* **raw\_input(‘What is your age?’)**
* **If a conditional is empty the keyword ‘pass’ must be used**
* **If statements use a colon rather than parenthesis**
  + i.e
    - *if self.request.GET:*
      * *fish.request.GET[‘fish’]*
* **Else if statements are written ‘elif’**
* **Functions are created with the keyword ‘def’**
  + i.e
    - *def print\_name:*
      * *print ‘Samantha’*
    - *print\_name()*
* **Casting** 
  + str = String int = integer float = decimals
* **Dictionaries in python are similar to objects and arrays. Set a variable equivalent to ‘dickt()’ then set the variable equal to an object** 
  + i.e
    - ice\_cream = dict()
    - ice\_cream = {‘Ben and Jerrys’: ‘Cookie Dough’, ‘Perrys’:’Chocolate’, ‘Bryars’:’Strawberry’}
* **Loops**
  + Range
    - i.e
      * *for I in range (1,100):*
  + Loop through array
    - i.e
      * *names = [Sam, Tina, Tom]*
      * *for I in names:*
        + *print i*
* **Format methods or big strings**
  + i.e
    - *your\_state = ‘Florida’*
    - *your\_name = ‘Samantha’*
    - *message = ‘’’ {your\_name} lives in {your\_state}.’’’*
    - *Message = message.format(\*\*locals())*
* **Booleans are capitalized**
  + i.e
    - *on = True*
    - *if on:*
      * *on = False*
    - *else:* 
      * *on = True*
* **app.yaml**
  + Adding css in app.yaml
    - i.e
      * *- url: /css/main\.css*
      * *static\_files: css/main.css*
      * *upload: css/main\.css*
* **Access modifiers**
  + Public – all variables and methods are by default
  + Protected – only accessable within the class and its subclasses
    - i.e
      * *class Place(object):*
        + *def\_ \_ init \_ \_ (self):*

*self.\_content = ‘ ‘*

* + - * + *plate = Plate()*
        + *plate.\_content = ‘Salmon’*
  + Private – nobody should gain access to it from outside the class
    - *Class Plate(object):*
      * *def \_ \_ init \_ \_(self):*
        + *self.\_ \_ content = ‘ ‘*
      * *plate = Plate()*
      * *plate. \_ \_ content = ‘Salmon’*
      * *\*\*\* Will product error, cannot be accessed*
* **Object Oriented programming concepts**
  + Encapsulation – hiding pieces of code from other people or your future self
    - Getters – Read only, looking at a variable
      * i.e
        + *class Average(object):*

*def \_ \_ init \_ \_ (self):*

*self.hw1 = 70*

*self.hw2 = 80*

*self.hw3 = 90*

*self.\_\_average = 0*

*@property*

*def average(self):*

*return self.\_\_average*

* + - * + *average = Average()*
        + *print average.average*
    - Setter – Write only, changing variable
      * i.e
        + *class Average(object):*

*def \_ \_ init \_ \_ (self):*

*self.hw1 = 70*

*self.hw2 = 80*

*self.hw3 = 90*

*self.\_\_average = 0*

*@property*

*def average(self):*

*return self.\_\_average*

*@average.setter*

*def average(self, new\_average):*

*self.\_\_average = new\_average*

* + - * + *average = Average()*
  + Abstraction – Classes created to hold attributes and methods to be used as a blueprint
    - i.e
      * *class Social\_media(object):*
        + *def \_ \_ init \_ \_ (self):*

*self.messaging = True*

*self.photos = True*

*self.friends = True*

* + Inheritance – Subclasses inherit attributes and methods from super class
    - i.e
      * *class Social\_media(object):*
        + *def \_ \_ init \_ \_ (self):*

*self.messaging = True*

*self.photos = True*

*self.friends = True*

* + - * *class Facebook(Social\_media):*
        + *def \_ \_ init \_ \_(self):*

*super(Facebook, self).\_ \_ init \_ \_ ()*

*self.messaging = True*

*self.photos = True*

*self.friends = True*

* + - * *class Instagram(Social\_media):*
        + *def \_ \_ init \_ \_(self):*

*super(Instagram, self).\_ \_ init \_ \_ ()*

*self.messaging = True*

*self.photos = True*

*self.friends = True*

* + - * *class Twitter(Social\_media):*
        + *def \_ \_ init \_ \_(self):*

*super(Twitter, self).\_ \_ init \_ \_ ()*

*self.messaging = True*

*self.photos = True*

* + - * *self.friends = True*
  + Polymorphism – Sub classes can override/repurpose a method that was set up in the super class
    - i.e
      * *class Room(object):*
        + *def \_ \_ init \_ \_(self):*

*self.doors = 0*

*self.outlets = 0*

*self.furniture = ‘ ’*

* + - * + *def printInfo(self):*

*print self.doors + self.outlets + self.furniture*

* + - * *class Bedroom(Room):*
        + *def \_ \_ init \_ \_(self):*

*super(Bedroom, self). \_ \_init \_ \_()*

*self.doors = 2*

*self.outlets = 8*

*self.furniture = ‘Bed, dresser’*

*self.closet = True*

* + - * + *def printInfo(self):*

*print self.doors + self.outlets + self.furniture + self.closet*

* **Aggregation – Implies a relationship where the child can exist independently of the parent.** 
  + Example: Class(parent) and Student(child). Delete the class and the Students still exist
* **Composition - implies a relationship where the child cannot exist independent of the parent**
  + Example: House (parent) and Room (child). Rooms don't exist separate to a House.
* **MVC – Model View Controller. Used to organize code. Separation of presentation makes it easier to edit, develop, and, collaborate.** 
  + Model– Data
    - Requesting, receiving, validating and sorting data
      * *class ConcertModel(object):*
        + *def \_ \_ init \_ \_ (self, concert):*
    - To pull from an api’s location
      * *self.\_ \_url = ‘http://xml.concertInfo/?q=’*
    - To assemble a request and concatenate the users inputed infromation
      * *self.\_ \_ request = urllib2.Request(self.\_ \_url + concert)*
    - Create object to get the url
      * *self. \_ \_opener = urllib2.buildopener()*
        + *def send(self):*
      * Use the url to get a result and request information from the api
        + *self.\_ \_ result = self.\_ \_opener.open(self.\_ \_request)*
        + *self.sort()*
        + *def sort(self):*
    - Parsing xml data and using it
      * *self.\_ \_xmldoc = minidom.parse(self.\_ \_result)*

