More on looping

You can loop over all kinds of stuff in Python.

Dictionaries

```
for key, value in user_dict.iteritems():
    …
```

Pandas rows

```
for row_index, row in movies_df.iterrows():
    …
```

SQL tables

```
user_table = Table(‘users’, metadata, autoload=True)
users = user_table.select()

for user in users:
    …
```

*this is using the `sqlalchemy` library, included in Anaconda

Another trick

Need to know which iteration of a loop you’re in?

Use enumerate

```
>>> thing_list = [“hello”, “there”, “class”]
>>> for i, thing in enumerate(thing_list):
…     print i + “ “ + thing

0 hello
1 there
2 class
```

```
def enumerate(collection):
    i = 0
    enumerated = []
    for thing in collection:
        enumerated.append(i, thing)
        i += 1
    return enumerated
```

- What if you wanted to use enumerate on a billion rows?

Enumerate is a “generator”.

```
def enumerate(collection):
    i = 0
    it = iter(collection)
    while 1:
        yield (i, it.next())
        i += 1
```

wat

`return` means “OK boss, there’s the answer. I’m done.”

`yield` means “Here’s one of them. Let me know when you need the next one.”

Why bother? (an example)

```
user_table = Table(‘users’, metadata, autoload=True)
users = user_table.select()

for user in add_tags(users):
    …

def add_tags(users):
    while True:
        user_row = users.next()
        # do some stuff with this SQL row
        …
        yield user_row
```

Classes

Sometimes lists and dictionaries don’t cut it.

You can define your own types of objects in Python using classes.

Definition:

```
class BootcampStudent():
    …
```

Making a new instance:

```
student = BootcampStudent()
```

Methods:

```
class BootcampStudent():
    def update_data_from_github(self):
        gh_api = GitHub()
        gh_api.users(self.github_username).get()

st = BootcampStudent()
```

The “constructor” method gets run every time a new instance is created.

```
class BootcampStudent():
    def __init__(self, name, github_username):
        self.name = name
        self.github_username = github_username

st = BootcampStudent(“Irmak Sirer”, “frrmack”)
```

CAREFUL!

You can set an attribute outside the **init**, at the class level.
This is called a **class attribute**, and if you change it one place it *changes for all instances of that class*.

```
class BootcampStudent():
    company = “CapitalOne”
    def __init__(self, name):
        self.name = name
        …
```

Other magic methods

- `def __repr__(self):` defines what happens when an instance of your class is printed
- `def __eq__(self, other):` defines what happens when checking if equal to something else
- `def __add__(self, other):` defines what happens when involved in an addition operator
- and many more!

Inheritance

Subclasses can “inherit” from parent classes and add onto the functionality of their parent class.

```
class Roster(list):
    def generate_pairs(self):
        …



roster = Roster()
for student in students:
    roster.append(student)
roster.generate_pairs()
```