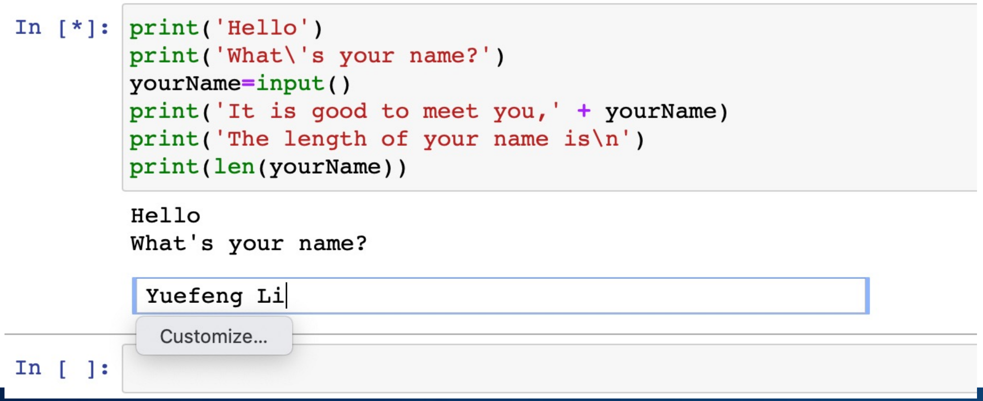
```
# this program says hello and asks for your name
print('Hello')
print('What\'s your name?')
yourName=input()
print('It is good to meet you,' + yourName)
print('The length of your name is\n')
print(len(yourName))
```

- Save (as) your file
- Run -> Run module
- OR using Jupyter

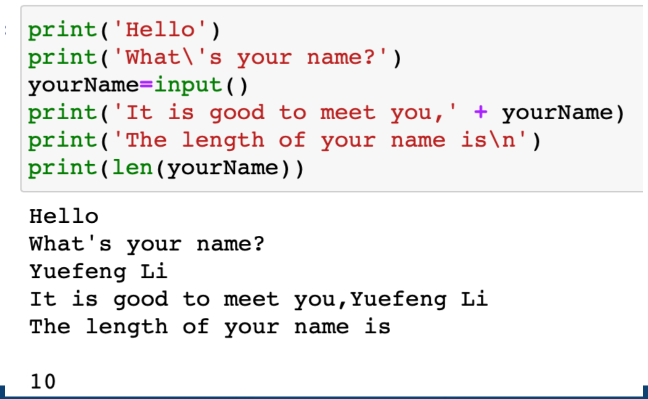

Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
hello.py - C:/2020/Python code/hello.py (3.8.1)
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
Ln: 9 Col: 0

```
'Textmining'
>>> '42'+3
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    '42'+3
TypeError: can only concatenate str (not "int") to str
>>>
================= RESTART: C:/2020/Python code/hello.py =================
Hello
What's your name?
Yuefeng Li
It is good to meet you,Yuefeng Li
The length of your name is

10
>>>
```

```
# this program                your name

print('Hello')
print('What\'s
yourName=input
print('It is good to meet you,' + yourName)
print('The length of your name is\n')
print(len(yourName))
```



```
In [*]: print('Hello')
        print('What\'s your name?')
        yourName=input()
        print('It is good to meet you,' + yourName)
        print('The length of your name is\n')
        print(len(yourName))

        Hello
        What's your name?

        Yuefeng Li

In [ ]:
```



```
print('Hello')
print('What\'s your name?')
yourName=input()
print('It is good to meet you,' + yourName)
print('The length of your name is\n')
print(len(yourName))

Hello
What's your name?
Yuefeng Li
It is good to meet you,Yuefeng Li
The length of your name is

10
```

Professor Yuefeng Li
School of Computer Science

Statements

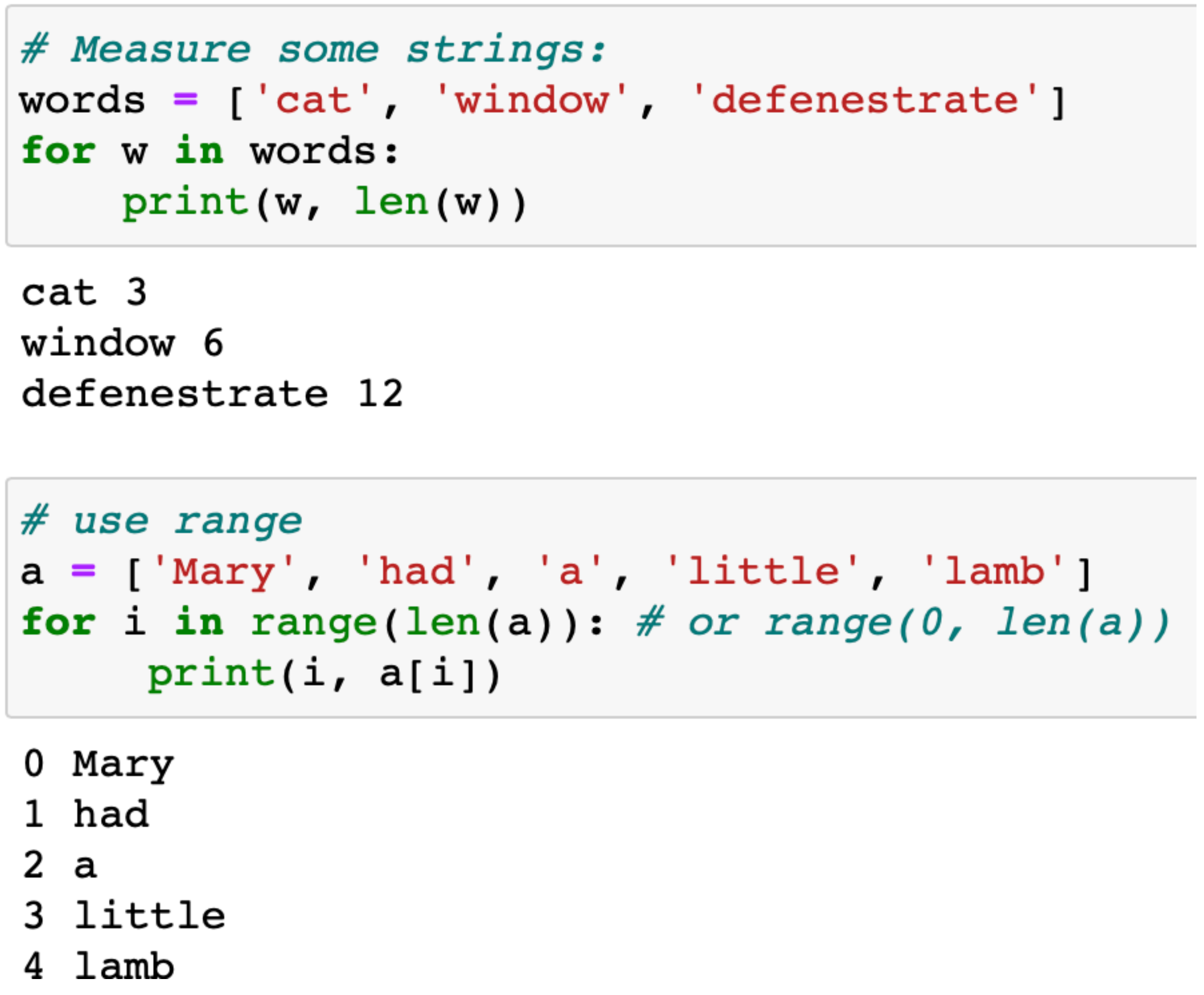
- Assignment statements: x = y
- Boolean values: True, False
- Comparison operators (==, !=, < >, <=, >=)
- Boolean operators: and, or, not
- If statements

```
x = int(input("Please enter an integer: "))
if x < 0:
     x = 0
     print('Negative changed to zero')
elif x == 0:
     print('Zero')
elif x == 1:
     print('Single')
else:
     print('More')
```

```
Please enter an integer: 1
Single
```

for Statements

- Examples



```
# Measure some strings:
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
```