*self.\_ \_populate = []*

*for i in self.\_ \_xmldoc[‘results’]:*

*do = ConcertData()*

*do.state = i[‘state’]*

*do.time = i[‘time’]*

*self.\_ \_populare.append(do)*

*@property*

*def populate(self):*

*return self.\_ \_populate*

* + - Storing returned data from api
      * *class ConcertData(object):*
        + *def \_ \_ init \_ \_ (self):*

*self.state = ‘ ‘*

*self.time = ‘ ‘*

* + View – What we see
    - Forms, gets info from user and sends it to the controller and/or model
      * *class ConcertView(object):*
        + *def \_ \_ init \_ \_(self):*

*self.\_ \_populate = ConcertData()*

* + - Looping through each recipes data to populate it
      * *def update(self, np):*
      * Variables to push content into
        + *self.\_ \_content = ‘ ‘*
    - Looping through each recipes data to populate it
      * *for i in np:*
        + *self.\_ \_populate = i*
        + *self.\_ \_content += "<div class='container sixteen columns results'>"*
        + *self.\_ \_content += '<p>State: ' + i.state + '</p>'*
        + *self.\_ \_content += '<p>Time: ' + i.time + '</p>'*
        + *self.\_ \_content += ‘</div>’*
    - Returning information through getter
      * *@property*
      * *def populate(self):*
        + *return self.\_ \_populate*
    - Using setter to set new values
      * *@populate.setter*
      * *def populate(self, new\_populate):*
        + *self.update(new\_populate)*
    - Returning information through getter
      * *@property*
      * *def concent(self)*
        + *return self.\_ \_content*
  + Controller – Master
    - Managing how model and view work together, manages changes
      * *if self.request.GET:*
        + *concert = self.request.GET[‘concert’]*
        + *cm = ConcertModel(concert)*
        + *cm.send()*
        + *cv = ConcertView()*
        + *cv.do = cm.do*
        + *self.response.write(cv.content)*
* **Import** 
  + Requesting api information
    - *Import urllib2*
  + JSON import
    - *import json*
  + xml import using minidom
    - *from xml.dom import minidom*
* **Loading in a url**
  + *self.\_ \_url = ‘http://xml.concertInfo/?q=’*
* **Parsing xml data**
  + *class BarModel(object):*
    - *def \_ \_ init \_ \_ (self, bar):*
      * *self.\_ \_url = ‘http://xml.barInfo/?q=’*
      * *self.\_ \_ request = urllib2.Request(self.\_ \_url + bar)*
      * *self. \_ \_opener = urllib2.buildopener()*
    - *def send(self):*
      * *self.\_ \_ result = self.\_ \_opener.open(self.\_ \_request)*
      * *self.sort()*
    - *def sort(self):*
      * *self.\_ \_xmldoc = minidom.parse(self.\_ \_result)*
* ***Parsing JSON data***
  + *class ConcertModel(object):*
    - *def \_ \_ init \_ \_ (self, bar):*
      * *self.\_ \_url = ‘http://xml.barInfo/?q=’*
      * *self.\_ \_ request = urllib2.Request(self.\_ \_url + bar)*
      * *self. \_ \_opener = urllib2.buildopener()*
    - *def send(self):*
      * *self.\_ \_ result = self.\_ \_opener.open(self.\_ \_request)*
      * *self.sort()*
    - *def sort(self):*
      * *self.\_ \_json\_data = json.load(self.\_ \_result)*
* **HTML population through superclass and subclass** 
  + Super Class – sub classes inherit from super classes
    - *class MainPage(object):*
      * *def \_ \_init \_ \_(self):*
  + Basic html set up to create basic html elements
    - * *self.\_head = ‘’’ <! DOCTYPE HTML><html><head></head><body> ‘’’*
      * *self.\_body = ‘’’ ‘’’*
      * *self.\_footer = ‘’’ </body></html> ‘’’*
    - *def print\_page\_info(self):*
      * *return self.\_head + self.\_body + self.\_footer*
  + Sub class – sub classes inhearite from super classes, in this case it is inheariting from the class MainPage
    - *class InputPage(MainPage):*
      * *def \_ \_init \_ \_(self):*
        + *super(InputPage, self).\_ \_init\_ \_()*
        + *self. \_ \_ input\_open = ‘’’ <form method=’GET’> ‘’’*
        + *self.\_ \_input = ‘’’ <input type='text' placeholder='Bar' name='bar'> <input type='submit' value='Search'> ‘’’*
        + *self.\_ \_input\_close = ‘’’ </form> ‘’’*
  + Polymorphism
    - Function from MainPage class is being repurposed for it’s sub class
      * *def print\_page\_info(self):*
        + *return self.\_head + self.\_ \_input\_open + self.\_ \_input + self.\_ \_input\_close + self.\_footer*
      * Instantiate subclass in the controller and use self .response.write(page.print\_page\_info())
* **docstring** = ‘’’ Comments in here to explain code ‘’’