Data Foundations: Python 101 Part II

Instructor: Anthony Rios

Outline

Introduction and Review

Using Python as a Programming Language
- Introduction
- Loops and Iteration
- Dictionaries

Interacting with the Outside World
- Reading from a File

Introduction and Review

Using Python as a Programming Language
- Introduction
- Loops and Iteration
- Dictionaries

Interacting with the Outside World
- Reading from a File

Covered Readings

Python for Everybody: Exploring Data In Python 3
by Charles Russell Severance
Free URL: **https://www.py4e.com/book**

Chapters covered this week: 2, 3, **5**, 6, **7**, and **8**!

Do not panic! Stop me and ask questions!

Homework 1 has been posted! Due **Wednesday**, next week.

Short Quiz (10 minutes max) on Wednesday.

Bring a pen/pencil.

Exercise 00

1. Download the latest in-class exercises from Blackboard
2. Open it in Jupyter Lab/Notebook.


3 minutes

Review

Mathematical Expressions		
Python	Math	English
$a + b$	$a + b$	Addition
$a - b$	$a - b$	Subtraction
$a * b$	$a \times b$	Multiplication
a/b	$a \div b$	Division
$a * *b$	a^b	Exponentiation
$a\%b$	$a \bmod b$	Modulo/Remainder after division

Boolean Expressions		
Python	Mathematics	Meaning
$x == y$	$x = y$	Equality
$x != y$	$x \neq y$	Not Equal
$x < y$	$x < y$	Less Than
$x <= y$	$x \leq y$	Less Than or Equal To
$x > y$	$x > y$	Greater Than
$x >= y$	$x \geq y$	Greater Than or Equal To

Review

example.py

```
myVar = 8
if 4 + 4 == myVar:
        print('4 + 4 = {}'.format(4+4))
else:
        print('Will never run this line')
```

anthony@MacBook:~$ python example.py

```
4 + 4 = 8
```

Introduction and Review

Using Python as a Programming Language
- Introduction
- Loops and Iteration
- Dictionaries

Interacting with the Outside World
- Reading from a File

for <loop variable> **in** <string>:

Loop Body

If the string has **length n**, then the loop body is executed **n times**.

Traversing a String Character-by-Character

example.py

```
my_string = 'abcd'
for c in my_string:          # c is a named variable
    print(c)
```

anthony@MacBook:~$ python example.py

```
a
b
c
d
```

for <loop variable> **in** <string>:

Loop Body

If the string has **length n**, then the loop body is executed **n times**.

Lists

The list type is a container that holds a **number of other objects, in a given order**.

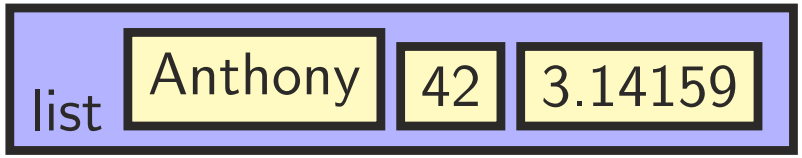
```
>>> my_list = []          # initialize an empty list
>>> name = ['Anthony', 'Michael', 'Rios']
>>> address = [123, 'sesame street']

>>> my_list.append('new element')          # append a new element
>>> my_list
['new element']

>>> my_list.append(3.14159)
>>> my_list
['new element', 3.14159]
```

Lists

```
>>> my_list = ['Anthony', 42, 3.14159]
```




```
>>>  my_list[0]          # List indexing starts at 0
Anthony

>>>  my_list[2]
3.14159

>>>  my_list[1:3]        # return list elements form start:end-1
[42, 3.14159]

>>>  my_list[0:2]
['Anthony', 42]
```

Note: A **string** can be interpreted as a **list of characters**.

Loops and Iteration

example.py

```
for x in y:
    # Loop Body
```

1. Let x = y[0] and then execute the loop body
2. Let x = y[1] and then execute the loop body
3. Let x = y[2] and then execute the loop body

   ....

4. Let x = y[n-1] and then execute the loop body

Lists: Range

the range(n) function returns a list of n elements.