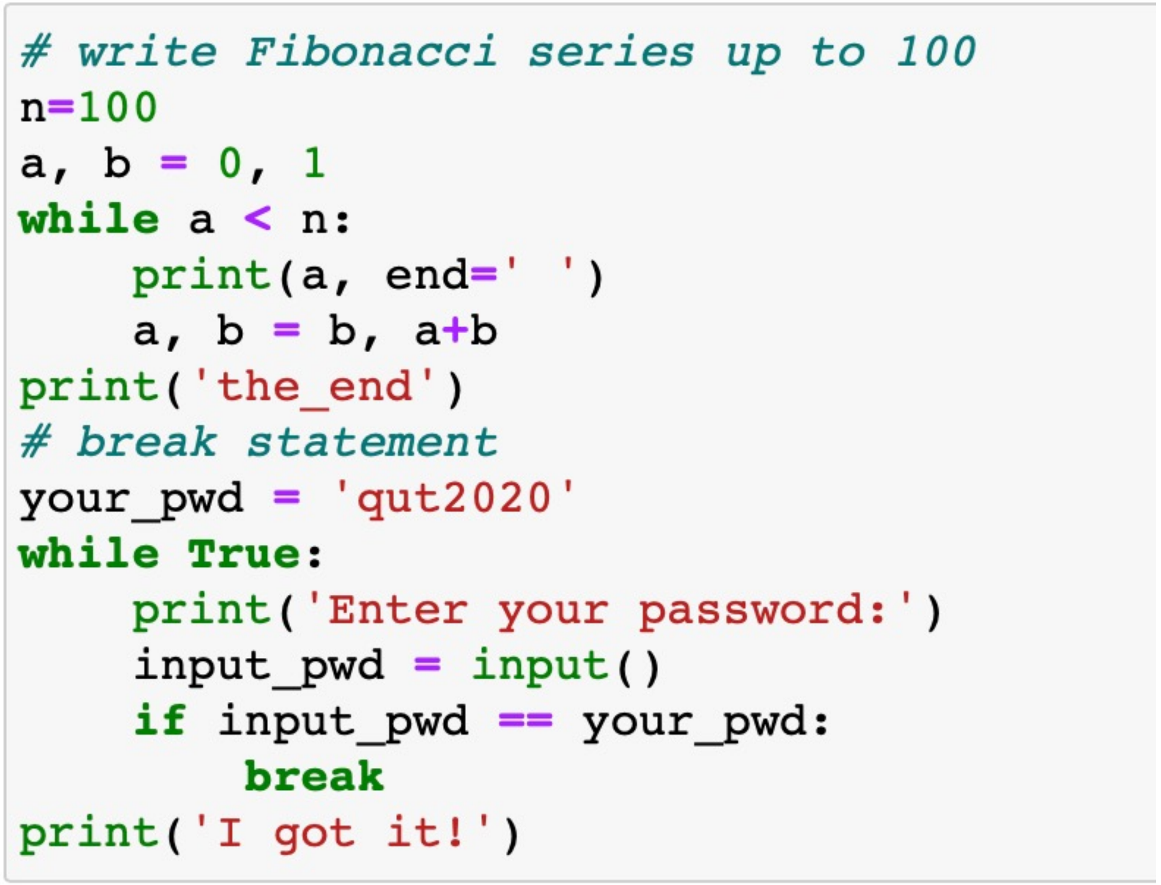
```
cat 3
window 6
defenestrate 12
```

```
# use range
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)): # or range(0, len(a))
     print(i, a[i])
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

While statements

- Examples

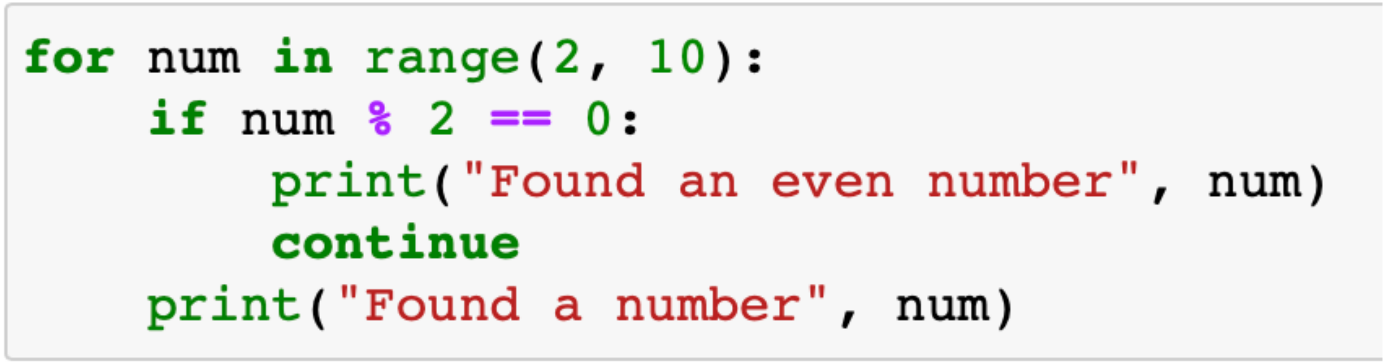


```python
# write Fibonacci series up to 100
n=100
a, b = 0, 1
while a < n:
    print(a, end=' ')
    a, b = b, a+b
print('the_end')
# break statement
your_pwd = 'qut2020'
while True:
    print('Enter your password:')
    input_pwd = input()
    if input_pwd == your_pwd:
        break
print('I got it!')
```

```
0 1 1 2 3 5 8 13 21 34 55 89 the_end
Enter your password:
qut2020
I got it!
```

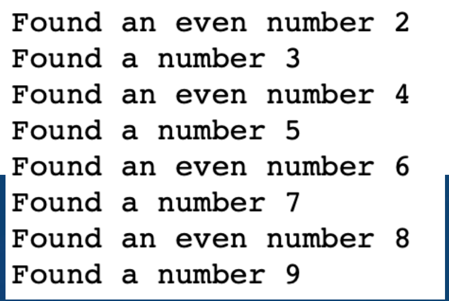
break and continue Statements

- The break statement, like in C, breaks out of the innermost enclosing for or while loop.
- The continue statement, also borrowed from C, continues with the next iteration of the loop:



```
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found a number", num)
```

- Please guess the outputs?



