

# Fundamentals of Programming

## Python short course, Lecture 2

Samuel Rivera

Department of Psychology  
The Ohio State University

Fall, 2015

# Outline

## 1 Pop Quiz

## 2 Objects

## 3 Functions and Methods

Defining functions

Methods

Scope

## 4 Data Structures

Basic data types

Lists and tuples

Dictionary

Strings

## 5 Exercises

## Pop Quiz

### Objects

### Functions and Methods

Defining  
functions  
Methods  
Scope

### Data Structures

Basic data types  
Lists and tuples  
Dictionary  
Strings

### Exercises

- 1 Create a list of integers from 0 to 10 (inclusive)
- 2 Loop through the list
- 3 During loop, if the item is even display 'even' and the list value. Otherwise just display 'odd'.

## Last lecture

Oops, didn't cover 'while loops'

While loops are different from for loops. They repeat while a condition is True. See lecture 1 updated slides.

```
x = True
while x:
    print( 'This will run once')
    x = False

while True:
    print( 'This will run forever' ) # Ctrl+C quit
    x = 1
```

# Everything in Python is an object

data types, functions, etc.

## Objects

- can be assigned to a variable or passed to a function
- can have attributes or methods associated with it (or both, or neither)

Programming can be:

- *Object oriented*: defining algorithm in terms of objects and how they interact with each other
- *Structured*: programming a sequence of steps in the algorithm

# Functions

Recipes for some task

```
print(X)  
len(X)  
range(X)
```

# Defining functions

```
def funcName( arg1, arg2, arg3 ):
    # here put the code
    pass # if you want to define later
```

# Fruitful vs Non-fruitful

```
def funcName( arg1, arg2, arg3 ):  
    x = arg1 + arg2  # the value is x lost forever
```

```
def nonFruitfulFunction( arg1, arg2, arg3 ):  
    x = arg1 + arg2  
    return x # now x can be used
```



# Methods

## Functions associated with other classes of objects

Lists, for example, are a python object that has methods (functions) associated with it. To use those methods, initialize the object, then call method:

```
x = [] # define an empty list  
x.append( 1 ) # add the value 1 to end of list
```

# Methods

I can't remember all of this.

You can find out about what methods exist by using help in the interpreter.

```
x = []  
help( x )  
help( 'string' )
```

# Scope

Scope: where variables exist

Functions typically have a local scope so that variables defined inside them don't interfere with those outside the function.

```
def makeX():  
    x = 5  
  
x = 2  
makeX()    # new x inside function scope  
print(x)   # x is still 2
```

# Scope

Be careful

If you pass a list to function, editing the list inside function will edit the original list in the outer scope. That is because lists are passed by *reference*, or based on a memory address.

```
def makeX(x):  
    x[0] = 5  
  
x = [2]  
makeX(x)    # call function  
print(x)    # you changed original x
```

# Basic data types

integers, floats, booleans

# Lists

- store 'lists' or arrays of things
- Mutable data type: you can replace single parts of it

# Lists

## Initialize

```
# initialize empty list
```

```
x = []
```

```
# initialize list of 10 ones
```

```
x = [1] * 10
```

# Lists

## Access and replace

```
x = [1,2,3,4] # initialize list  
print(x[0] )  # prints the value 1
```

```
x = [1,2,3,4] # initialize list  
x[0] = 5     # replace 1 with 5  
print( x)
```



# Tuples

Like a list.

Tuples have 2 key differences from lists:

- ① *immutable*: you cannot just change individual elements.  
you have to replace the whole thing
- ② different syntax to create tuple (same to access elements)

```
# create a tuple
y = (1,2,3)

# show first entry
print( y[0] )
```

# Dictionary

Lets you build custom data types

```
# make a dictionary
y = dict()

# add a 'value' to a 'key'
y['name'] = 'bob'
y['age'] = 20

# alternatively, initialize with values
y = {'age': 20, 'name': 'bob'}
```

# Dictionary

You can iterate over dictionary entries

```
# make a dictionary
someDict = {'age': 20, 'name': 'bob'}
for k, v in someDict: # for key, value in
    print( 'key' )
    print( k )
    print( 'value' )
    print v
```

# Strings

Also immutable, so you cant replace characters.

```
# make a string  
y = 'some string'  
  
# concatenate strings  
z = 'some' + ' string'
```

# Strings

```
# initialize string of 10 spams
```

```
x = 'spam' * 10
```

```
# make space separated string of letters abcd
```

```
y = ' '.join('abcd')
```

# Strings

## Special characters

```
'\t' # tab
```

```
'\n' # new line
```

# Strings

## Format operator

```
"I am %s. This is lecture %d!" % ('sam', 21)
```

### Format symbols

- %d - integer
- %f - float
- %s - string

# Exercise 1

## Sort a list

Let us assume you have a list of integers. Make a function to sort it in ascending order without using the sort method.



## Exercise 2

### Guess my number

Create a function implementing the binary search algorithm. Have the function take one argument for the true number to guess. Assume that you give the computer a number between 0 and 1000. At each iteration of a while loop, the computer should make a guess within the valid range, check if the true number is higher, lower, or equal to the actual number, and then update the valid range for guessing.

## Exercise 3

Draw an ascii Christmas tree!

Create a function implementing the binary search algorithm. Have the function take one argument for the true number to guess. Assume that you give the computer a number between 0 and 1000. At each iteration of a while loop, the computer should make a guess within the valid range, check if the true number is higher, lower, or equal to the actual number, and then update the valid range for guessing.