>>> range(3) **# by default, elements are ordered starting at 0**
[0,1,2]

Basic format: range([start], stop[, step])

>>> list(range(3, 10, 2)) **# will return a "generator", need to cast to list()**
[3,5,7,9]

The for-loop Revisited: Step-by-Step

example.py

```
for i in range(10):
        if i % 2 == 1: # i == 0, remainder 0
                print('{} is an odd number.'.format(i))
```

```
anthony@MacBook:~$ python example.py
```

The for-loop Revisited: Step-by-Step

example.py

```
for i in range(10):
        if i % 2 == 1: # i == 1 remainder 1
                print('{} is an odd number.'.format(i))
```

anthony@MacBook:~$ python example.py

```
1 is an odd number
```

The for-loop Revisited: Step-by-Step

example.py

```
for i in range(10):
        if i % 2 == 1: # i == 2 remainder 0
                print('{} is an odd number.'.format(i))
```

anthony@MacBook:~$ python example.py

```
1 is an odd number
```

The for-loop Revisited: Step-by-Step

example.py

```
for i in range(10):
        if i % 2 == 1: # i == 3 remainder 1
                print('{} is an odd number.'.format(i))
```

anthony@MacBook:~$ python example.py

```
1 is an odd number
3 is an odd number
```

The for-loop Revisited: Step-by-Step

example.py

```
for i in range(10):
        if i % 2 == 1: # i == 10 remainder 0
                print('{} is an odd number.'.format(i))
```

anthony@MacBook:~$ python example.py

```
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.
```

Exercise 1: Putting it all together

Write a program that does the following:

- Prints the number 1 to 100
- For the numbers 10 to 25 (including 10 and 25), instead of printing the number, print the word “cheese”.
- For number 55 to 100 (including 55, but not including 100), instead of printing the number, print the word “cake”.
- For number 100, print the word ”Done!”


10 minutes

Open-Ended Iteration

- So far, we have only addressed iterative problems in which we know (in advance) the **required number of repetitions**.
- Not all iteration problems are like that.
- Some iteration problems are open-ended
- Stir for **5 minutes** vs Stir **until fluffy**.

While-loop Mechanics

while boolean-expression:

Loop Body

- The while loop will continue until the "boolean-expression" is False.
- The while loop **could repeat forever!**

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # "cnt += 1" is equivalent to "cnt = cnt + 1"
        accumulator += cnt # accumulator = accumulator + cnt
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
The count is 1
The count is 2
The count is 3
The count is 4
accumulator: 15
```

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # 0 to 1
        accumulator += cnt # 0 to 1
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
```

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # 1 to 2
        accumulator += cnt # 1 to 3
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
The count is 1
```

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # 2 to 3
        accumulator += cnt # 3 to 6
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
The count is 1
The count is 2
```

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # 4 to 5
        accumulator += cnt # 10 to 15
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
The count is 1
The count is 2
The count is 3
```

The While-Loop

example.py

```
cnt = 0
accumulator = 0
while cnt < 5:
        print("The count is {}".format(cnt))
        cnt += 1 # 3 to 4
        accumulator += cnt # 6 to 10
print('accumulator: {}'.format(accumulator))
```

anthony@MacBook:~$ python example.py

```
The count is 0
The count is 1
The count is 2
The count is 3
The count is 4
```

The Infinite Loop

example.py

```
print("keeps going")
while True: # Will never be False
        print("and going")
```

anthony@MacBook:~$ python example.py

```
keeps going
and going
and going
and going
and going
...
```

Take input from the user

The **input()** function **prompts** the user to type something in the terminal.

```
>>> name = input("What is your name?")
What is your name? Anthony
>>> name
'Anthony'
```

```
>>> age = input("What is your age?")
What is your age? 102
>>> age
'102'
```

In Python 3.x, **input()** will always be a **string**. Cast to float with float() or int with int().

```
>>> age = int(age)
>>> age
102
```

Catching exceptions with try/except

Two types of errors:

- **Syntax Errors** (missing ":" after True)

```
>>> while True print('Hello World!')
File "<stdin>", line 1
while True print('Hello world')
                ^
SyntaxError: invalid syntax
```

- **Exceptions**

```
>>> name = 'Anthony'
>>> float(name)
File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'Anthony'
```

Catching exceptions with try/except

example.py

```
try:
        number = input("Please enter a number:")
        number = float(number) # cast from string to float
        print('You entered: {}'.format(number))
except:
        print('Oops!  That is not a number')
```

anthony@MacBook:~$ python example.py

```
Please enter a number: Anthony
Oops!  That is not a number
```

Python Standard Exceptions

- **Exception** – Base class for all exceptions.
- **ZeroDivisionError** – Raised when division or modulo by zero takes place.
- **ValueError** – Raised when built-in function for a specic data type is passed the wrong type (i.e., needs int, but given string)
- **IndexError** – Raised when an index is not found in a sequence (i.e., running myList[10] on a list with only 3 elements).

Catching exceptions with try/except

example.py

```
try:
        number = input("Please enter a number:")
        number = float(number) # cast from string to float
        print('You entered: {}'.format(number))
except ValueError:
        print('Oops!  That is not a number')
```

anthony@MacBook:~$ python example.py