```
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

Importing Modules

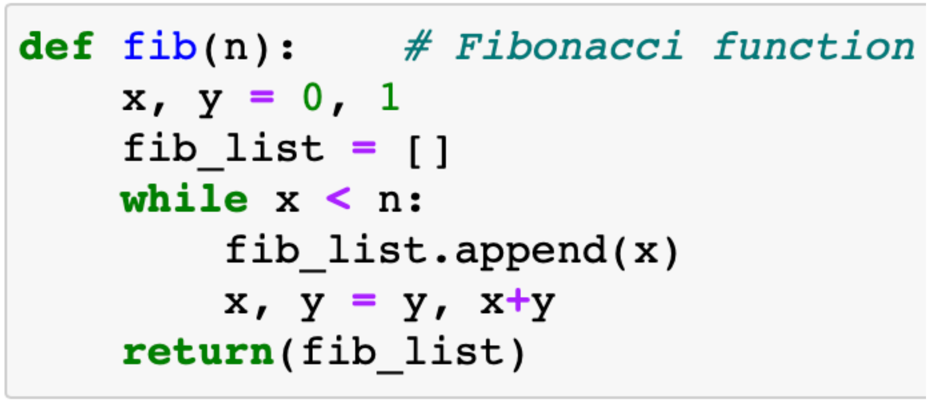
- A module can contain executable statements as well as function definitions.
- Also, all Python programs can call **built-in functions** (a basic set, e.g., input(), print(), len(), int(), float(), str())
- Python also comes with the **standard library** (a set of modules, e.g., math, random, sys, os)
- The import keyword (e.g., import random; the we can use random.randint(a, b) function to get a random number between a and b)
- Standard Modules: sys
  - dir() function is used to find out which names a module defines.

Packages

- Packages are a way of structuring Python’s module namespace by using “dotted module names”.
- For example, the module name A.B designates a submodule named B in a package named A.
- Users of the package can import individual modules from the package, for example:
  - **import sound.effects.echo** #It must be referenced with its full name, or
  - **from sound.effects import** echo #makes it without its package prefix

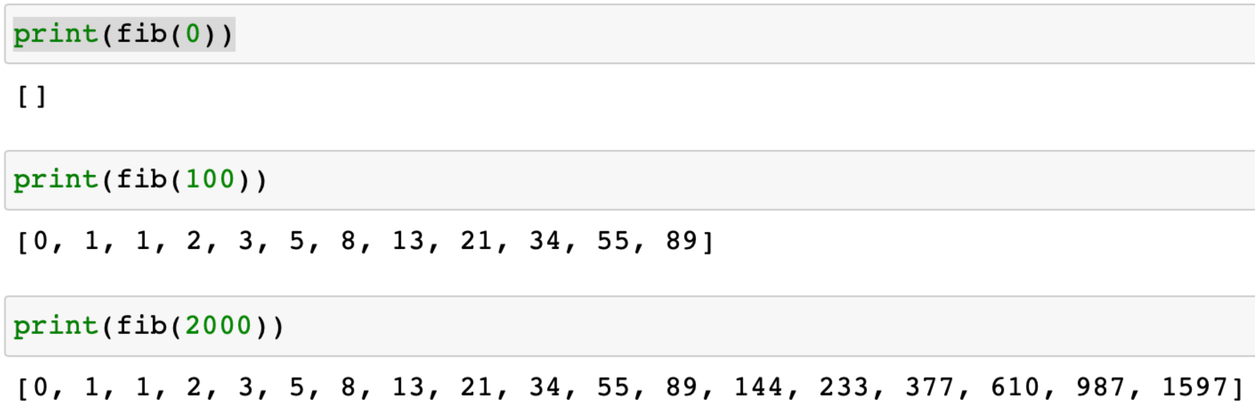
Functions

- The keyword **def** introduces a function *definition*.
- It must be followed by the function name and the parenthesized list of formal parameters.
- The statements that form the body of the function start at the next line and must be indented.
- You can also use a **return** statement to return a value or use print() to return **None** value (null or nil used in other languages).



```python
def fib(n):     # Fibonacci function
    x, y = 0, 1
    fib_list = []
    while x < n:
        fib_list.append(x)
        x, y = y, x+y
    return(fib_list)
```

Call the function to test it



```
print(fib(0))
```

```
[]
```

```
print(fib(100))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

```
print(fib(2000))
```

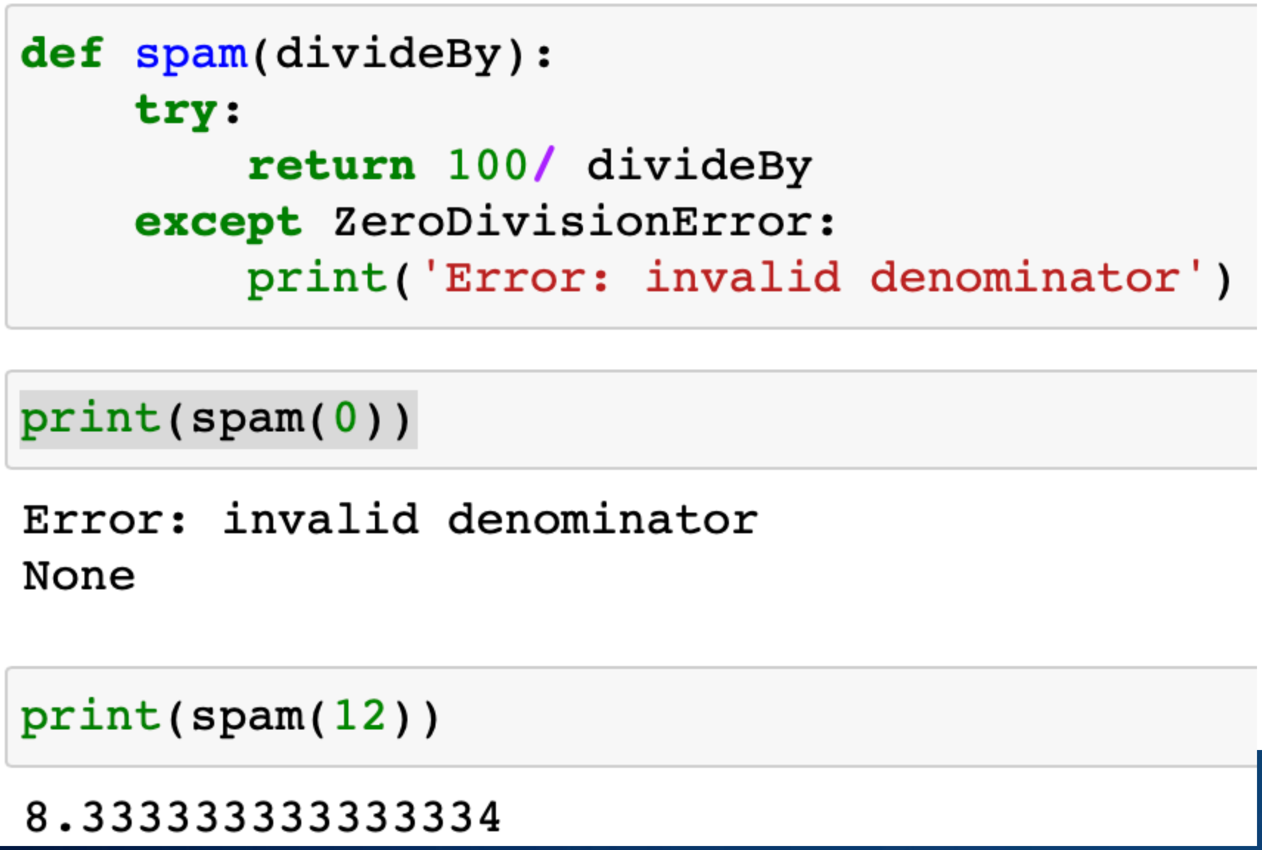
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597]
```

Local and Global Scope

- Local variables are parameters or variables that are used in a local scope, e.g., *x*, *y* and *n* used in *fib* are said to exist in that function’s local scope.
- Otherwise, they are global variables.
- Local variables cannot be used in the global scope or in other local scopes.
- Global ones can be read from a local scope.

