INFO 450 Fall 2020

Week 2

Agenda

- Lists
- Sorting
- List Comprehension

Zen of Python

```
import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
```

- Simple is better than complex.
- Complex is better than complicated.
- Readability counts
- There should be one way to do it
- Now is better than never.

Lists

A list is a collection of items in a particular order.

You can put anything you want into a list

Bound by brackets [ ]

Lists should be named something plural e.g. 'students', 'customers', 'deer'

(See what I did there?)

```
import logging
logging.basicConfig(level=logging.DEBUG)
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
logging.debug("Length of bicycles: %d", len(bicycles))
logging.debug(bicycles)
```

- I probably won't always enter the import logging and logging.basicConfig calls in each code example, but, you will need to if you use logging.

Printing a List

When you print a list in Python (print or logging, I prefer logging) - Python returns a string representation of your list, brackets included.

```
>>> logging.debug("Length of bicycles: %d", len(bicycles))
DEBUG:root:Length of bicycles: 4
>>> logging.debug(bicycles)
DEBUG:root:['trek', 'cannondale', 'redline', 'specialized']
```

Access Elements in a list

The list contains elements in an order. A developer can access each item by its index:

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> logging.debug(bicycles[0])
DEBUG:root:trek
>>> logging.debug(bicycles[1])
DEBUG:root:cannondale
>>> logging.debug(bicycles[0].title())
DEBUG:root:Trek
```

Python starts counting/indexing at 0

because it's a real programming language

CRUD

- Create
- Read
- Update
- Delete

```
>>> motorcycles = ['honda','yamaha','harley-davidson']
>>> motorcycles
['honda', 'yamaha', 'harley-davidson']
>>> motorcycles[0] = 'ducati'
>>> motorcycles
['ducati', 'yamaha', 'harley-davidson']
>>> motorcycles.append('honda')
>>> motorcycles
['ducati', 'yamaha', 'harley-davidson', 'honda']
```

- append(itm) - Adds to the end of the list. New, highest index.

Dynamically creating and adding

```
>>> motorcycles = []
>>> motorcycles.append('honda')
>>> motorcycles.append('yamaha')
>>> motorcycles.append('harley-davidson')
>>> motorcycles
['honda', 'yamaha', 'harley-davidson']
```

Inserting , not appending

```
>>> motorcycles
['honda', 'yamaha', 'harley-davidson']
>>> motorcycles.insert(2, 'suzuki')
>>> motorcycles
['honda', 'yamaha', 'suzuki', 'harley-davidson']
```

Deleting items from the list

Python has a keyword: to delete. This deletes variables from memory, items in the list, etc

```
>>> motorcycles
['honda', 'yamaha', 'suzuki', 'harley-davidson']
>>> del motorcycles[2]
>>> motorcycles
['honda', 'yamaha', 'harley-davidson']
>>> bike = motorcycles.pop()
>>> motorcycles
['honda', 'yamaha']
>>> bike
'harley-davidson'
>>> new_bike = motorcycles.pop(0)
>>> motorcycles
['yamaha']
>>> new_bike
'honda'
>>> yamaha_bike = 'yamaha'
>>> motorcycles.remove(yamaha_bike)
>>> motorcycles
[]
```

Organizing a list

- Permanent sort

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> cars
['bmw', 'audi', 'toyota', 'subaru']
>>> cars.sort()
>>> cars
['audi', 'bmw', 'subaru', 'toyota']
>>> cars.sort(reverse=True)
>>> cars
['toyota', 'subaru', 'bmw', 'audi']
```

Temporary Sort

Maintain the original ordered list, but provide a sorted 'copy'

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> sorted(cars)
['audi', 'bmw', 'subaru', 'toyota']
>>> sorted_cars = sorted(cars)
>>> sorted_cars
['audi', 'bmw', 'subaru', 'toyota']
>>> cars
['bmw', 'audi', 'toyota', 'subaru']
```

Other things

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> cars
['bmw', 'audi', 'toyota', 'subaru']
>>> cars.reverse()
>>> cars
['subaru', 'toyota', 'audi', 'bmw']
>>> len(cars)
4
>>> cars[-1]
'bmw'
>>> cars[0]
'subaru'
>>> cars[0:1]
['subaru']
>>> cars[1:3]
['toyota', 'audi']
>>> cars[55]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Working with lists

One of the most used control structures in Python has to do with iterating through every element in a list.

Example from the book: in a game, you might want to move every element on the screen.

- Print out todays stock market
- List all orders
- Verify inventory

Old way

```
magicians = ['alice', 'pat', 'pallavi', 'sam']
x = 0
while x < len(magicians):
    logging.debug(magicians[x])
    x += 1

DEBUG:root:alice
DEBUG:root:pat
DEBUG:root:pallavi
DEBUG:root:sam
```

for

not really sure why I showed the old way.

The `for` loop construct allows you to not worry about how many items are in a list

```
magicians = ['alice', 'pat', 'pallavi', 'sam']
for magician in magicians:
     logging.debug(magician)
logging.debug("DONE")
DEBUG:root:alice
DEBUG:root:pat
DEBUG:root:pallavi
DEBUG:root:sam
DEBUG:root:DONE
```

```
magicians = []
for magician in magicians:
     logging.debug(magician)

DEBUG:root:DONE
```

Lots of errors in the book

Indenting where you shouldn't

Forgetting to indent where you should

Forgetting the colon :

```
for magician in magicians
    logging.debug(magician)
```

Numerical Lists

https://docs.python.org/3/library/functions.html#func-range

The `range` function generates a series of numbers

```
class range(stop)
class range(start, stop[, step])
Rather than being a function, range is actually an immutable sequence type,
as documented in Ranges and Sequence Types — list, tuple, range.
```

```
>>> for x in range(5):
...     logging.debug(x)
...
DEBUG:root:0
DEBUG:root:1
DEBUG:root:2
DEBUG:root:3
DEBUG:root:4
```

```
>>> for x in range(3,9):
...     logging.debug(x)
...
DEBUG:root:3
DEBUG:root:4
DEBUG:root:5
DEBUG:root:6
DEBUG:root:7
DEBUG:root:8
```

.... immutable sequence type?

Let's look at some Python fun to see what's going on.

```
>>> x = range(10)
>>> type(x)
<class 'range'>
>>> numbers = list(range(10))
>>> type(numbers)
<class 'list'>
>>> numbers
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Comprehension

Allows you to perform some actions on a list in one line of code.

Combines the for loop and creation of new elements into one line.

Book: "List comprehensions are not always presented to beginners, but I have included them here because you'll most likely see them as soon as you start looking at other peoples code."

```
>>> squares = [value ** 2 for value in range(1,11)]
>>> squares
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

HOMEWORK

Due Wednesday, 2nd by 23:59:59 ET (EST?)

[Your github name]/[your github repo]/week2/homework.py

Complete the function "squared threes"

```
import logging
logging.basicConfig(level=logging.DEBUG)

def squared_threes():
    return_value = []
    # YOUR CODE GOES HERE

    # END SHOULDNT GO BEYOND HERE
    return return_value

if __name__ == "__main__":
   for x in squared_threes():
       print(x)

   # should print out all numbers between 0 and 99 (inclusive)
   # that are evenly divisible by 3, then squared.
   # e.g
   # 9
   # 36
   # 81
   # .....   etc
```