### *CSS*

**BASIC**

* **COMMENT** - /\* … \*/
* LINK CSS file (here, stylesheet.css) to the index.html sheet with <**link type = "text/css" rel = "stylesheet" href = "stylesheet.css"/>** within <head> code
* **Self-closing tags** - eg. img, link - if nothing in between <></>, then can make it **</>**

**PRIORITY:**

* **CASCADE ORDER: LAST CSS FORMAT APPLIED TO ANY ELEMENT IS WHAT SHOWS!**
* Can override using **!important** in css file 🡪 MOST imp, will show no matter what order of external links/inline styles
* **SPECIFICITY**: More specific css styling will apply (eg. div#menu styling will override div style, independent of Cascade order)
  + IDs > Classeses > Elements; ## elements > # elements (eg. .div1.div2 { } > .div{ })

**STYLING:**   
**SELECTOR { PROPERTY: VALUE}** Eg. .header p {color: blue; …}

* **GROUPING (CLASS/ID)**
  + **CLASS** - **bunch** of elements. **<div class = "**square**">**... Identified in CSS as **.**classname **{}**
  + **IDs**- **single** element being stylized. **<div id = "**circle">... In CSS as**#**circle**{**color: "blue";**}**
  + **Sub-styling within classes/IDs:**
    - **p.**classname { } --> only applies to classes of classname that are PARAGRAPHS!
    - **p[name** = " xx "**]** { } --> only applies to classes that are paragraphs and have attribute name called xx
    - **p[name ^/ $ / \*=** "bacon"] { } --> any value BEGINNING with /ENDING with/ CONTAINS "bacon" will be affected
* **COLORS**
* **Color:** hexadecimal values (#6 nos./letters), RGB values (0,255,32), Color names
* **background-color:**
* **FONTS**
* **font:** bold 20px Tahoma
* **font-family:** sans-serif;
* **font-size:** 60 px; OR 1 **em** (relative measure of the screen size, interchangeable across diff size screens vs. px)
* **OTHER**
  + **TEXT-ALIGN: center/right/left**
* **Background-image: url (‘**http://…’);
* **MARGIN**
  + Space **OUTSIDE** the element; adjust margin to move elements closer or farther from each other
  + **margin: auto** (centers the element)
  + **margin-right: auto** (takes as much space as possible on the right side, causing the box to appear at the left edge of the screen)
  + **margin-top**: 1px or **margin-bottom/right/left:** or
  + **margin:**1px 2px 3px 4px - top margin and around clockwise
* **BORDER**
  + Edge of the element
* **BORDER: 1px solid black**;  (width: px, style: solid/dashed/outset, color)
  + **border-RADIUS:5px**etc. to get rounded edges
* **PADDING:**
  + Space between content/text and border, **INSIDE** the border; adjust padding to move content closer or farther from border
  + **Padding-top/bottom/right/left:** xpx xpx xpx xpx or
  + **Padding:** value value value value or **padding:10px** for same padding on all sides

***Notes***:

* + TP, TB, TM - top padding,  border, margin; **negative padding** or margin values moves the object in the opp direction.
  + Margin, Border, Padding work in that order (top-to-bottom) of a text element
* **DISPLAY:**
  + **BLOCK** - takes up full width of a line (unless o/w stated in css as width = 100px), displays on a new line, default behaviour
  + **INLINE-BLOCK -**element width of a box but other elements can sit on the SAME LINE; use for menu items/tabs
  + **INLINE** - element only takes its own width, not box width, best for menu bar, headers and paras
  + **NONE** - element disappears, doesn’t taken any space
  + **HIDDEN** - element disappears/doesn't show but takes space!
* **POSITION**
  + **POSITION:ABSOLUTE / RELATIVE** - add on margin functions to support positioning
* **TOP/LEFT/BOTTOM/RIGHT : 10 px** 🡪 shifts an element from the top/left..; must use position:relative for this!
  + **POSITION: FIXED** - anchors an element to a browser’s window, doesn't change even if scroll up/down, eg. Navigation menu on webpage
* **FLOAT:**
  + **FLOAT:right/left** 🡪 moves an element to the far left or right of the page
  + **CLEAR:right/left/both** - gets out of elements floating on the left/right and moves element to the bottom of the page (e.g. copyright 2011 etc.)
  + **Z-INDEX:1** gives an element highest priority

# **2. PYTHON (.py)**

**BASICS**

* **python.org -->** Download --> Installer
* **Comment - #** or **""" xxx """**  for multi-line comment or select the lines and Cmd ?
* Run file in Terminal by typing **python [foldername/filename.py]**
* **HELP** (object) or **DIR (**object eg. 5) 🡪 help on int etc or details out the object’s methods which you can dive into detail as <**object>.\_\_methodname\_\_** (eg. abs.\_\_doc\_\_)
* **Python** functionname 🡪 describes the function
* **Pydoc** functionname (eg. open, file, os, sys) 🡪 documents the function; **q** to quit
* **Syntax:**
  + 4 spaces, not tabs
  + Variable names – lowercase, underscore separators i.e. my\_func not myFunc

**INPUT/OUTPUT**

* **PRINT** "Welcome to Python" (or 'xx' ) or print 3, 5\*\*2, var1, var2
  + To print all characters: **for** char **in** string1: **print** char**,** --> , prints each char on the same line
* name = **RAW\_INPUT** ("Enter name:") --> takes what user inputs and treats it as a STRING vs. **INPUT** ("Enter name:") --> NOT a string
  + var = **int(raw\_input())** 🡪 converts it into an int
  + raw\_input **("Press<enter>")** --> allows you to see the program so it doesn't exit immediately after running but waits till you press enter

**MATH OPERATORS / LOGIC & COMPARISONS**

* +, -, \*, / --> division computes floats (eg. 5/2 = 2.5) OR // --> division computes ints (eg. 5//2 = 2 i.e. largest whole number less than the answer)
* Precedence - PEDMAS
* += 5 , -= --> x = x + 5
* Exponentiation \*\* - eg. 2 \*\* 3 = 2 \* 2 \* 2
* MODULO/us % remainder - eg. 10 % 3 = 1
* ones = hour % 10; tens = hour // 10
* COMPARATORS: **= =, !=, < >, <=;**
* OPERATORS: **NOT**, **AND, OR**--> evaluated in that order; change order using ( )
* print x = 3==3 --> prints Trues

**DATA TYPES/VARIABLES: --> n = #**

* **int** --> int (3.7) outputs just the number (3), no rounding
* **float** --> stores 15 decimal digits of accuracy
* **booleans** (True/False - capital T/F); if friend: --> if friend == True
* print “text… {} … {}”**.FORMAT (**vara, varb)
  + ~~Old way - print “My name is~~ **~~%s or %d or %i or %r~~**~~” (~~**~~%stringname~~**~~, %doublename, %int), %character no matter what~~
* **STRING**: variable = "…" or '..' or str (5)
* SPECIAL CHARACTERS
  + **\n** 🡪 new line (can be included within the “ \n string “
  + **\t** 🡪 tabbed in
  + **\\** 🡪 escape character eg. var = ‘I\'m’ or \\
  + print var1, var2 🡪 w/o comma, prints var2 on new line, else on same line
* **GLOBAL** (create, modify etc. outside functions) vs.
* **LOCAL** (inside functions) variables; To update/assign global variable in a function - GLOBAL varx