Exception Handling

- An error, or exception, can crash the entire program.
- Errors can be handled with try and except statements.



```
def spam(divideBy):
    try:
        return 100/ divideBy
    except ZeroDivisionError:
        print('Error: invalid denominator')
```

```
print(spam(0))
```

```
Error: invalid denominator
None
```

```
print(spam(12))
```

```
8.333333333333334
```

Professor Yuefeng Li
School of Computer Science

5. Dictionaries, Structuring Data & Classes

- A dictionary is a *set* of ***key: value*** pairs, with the requirement that the keys are unique (within one dictionary).
- A pair of braces creates an empty dictionary: {}.
- Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.
- Unlike list, items in dictionaries are *unordered*. So, **dictionaries cannot be sliced like lists**.

```
>>> spam = ['cats', 'dog', 'moose']
>>> bacon= ['dog', 'moose', 'cats']
>>> spam==bacon
False
>>> dspam = {'cats':2, 'dog':3, 'moose':10}
>>> dbacon= {'dog':3, 'moose':10, 'cats':2}
>>> dspam == dbacon
True
```

Review Methods for List objects

- list.append(*x*) Add an item to the end of the list.
- list.extend(*iterable*) Extend the list by appending all the items from the *iterable*.
- list.insert(*i*, *x*) Insert an item at a given position. The first argument is the index of the element before which to insert.
- list.remove(*x*) Remove the first item from the list whose value is equal to *x*. It raises a ValueError if there is no such item.
- list.pop([*i*]) Remove the item at the given position in the list and return it.
- list.clear() Remove all items from the list. Equivalent to del a[:].
- list.index(*x*[, *start*[, *end*]]) Return zero-based index in the list of the first item whose value is equal to *x*. Raises a ValueError if there is no such item.
- list.count(*x*) Return the number of times *x* appears in the list.
- list.sort(*key=None*, *reverse=False*) Sort the items of the list in place.
- list.reverse() Reverse the elements of the list in place.
- list.copy() Return a shallow copy of the list. Equivalent to a[:].

List Comprehensions

- List comprehensions provide a concise way to create lists.
- The following is an example to create a list: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```
squares = []
for x in range(10):
        squares.append(x**2)
```

- OR equivalently (list comprehension):

```
squares = [x**2 for x in range(10)]
```

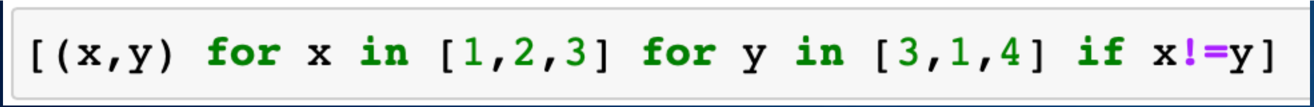
- The following program combines the elements of two lists x and y if they are not equal:

```
combs = []
for x in [1,2,3]:
        for y in [3,1,4]:
                if x != y:
                        combs.append((x, y))
```

- **Write** a list comprehension to create combs using two list x = [1,2,3] and y = [3,1,4] as follows:

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

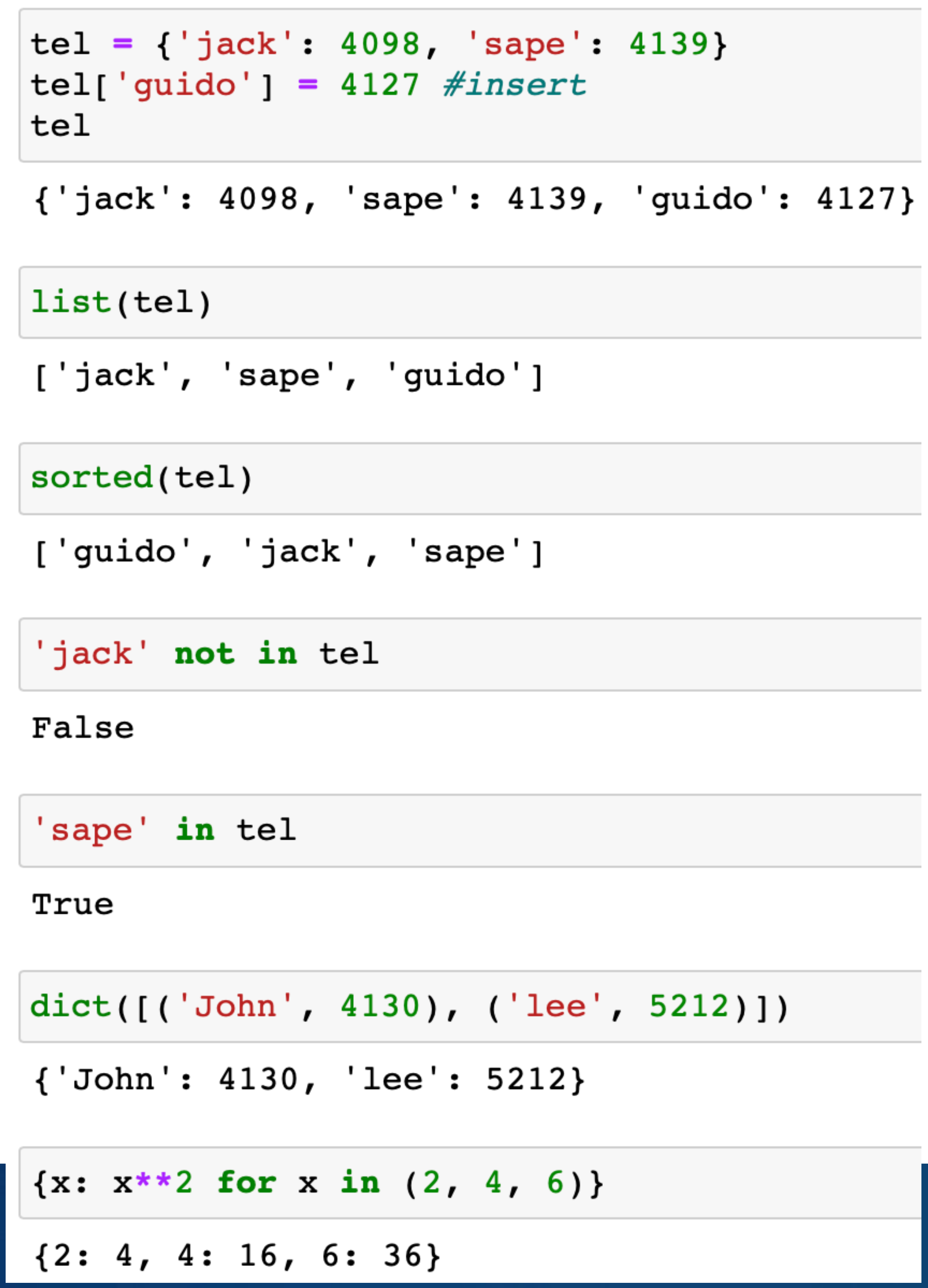
Professor Yuefeng Li
School of Computer Science



