LISTS AND LOOPS

Shannon Turner
Twitter: @svthmc
**Github: http://github.com/
shannonturner**

HEAR ME
CODE

OBJECTIVE

- Review Lesson One

- Learn what lists are

- Learn how to add and remove items

- Learn the situations lists are useful for

- Learn how to use loops and lists together to make your programs powerful and flexible

HEAR ME
CODE

LIGHTNING REVIEW

- Variables are names that you can assign values to

- Variables can contain numbers, strings, lists, True/False, any type of information you want to store!

- Variable names can contain letters and underscores and should be descriptive (can you tell what it's used for?)

HEAR ME
CODE

LIGHTNING REVIEW

- Strings (text) can contain anything that you can type out on the keyboard

- Strings are commonly used for names, phone numbers, email addresses, other addresses, URLs, and so much more!

- Slicing is used to see parts of a string

HEAR ME
CODE

LIGHTNING REVIEW

- String methods allow you to do special actions on strings (find, replace, count, lowercase, etc)

- .format( ) is the string method we'll use most often.

```
1   name = "Shannon"
2   age = 29
3
4   print "My name is {0} and my age is {1}".format(name, age)
```

LIGHTNING REVIEW

- Conditionals allow you to change the behavior of your program

- Program behavior is based on your variables:

```python
if location == 'DC':
    print "District of Columbia"
elif location == 'MD':
    print "Maryland"
elif location == 'VA':
    print "Virginia"
else:
    print "Somewhere else"
```

HEAR ME
CODE

WE KNOW WHAT A LIST IS

- We're already very familiar with lists. We just took attendance!

- Lists are containers that can hold multiple pieces of information. Lists are commonly used to hold:

  – strings (ex: list of attendees' names)

  – numbers (ex: number of attendees for each class)

DON'T DO THIS!

- If we had to do this, it would be a pain:

- attendee1 = 'Shannon'

- attendee2 = 'Jenn'

- attendee3 = 'Grace'

- What if Jenn had to cancel? Do we have to do this for <u>every</u> attendee?

HEAR ME
CODE

CREATING A LIST

- Lists are are created by placing items inside of [ ]

- **attendees = ['Shannon', 'Jenn', 'Grace']**

- Items are separated by commas

- An empty list looks like this:
  - **people_who_didnt_do_playtime = []**

HEAR ME
CODE

HOW LONG IS MY LIST?

- **`attendees = ['Shannon', 'Jenn', 'Grace']`**

- **`print len(attendees) # 3`**

  or

- **`number_of_attendees = len(attendees)`**
- **`print number_of_attendees # 3`**

HEAR ME
CODE

REMEMBER SLICING?

- `attendees = ['Shannon', 'Jenn', 'Grace']`

- `print attendees[0] # Shannon`
- `print attendees[1] # Jenn`
- `print attendees[2] # Grace`
- `print attendees[0:2] # Shannon, Jenn`

- What happens if we `print attendees[3]` ?

HEAR ME
CODE

ADDING ITEMS TO A LIST

- `.append()` adds an item to the end of a list

- `class_sizes = []`

- `class_sizes.append(28)`
- `print class_sizes # [28]`

- `class_sizes.append(27)`
- `print class_sizes # [28, 27]`

HEAR ME CODE

CHANGING LIST ITEMS

- **`print class_sizes # [28, 27]`**

- **`# Someone showed up late to the first class, so let's change the first list item`**
- **`class_sizes[0] = 29`**

- **`print class_sizes # [29, 27]`**

HEAR ME
CODE

QUICK EXERCISE

- **`days_of_week = ['Monday', 'Tuesday']`**

- **`days_of_week.append('Wednesday')`**

- Append the rest of the days in the week, printing your progress as you go:

- **`print days_of_week`**
- **`print len(days_of_week)`**

HEAR ME
CODE

DELETING LIST ITEMS

- **`print days_of_week`** after each time you make a change to your list.

- **When in doubt, print it out!**

- There are two main ways to delete an item from a list. This one will just get rid of the list item:

- **`days_of_week.pop()`**
- **`print days_of_week`**

HEAR ME
CODE

DELETING LIST ITEMS

- **When in doubt, print it out!**

- And this one will get rid of the list item and save it into a variable:

- `day = days_of_week.pop()`
- `print day`
- `print days_of_week`

HEAR ME
CODE

DELETING LIST ITEMS

- By default, `.pop()` will remove the last item in your list.

- But you can specify a position, and it will remove that item instead.

- `day = days_of_week.pop(3)`
- `print day`
- `print days_of_week`

HEAR ME
CODE

QUICK EXERCISE

- `months = ['January', 'February']`

- `months.extend(['March', 'April' … ])`

- `.append()` adds one to the end

- `.extend()` adds many

HEAR ME
CODE

ADD/REMOVE FROM THE BEGINNING OF A LIST

- **# Remove the first month**
  months.pop(0)

- **# Insert 'January' before index 0**
  months.insert(0, 'January')

HEAR ME
CODE

STRINGS -> LISTS

- **`address = "1133 19th St NW Washington, DC 20036"`**

- **`address_as_list = address.split(" ")`**

- In this example, every time Python sees a space, it will use that to know where to split the string into a list (but you can use any character)

HEAR ME
CODE

IS THIS IN MY STRING?