**LOOPS**

* **WHILE** count < 5**: …**count += 1 **ELSE ..**
* IF … : **BREAK**
* **if…: ELIF….: else:**
* If/While .. **ELSE:**
* **FOR if .. ELSE**
  + **List elements**: **FOR** num **IN** array\_numbers [:2]: --> print num
  + **Strings:** for char in range (**len(string) – 1):** …
  + **Numbers**: for i in **RANGE (10):** --> for i in range 0-9
  + **Characters - for** char **in** string1: **print** char --> prints each **character** of string1
  + **Lists**: for **index, value** in list.**ENUMERATE()…**
  + **Dictionaries**: FOR key, value in dict.**ITERITEMS()**: print key, value or **PRINT “{}: {}”.FORMAT(k,v)** 🡪 prints out key: value i.e. 1st value in first {}, 2nd in 2nd {} etc; can also do “{} is {} years old”.format(person[‘name’], person[‘age])
* **BREAK:** eg. if name = ‘x’: break

**STRING METHODS**:

* **Note:**
  + **STRINGS ARE IMMUTABLE!** 🡪 greet = “Yo”; new\_greet = “H” + greet[1:] NOT greet[0] = “H”
  + Concatenate**: +** only works on strings --> print "U " + "rock" + str(2) --> output "U rock 2" but VERY inefficient 🡪 use “{} {}”.format(w1, w2)!
* print **LEN(var)** - calculates length of text
* **string[-1]** --> prints last letter of string OR **string[len (string)-1]**
* str1 = "poona", str2 = "blr" --> print "Let's go to **%s** or **%s**" **%(str1, str2)** --> Let's go to poona or blr
* Can **SLICE** strings: eg. for char **in range (len(string)-1)**: if **string [char:char+2]** == “hi”: return True; won’t work for ~~char in string~~
* string.**SPLIT (“**x**”);** 🡪 creates an **ARRAY** of strings split at character x
* string.**FIND (**'text') --> starts counting at 0, returns character at which the word starts, incl. spaces
* var**.LOWER()**/.**UPPER()./TITLE()/.SWAPCASE** - makes all text lower/UPPER /Proper case/sWAPS cASE--> dot notation works only on strings
* var1.**JOIN**(list) or **“ “.JOIN**(splitwords)🡪 joins var1 to each of the words of the list (eg. heyvar1therevar1)
* string.**REPLACE(**‘wordtofindinstring’, ‘wordtoreplacewith’) 🡪 for all occurrences
* string.**TRANSLATE(**whattoreplacewith, whattoreplace) -> eg. replace special characters ‘!@#$%<>?,./;:’ or **‘[^A-Za-z0-9]’** with nothing 🡪 .translate (None, ‘…’)
* **SORTED('string', key =)** --> returns first capitals, then lower case letters unless key is specified
* **STR(var)** - makes a string out of the parameter if not a string
* Str1.**ENDSWITH (**str2) 🡪 checks if string 1 ends with string 2
* var.**isalpha( )** -> checks if var contains only alphabetical characters; true/false output
* To map letters to other letters:
  + table = string.**MAKETRANS (**“a,b,c….”, “c,d,e…”) 🡪 maps a to c, b to d etc.
  + originalstring.**TRANSLATE(TABLE)**
* char **IN “aeiouAEIOU”:** 🡪 checks if char is a vowel
* To check **if str is an int:**
  + String.**isdigit()** 🡪 for positive #s (can do it for s[1:].isdigit() 🡪 T/F
  + Write a helper function **- try: int(s) return True except ValueError: return False**