```
Please enter a number: Anthony
Oops!  That is not a number
```

Breaking the Loop

example.py

```
sum = 0
while True:
        try:
                number = input('Enter a number:')
                number = float(number)
                sum += number
        except:
                print('"{}" is not a number!'.format(number))
                break # the break keyword will exit a loop
print('Sum: {}'.format(sum))
```

anthony@MacBook:~$ python example.py

```
Enter a number: 3
Enter a number: 7
Enter a number: Cake
"Cake" is not a number!
Sum: 10
```

Exercise 2

Write a program which repeatedly reads numbers until the user enters "done". Once "done" is entered, print out the total, count, and average of the numbers. If the user enters anything other than a number, detect their mistake using try and except and print an error message and skip to the next number.

```
anthony@MacBook:~$ python example.py
Enter a number: 4
Enter a number: 5
Enter a number: bad data
Invalid input
Enter a number: 7
Enter a number: done
16 3 5.33333333333
```


15 minutes

Dictionaries

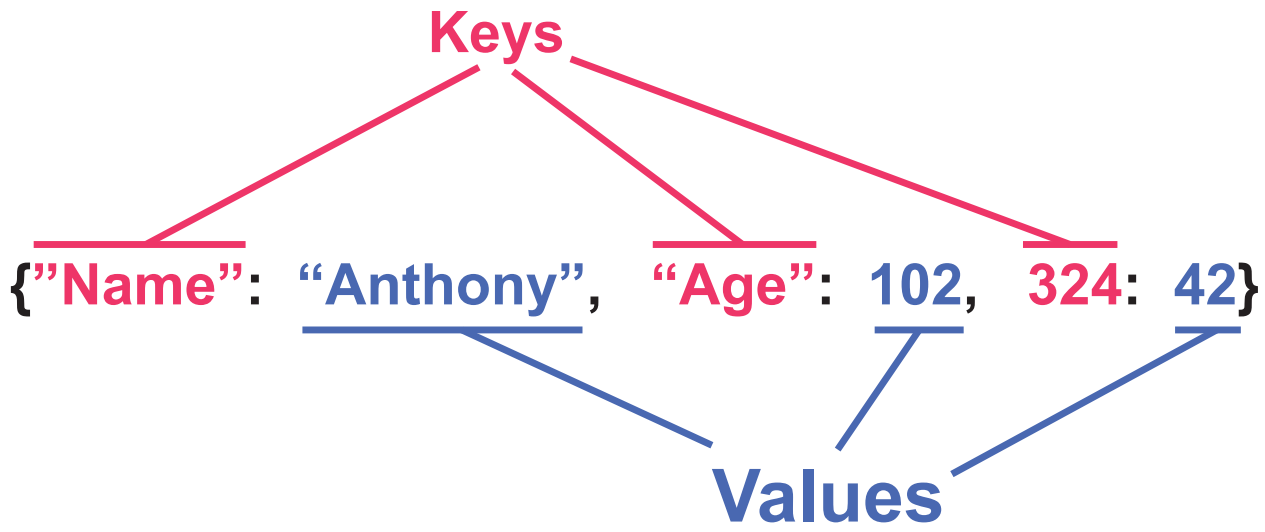
A dictionary is **like a list**, but more general.

- For a list, index positions **must be integers**.
- For a dictionary, indices can be (almost) **anything**!
- dictionaries have **no** particular order.

Keys	Values
'Name'	'Anthony'
'Age'	102
'key'	0
2018	['a','b','c']

>>> myVar = {'Name': 'Anthony', 'Age': 102, 'key': 0, 2018: ['a','b','c']}

Dictionaries




```
>>> myVar = {"Name": "Anthony", "Age": 102, 324: 42}
>>> myVar
{"Name": "Anthony", "Age": 102, 324: 42}
>>> myVar["Name"]
'Anthony'
>>> myVar[324]
42
```

Dictionaries: Indexing

```
>>> myVar = {"Name": "Anthony", "Age": 102, 324: 42}
```

```
>>> myVar["weight"]
File "<stdin>", line 1, in <module>
KeyError: 'weight'
```

```
>>> myVar.get("weight", 400) # myVar.get(KEY, Default Value)
 400
```

Dictionaries: Adding and Modifying New Keys/Values