```
[(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]
```

Main operations on a dictionary

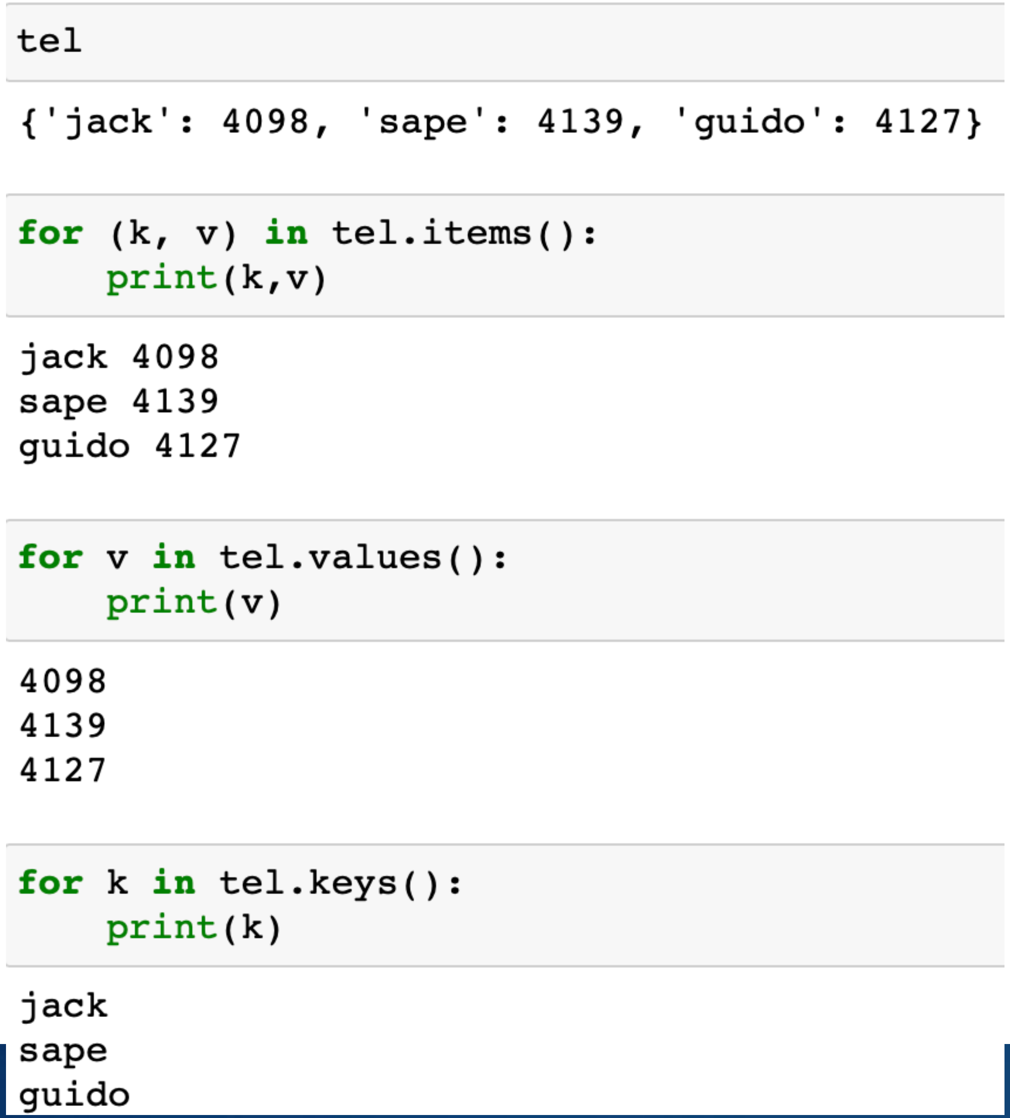
- They are storing a value with some key and extracting the value given the key.
- It is also possible to delete a key:value pair with **del**.
- If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.
- The dict() constructor builds dictionaries directly from sequences of key-value pairs.
- In addition, **dict** comprehensions can be used to create dictionaries from arbitrary key and value expressions.



```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127 #insert
tel
```
```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```
```
list(tel)
```
```
['jack', 'sape', 'guido']
```
```
sorted(tel)
```
```
['guido', 'jack', 'sape']
```
```
'jack' not in tel
```
```
False
```
```
'sape' in tel
```
```
True
```
```
dict([('John', 4130), ('lee', 5212)])
```
```
{'John': 4130, 'lee': 5212}
```
```
{x: x**2 for x in (2, 4, 6)}
```
```
{2: 4, 4: 16, 6: 36}
```

keys(), values() and items() methods

- When looping through dictionaries, the key and corresponding value can be retrieved at the same time using these methods.
  - items()
  - values()
  - keys()



```
tel
```

```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```

```
for (k, v) in tel.items():
    print(k,v)
```

```
jack 4098
sape 4139
guido 4127
```

```
for v in tel.values():
    print(v)
```

```
4098
4139
4127
```

```
for k in tel.keys():
    print(k)
```

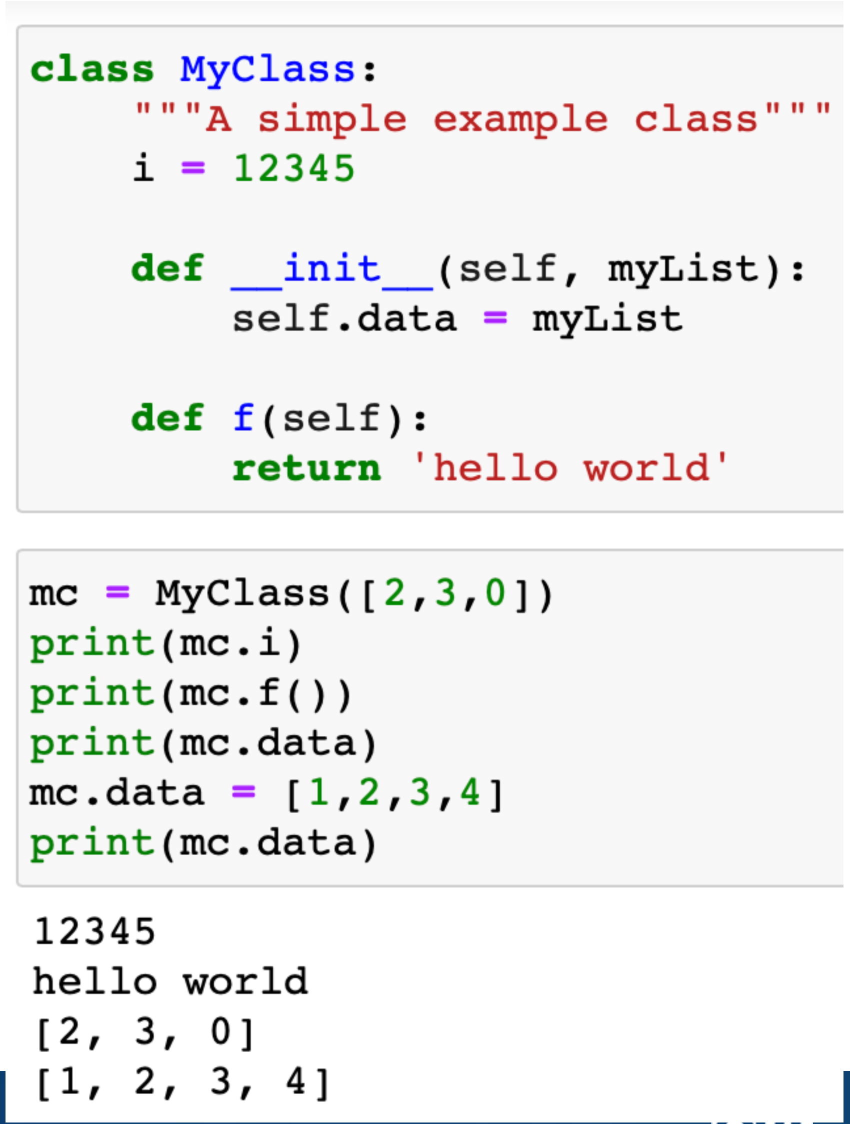
```
jack
sape
guido
```

Classes

- Classes provide a way of bundling data and functionality together.
- Python classes provide all the standard features of Object-Oriented Programming
- Class Definition Syntax

```
class ClassName:
        <statement-1>
        . . .
        <statement-N>
```

- Class objects support two kinds of operations: attribute references and instantiation.
  - Attribute references use the standard syntax: *obj.name*
  - The instantiation creates an empty object.
  - A specific initial state, define a special method named __init__().



