

LISTS AND LOOPS

Shannon Turner

Twitter: @svt827

Github: <http://github.com/shannonturner>

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

AGENDA

- Lightning Review
- Lists
 - The basics
 - Slicing (it's back!)
 - Adding/removing items
 - List methods
- Loops

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 exactly what it does?)

LIGHTNING REVIEW

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

LIGHTNING REVIEW

- Conditionals allow you to change the behavior of your program
- Program behavior is based on your variables:
 - **`age >= 21`**
 - **`bread == 0`**
 - **`gender.lower() == 'f'`**
 - **`len(attendees) > 30`**

LISTS: WHAT ARE THEY?

- 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)

LISTS: WHAT ARE THEY?

- If we had to do this, it would be a pain:
- attendee1 = 'Shannon'
- attendee2 = 'Jenn'
- attendee3 = 'Grace'

LISTS: SYNTAX

- Lists 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_pbj = []**

LISTS: 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]` ?

LISTS: LENGTH

- `attendees = ['Shannon', 'Jenn', 'Grace']`
- `print len(attendees) # 3`

or

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

LISTS: ADDING ITEMS

- `list.append()` adds an item to the end
- `attendees_ages = []`
- `attendees_ages.append(28)`
- `print attendees_ages # [28]`
- `attendees_ages.append(27)`
- `print attendees_ages # [28, 27]`

LISTS: CHANGING EXISTING ITEMS

- `print attendees_ages # [28, 27]`
- `attendees_ages[0] = 29`
- `print attendees_ages # [29, 27]`

LISTS: QUICK EXERCISE

- `days_of_week = ['Monday' , 'Tuesday']`
- `days_of_week.append('Wednesday')`
- Append the rest of the days in the week, then:
- `print days_of_week`
- `print len(days_of_week)`

LISTS: DELETING EXISTING ITEMS

- `print days_of_week`
- `day = days_of_week.pop()`
- `print day` # What do you get?
- `print days_of_week`
- `day = days_of_week.pop(3)`
- `print day` # What do you get?
- `print days_of_week`

LISTS: QUICK EXERCISE

- `months = ['January', 'February']`
- `months.extend(['March', 'April' ...])`
- `list.append()` adds one to the end
- `list.extend()` adds many

LISTS: ADD/REMOVE FROM THE BEGINNING

- `# Remove the first month`
`months.pop(0)`
- `# Insert 'January' before index 0`
`months.insert(0, 'January')`

LISTS: STRINGS TO 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)

LISTS: MEMBERSHIP

- The **in** keyword allows you to check whether a value exists in the list
- Also works with strings!
- `'ann' in 'Shannon' # True`
- `'Frankenstein' in python_class # False ... what a relief!`

LISTS: EXERCISE & LUNCH

Use `raw_input()` to allow a user to type an address

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

Allow user to enter 3 addresses; after three, print the length and contents of each list.

LISTS: RANGES OF NUMBERS

- # Most common: range from 0 to ...
`range(5)` # `[0, 1, 2, 3, 4]`
- # `range(start, stop)`
`range(5, 10)` # `[5, 6, 7, 8, 9]`
- Use this when you need to do a task a certain number of times

LISTS: RANGES OF NUMBERS

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

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

LOOPS: FOR LOOP EXERCISE

Change your quadrant exercise to use a for loop instead of repeating the same code three times.

Syntax looks a little like this:

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

LOOPS: FOR LOOP

```
days_of_week = ['Monday', 'Tuesday', ...]
```

```
for day in days_of_week:  
    print day
```

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

LOOPS: FOR LOOP

```
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]`

LOOPS: NESTED FOR LOOPS

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

```
for day in days_of_week:  
    print day
```

LOOPS: NESTED FOR LOOPS

```
for month in months_in_year:  
    print month
```

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

```
for day in days_of_week:  
    print day
```

LOOPS: 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.

LOOPS: ZIP

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.

LOOPS: WHILE

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?"

LOOPS: WHILE

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

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

EXERCISES

On my Github's [python-lessons](#) repo, go to the playtime folder:

- pbj_while.py
- states.py
- movies.py