```
>>> myVar = {"Name": "Anthony", "Age": 102, 324: 42}

>>> myVar
{'Name': 'Anthony', 'Age': 102, 324: 42}

>>> myVar['weight'] = 400

>>> myVar
{'Name': 'Anthony', 'Age': 102, 324: 42, 'weight': 400}

>>> myVar['age'] = 0

>>> myVar
{'Name': 'Anthony', 'Age': 0, 324: 42, 'weight': 400}

>>> myVar['age'] += 50

>>> myVar
{'Name': 'Anthony', 'Age': 50, 324: 42, 'weight': 400}
```

Dictionaries: Testing if Key Exists

example.py

```
myVar = {'Name': 'Anthony', 'Age': 102, 324: 42}
if 'weight' in myVar:
        print('Your weight is {}.'.format(myVar['weight']))
else:
        print('I do not know your weight')
        myVar['weight'] = int(input('What is your weight?'))
print('Your weight is {}.'.format(myVar['weight']))
```

anthony@MacBook:~$ python example.py

```
I do not know your weight.
What is your weight? 1024
You weight is 1024.
```

Exercise 3

Write a program that counts how many times each letter appears in a string. The counts for each character should be stored in a dictionary where the character is the key and the value is the count.

For example, given the string **'aaabbc'**, your code should output the dictionary
{'a':3, 'b':2, 'c':1}.

We are effectively computing a **histogram**, which is a statistical term for a set of counters (or frequencies).

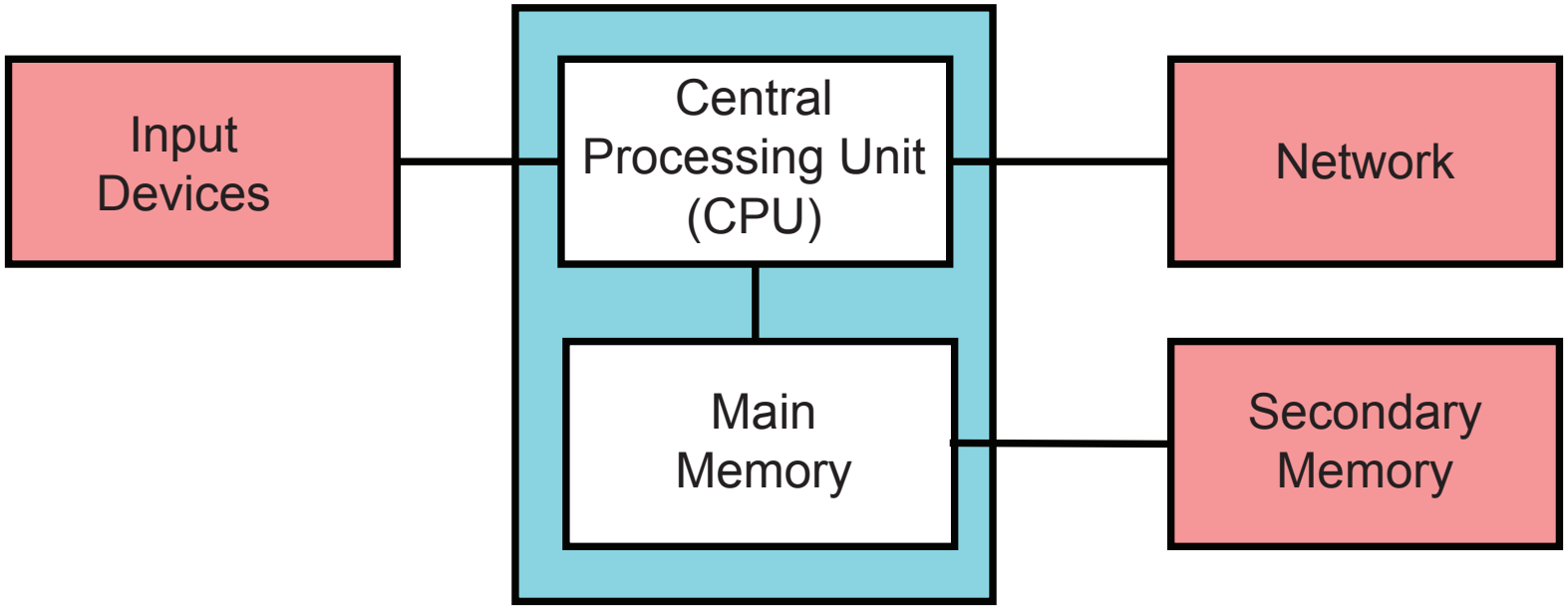

5 minutes

Introduction and Review

Using Python as a Programming Language
- Introduction
- Loops and Iteration
- Dictionaries

Interacting with the Outside World
- Reading from a File

File IO

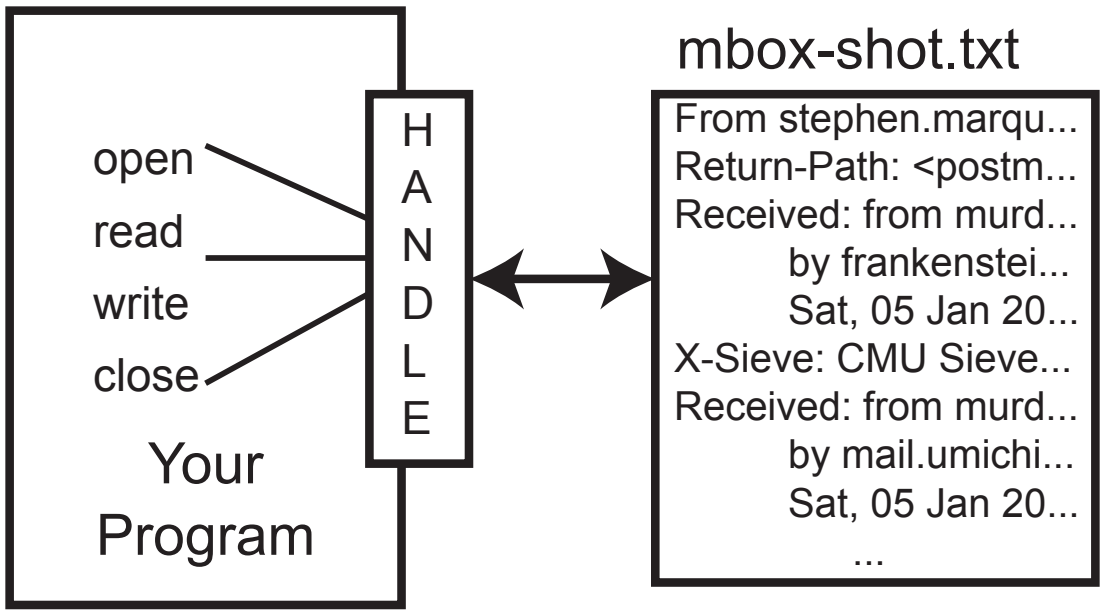


- Main memory stores **all variables**/loaded data (lists, ints, floats, strings, ...)
  - Erased when computer restarts. Applications lose access after they are closed.
- Secondary memory (hard drives, usb drives, ...)
  - Slower than main memory, but the information is not deleted when the CPU is powered off.

`https://www.youtube.com/watch?v=DKGZlaPlVLY`

Opening a File

