Let's Learn Python!

Getting started:

1. Join the wifi network
   Network:
   Password:
2. Get Python installed
3. Start the Python interactive shell
4. Get ready to have fun!

Find these slides here:
http://www.meetup.com/PyLadies-ATX/files/

Meet your teachers:

Barbara Shaurette

The TA Team

What to expect:

- Programming fundamentals
- Lessons in Python
- Working with neighbors
- Lots of practice

Let's start with a quiz: http://pyladi.es/atx-quiz

What is programming?

★ A **computer** is a machine that **stores** and **manipulates** information

★ A **program** is a detailed set of **instructions** telling a computer exactly what to do.

Things you can do with programs:

- Make music and videos
- Make web sites
- Play games
- Automate a household chores list
- Calculate this year's taxes
- What are some other ideas?

Algorithms

97 Simple Steps to a PB&J

Is making PB&J difficult?

How many steps does it feel like?

Let's talk to Python!

Why Python?

- ★ Readable syntax
- ★ Powerful
- ★ Awesome community
- ★ Interactive shell

Python Interactive Shell

aka, the Python interpreter

- ★ Easy to use
- ★ Instant feedback

The **prompt** - prompts you to type stuff:

```
>>>
```

Numbers

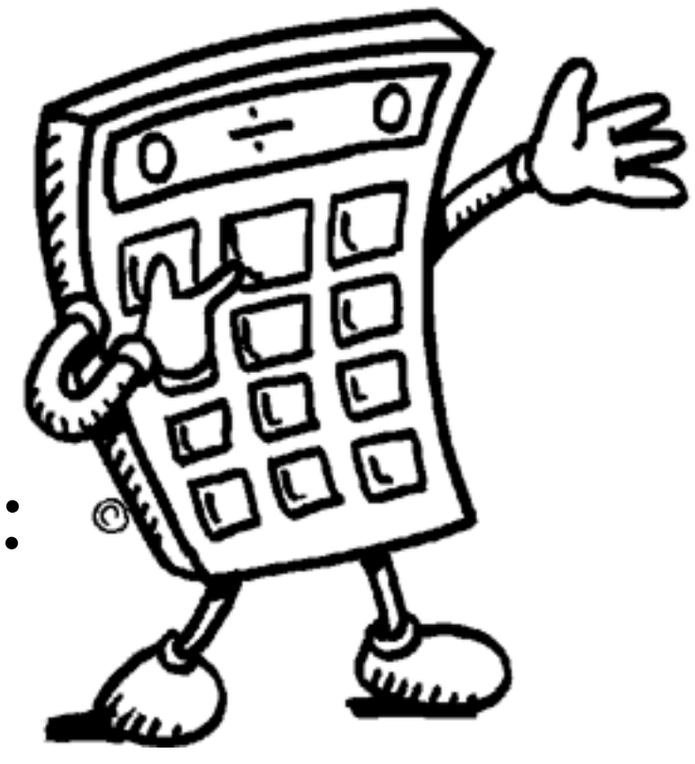
Numbers

Arithmetic operators:

addition: +

subtraction: -

multiplication: *



Try doing some math in the interpreter:

```
>>> 1 + 2
>>> 12 - 3
>>> 6 * 5
```

Numbers

Another arithmetic operator:

division: /

Try doing some division in the interpreter:

```
>>> 8 / 4
>>> 20 / 5
>>> 10 / 3
```

Is the last result what you expected?

Numbers

Integers (whole numbers):

```
9
-55
```

```
>>> 11/3
3
```

Floats (decimals):

```
17.318
10.0
```

```
>>> 11.0/3.0
3.6666666666666665
```

Rule:

★ If you want Python to respond in floats, you must talk to it in floats.

Numbers

Comparison operators:

==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Numbers

Comparison practice:

```
>>> 5 < 4 + 3
```

```
>>> 12 + 1 >= 12
```

```
>>> 16 * 2 == 32
```

```
>>> 16 != 16
```

```
>>> 5 >= 6
```

Numbers

Comparison practice:

```
>>> 5 < 4 + 3
True
>>> 12 + 1 >= 12
True
>>> 16 * 2 == 32
True
>>> 16 != 16
False
>>> 5 >= 6
False
```

Strings

Strings

```
>>> “abcdef”
>>> “garlic breath”
>>> “Thanks for coming!”
```

Try typing one without quotes:

```
>>> apple
```

What’s the result?

