- **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*

- **<u>Aggregation</u> – 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
- **<u>Composition</u> - 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.
- **<u>MVC</u> – Model View Controller. Used to organize code. Separation of presentation makes it easier to edit, develop, and, collaborate.**
    - <u>Model</u> – Data
        - Requesting, receiving, validating and sorting data
            - *class ConcertModel(object):*
                - *def _ _ init _ _ (self, concert):*
                    - *self._ _url = 'http://xml.concertInfo/?q='*
                    - *self._ _ request = urllib2.Request(self._ _url + concert)*
                    - *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)*
                    - *self._ _populate = []*
                    - *for i in self._ _xmldoc['results']:*
                        - *do = ConcertData()*
                        - *do.state = i['state']*
                        - *do.time = i['time']*
                        - *do.artist = i['artist']*

- *do.venue = i['venue']*
- *self._ _populare.append(do)*

- *@property*
- *def populate(self):*
  - *return self._ _populate*

- *class ConcertData(object):*
  - *def _ _ init _ _ (self):*
    - *self.state = ' '*
    - *self.time = ' '*
    - *self.artist = ' '*
    - *self.venue = ' '*

- 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()*
      - *def update(self, np):*
        - *self._ _content = ' '*
        - *for i in np:*
          - *self._ _populate = i*
          - *self._ _content += " "*
          - *self._ _content += ' State: ' + i.state + ' '*
          - *self._ _content += ' Time: ' + i.time + ' '*
          - *self._ _content += ' Artist: ' + i.artist + ' '*
          - *self._ _content += ' Venue: ' + i.venue + ' '*
          - *self._ _content += ' '*
      - *@property*
      - *def populate(self):*
        - *return self._ _populate*
      - *@populate.setter*
      - *def populate(self, new_populate):*
        - *self.update(new_populate)*
      - *@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**
  - *class MainPage(object):*
    - *def _ _init _ _(self):*
      - *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*

- *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> ‘’’*
  - *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())

- **<u>docstring</u>** = ‘’’ Comments in here to explain code ‘’’