**LISTS/SEQUENCES** (or STRINGS) (Can't use for ints):

* listx = [a, b, "c"]; listx[0] = a, listx[-1] --> last element
* **LIST COMPREHENSION: Python syntax to create a list:** 
  + New\_list = [***function* for *item* in [*list or range*] if *condition*]**
    - Eg: evens\_to\_50: new\_list = [i **for** i **in** **range**(51) **if** i % 2 == 0]
    - Eg. newwords = [“{}!”.format(word) for word in [“boom”] = [“boom!”]
* **TUPLE**: tuplename = (“dog”…) i.e. **( ) syntax**, much **faster** than lists but **CAN’T MODIFY** the tuple!
* **BASIC:**
  + **LEN** (lists) = 3
  + **MAX/MIN (**list) --> returns max/min value
  + **LIST**('string') --> makes string into a list/array of its characters; To COPY a list without mutating: a = [1,2,3]; b=a; a[0] = 5 --> a = b = [5,2,3] vs. not mutating **b = LIST(a)**; a[0] =5 --> a = [5,2,3], b = [1,2,3]
  + listx**.INDEX**("c") - returns FIRST index in which "c" is --> 2
  + **"Multiplying"**: string \* 10 --> stringstringstring or [int] \* 10 --> [int int int] (vs. 2 \* 10 --> 20)
  + Element **IN**a sequence? **Boolean**value --> 'x' (letter) **IN**word or 'word' in list etc.
  + **JOIN** (“ “**,** listname) or (“#”, list [3:6]) 🡪 joins the list with the “character” (either entire list of from 3 to 6-1 = 5th index)
  + **ANY** ([I % 3 **for** i **in** [3, 4, 5])) 🡪 returns true if condition is true for ANY item in the list; here 4 & 5 % 3 == 1, which is true
  + **SUM** (1 for i in [3,4,4] if i == 4) 🡪 returns 2; checks for how many items a condition is true and since it’s 2 4s and it’s “summing” 1s, that =s 2
* **SLICING**   
  **[start:end:stride]** where stride is space between items in the sliced list (eg. stride = 2 would be every other item)
  + **listx[0:2]** - starting at element 0 ending at element 2 (excl.) i.e. 0 & 1
  + **listx [:5]** - first through 6th element or **listx[5:]** - 6th through last element
  + **list[-1]:** last element --> indexing from back starts at -1, -2...
  + **listx[ : ]** - entire list
  + listx [::2] – entire list but every other element (stride = 2)
  + list[::-1] – entire list backwards! **Reverses the list**
  + listx [ 1:5] = **[ ]** --> **deletes** elements 1 through 4 from the list
  + **listx[1:1]** = [3,3,3] --> inserts new array in the middle of listx, after 1st element (0th index)
  + listx [ 1 : 8 : 2] --> element 1 through 8 skipping by 2
  + listx [ 10: 0 : -2] or list [: : -2]--> **counting backwards by 2** for part of list or whole list
* **MODIFYING LISTS** – *lists modified in place, vs. strings (creates new string)*
  + **CONCATENATE** - list1 **+** list2 or string1 **+** string 2--> combination of the two, but can't join 2 diff data types
  + listx**.APPEND**("d"); - adds d at the end of the list; doesn't work for strings?
  + Listx.SHIFT(“x”) 🡪 adds x at the FRONT of the list (to confirm)
  + listx**.INSERT** (3, "d") --> inserts d at 3rd index and moves the rest down 1 or
  + list1.**EXTEND** (list2) --> modifies list1 with list2 added on at the end
  + listx.**REMOVE** ("d") --> removes the FIRST occurrence of value, not the index; modifies the list
  + list.**POP** - list.pop(4) --> returns element at INDEX 4 that was "popped" i.e. kills the element & returns it (if list.**pop()** --> pops last element by default)
  + list.**DEL(**array[1]) --> like pop, deletes the element at INDEX 1, but doesn't return the element
  + list.**SORT(key** = str.lower()**)** --> sorts the list alphabetically, based on the key (here by alphabet); doesn't return the list until you call it; MODIFIES list!
  + **SORTED('string' or list or dict, [key =,] [reverse=True])** --> returns a new sorted list based on keys in reverse direction; can have **key = lambda** x: len(x[1]) 🡪 sorts by LENGTH of 1st index of array!
  + list.**REVERSE()** --> reverses the list
  + list.**COUNT** ('word') --> returns # of occurrences of word in the list
  + **ENUMERATE (dict, start = 0)** 🡪 lists out the dictionary values and keys, starting at index specified and counting indices;
    - To get index: **for index, item in enumerate (dict): print index, item**
  + **ZIP (lista, listb, listc..)** 🡪 iterates over multiple lists at once; creates pairs/triplets.. of elements and will stop at the end of shortest list; eg. *for a,b in zip (lista, listb): print max(a,b)*
  + **HASH (**key) 🡪 returns hash value i.e. integer of index of the key/object; if used in a for loop, run it for as many indices there are i.e. **LEN (**dict)
* **DICTIONARY / HASHES {‘key’: value}**: but call using dict [ ]
  + d = {'key1' **:** 9, 'key2' : my\_func} where 'x' is the **KEY**and 9 is the **VALUE**;
    - eg. for key in d: if d[key] == 9, print key; key can be a FUNCTION!
  + dict.**LEN()** --> gives the number of PAIRS
  + dict.**ITEMS ( )** 🡪 prints tuple of items (pairs)
  + dict.**VALUES** ( ) or dict.**KEYS** ( ) 🡪 returns array of dictionary’s values or keys
  + for k, v **IN** dict.**ITERITEMS():** to loop over the items of a dictionary
  + To **FORMAT** output**: “{} : {}”FORMAT(k,v)** 🡪 prints out 1st value in first {}, 2nd in 2nd {}; eg. “{} is {} years old”.format(person[‘name’], person[‘age])
  + **DEL** dictname [keyname]
  + dict.**CLEAR()** 🡪 removes all the keys and values
  + To **COPY** a list, don't just say lista = listb but instead listnew = **LIST(**oldlist) or **import copy** newlist = **copy.copy**(old\_list)
  + dic2 = dict1.**COPY ()** 🡪 makes a new copy of old dict
  + dict.**HAS\_KEY (‘key’)** 🡪 returns true/false
  + **FOR** keyname, valuename **IN** dictionary name: print keyname, valuename
  + **If not sure if the value exists:** value = dictname.**GET** (“keyname”, “Doesn’t exist”) 🡪 returns the second if keyname isn’t there
  + **Notes**:
    - Can only have immutable values as keys e.g.. tuple (a, b, c) but can't have a mutable element (lists, empty dicts) as keys
    - Dictionaries aren’t ordered in a specific way 🡪 can’t do dict[0]
* **MORE**
  + **LAMBDA** 🡪 **Anonymous function,** can pass it like a variable in a function, usually used for SORTING functions; *In all of these, “FUNCTION” can be defined as “lambda x : x\*\*2”* 
    - Eg. people = [(‘Jack’, 35), (‘Jill’, 23)];  
      **SORTED (**people, key = lambda person: person[1])) 🡪 sorts by age
  + **FILTER (FUNCTION, SEQUENCE)** 🡪 returns set of items (string, tuple or list) from the sequence for which the function (item) is true; eg. func – def f(x): return x\*x
    - Eg. print FILTER **(LAMBDA x : x % 3 ==0**, mylist)
  + **MAP (FUNCTION, SEQ1, SEQ2,…)** 🡪 calls function for each sequence (# of Seq arguments is based on # of args in the function) & returns a list of values
    - Eg. map (lambda x, y : x+y, seq1, seq2) 🡪 returns x+y for each element of seq 1 and seq2
  + **REDUCE (FUNC, SEQ)** 🡪 returns a single value constructed by calling the function on first 2 items of the sequence, then the result on the next item etc:
    - Eg. f = [1,2,3,4] print **REDUCE (LAMBDA x,y : x+y, f) 🡪** takes the first 2 elements (x, y and adds them x+y) and then takes that sum and adds it to the 3rd element…
* **OTHER**
  + **CHR**(key) --> converts key into a string (character)
  + **TUPLE**
    - List that can't be changed; check - print tuple1(a)
    - tuple1 = **(**x,y,z**)**

**FUNCTIONS**:

* When have ( ) 🡪 RUNS/EXECUTES the function!
* **def** my\_function **(parameter1, parameter 2)**:
  + **RETURN**- what function outputs; If no "**return** xxx", then the function returns "none")
  + If **RECURSIVE** function i.e. call the function within itself, make sure to “**RETURN”** after calling the function again
* For **default values of parameters**, def funcname (para1 = “x”, para2 = “y”):
  + If call funcname () 🡪 returns x, y
  + If user calls funcname (a, b) 🡪 returns a, b or funcname (para2 = b) 🡪 returns (x, b)
* **Multiple unknown parameters (**don't how many arguments) :
  + If parameters is a **tuple** \*: def function X **(\*args)**: print **args**
  + If parameters is a **dictionary** \*\*: def funcname (**\*\*dictname**); when call it, say funcname (key1 = val1, key2 = val2)
  + For combo: def funcname (var1, var2, \*tuple, \*\*dict): when call it, say funcname (name, age, 32, 43, bacon =3) or funcname (name, age, \*tuplename, \*\*dictname)
  + To ensure # args > 1: within the func, **IF args/dictname is NOT NONE:** …
* Can call a function within another function

**CLASSES (OOP)**

* **CREATING A CLASS AND OBJECT**
  + **CLASS** ClassName (**OBJECT):** // creating a class
    - **def** \_\_**INIT\_\_ (SELF,** param2 = **“default”**, …): self.param2 = param2; self.color = “red” etc. 1st param has to be SELF, default if no param
    - **def \_\_REPR\_\_(SELF) : return** “{} {}”.format(self.param1, self.param2…) i.e. // representation i.e. prints out the care object details defined in init
    - **def** method1 (**SELF, param1, param2…):** self.name = name
    - **variable1** = string/int/Boolean… //attributes of the class!
  + **Objectname = ClassName ( )** // creating an object instance of the class
  + Objectname.variable1 or objectname.methodname (param1, param2,..)
    - Access info of the class; don’t need SELF as a param when call method
    - **Don’t need () for variables, only for functions!**
* **SUBCLASS and SUPERCLASS**

CHILD class inherits ALL characteristics of parent class(es)!