Strings

If it's a string, it must be in quotes.

```
>>> "apple"
>>> "What's for lunch?"
>>> "3 + 5"
```

Rules:

- ★ A string is a character, or sequence of characters (like words and sentences).
- ★ A number can be a string if it's wrapped in quotes

Strings

String operators:

concatenation (joining words together): +

multiplication: *

Try concatenating:

```
>>> “Hi” + “there!”
‘Hithere!’
```

Try multiplying:

```
>>> “HAHA” * 250
```

Strings: Indexes

Strings are made up of **characters**:

```
>>> “H” + “e” + “l” + “l” + “o”
‘Hello’
```

Each character has a position called an *index*:

```
H e l l o
0 1 2 3 4
```

In Python, indexes start at **0**

Strings: Indexes

```
>>> print “Hello”[0]
H
>>> print “Hello”[4]
o


>>> print “Hey, Bob!”[6]
o
>>> print “Hey, Bob!”[6 - 1]
B
```

Strings: Indexes

```
>>> print “Hey, Bob!”[4]
```

What did Python print?

Rules:

- ★ Each character’s position is called its *index*.
- ★ Indexes start at 0.
- ★ Spaces inside the string are counted.

The print command

print

Without print, you can concatenate with the '+' operator:

```
>>> "This" + "isn't" + "great."
Thisisn'tgreat.
```

With print, you get spaces between items:

```
>>> print "This", "is", "awesome!"
This is awesome!
```

Practicing with print

```
>>> print “Barbara has”, 2, “dogs.”
Barbara has 2 dogs.
```

```
>>> print 6+6, “eggs make a dozen.”
12 eggs make a dozen.
```

Try printing two sentences with numbers outside the quotes.

Variables

Variables

Calculate a value:

```
>>> 12 * 12
144
```

How can you save that value, 144?

Assign a name to a value:

```
>>> donuts = 12 * 12
>>> color = “yellow”
```

A **variable** is a way to *store a value*.

Variables

```
>>> donuts = 12 * 12
>>> color = “yellow”
```

Assign a <u>new</u> value:

```
>>> color = “red”
>>> donuts = 143

>>> color = “fish”
>>> color = 12
>>> color
12
```

Variables

- ★ Calculate once, keep the result to use later
- ★ Keep the name, change the value

Some other things we can do with variables:

```
>>> fruit = “watermelon”
>>> print fruit[2]
>>> number = 3
>>> print fruit[number-2]
```

Variables

Converting variables:

Turn a string into a number (use int or float).

```
>>> pets = '4'
>>> num_pets = int(pets)
```

Turn a number into a string:

```
>>> str_pets = str(num_pets)
```

Variables Practice 1

```
>>> name = “Barbara”
>>> color = “blue”
>>> print “My name is”, name, “and my
favorite color is”, color

>>> name = “Sara”
>>> color = “purple”
>>> print “My name is”, name, “and my
favorite color is”, color
```

Variables Practice 1: Answers

```
>>> print “My name is”, name, “and my
favorite color is”, color
```

Output:

```
My name is Barbara and my favorite
color is blue

My name is Sara and my favorite color
is purple
```

Variables Practice 2

```
>>> name = “Andrew”
>>> age = 30
>>> dog_year_length = 7
>>> dog_years = age * dog_year_length


>>> print name, “is”, dog_years,
    “in dog years!”
```

Variables Practice 2: Answers

```
>>> print name, “is”, dog_years,
    “in dog years!”
Andrew is 210 in dog years!
```

Variables Practice 3

Use decimal numbers if needed for precision:

```
>>> age = 32
>>> decade = 10
>>> print “I’ve lived for”,
    age/decade, “decades.”

>>> decade = 10.0
>>> print “I’ve lived for”,
    age/decade, “decades.”
```

Variables Practice 3: Answers

```
>>> print “I’ve lived for”,
    age/decade, “decades.”
I’ve lived for 3 decades.

I’ve lived for 3.2 decades.
```

Errors

Errors

```
>>> “friend” * 5
‘friendfriendfriendfriendfriend’

>>> “friend” + 5
Error

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Do you remember what ‘concatenate’ means?

What do you think ‘str’ and ‘int’ mean?

Errors

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
    ^                      ^           ^          ^
```

- Strings: ‘str’
- Integers: ‘int’
- Both are objects
- Python cannot concatenate objects of different *types*

Errors

Here's how we would fix that error:

```
>>> "friend" + 5
Error
```

Concatenation won't work.