```python
class MyClass:
    """A simple example class"""
    i = 12345

    def __init__(self, myList):
        self.data = myList

    def f(self):
        return 'hello world'
```

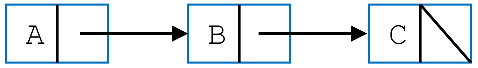
```python
mc = MyClass([2,3,0])
print(mc.i)
print(mc.f())
print(mc.data)
mc.data = [1,2,3,4]
print(mc.data)
```

```
12345
hello world
[2, 3, 0]
[1, 2, 3, 4]
```

Example of Classes

Linked List

- Define a **Node** class that includes two attributes: *data* and *next*
- Class **Linkedlist**
  - Attribute: *head*
  - Mthods: *insert*, *lprint*
- Please use the two classes to create a linked list:



Then print the list as

A --> B --> C

```
class Node:
    def __init__(self, data, next=None):
        self.data=data
        self.next=next
```

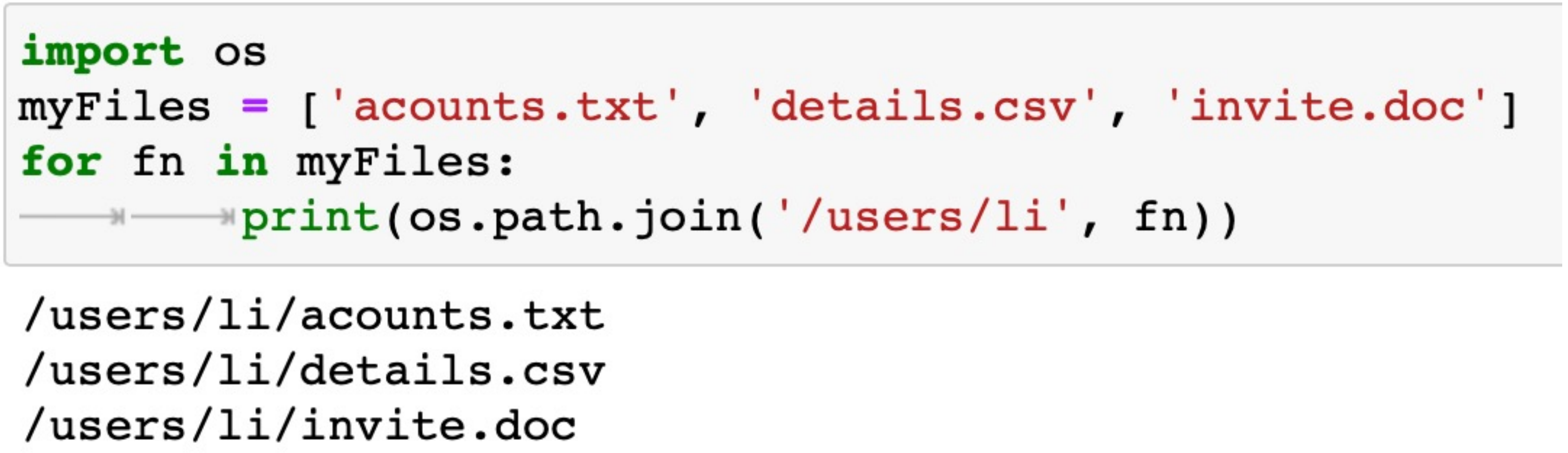
```
class Linkedlist:
    def __init__(self, hnode):
        self.head=hnode
    def insert(self, nnode):
        if self.head != None:
            p = self.head
            while p.next != None:
                p=p.next
            p.next=nnode
    def lprint(self):
        if self.head != None:
            p = self.head
            while p!= None:
                print(p.data,end =" " )
                if p.next != None:
                    print ('-->', end=" ")
                p=p.next
        else:
            print('The list is empty!')
```

```
myList = Linkedlist(Node("A", None))
myList.insert(Node('B', None))
myList.insert(Node('C', None))
myList.lprint()
```

```
A --> B --> C
```

6. Reading and Writing Files

- A file has two key properties: a filename and a path (root in window is c:\, and on OS X or Linux, the root is /)
- Creating string for filenames
  - import os
  - os.path.join(‘usr’, ‘bin’, ‘spam’)
  - In window, it will be *‘usr\\bin\\spam’* # backslash needs to be escaped; in OS X or Linux, it will be *‘usr/bin/spam’*
- Example



```python
import os
myFiles = ['acounts.txt', 'details.csv', 'invite.doc']
for fn in myFiles:
        print(os.path.join('/users/li', fn))
```

```
/users/li/acounts.txt
/users/li/details.csv
/users/li/invite.doc
```

os.path Module

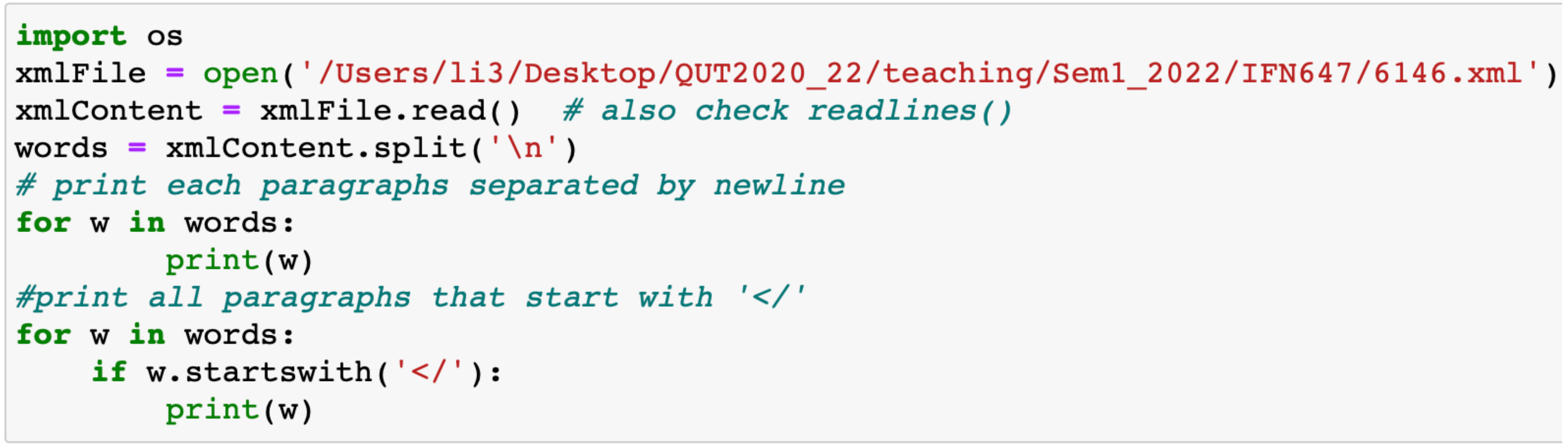
- Get and change the current working directory

