Python for Data Analysis

Andreas Weller, PhD

WTCHG - NHS

23.4.2014

Intro

Overview

Data types

- Basic materials (Wood / Stone)

Overview

Data types

- Basic materials (Wood / Stone)

Methods

- Associated activities (Wood sawing / Stone chipping)

Overview

Data types

- Basic materials (Wood / Stone)

Methods

- Associated activities (Wood sawing / Stone chipping)

Data structures

- Building structures (Wooden wall / Stone wall)

Overview

Data types

- Basic materials (Wood / Stone)

Methods

- Associated activities (Wood sawing / Stone chipping)

Data structures

- Building structures (Wooden wall / Stone wall)

Flow control

- Command builders ('Build me a 10 foot stone wall!')

Data types

Data types: numbers

- int (Integers): whole numbers
- float: floating-point numbers

Data types: numbers

- int (Integers): whole numbers
- float: floating-point numbers

Variables

- a = 4
- b = 2
- a + b
  6

Data types: Boolean

- Switch with 2 states: True or False
- Answers to a yes/no question

Data types: Boolean

- Switch with 2 states: True or False
- Answers to a yes/no question

Comparisons

- 3 == 4
  False
- 2 < 4
  True

Data types: Strings

- Strings of letters without inherent meaning
- Always in quotation marks

Data types: Strings

- Strings of letters without inherent meaning
- Always in quotation marks

String manipulation

- name = "Andreas"
- last = "Weller"
- name + last
  "AndreasWeller"
- "ll" in last
  True
- "First name: %s, last name: %s." % (name, last)
  First name: Andreas, last name: Weller.

Slicing

- Ways to retrieve data from an iterable
- Scheme: iterable [first item : item AFTER last item : step]

a	n	d	r	e	a	s
0	1	2	3	4	5	6

Slicing

- name = "Andreas"
- name[3:5]
  "re"
- name[3:]
  "reas"
- name[::2]
  "Ades"

Functions and Methods

Methods

- Methods are the verbs of Python
- Scheme: object.method(argument)

English	Python	Abstract
"The dog sleeps"	dog.sleep()	obj.method()
"The dog jumps high"	dog.jump("high")	obj.method(arg)
"The dog eats a cat"	dog.eat(cat)	obj.method(obj)

Methods II

- Methods are specific to each data type.

String methods

- name = "Andreas"
- name.upper()
  "ANDREAS"
- name.lower()
  "andreas"
- name.strip("as")
  "Andre"
- name.replace("dre", "XXX")
  "AnXXXas"

Data structures

Lists

- Lists are **ordered** arrays of objects.
- The items in a list can be of any type.
- Lists are defined in **square brackets**.

Lists

- Lists are **ordered** arrays of objects.
- The items in a list can be of any type.
- Lists are defined in **square brackets**.

List methods

- students = ["Pauline","Carme","Anthony"]
- "Andreas" in students
  False
- students[1]
  "Carme"
- students.append("Pauline")
- students
  ["Pauline","Carme","Anthony","Pauline"]
- students.count("Pauline")
  2

Dictionaries

- Dictionaries are a collection of **unordered** {key:value} pairs
- Keys need to be of type str, values can be anything
- Dictionaries defined in **curly brackets**, but selected in **square brackets**

Dictionaries

- Dictionaries are a collection of **unordered** {key:value} pairs
- Keys need to be of type str, values can be anything
- Dictionaries defined in **curly brackets**, but selected in **square brackets**

Dictionary methods

- grades = {"Pauline":2, "Carme":4, "Anthony":1}
- grades["Carme"]
  4
- grades["Reem"] = 1
- grades
  {"Pauline":2, "Reem":1, "Carme":4, "Anthony":1}
- grades.keys()
  ["Pauline","Reem","Carme","Anthony"]

Flow control

Simple scripting

- A script is simply a text file ending in ".py"
- Scripts are executed on the commandline
- 'python script.py'

```
# outputs "Hello World!"

message = "Hello␣World!"
print message
```

If statements

- Run some code **IF** a condition is True
- Code to be run is indented with tabs

```
students = 8
max_students = 5

if (students > max_students):
    print "Too␣many␣students!"
else:
    print "Right␣amount␣of␣students!"

print "Welcome␣to␣the␣class!"
```

For loops

- Run some code **FOR** each item in an iterable
- Looping code is indented with tabs

For loops

- Run some code **FOR** each item in an iterable
- Looping code is indented with tabs

```
# greet all students in class

students = ["Pauline","Carme","Anthony"]

for name in students:
    message = "Hello " + name + "!"
    print message

print "Welcome to the class!"
```

Working with files

File manipulation I

```
# read file as a list of rows
handle = open("quasar.tsv", "r")
rows = handle.readlines()
handle.close()

# print rows
for row in rows:
    print row

# write file
output = open("test.txt", "w")
output.write("This␣is␣a␣test.")
output.close()
```

File manipulation II

```
# read file as a list of rows
handle = open("quasar.tsv", "r")
rows = handle.readlines()
handle.close()

# print chromosome and position of the first sample
for row in rows:
    f = row.split("\t")

    sample = f[0]
    chrom = f[1]
    pos = f[2]

    if sample == "Q2PL1_A01_v1_T":
        print chrom, pos
```

File manipulation III

- The **WITH** statement is the best way to open files
- Python walks through the rows one by one, without loading the whole file into memory

File manipulation III

- The **WITH** statement is the best way to open files
- Python walks through the rows one by one, without loading the whole file into memory

```
# print chromosome and position of the first sample

with open("quasar.tsv") as variant_file:
    for row in variant_file:
        f = row.split("\t")

        sample = f[0]
        chrom = f[1]
        pos = f[2]

        if sample == "Q2PL1_A01_v1_T":
            print chrom, pos
```

Modules

- Many useful modules are not loaded by default.
- We can use any Python code in our script by **importing** it.
- The module exists as an object with methods to call

Modules

- Many useful modules are not loaded by default.
- We can use any Python code in our script by **importing** it.
- The module exists as an object with methods to call

```
# Use the os (operating system) module
# to read all files in a folder as a list

import os
files = os.listdir("./")

# Use the sys module to parse command line arguments

import sys
argument1 = sys.argv[1]
argument2 = sys.argv[2]
```

Functions

- Functions are abbreviations for code.
- They are define using the **def** keyword.

Functions

- Functions are abbreviations for code.
- They are define using the **def** keyword.

```
# define a function that greets the class

def greet_class(all_students):
    for student in all_students:
        print "Good␣morning␣%s!" % student

# use the function

all_students = ["Carme", "Suska"]
greet_class(all_students)
```

Functions

- Functions are abbreviations for code.
- They are define using the **def** keyword.

```
# define a function that greets the class

def greet_class(all_students):
    for student in all_students:
        print "Good␣morning␣%s!" % student
    return len(all_students)

# use the function

all_students = ["Carme", "Suska"]
student_count = greet_class(all_students)
print "I␣have␣%s␣students." % student_count
```

Functions

- Functions are abbreviations for code.
- They are define using the **def** keyword.

```
# define a function that greets the class

def greet_class(all_students):
    for student in all_students:
        print "Good morning %s!" % student
    return len(all_students)

# use the function

my_students = ["Carme", "Suska"]
student_count = greet_class(my_students)
print "I have %s students." % student_count
```