Let's use the `print` command for display:

```
>>> print "friend", 5
friend 5
```

No concatenation, no problem!

Exercise Set 1

1. Store your name, height and favorite color in variables. Print that information in a sentence.

2. Calculate the number of 2-week disposable contact packs you need in a year and store that value in a variable. Print, in sentence form, the number of disposable contact packs you need to buy to be stocked for two years.

3. Calculate how many seconds all attendees will spend in this workshop and store it in a variable. Print the answer as a sentence.

4. Calculate the number of donuts made in a week if 15 dozen are made each day. Print, in sentence form, the number of donuts 100 people would have to eat in order to finish them all.

Exercise Set 1: Review

Store your name, height and favorite color in variables. Print that information in a sentence.

```
>>> name = “Barbara”
>>> height = “67”
>>> color = “blue”
>>> print name, “is”, height,
    “inches tall and loves”, color, “!”

Barbara is 67 inches tall and loves blue!
```

Exercise Set 1: Review

Calculate the number of 2-week disposable contact packs you need in a year and store that value in a variable.

```
>>> contact_packs = 52 / 2
```

Print out, in sentence form, the number of disposable contact packs you need to buy to be stocked for two years.

```
>>> print “I will need to buy”, contact_packs, “contact packs this year.”

I will need to buy 26 contact packs this year.
```

Exercise Set 1: Review

Calculate how many seconds all attendees will spend in this workshop.

Store that value in a variable.

```
>>> seconds = 60 * 60 * 6 * 32
```

Print the answer in a sentence.

```
>>> print “Attendees will spend a total
of”, seconds, “seconds in this workshop.”

Attendees will spend a total of 1152000
seconds in this workshop.
```

Exercise Set 1: Review

Calculate the number of donuts made in a week if 15 dozen are made each day.

```
>>> number_of_donuts = 15 * 12 * 7
```

Print, in sentence form, the number of donuts 100 people would have to eat in order to finish them all.

```
>>> print “Each person will eat”,
number_of_donuts / 100.0, “donuts.”

Each person will eat 12.6 donuts.
```

Types of data

Data types

Three types of data we already know about:

“Hi!”	string
27	integer
15.238	float

Python can tell us about types using the `type()` function:

```
>>> type(“Hi!”)
<type ‘str’>
```

Can you get Python to output `int` and `float` types?

Data type: Lists

Lists

List: a sequence of objects

```
>>> fruit = [“apple”, “banana”, “grape”]
>>> numbers = [3, 17, -4, 8.8, 1]
>>> things = [“shoes”, 85, 8.8, “ball”]
```

Guess what these will output:

```
>>> type(fruit)
>>> type(numbers)
>>> type(things)
```

Lists

Guess what these will output:

```
>>> type(fruit)
<type ‘list’>

>>> type(numbers)
<type ‘list’>

>>> type(things)
<type ‘list’>
```

Lists

Lists have indexes just like strings.

```
>>> fruit
[‘apple’, ‘banana’, ‘grape’]
```

```
>>> print fruit[0]
‘apple’
```

How would you use `type()` to verify the type of each element in the list?

Lists

Make a **list** of the four Beatles.

Use an **index** to print your favorite one's name.