- The **in** keyword allows you to check whether a (smaller) string appears within a (larger) string

- **'ann' in 'Shannon' # True**

- **'SE' in address: # False**

- This works *just a little* differently with lists.

HEAR ME
CODE

IS THIS IN MY LIST?

- The **in** keyword allows you to check: does this exact list item appear in this list?

- `'Shannon' in attendees # True`

- `'ann' in attendees # False`

- `'Frankenstein' in attendees`
  `# False` … `what a relief!`

HEAR ME
CODE

GROUP EXERCISE & LUNCH

Create a program that checks which quadrant an address belongs to.

If an address contains a quadrant (NW, NE, SE, SW), then add it to that quadrant's list.

Do this with a handful of addresses and print the contents of each list.

HEAR ME
CODE

FOR EACH ITEM IN THIS LIST ...

```
attendees = ['Shannon', 'Jenn',
'Grace', ...]

for name in attendees:
    print name
```

For each item in this list:

      do something with that item

HEAR ME
CODE

… DO SOMETHING WITH THAT ITEM

```
days = ['Monday','Tuesday',…]

for day in days:
    print day
```

For each item in this list:

      do something with that item

HEAR ME
CODE

IF WE HAVE A LIST, WE CAN LOOP OVER IT

- **`# Most common: range from 0 to` …**
  `range(5) # [0, 1, 2, 3, 4]`

- **`# range(start, stop)`**
  `range(5, 10) # [5, 6, 7, 8, 9]`

- **`range()`** creates a list of numbers.
- If we have a list, we can loop over it.

HEAR ME
CODE

FOR EACH NUMBER IN THIS LIST

```
for number in range(10):
    print number
```

- Use this when you need to do a task a certain number of times

- Remember: **range()** creates a list, and a **for** loop will do something for each item in a list.

HEAR ME
CODE

FOR EACH NUMBER IN THIS LIST

```
for week in range(1, 5):
    print "Week {0}".format(week)
```

For each item in this list:
        do something with that item

range(1, 5) is equivalent to [1, 2, 3, 4]

HEAR ME
CODE

NESTED FOR LOOPS

```python
for week in range(1, 5):
    print "Week {0}".format(week)

    for day in days:
        print day
```

Notice how the days loop is indented.

What happens if it's at the same indentation level as the weeks loop?

HEAR ME
CODE

LOOPS WITHIN LOOPS

```python
for month in months:
    print month

    for week in range(1, 5):
        print "Week {0}".format(week)

        for day in days:
            print day
```

HEAR ME
CODE

ENUMERATE( )

Normally, a **`for`** loop gives you each item in a list one at a time

`enumerate()` is a function that you use with a for loop to get the index (position) of that list item, too.

Commonly used when you need to change each item in a list one at a time.

HEAR ME
CODE

ENUMERATE( )

We're going to beat the lunchtime slump by taking attendance again now - twice!

```python
for name in attendees:
    print name
```

```python
for index, name in enumerate(attendees):
    print index, name
```

HEAR ME
CODE

ZIP( ) MULTIPLE LISTS TOGETHER LIKE A ZIPPER

Normally, a **`for`** loop lets you use each item in a single list one at a time

`zip()` is a function that you use with a for loop to use each item in multiple lists all at once.

```python
state_abbrevs = ['AL', 'AK', 'AZ']
state_names = ['Alabama', 'Alaska', 'Arizona']

for abbrev, name in zip(state_abbrevs, state_names):
    print "The abbreviation of {0} is {1}".format(name, abbrev)
```

HEAR ME
CODE

WHILE LOOPS

A **for** loop lets you use each item in a single list one at a time, which is great for performing actions a certain number of times.

while loops are the cousins of conditionals.

Like an if statement, while will ask "is this true?"

HEAR ME
CODE

WHILE LOOPS: AS LONG AS THIS IS TRUE

```
if bread >= 2:
    print "I'm making a sandwich"


while bread >= 2:
    print "I'm making a sandwich"
    bread = bread - 2
```

What if you forget "bread = bread - 2"

HEAR ME
CODE

EXERCISES

On my Github's **python-lessons** repo, in the playtime folder:

- Beginner: PB&J While Loop
- Beginner: 99 bottles of beer on the wall
- Intermediate: States Drop-down menu

HEAR ME
CODE

CODE SAMPLES

All about Lists

https://github.com/shannonturner/python-lessons/tree/master/section_04_(lists)

All about Loops

https://github.com/shannonturner/python-lessons/tree/master/section_05_(loops)

All about Strings to lists, lists to strings

https://github.com/shannonturner/python-lessons/tree/master/section_06_(str-list)

HEAR ME
CODE