* **IMPLICIT: class ChildClass** (**parentClass1, parentClass2…**)
  + ParentClass2 is usually a **MIXIN class,** which should **NOT have \_\_init\_\_ functions!**
  + Def \_\_ init\_\_ (self, param1, param2):
    - parentClass.\_\_init\_\_ (self, parentparam1val, parentparam2val) or
    - **super(ChildCass, self).\_\_init\_\_(param1)**
    - self.param2 = param2 (to override after initiatilizing….)
* **OVERRIDE:** For ChildClass to inherit only SOME characteristics and override rest of variables/funcs– instead of pass, **explicitly rewrite relevant functions / variables with same names** of parentclass in childclass as what you want it
  + Eg. Class child (parentClass) : var2 = “toast” (vs. “bacon” in parent class)
* **ALTER:** 
  + To make a child have a subset of characteristics, under def \_\_ init \_\_ **SUPER (CHILDCLASS, self) . \_\_INIT \_\_ (parentparam1, parentparam2)**
  + To use/alter parent commands in the child, def function (self, param1..): **SUPER (CHILDCLASS, self).parentClassFunction(self, param1…)**
* **TYPES OF METHODS:**
  + **INSTANCE METHOD:**
    - Needs **self** as parameter, eg. def func(x): **self**.x = x
  + **STATIC METHODS**:   
    Doesn’t edit/manipulate any instance of an object or any class – has nothing to do with the specific class (but are housed in classes)Class Vehicle(object):  
    **@staticmethod** //decoratordef random\_color():  
     colors = [“red”, “blue”, “green”]  
     return random.choice(colors) // no SELF!  
    *When calling:* color = Vehicle.random\_color(), don’t need object instance
  + **CLASS METHODS:**Related to Class, not specific object, so need to ref Class; Almost always used for **CREATING A NEW OBJECT** of any class you input  
    Class Vehicle (object):  
     max\_speed = 100  
    **@classmethod**  
    def get\_max\_speed(**CLS):** random\_color = **cls.**random\_color()  
     return newobj = cls(random\_color) //creates obj of class cls w/ rand color
  + **DECORATORS: 🡪 MUST RETURN FUNCTIONS!** (not int/value etc.)  
    Wrap functions to “augment”/give another ability; eg: **@log\_in\_required  
    def** calm\_down**(function):  
     def inner():  
     return function().**replace(“!”, “.”)  
     **return inner** //decorators return a function!  
     To use: @calm\_down right before the func. Being called
  + **GENERATORS**: Generates an interable list to infinity, yields another value every time you call **NEXT,** then stops until you call it again  
    **To create**: def integers(): i = 1 while True: **YIELD i ,** i+= 1  
    **To call:** ints = integers() 🡪 ints.**NEXT()** 🡪 1, ints.NEXT() = 2 etc. OR   
     for num in integers(): print num
* **COMPOSITION:**
  + One class: class Other (object): def func(self): print “Other’s func”)
  + Another class: class Another(object):
    - def \_\_init\_\_(self): **self.other = Other()**
    - def func(self): **self.other**.func()
* **VARIABLES**
  + **GLOBAL / MEMBER / INSTANCE** 🡪 variables (and/or functions) available everywhere / only to members of a certain class / only to particular instances of a class
  + Can **REFERENCE a global variable** w/in a function but **CANNOT ASSIGN/MODIFY the global variable** w/in a func; if you plan to modify:
    - Must declare a LOCAL variable of the same name or
    - **Pass in the global variable as an argument in the function**
    - Declare the variable as **GLOBAL** in the function (last option)
* **CONSTRUCTORS**
  + Initializes the object and runs the methods immediately (rather than having to explicitly call the methods of the class after you create the object)
  + Instead of objname = classname ( ) and then objectname.methodname ( ) to run each individual method, just objectname = classname ( ) and it automatically runs the methods of the class
* **APIs**
  + **HTTP REQUEST:**
    - **GET –** retrieves information from specified source
    - **POST –** sends new information to the specified source
    - **PUT/PATCH** – updates existing information of the source
    - **DELETE –** removes existing info from source
  + **REQUEST** consists of :
    - **Request LINE -** tells server what kind of request is being sent
    - **HEADER**– sends the server additional info
    - **BODY** – empty or contains data if you’re POST/putting info
  + **MAKE REQUEST:**
    - From **urllib2** import **urlopen** *and*From **json** import **load**
    - kitten **= urlopen(‘**http://placekitten.com’)  
      response = kitten.**READ()** 🡪 GET request  
      *or*

Import **requests**kitten = **requests.get(**‘http://placekitten.com’) 🡪 GET request

* + - print kitten.**text[**559:1000] 🡪 displays portion of text string
  + **Server RESPONSE**
    - Response = **requests.get** (“www.xx.com”))
    - Consists of 3# **status code** (response.**status\_code**) starting w/ 1-5:
      * 1xx – got it, working on your request
      * 2xx – okay! Successfully responding (eg. 200 – ok!)
      * 3xx – I can do what you want, but working on smtn else first
      * 4xx – mistake (eg. 404 – page not found)
      * 5xx – server goofed up and can’t respond to your request
  + **APIs:**
    - **API KEY –** long alphanumeric combination; goes through **OAuth**
    - APIs respond with either **XML** (~HTML but self-defined tags eg. <name> vs. <p>) or **JSON** (~javascript); read documentation to know which language response is in!
* **OTHER FUNCTIONS**
  + **IS-A:** something inherits from another (eg. salmon is a fish); talk about objects related to classes
  + **HAS-A:** something has something (eg. salmon has a mouth); objects reference classes
  + **ISINSTANCE(**object, Class) 🡪 To check is the object is an instance of a class
  + **HASATTR**(object, name) 🡪 returns True is object has an attribute “name”
  + **Def \_\_EQ\_\_(self, OTHER):** return self.name = other.name and …// to test equality of objects
* **TIPS**:
  + Put details/definitions of each individual class in a diff .py file
    - Link files with “from filename import ClassName”
    - Only need to import at the top of the file if a new object of that class is created
  + If pycharm has red squiggly line under a word, **alt+enter** to debug
  + **Return x OR NONE** (if x is referenced later on and you need a value for it!)

**KEY DOWN:**

* **From msvcrt import getch**
* **GETCH()** 🡪 gets the character or msvcrt.getch()
* **ORD**(getch()) 🡪 converts the key into an int value (eg. 27 ESC, 13 Enter, 224 special keys, 80, down arrow, 72, up arrow)