Lists

Make a **list** of the four Beatles.

```
>>> beatles = [‘John’, ‘Paul’, ‘George’, ‘Ringo’]
```

Use an **index** to print your favorite one’s name.

```
>>> print beatles[2]
```

Homework

Read about other ways of managing sequences of data:

dictionaries - http://bit.ly/U3J19c
tuples - http://bit.ly/1068Drk
sets - http://bit.ly/ZoK9qK

Data type: Booleans

Booleans

A boolean value can be: `True` or `False`

Is 1 equal to 1?

```
>>> 1 == 1
True
```

Is 15 less than 5?

```
>>> 15 < 5
False
```

Booleans

What happens when we type Boolean values in the interpreter?

```
>>> True
>>> False
```

When the words ‘True’ and ‘False’ begin with capital letters, Python knows to treat them like Booleans and not strings or integers.

```
>>> true
>>> false
>>> type(True)
>>> type(“True”)
```

Booleans: Comparisons

`and`

Both sides of the expression must be True.

```
>>> True and True
>>> 1 == 1 and 2 == 2
>>> True and False
```

`or`

Only one side of the expression needs to be True.

```
>>> True or True
>>> True or False
>>> 1 == 1 or 2 != 2
```

Booleans: Reverse

`not`

When you use **`not`**:

- something `True` becomes `False`
- something `False` becomes `True`

```
>>> not 1 == 1
>>> not True
```

Booleans: Practice

Try some of these expressions in your interpreter:

```
>>> True and True
>>> False and True
>>> 1 == 1 and 2 == 1
>>>"test" == "test"
>>> 1 == 1 or 2 != 1
>>> False and False
>>> True and 1 == 1
>>> False and 0 != 0
>>> True or 1 == 1
>>>"test" == "testing"
>>> 1 != 0 and 2 == 1
>>> False or False
```

For practice later:
http://bit.ly/boolean-practice

Logic

`if` Statements

`if` Statements

Making decisions:

"**If** you're not busy, let's eat lunch now."
"**If** the trash is full, go empty it."

If a condition is met, perform the action that follows:

```
>>> name = "Jess"
>>> if name == "Jess":
...     print "Hi Jess!"

Hi Jesse!
```

`if` Statements

Adding more choices:

"**If** you're not busy, let's eat lunch now.
Or **else** we can eat in an hour."

"**If** there's mint ice cream, I'll have a scoop.
Or **else** I'll take butter pecan."

The else clause:

```
>>> if name == "Jess":
...     print "Hi Jess!"
... else:
...     print “Impostor!”
```

`if` Statements

Including many options:

"**If** you're not busy, let's eat lunch now.
Or **else if** Bob is free I will eat with Bob.
Or **else if** Judy's around we'll grab a bite.
Or **else** we can eat in an hour."

The elif clause:

```
>>> if name == "Jess":
...     print "Hi Jess!"
... elif name == “Sara”:
...     print “Hi Sara!”
... else:
...     print “Who are you?!?”
```

`if` Statements: Practice

Write an if statement that prints "Yay!" if the variable called color is equal to "yellow".

Add an elif clause and an else clause to print two different messages under other circumstances.

`if` Statements: Practice

Write an if statement that prints "Yay!" if the variable called color is equal to "yellow".

Add an elif clause and an else clause to print two different messages under other circumstances.

```
>>> color = "blue"
>>> if color == "yellow":
...     print "Yay!"
... elif color == "purple":
...     print "Try again"
... else:
...     print "We want yellow!"
```

Loops

Loops

Loops are chunks of code
that repeat a task over and over again.

★*Counting* loops repeat a certain number of times.

★*Conditional* loops keep going until a certain thing happens (or as long as some condition is True).

Loops

Counting loops repeat a certain number of times.

```
>>> for mynum in [1, 2, 3, 4, 5]:
...         print “Hello”, mynum
```

```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

The *for* keyword is used to create this kind of loop, so it is usually just called a *for loop*.

Loops

Conditional loops repeat until something happens.

```
>>> count = 0
>>> while (count < 4):
...         print 'The count is:', count
...         count = count + 1

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

The *while* keyword is used to create this kind of loop, so it is usually just called a *while loop*.

Loops: Practice

Create a list of some of your classmates' names

Loop over the list and say hello to each person.

Remember: The second line should be indented 4 spaces.

Loops: Practice

Create a list of some of your classmates' names

```
>>> names = [“Barbara”, “Jay”, “Maddy”]
```

Loop over the list and say hello to each person.

```
>>> for person in names:
...         print “Hello”, person
```

Functions

Functions

Remember our PB&J example?

