Introduction to Python
Biostatistics Computing Workshop

Adam Peterson
University of Michigan Department of Biostatistics

18 November 2015

This handout, and other reference material can be found online at github

1 Data Structures

1. lists

   ordered array

   Main data structure for Queues,Stacks,etc.

   zero indexed

```
>>list_1 = [1,2,3,4,5,"pineapple"] ## six value array
>>list_1[0] == 1 ## accesses first element of list,
### checks to see if it is the same as 1
### evaluates to True
>>len(list_1) == 5 ## evaluates to True
>>list_2 = list() ## empty list
>>list_2.append(1) ## will place the element 1 into the
0th position of the list
```

1. sets

   unordered collection

   Removes Duplicates

   same operatiosn as "Math" sets

```
>>set_1 = {1,2,3,"cats"}
>>print(set_1)
  "cats" 1 ,3 ,2  ## notice the lack of order
>> set_2 = {"dogs","canaries","cats"}
>> new_set = set_1.union(set_2)
>> print(new_set)
3, 2, 1, "cats", "canaries", "dogs"
>> print(set_1.intersection(set_2))
"cats"
```

1. dictionaries (dicts)

   associative arrays

   Great for simulating one-to-one/onto functions (small cases)

```
>>my_first_dict = {"John":"Appleseed","Banana":"Pie"}
>>print(my_first_dict["John"])
  "Appleseed"
```

2 Control Flow, Variable Declaration

All the things you do *with* your data, once you've declared them in memory

2.1 boolean operators

Python Operator	Result
&,and	Boolean 'And" operator
or	Boolean 'Or' operator
=	Assignment operator
==	Equality Operator
!=	Not equal
%	Modulus Operator

2.2 keywords

1. if statement

```
foo = True ## For all below - INDENTATION MATTER!!!
if foo: ## Interpreter will assume you mean "if foo==true"
        print "HelloWorld"
>> "HelloWorld"
if not foo:
        print "Goodbyeworld"
elif foo == 7: ## chained if statements
        print "ElloGuvner"
else:
        print "HelloWorld"
>> "HelloWorld"
```

1. while statement

```
index_var = 0
while index_var < 3:
        print index_var
        index_var +=1 ## increment index_var
>> "0"
   "1"
   "2"
```

1. def function

```
def my_first_function(parameter_argument_1):
        """
        This is a doc string that explains
        this functions purpose in the world
        this function tells the user
        whether the function is even or odd
        """
        if type(parameter_argument_1) != int:
                return "Pleaseonlyenterintarguments"
        if parameter_argument_1 % 2 == 0:
                return "Even"
        else:
                return "Odd"
```

3 classes

Python can be used as a functional language or an Object Oriented Language (OOP)

1. Classes (dicts)

   Custom data structures

   Composed of arrangement of all previous data structures/functions

```
class Student:
"""
This is a custom digital object that stores information about a student
"""
        def __init__(self,name):
                self._name = name
                self._grades = [] ## makes an empty list here

        def enter_grade(self,grade):
                """
                For entering a student's grade
                """
                self._grades.append(grade)

        def is_good_student(self):
                """
                Subjective notion of Student's worth
                ""
                if sum(self._grades)==100:
                        print "yeah!"
                else:
                        print "ehh"

                def get_name(self):
                        return self._name
```

4 The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

5 Reference Materials

1. Classes
    - Coursera
        - Python for Everybody Specialization
        - Fundamentals of Computing Specialization
2. Problem Sets
    - Codeacademy
    - hackerrank
    - Rosalind BioInformatics Problems
    - Project Euler
3. Packages/Software Worth Knowing
    - Scipy Lectures
    - Introduction to Ipython
    - Pandas
    - Statsmodels
    - scikit-learn