**IMPORT**:

* **IMPORT [libname]** or **FROM [libname] IMPORT [funcname]**
* **from SYS import ARGV** 🡪 argument variable or just **IMPORT SYS**
  + Argv – list of arguments you passed to interpreter
  + If script **inputs on command line**, then argv (else raw\_input()); eg. in terminal, say **python filename.py (script) var1, var2** and can say “print argv[1]” in the file, which prints out var1 (?)
  + **script, var1, var2 = argv**
* **Import RANDOM**, esp function **random.randrange()**; also **random.randint()**
  + Randint (0,5) – integers between 1 - 5 inclusive
  + Random.**CHOICE(**listx) 🡪 random word from listx
* **Import TIME🡪 Time.sleep()** 🡪 adds a second between each statement
* **Import** [**TIMEIT**](https://docs.python.org/2/library/timeit.html)
  + **>>> timeit.timeit**(stmt = ‘pass’, ….)
* **Import RE** (regular expressions)
  + **re.findall(r’\d+’,** bibliography) 🡪 \d+ means find >1 (+) digits (d) in string bibliography (or ‘\d{4}’ for exactly 4 digits or {1} to return each digit of all #s
  + **match** = **re.search(r’^\w+’,** “Sept 18th, 2014”) 🡪 get 1 match at start (^) (or end ($)) of alphanum words (w);
    - match.**group()** 🡪 returns all the results
    - match**.group(n)** 🡪 returns nth result if multiple searches eg. re.search(r’^(\w+)**:** (\w{3}-\w{4}), put each search in **( )**
    - match = re.search(r’.>(.+)<’, line) 🡪 searches for text within html tags eg. <p> This is text </p>; use **.+** not w+ coz want to return **spaces** too!
  + **Note**: To match “.” or “$”, need a \. Or \$ to escape! (else . = match charac and $ = march last character\_
  + **Note:** To find **ALL** instances, used **findall**; for **1 instance,** use **search-match**
  + **Note:** For any alphanumeric – findall(**r’([A-Za-z0-9\.\+\_-]**+@\w+\.com)
* **IMPORT CSV**
* Importing self-created modules (bunch of variables etc. stored) – **IMPORT** filename (w/o .py)
  + Call module as filename.methodname ( ) or var = filename.methodname and then var ( ) wherever you need
  + If make edits to the file, **RELOAD (filename),** don’t re-import, it wont work;or need to clos
  + e tab, open new tab, re-import and re-run
* Importing modules -**import MATH**– can then use functions like **math.sqrt** (25)
* TIPS
  + **DIR (MODULENAME)** eg. dir (math) - to find out what info contained in module; don't usually need the ones with \_\_func\_\_
  + **HELP (modulename)** 🡪 gives details on each functions in the module
  + Modulename**. \_\_DOC \_\_** 🡪 gives summary about the module, double underscore
* Importing functions within modules - from module import function i.e. from math import sqrt OR from module import \* - sqrt (25) - only be careful because sqrt may conflict with a sort function you create separately, so best to import individual functions
* Other built-in functions: **max** (\*args), **min** (\*args), **abs** (arg), **type** (arg) - type of data eg. int, float 4.3, string etc.
* **from os.path import exists** 🡪 **exists**(filename) - returns True if a file exists, else False

**FILES INPUT-OUTPUT:** Writing text, reading from a file

* Download **iTerm** 🡪 better than mac’s inbuilt terminal
* **WITH** *open(“filename.txt”, “w”)* **AS** *file*: file.writelines(lines) 🡪 auto closes the file!
* **CREATE** a file:
  + **Touch** filename.txt (should be in the folder you want the file to be in!)
* **OPEN** a file
  + **FILE OBJECT:** filename = **OPEN (‘filepath/dir’, ‘mode’ – ‘r’ to read, ‘w’ to write, ‘r+’ to read & write, ‘a’ for append, “rb” for csv file)** eg. fob = open(‘c:/test/a.txt’,’w’)
  + If already created a file previously, just fileobject = **OPEN (“filename”)**
  + **ITERATE** over file: eg. file = open (“hello.txt”, “r”), **for line in file: print line**
* To **WRITE** files
  + **Fileobject**.**WRITE (‘**whats up \n line2 \nfinal line’)
  + Can loop over: open file, then **for** line in lines [“line1”, “line2”]: file.write(line)
  + Fileobject.**WRITELINES (**listx)
* To **READ** files:
  + **Fileobject**.**READ ( )🡪** if int parameter, reads # characters/byes of the file; w/o parameters, reads whole fie
  + print fileobject.**READLINE ( )**; to read a line upto first break: adds \n at the end of the line, but can change that by adding a comma
  + To read remaining lines, line by line, returns each line as a **list**: listx = **print** fileobject.**READLINES ( )**
    - Can edit listx [2] = “bahh”
* **CSV FILES:**
  + **Import csv**
  + **With open**(‘suess.csv”, “**rb**”(read) or “**wb**” (write)) **as** file: // same as a normal file
    - **To read:** for row in **csv.reader**(file):  
       print row // this is a list! **row[0]** = 1st col etc
    - **To write**: csv\_writer = **csv.writer**(file):  
       csv\_writer.**writerows**(lines) or **writerow**(singlerow)
  + All values from csv files are **STRINGS** 🡪 need to convert to ints if needed
  + With
* Other commands:
  + Fileobject.**CLOSE( )** 🡪 must close the files!
  + Filename.**CLOSED** 🡪 returns True/False i.e. file closed/not; no ( ) needed
  + Filename.**TRUNCATE** () 🡪 empties the file!
  + Indata = in\_file.read(); then **len** (indata) 🡪 # bytes input file is
  + Filename.**SEEK (n)** 🡪 seeks back to nth byte, 0 if it’s right to the beginning
  + To automatically \_\_enter\_\_() or \_\_exit\_\_() a file, call **WITH OPEN (“**filename”, “w”) **AS** textfilename: textfilename.**write**(“xxx”) 🡪 automatically buffers and closes the file without calling filename.close()
* Within Python Interpreter (**>>>**) or **IMPORT OS** in pycharm and create funcs
  + Library **OS** contains scripting utility functions
  + **os.getcwd()** 🡪 gets name of current director
  + **os.mkdir**(“dirname”) 🡪 makes new directory
  + **os.chdir**(“dirname”) 🡪 changes directory into new directory
  + **os.listdir(**“.”) 🡪 lists all files in **current** directory

[**ERRORS**](https://docs.python.org/2/tutorial/errors.html) **& DEBUGGING**:

* def somefunc () :
  + **TRY:** 10/0
  + **EXCEPT zeroDivError**: print “oops” *//* ***if error****, do this*
    - Can also be **except (RuntimeError, TypeError):** pass
    - Must be an ERROR, not just returning false!
  + **ELSE: pass** (exception didn’t occur) *// if no error, do this*
  + **FINALLY:** print “donzo!” (similar to Javascript) *// error or not, do this*
* **Types of errors:**
  + **NameError** - if name of var/funcnot defined/misspelled/no import stmt
  + **AttributeError** - misspelled part of function after period (eg. math.PI vs .pi)
  + **TypeError** - wrong # inputs into functions etc
  + **SyntaxError** - missing colons (:) etc, = vs ==
  + **ValueError**: if not a valid number
* **Debugging in PyCharm:**
  + Click on the LHS for red dot to “**DEBUG**” (right click and debug example – runs program till the line you choose) **(Ctrl Shift D)**