Which is easier?:

1. Get bread
2. Get knife
4. Open PB
3. Put PB on knife
4. Spread PB on bread ...

1. Make PB&J

Functions are a way to *group* instructions.

Functions

What it's like in our minds:

"Make a peanut butter and jelly sandwich."

In Python, it could be expressed as:

```
make_pbj(bread, pb, jam, knife)
```

function **name** function **parameters**

Functions

Let's create a function in the interpreter:

```
>>> def say_hello(myname):
...     print 'Hello', myname
```

Remember: The second line should be indented 4 spaces.

Functions

def is the **keyword** we always use to define a function.

‘myname’ is a **parameter**.

```
>>> def say_hello(myname):
...     print ‘Hello’, myname
```

Functions

Now we'll *call* the function:

```
>>> say_hello("Katie")
Hello, Katie

>>> say_hello("Barbara")
Hello, Barbara
```

Use your new function to say
hello to some of your classmates!

Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

```
>>> def double_number(number):
...     print number * 2

>>> double_number(14)
28
```

Functions: Practice

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

```
>>> def multiply(num1, num2):
...         print num1 * num2

>>> multiply(4, 5)
20
```

Functions: Output

`print` displays something to the screen. But what if you want to save the value that results from a calculation, like your doubled number?

```
>>> def double_number(number):
...     print number * 2

>>> new_number = double_number(12)
24

>>> new_number
>>>

>>> new_number = double_number(12)
24
```

Functions: Output

```
>>> def double_number(number):
...     return number * 2
```

```
>>> new_number = double_number(12)
24

>>> new_number
24
```

Functions

Rules:

- ★ Functions are **defined** using **`def`**.
- ★ Functions are **called** using **parentheses**.
- ★ Functions take **input** and can return **output**.
- ★ `print` **displays** information, but does not give a value
- ★ `return` gives a **value** to the caller (you!)

Comments

- ★ *Comments* are used as reminders to programmers.
- ★ Computers ignore comments, but they are useful to humans.
- ★ Use # to start comments

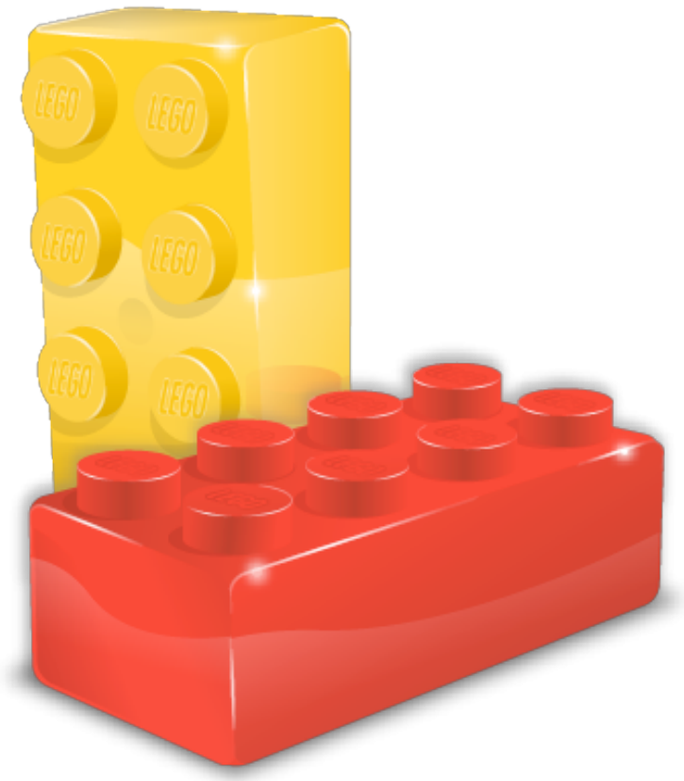
```
>>> def double_number(number):
...     # Here's where we double the number:
...     return number * 2

>>> new_number = double_number(12)
24

>>> # You can also have a comment by itself
```

Modules

Modules



A module is a block of code that can be combined with other blocks to build a program.

You can use different combinations of modules to do different jobs, just like you can combine the same LEGO blocks in many different ways.