```
>>> os.getcwd()
'/Users/li3/Desktop/2020/Python code'
>>> os.chdir ('/Users/li3/Desktop')
>>> os.getcwd()
'/Users/li3/Desktop’
```

- Other commands
  - Os.makedirs()
  - Os.path.abspath()
  - Os.path.getsize()
  - Os.path.exists()

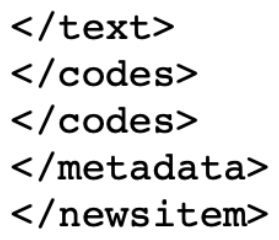
Open and Read files

- Open() function returns a File object (Close() is used to close the file)
- import glob #open(file_) for file_ in glob.glob("*.xml")
- Read() or write() methods on the File object.



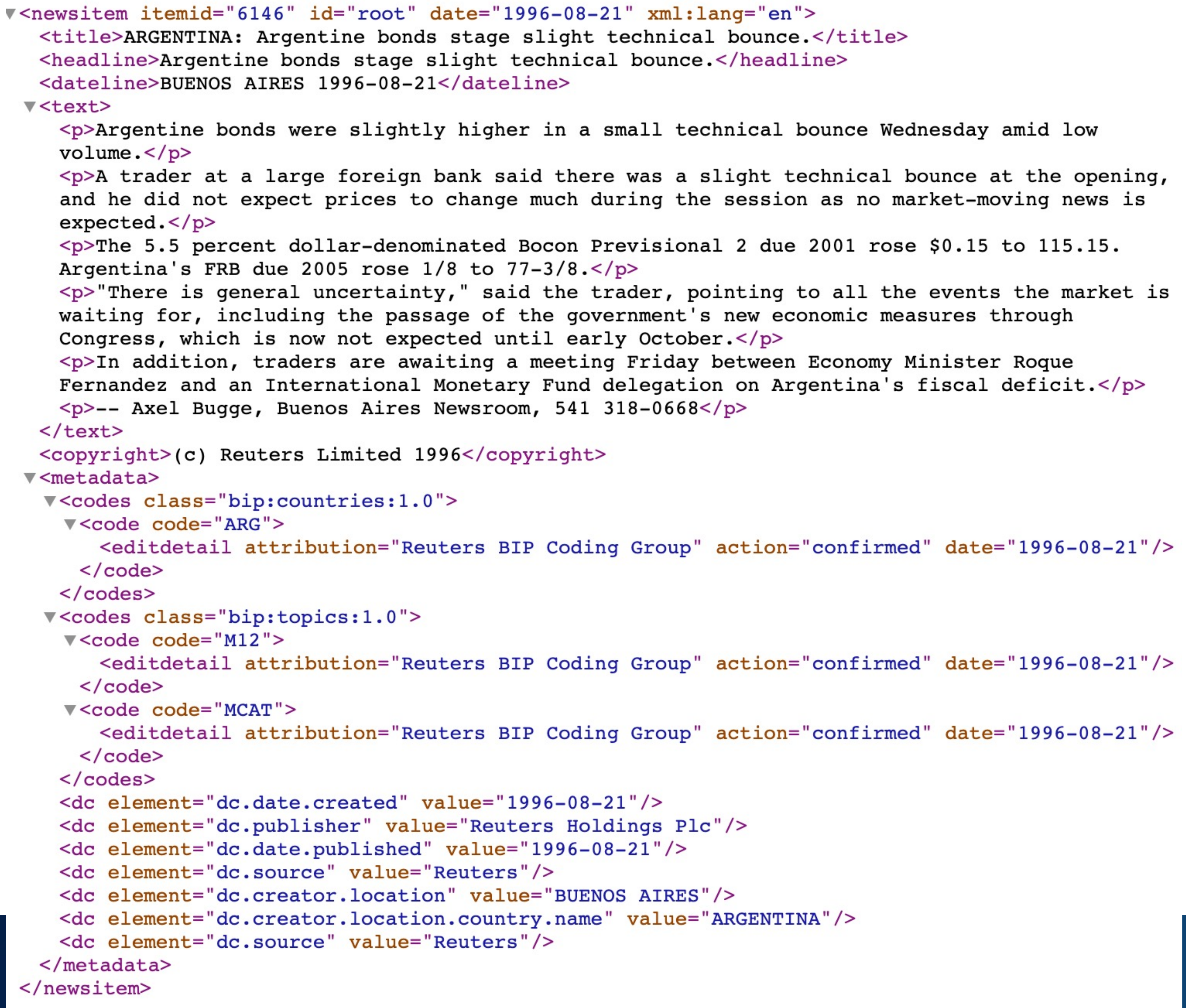
```python
import os
xmlFile = open('/Users/li3/Desktop/QUT2020_22/teaching/Sem1_2022/IFN647/6146.xml')
xmlContent = xmlFile.read()  # also check readlines()
words = xmlContent.split('\n')
# print each paragraphs separated by newline
for w in words:
        print(w)
#print all paragraphs that start with '</'
for w in words:
    if w.startswith('</'):
        print(w)
```

All paragraphs that start with '</'



```
</text>
</codes>
</codes>
</metadata>
</newsitem>
```

Example File 6146.xml



```
▼<newsitem itemid="6146" id="root" date="1996-08-21" xml:lang="en">
   <title>ARGENTINA: Argentine bonds stage slight technical bounce.</title>
   <headline>Argentine bonds stage slight technical bounce.</headline>
   <dateline>BUENOS AIRES 1996-08-21</dateline>
  ▼<text>
     <p>Argentine bonds were slightly higher in a small technical bounce Wednesday amid low
     volume.</p>
     <p>A trader at a large foreign bank said there was a slight technical bounce at the opening,
     and he did not expect prices to change much during the session as no market-moving news is
     expected.</p>
     <p>The 5.5 percent dollar-denominated Bocon Previsional 2 due 2001 rose $0.15 to 115.15.
     Argentina's FRB due 2005 rose 1/8 to 77-3/8.</p>
     <p>"There is general uncertainty," said the trader, pointing to all the events the market is
     waiting for, including the passage of the government's new economic measures through
     Congress, which is now not expected until early October.</p>
     <p>In addition, traders are awaiting a meeting Friday between Economy Minister Roque
     Fernandez and an International Monetary Fund delegation on Argentina's fiscal deficit.</p>
     <p>-- Axel Bugge, Buenos Aires Newsroom, 541 318-0668</p>
   </text>
   <copyright>(c) Reuters Limited 1996</copyright>
  ▼<metadata>
    ▼<codes class="bip:countries:1.0">
      ▼<code code="ARG">
         <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
       </code>
     </codes>
    ▼<codes class="bip:topics:1.0">
      ▼<code code="M12">
         <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
       </code>
      ▼<code code="MCAT">
         <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
       </code>
     </codes>
     <dc element="dc.date.created" value="1996-08-21"/>
     <dc element="dc.publisher" value="Reuters Holdings Plc"/>
     <dc element="dc.date.published" value="1996-08-21"/>
     <dc element="dc.source" value="Reuters"/>
     <dc element="dc.creator.location" value="BUENOS AIRES"/>
     <dc element="dc.creator.location.country.name" value="ARGENTINA"/>
     <dc element="dc.source" value="Reuters"/>
   </metadata>
</newsitem>
```

Paragraphs separated by newline

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<newsitem itemid="6146" id="root" date="1996-08-21" xml:lang="en">
<title>ARGENTINA: Argentine bonds stage slight technical bounce.</title>
<headline>Argentine bonds stage slight technical bounce.</headline>
<dateline>BUENOS AIRES 1996-08-21</dateline>
<text>
<p>Argentine bonds were slightly higher in a small technical bounce Wednesday amid low volume.
</p>
<p>A trader at a large foreign bank said there was a slight technical bounce at the opening, a
nd he did not expect prices to change much during the session as no market-moving news is expe
cted.</p>
<p>The 5.5 percent dollar-denominated Bocon Previsional 2 due 2001 rose $0.15 to 115.15. Argen
tina's FRB due 2005 rose 1/8 to 77-3/8.</p>
<p>&quot;There is general uncertainty,&quot; said the trader, pointing to all the events the m
arket is waiting for, including the passage of the government's new economic measures through
Congress, which is now not expected until early October.</p>
<p>In addition, traders are awaiting a meeting Friday between Economy Minister Roque Fernandez
and an International Monetary Fund delegation on Argentina's fiscal deficit.</p>
<p>-- Axel Bugge, Buenos Aires Newsroom, 541 318-0668</p>
</text>
<copyright>(c) Reuters Limited 1996</copyright>
<metadata>
<codes class="bip:countries:1.0">
  <code code="ARG">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
  </code>
</codes>
<codes class="bip:topics:1.0">
  <code code="M12">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
  </code>
  <code code="MCAT">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-08-21"/>
  </code>
</codes>
<dc element="dc.date.created" value="1996-08-21"/>
<dc element="dc.publisher" value="Reuters Holdings Plc"/>
<dc element="dc.date.published" value="1996-08-21"/>
<dc element="dc.source" value="Reuters"/>
<dc element="dc.creator.location" value="BUENOS AIRES"/>
<dc element="dc.creator.location.country.name" value="ARGENTINA"/>
<dc element="dc.source" value="Reuters"/>
</metadata>
</newsitem>
```