**PYGAME:**

*Initialization, User Input, Animation, Rendering*

* **Import pygame** … **Pygame.init()**
* Display surface: **Pygame.display:**
  + screen = **pygame.display.set\_mode(resolution = (width, height),** flags=0, depth = 32) 🡪 can leave flags, pixel depth out; res (640, 480)
  + **pygame.display.set\_capture(**“hello world”)
  + **screen.fill**((255,155,55)) and then **pygame.display.update()** 🡪 draws orange screen
  + **pygame.display.info()** 🡪 info on screen res of comp etc.
  + **pygame.display.flip()** 🡪 will flip surfaces
* Display surface:
  + surface = **pygame.Surface**((width, height), flags = 0, depth = 30)
  + surface1.**BLIT(Surface2, location) 🡪** using Block Image Transfer, draws surface2 on surface1 or **screen.blit(**pygame.image.load(‘car.png’), (50,100))
  + surface.**FILL(**color, rect=None) 🡪 fills surface with inputted color
  + surface.**get\_rect()** 🡪 gets rect area of surface
  + **ANIMATION**: for i in range(100): **screen.fill**((0,0,0) **screen.blit**(car, (i,0))
* **Input handling:**
  + **Pygame.event.wait/poll/get()** 🡪 waits for next event / sees whether any other events for processing else returns NOEVENT / returns all outstanding events
* **Shapes**:
  + **Pygame.draw.circle/polygon/line/rect (Surface, color…)**
  + **Rect = pygame.Rect((left,top), (width,height))**
  + Can dorect1.**move(xy)** or **.clamp**(Rect2) or **.union/.clip/.collide…**
* **Events:**
  + Pygame.**KEY.GET\_PRESSED ()** 🡪 returns state of ALL the keys
  + Pygame.**TIME.WAIT(milisecs)** 🡪 waits the amount of time in milisecs
* **TIMING:**
  + Clock **= pygame.time.Clock()**; FRAMES\_PER\_SECOND = 30; clock.tick(FRAMES\_PER\_SECOND) 🡪 clock object pauses until 1/30th of a second has pissed to call tick; limits 30 ticks per second
* **KEYS:**
  + **EVENT.TYPE** == **KEYDOWN** ie. Checks if key is down or up
  + If **EVENT.KEY** == **K\_RIGHT/K\_LEFT/K\_DOWN/K\_ESCAPE**
* **PYGAME.MOUSE**:
  + pygame.mouse.**get\_pressed()** – returns l/center/r of mouse button state
  + pygame.**mouse.get\_pos()** – returns x, y position
  + **.get\_rel()** 🡪 returns x, y movement since last, can move objects 2x as fast
  + **.get\_focused()** 🡪 returns active if mouse is in/out of screen
  + **.set\_visible(bool)** 🡪 allows you to if mouse if visible or not
  + **.set\_pos()** 🡪 allows you to set position of mouse
* **Exit:**
  + To close game on closing the screen: **for** event **in** **pygame.event.get():** if event**.type == pygame.QUIT(): sys.exit()**
* **SPRITES:**
  + **Creates a class of an image etc.which holds it’s information**
  + **Class** CarSprite**(pygame.sprite.Sprite):**MAX\_SPEED = 10 **def \_\_init\_\_ (self,** image, position):  
    **pygame.sprite.Sprite.\_\_init\_\_(self)  
    self.**src\_image = pygame.image.load(image)  
    …
  + Can check for **“collisions”:** collisions **= pygame.sprite.spritecollide (**car\_group, pad\_group)
  + Can also **DRAW** and **CLEAR:** car\_group.**clear**(screen, background)
* Other: Pygame.**fullscreen,**

**SIMPLE GUI**

* Basic
  + import simplegui
  + FRAME (have to frame.start!!!!), CANVAS
  + Check-list: Global variables, Helper functions, Classes, Define event handlers --> create frame, register event handlers, start frame & timers (frame.start, timer.start)
  + def Input\_handler (text) - text is ALWAYS a string! Best to use float (text) to convert to a number
  + PRINT and label values - input and result - to debug! (Eg. print "x input = ",  x)
* Canvas - DRAW HANDLERS
  + simplegui.create\_frame("Title", width, height)
  + ORIGIN is always in upper right
  + define draw handler - def draw (canvas): canvas.draw\_text ("Hello', [100,100], 24, "White")
  + register draw handler - frame.set\_draw\_handler (draw)
* Timer
  + timer=simplegui.create\_timer (interval, handler); interval is in milliseconds ie 2000 = 2 secs
  + handler - def tick  ()
  + Diff event handlers don't interfere with each other - Tick handler doesn't draw, only times; Draw handler doesn't time, only draws!
  + Keyboard inputs:
  + frame.set\_keydown/up\_handler (keydown/up)
  + simplegui.KEY\_MAP ("left") (or right, up, down)
* MOTION
  + p(t + 1) = p(t) + 1 \* v(t) --> p[0] = p[0] + v[0]; p[1] += v[1]
  + Point = point + vector --> p = p + a\*v; p[0] += a\*v[0] - Modify each element of the vector at a time
  + RHS of the screen - if width = 200, last pixel is width - 1 = 199, so if p[t] > 199, off the canvas
  + Collision with wall:
    - Ball hitting left wall - if ball\_pos[0] <= Ball\_radius: vel [0] = -vel[0] --> continues y trajectory, change x trajectory
    - Ball hitting right wall - if ball\_pos[0] > (width - 1) - Ball\_radius : vel [0] = -ve[0]
    - Ball hitting top/bottom wall - same but ball\_pos [1] and modify vel [1]
  + Velocity control - p[0] += v[0]; key handler - v[0] += c (right arrow) - but use it on keydown, stop on keyup
* MOUSE
  + frame.set\_mouseclick\_handler (mouseclick\_handler); def mouseclick\_handler (position): --- (position - tuple of 2 integers)
* FOR Loops:
  + for ball\_pos in ball\_list: canvas.draw….
  + Can NOT remove elements from a list if iterating over that specific list!
  + return [n \*\* 2 for n in numbers] OR for n in numbers, result.append (n\*\*2)
* IMAGES
  + im = simplegui.load\_image(URL)
  + canvas.draw\_image(im, src\_center, src\_size, dst\_center, dst\_size) - last 4 are 2 data points each; src center/size: metrics of image, dst center/size: metrics of the canvas

**BITWISE PROGRAMMING**

* **Base 2:** In python, it’s 0bxxxxx, starting from the right where the right most is 2^0 then to it’s left is 2^1, then 2^2… so 1 = 0b10 and 12 is 0b1100 = 2^3 + 2^2 + 0 + 0
* **BIN** (num) 🡪 takes an int and returns binary rep of the int as a string
* **OCT (**num**) or HEX(**num**)** 🡪 base 8 or 16
* **INT (“#”, base)** 🡪 converts the number from it’s current base to base 10
* **Shifting bits right/left** **>>** or **<<** 🡪 moves all the 1s over (left or right) the number of bits Eg. 0b000101 << 3 = 0b101000 (left bit shift 3)
* **AND (&)** - Compares 2 bit #s and returns number where the bits of **BOTH** the numbers is 1 Eg. a 00101 and b 01100 🡪 a&b = c 01000 i.e. only 1&1 = 1, 0/1 & 1/0 = 0
* **OR (|)** – Compares 2 bit #s and returns number where **EITHER** of bits are 1
* **XOR (^)** – Returns number where bit of **EITHER, but NOT BOTH** numbers is 1
* **NOT (~)** – flips all bits in a single number => adding 1 to it and reversing sign Eg. ~1 = -2 and ~-42 = 43
* **BIT MASK** 🡪 variable that helps with bitwise ops.
  + Eg To find if 3rd bit of num is on or off (1 or 0), num & mask (0b0100) > 0🡪 bit is on!
  + Eg. To turn a bit on whether it’s on or off – num | mas (0b1)
  + Eg. To flip bits, use XOR

## 

# **3. DJANGO**

Videos: [Kenneth Love](https://www.youtube.com/watch?v=KZHXjGP71kQ&feature=youtu.be) or [Eventbrite dude](https://www.youtube.com/watch?v=NfsJDPm0X54)

Isolation - virtualenv

Determinism – dependency management eg. setup.py file, pip, buildout, specify versions!