Modules

There are lots of modules that are a part of the Python Standard Library

How to use a module:

```
>>> import random
>>> print random.randint(1, 100)

>>> import time
>>> time.time()

>>> import calendar
>>> calendar.prmonth(2013, 3)
```

Modules

A few more examples:

```
>>> import os
>>> for file in os.listdir("~/Desktop"):
...     print file

>>> import urllib
>>> myurl = urllib.urlopen('http://www.python.org')
>>> print myurl.read()
```

You can find out about other modules at: http://docs.python.org

Objects

Objects

Real objects in the real world have:

- things that you can do to them (actions)
- things that describe them (attributes or properties)

In Python:

- "things you can do" to an object are called *methods*
- "things that describe" an object are called *attributes*

Objects

This ball object might have these *attributes*:

```
myBall.color
myBall.size
myBall.weight
```

You can display them:

```
print myBall.size
```

You can assign values to them:

```
myBall.color = 'green'
```

You can assign them to attributes in other objects:

```
anotherBall.color = myBall.color
```

Objects

The ball object might have these *methods*:

```
ball.kick()
ball.throw()
ball.inflate()
```

Methods are the things you can *do* with an object.

Methods are chunks of code - *functions* - that are included *inside* the object.

Objects

In Python the description or blueprint of an object is called a *class*.

```
class Ball:

    color = ‘red’
    size = ‘small’
    direction = ‘’

    def bounce(self):
        if self.direction == ‘down’:
            self.direction == ‘up’
```

Objects

Creating an instance of an object:

```
>>> myBall = Ball()
```

Give this instance some attributes:

```
>>> myBall.direction = “down”
>>> myBall.color = “blue”
>>> myBall.size = “small”
```

Now let’s try out one of the methods:

```
>>> myBall.bounce()
```

Practice Exercises

Practice Exercises

http://codingbat.com/python

Some Simple Programs

Some Simple Programs

Log in to Appsoma: https://appsoma.com/code
Click on the 'Code' tab and type the following:

```
def greeting():
    your_name = raw_input('Type your name:')
    if your_name == 'Matt':
        print "Hi Matt!"
    else:
        print "Hello", your_name

greeting()
```

Click on the 'Save As' button and save the file as `name.py`
Click on the 'Run' button and follow the prompt.

Some Simple Programs

Close both open tabs, then click on the 'New' button and type this:

```
secret_number = 7

guess = input("What's the secret number? ")

if secret_number == guess:
    print "Yay! You got it."
else:
    print "No, that's not it."
```

Save the file as `guess.py` and click on the 'Run' button.

Some Simple Programs

Close the tab on the right, then make these changes to the game and save your file. When you finish, click the 'Run' button again:

```
from random import randint
secret_number = randint(1, 10)

while True:
    guess = input("What's the secret number? ")
    if secret_number == guess:
        print "Yay! You got it."
        break
    else:
        print "No, that's not it."
```

Some Simple Programs

Close the tab on the right, then make these changes to the game and save your file. When you finish, click the 'Run' button again:

```
from random import choice

secret_number = choice(range(1, 20))
while True:
    guess = input("What's the secret number? ")
    if secret_number == guess:
        print "Yay! You got it."
        break
    elif secret_number > guess:
        print "No, that's too low."
    else:
        print "No, that's too high."
```

Congratulations!

You're now a Pythonista!

Remember this?

Let's go back to this quiz: http://pyladi.es/atx-quiz

What do to next:

Find hack nights, local user groups, keep practicing!

PyLadies Austin
http://www.meetup.com/PyLadies-ATX/

Austin Web Python
http://www.meetup.com/austinwebpythonusergroup/

Python Web Houston
http://www.meetup.com/python-web-houston/

If there's no user group in your area, start your own!

Where to learn more:

Self-paced tutorials
http://learnpythonthehardway.org/book/
http://www.codecademy.com/tracks/python
https://www.khanacademy.org/cs/tutorials/programming-basics

Online Classes
https://www.udacity.com/course/cs101
https://www.udacity.com/course/cs253
http::/www.coursera.org
(add Google Developer Python course)

Learn more about lists and zero indexing
https://www.khanacademy.org/science/computer-science/v/python-lists
http://en.wikipedia.org/wiki/Zero-based_numbering