7. Web Scraping

- The webbrowser module comes with Python and opens a browser to a page.

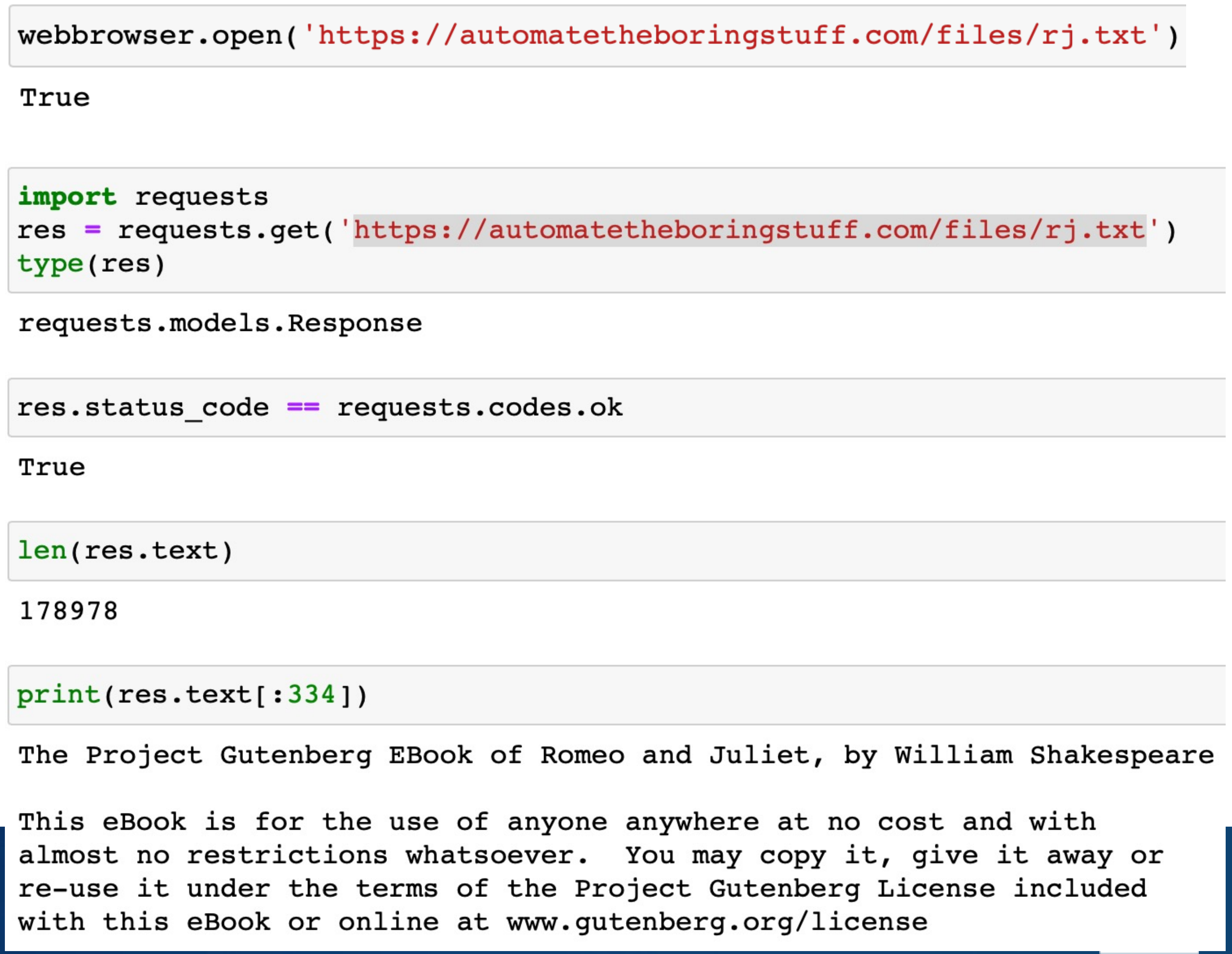
```
>>> import webbrowser
>>> webbrowser.open('http://inventwithpython.com/')
True
```

- Download a Web page
- install requests module
  - pip install requests
- Requests.get() function
- Raise_for_status() is used to
  stop if an error happens

Invent with Python

Learn to Program. For Free.

Learning to program makes you a smarter and more capable human being.



```
webbrowser.open('https://automatetheboringstuff.com/files/rj.txt')
```

```
True
```

```
import requests
res = requests.get('https://automatetheboringstuff.com/files/rj.txt')
type(res)
```

```
requests.models.Response
```

```
res.status_code == requests.codes.ok
```

```
True
```

```
len(res.text)
```

```
178978
```

```
print(res.text[:334])
```

```
The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org/license
```

Processing HTML

```
import requests, bs4
res = requests.get('https://nostarch.com/')
res.raise_for_status()
noStarchSoup = bs4.BeautifulSoup(res.text)
metas = noStarchSoup.select('meta')
titles = noStarchSoup.select('title')
print(titles)
#print(metas)
```

```
[<title>No Starch Press | "The finest in geek entertainment"</title>]
```

```
type(noStarchSoup)
```

```
bs4.BeautifulSoup
```

```
print(res.text[:200])
```

```
<!DOCTYPE html>
<html lang="en" dir="ltr" xmlns:og="http://ogp.me/ns#">
<head>
<script src="/cdn-cgi/apps/head/j5v88GAcO1Pymf91CQYvgLZqNao.js"></script><link rel="profile" h
ref="https://www.w3.org/199
```

```
ef = open('example.html')
es = bs4.BeautifulSoup(ef)
authors = es.select('#author')
print(authors)
```

```
[<span id="author"> Y Li </span>]
```

References

[1] https://docs.python.org/3/tutorial/

[2] Al Sweigart, Automate the boring stuff with Python, 2015

[3] K. A. Lambert, Foundations of Python: data structures, Cengage Learning PTR, 2014.