Similarity – Vagrant: tool for managing virtual machines for dev purposes

**Points:**

* Rails etc. use MVC (model, view, controller) approach vs. Django **MTV (model** - data**, template/view –** visual output i.e. data display/HTML**, view/controller –** logic between model and template i.e. what data outputs in the view**)**
* **VIRTUALENV:** Allows use of diff python/django packages/new or old versions in each virtualenv without conflicts
* Bash Profile path: **open ~/.bash\_profile**

[**To deploy Django**](http://effectivedjango.com/latex/EffectiveDjango.pdf) **:**

* **STEPS to connect to an existing virtual project:**
  + CD into django folder
  + Connect to virtualenv - **WORKON virtualenvname** (just WORKON gives list of virtualenvs)
  + In new Term tab - Run DB: **PSQL POSTGRES** and then **\CONNECT** **dbname**
  + In old Terminal: Connect to Django interpreter $**python manage.py** **SHELL**
* **NOTES**:
  + To “deactivate” i.e. exit from virtualenv 🡪 **EXIT** or **DEACTIVATE;**
  + To read Django code – **CMD+mouseover** a class/object etc
  + To run file in Django shell: >>> **execfile**(‘path/filename.py’)
  + Django **SuperUser**: gmail, roxnairani, django(axe)

**TO CREATE A BRAND NEW PROJECT**

* **SET UP:**
  + CD into folder you want to create your project
  + **mkvirtualenv** foldername
    - Make sure prompt changes to (virtualenvfoldername) $ …
    - Can check $**which** python 🡪 should be in virtualenv folder
    - If made virtualenv in wrong place, **rmvirtualenv** and then re-create
  + $ **pip install django** == 1.6.2 (latest version if == *version* not specified)
  + **$ pip install psycopg2** // python package helps python talk to postgresql
* **CREATE NEW PROJECT**:
  + **$ django-admin.py** **startproject** projectname **.** (eg. addressbk)
  + Contains **settings.py** (specify DB connections, template locations etc.), **urls.py** (to write your URLs), **wsgi.py**
* **Create DB for PROJECT & link to postgres** for project:
  + Run postgres:
    - **psql postgres** then **create database** dbname; then **\connect** dbname
    - If DB not running - $ postgres –D /usr/local/var/postgres
  + In level with manage.py, create a new file called **local\_settings.py** and paste code below to link project to DB created:
    - **DATABASES = {‘default’: {‘ENGINE’: ‘django.db.backends.postgresql\_psycopg2’, ‘NAME’: ‘***dbname’***,}}**
  + Inform **settings.py** about local\_settings.py: At the bottom of settings.py, add:
    - **try: from local\_settings import \* except ImportError: pass**
  + Tell **PyCharm** about virtualenv:
    - PyCharm Prefs –> Project Interpreter –> Python Interpreter –> Settings/Local –> Home Direc - .virtualenvs – writers – bin – python
    - PyCharm Prefs 🡪 Django - 1st line, show link to working directory (where files are) and in 2nd line, link it to settings.py (not local\_settings.py)
    - **Run Django** - Green play button in pycharm or **CD into projectname** and then **$python manage.py runserver**
      * Links to 127.0.0.1:8000/admin
    - If it doesn't:
      * Drop-down by green play - Edit Configurations
      * + Django Server, Give it a relevant name and make sure Prefs (Interpreter and Django) are linked to the virtualenv and working directory respectively
* **INSTALL / LINK TO SOUTH**:
  + **$pip install south** (in virtualenv!) *#won’t have to do in Django 1.7*
  + Add **‘south’** to settings.INSTALLED\_APPS
  + CD into project folder, then: **$python manage.py syncdb** - syncs models to DBs / tables; should have appname\_classname table
  + Create **superuser** (if no auto pop-up - $python manage.py **createsuperuser**)
* **CREATE AN APP:** 
  + Inside folder of manage.py in terminal: **$ python manage.py startapp** appname (eg. blog)
* **REGISTER APP** with the project:
  + In **settings.py**, under **INSTALLED\_APPs** tuple, add ‘appname’
* **MODELS (DATA TABLES FOR APP) in models.p**y:
  + **Model field types**: AutoField (auto-incrementing int i.e. primary key), BooleanField, CharField, Date/DateTime/Decimal/Email/File/Image/Integer /Text**Field**
  + **Model Inheritance:**
    - ***Abstract Inheritance:***
      * *Abstract Models parent class doesn't have a DB – eg.* ***AbstractUser***
      * *Can create an Abstract Class & have other classes inherit from it (eg. name attribute); declare Abstract in class (models.Model):* ***class Meta: abstract = True***
    - ***Multi-table inheritance:***
      * Models inherit from other models; be cautious using it
  + **Class** classname (**models.Model):** *//handles CRUD tasks, DB elements etc*
    - ***Note****:* ***ID*** *key auto-added UNLESS o/w specified as* ***PRIMARY KEY***
    - first\_name = **models.CharField**(**max\_length =** 255)
    - email = models.**EmailField(null=True** i.e. can be null!**)**
    - pub\_date = **models.DateTimeField (‘**date published’)
    - **slug = models.SlugField (..)**
    - image = **models.ImageField(upload\_to=**’card\_images’, blank=True, null=True)
      * **src=”{{**card**.images.url}}”** – gets URL, w/o url, would get path!
    - SHIRT\_SIZES = ((‘S’, ‘Small’), (‘M’, ‘Med’))  
      shirt = **models.CharField(max\_length=1, CHOICES=**SHIRT\_SIZES)  
      *Note: In template, can use “shirt.****get****\_shirt\_****display****”) to display “Small” instead of “S” that is stored in DB*
    - body **= models.TextField(null = True, verbose\_name=’**Text….’**)**
    - **In ONE-MANY, FOREIGN KEY (i.e. chicken feet) is on MANY:** i.e.*In class “Post” -* author **= models.ForeignKey(**className eg. Author, **related\_name**=’post’**)***Name the field name of the Table it relates to (here, author)*
    - **In MANY-MANY, FOREIGN KEY can go on either table** i.e.  
      *In* **class Tag – posts = models.ManyToManyField(**classNamePost, **related\_name = ‘tag’)***Django makes the THROUGH table automatically!* **def \_\_unicode\_\_(self): /**/same as \_\_repr\_\_ but allows non-Eng chars **return u”{}”.format(self.**name**)**
  + **CUSTOM USERS:**
    - **Settings.py:** add **AUTH\_USER\_MODEL =** “**appname**.**modelName**” (eg. cards.Player)
    - **Models.py:** **class** Player(**AbstractUser**): phone=models.CharField…
    - **Forms.py:** Update class Meta: model = **Player**; Override **clean\_username**(self): *explicitly pass “Player” not “User”*
    - Run schemamigration, python manage.py createsuperuser (if don't have 1)
  + **Note:**
    - See all tables (appname\_tablename) in Terminal **psql writers** … **\dt**
    - To see all Django commands – **python manage.py help**
* **UPDATE admin.py:**
  + Add: **from** appname.**models** **import** model1, mod2,
  + Update: **class CommentAdmin(admin.ModelAdmin)**:
    - **Fieldsets = [ (None, {‘fields’: [**‘question’]}), (‘Date info’, **{fields’: [**‘pub\_date’], **‘classes’: [‘collapse’]})**
    - **list\_display** = (‘name’, ‘user’, ‘funcname’) *- shows in website/admin*
    - **List\_filter = [**‘pub\_date’]
    - **Search\_fields = [**‘question]
  + Register & Relate them **admin.site.register(Comment, CommentAdmin)**
* **MIGRATIONS (SOUTH):**
  + **Transition:**
    - **First time** to create DBs in South**:   
      $ python manage.py schemamigration** appname **–-initial**