```
>>> file_handle = open('mbox-short.txt')
>>> file_handle
<_io.TextIOWrapper name='mbox-short.txt' mode-'r' encoding='UTF-8'>
```



If successful, open returns a **file handle**. The file handle is **not** the actual data contained in the file.

File input

```
>>> file_handle = open('mbox-short.txt')

>>> file_data = file_handle.read() # Will read the entire file as a
string

>>> file_data[:31] # prints the first 31 characters of the string
'From stephen.marquard@uct.ac.za'

>>> file_handle.close() # Close the file

>>> c = file_handle.read()
File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

Strings Revisited

```
>>> '\n'
'\n'
>>> print('\n') # represents a blank line


>>> print(repr('\n'))
'\n'
```

- "\n" is the new line symbol.
- print() always adds a new line

```
>>> 'Ant' in 'Anthony' # "in" can be used to check for sub-strings
True
>>> 'Test' in 'Anthony'
False
```

Looping Over a File Line-by-Line

anthony@MacBook:~$ cat myfile.txt

```
line 1.
line 2.
line 3.
```

example.py

```
to_open = open('myfile.txt')
for line in to_open:
        print(line)
```

anthony@MacBook:~$ python example.py

```
line 1.

line 2.

line 3.

```

Looping Over a File Line-by-Line

anthony@MacBook:~$ cat myfile.txt

```
line 1.
line 2.
line 3.
```

example.py

```
to_open = open('myfile.txt')
for line in to_open:
        print(repr(line))
```

anthony@MacBook:~$ python example.py

```
line 1.\n
line 2.\n
line 3.\n
```

Looping Over a File Line-by-Line

anthony@MacBook:~$ cat myfile.txt

```
line 1.
line 2.
line 3.
```

example.py

```
to_open = open('myfile.txt')
for line in to_open:
        # .strip() removes white
        # space at the end and
        # the start of a string
        print(line.strip())
```

anthony@MacBook:~$ python example.py

```
line 1.
line 2.
line 3.
```

Exercise 4

Read the file “mbox.txt” line-by-line and calculate and print the following:

- The total number of lines in the file.
- The number of lines that contain the substring “From:”.


10 minutes