**Every time after:**$python manage.py **schemamigration** appname **–-auto**

* + - * *Will then have a folder migrations with init.py & initial.py (with* ***forwards*** *methods – changes being made,* ***backwards*** *methods – how to undo changes, &* ***models*** *- state of DB)*
    - Apply those changes: $python manage.py **migrate** appname

| **Step** | **South** | **Django migrations** |
| --- | --- | --- |
| initial migration | 1. run syncdb 2. then ./manage.py schemamigration <appname> --initial | ./manage.py makemigrations <appname> |
| apply migration | ./manage.py migrate <appname> | ./manage.py migrate <appname> |
| non-first migration | ./manage.py schemamigration <appname> --auto | ./manage.py makemigration <appname> |

* ***ADD DATA IN MODELS (TABLES)***
  + ***From*** *blog.****models******import*** *Author  
    Author.****OBJECTS.CREATE(name=****‘xro’,* ***…****) (Creates & saves new object!)  
    cards = [Card{suit=suit, rank=rank) for suit in suits for rank in ranks]*Cards.objects.**BULK\_CREATE(cards)** *🡪 bulk creates 52 objs with 1 hit to DB*
  + *For* ***ONE-MANY****, to* ***link*** *2 classes (tables), can do:  
    >>>* ***author*** *= Author.objects.get(pk=2)  
    >>> Post.****objects.create****(title=’xx’ body=’xx’, author=****author)***
  + *For* ***MANY-MANY:****To add data in through tables – eg. link posts-tags:  
    >>> food\_tag = Tag.objects.get(pk=1)  
    >>> food\_tag.post****.ADD(2, 3, …)*** *🡪 Add tag ‘food’ to posts 2, 3 and …****Can ADD (PRIMARY KEY) or can ADD (OBJECT)***
* **UPDATE/DELETE DATA**
  + author = Author.objects.get(pk=4)  
    **UPDATE**: author.twitter = ‘@NewTwitter’  
    **SAVE: author.save()**
  + **DELETE: author.delete()**
* **QUERY DATA:**
  + In Django, methods (.method( )) map to SQL’s queries (eg. WHERE, FROM...)
  + In Django shell – **python manage.py shell**

**GET ALL:**

* + - >>> all authors = Author**.objects.ALL()** *// equiv. to SELECT \* FROM author from Model Author  
      >>>* **for** author **in all\_authors:** *// all\_authors is an iterable variable!* print author.name, author.twitter

[**GET SPECIFIC**](https://docs.djangoproject.com/en/1.6/ref/models/querysets/#exact):

* + - **SINGLE:** >>> dr\_seuss = Author.**objects.GET(pk**=1) where pk is primary key (or id); use GET only when there’s **ONE** object to get!
      * x = Model.objects.**SELECT\_RELATED(**‘field1’, ‘feld2’…).GET(ID=4); x.field1 doesn’t hit DB (vs. if you did .get() and then x.field1)
    - **MULTIPLE:** >>> authors = Author.**objects.FILTER** (~*where*)
      * ***// Filter outputs a LIST!*** *(so if only 1 item, use GET)*
      * **.filter(pk\_\_gt(e)/lt(e)**=1) 🡪 authors with id >(=)/ <(=)
      * .filter**(pk\_\_in**=[1,4,7]) 🡪 authors with pk 1, 4 and 7;
      * .filter(fieldname\_\_**isnull=False)** 🡪 looks for non-null fields
      * **MATCHES:**
        + .filter(name\_\_**starts/endswith/(i)exact/(i)contains**=’X’)
    - **.EXCLUDE(**colname\_\_gte=**datetime.date.today())** 🡪 opp. of “filter”
    - author.post.**COUNT( ), ANNOTATE( )**
    - **.AGGREGATE ( Avg/Max(**‘price’**)** 🡪 averages of price
    - .filter(…)**.VALUES(**‘id’, ‘name’**)** 🡪 DICTIONARY of details {name:’x’, id:1…}
    - **VALUES\_LIST** (‘id’, flat=True) 🡪 TUPLES of ids, flat=True 🡪 [1,2,..] instead of [(1), (2)…] one-tuples
    - **order\_by(‘**author’)**.DISTINCT( )** 🡪 must call distinct on an order\_by, returns a list of non-duplicate rows
    - **from django.db.models import F, Q**
      * **F** compares value of 2 fields on **same model:** .filter(comments\_\_gt=F(‘likes’)) i.e. # comments > # likes
      * **Q** executes queries with **OR** ( **|** )or **AND** statements:  
        **.get(Q** (pub\_date==date(2005,5,2) **|** **Q**(pub\_date=date(2005,5,6))
  + **New instance** of an object**:** new\_blog**.pk = None,**  new\_obj.save()
  + Use **.EXISTS()** vs. if loop, **count()** vs. **LEN()** for efficiency
  + Author.objects.**EXTRA(select={**‘is\_recent’: “pub\_date > ‘2006-01-01’”}) – adds extra attribute to results

**ORDER / LIMIT**

* + - >>> alphabet\_authors = Author.objects.**ORDER\_BY (‘**name’) or (‘pk’)
    - For RANDOM ordering (1st 2 elements): Card.objects.order\_by**(‘?’)**[:2]
    - >>> reverse\_authors = Author.objects.**ORDER\_BY (‘**-name’) 🡪 DESC
    - **SLICE** lists to LIMIT: >>> authors = Author.objects.all() **[:1]** 🡪1 result!
    - Queryset.**REVERSE( ) [:5]**  🡪 reverses i.e. gives LAST 5 objects
  + **ONE-MANY QUERIES:**
    - **Get post obj from post\_id** using: post = Post.objects.get(pk=1)  
      -- > author=post.author & authorID = post.author\_id
    - To **join/query across** tables: **tablename\_\_columnname**eg. posts = Post.objects.filter**(author\_\_name**=’Dr. Suess’)
  + **MANY-MANY QUERIES:**
    - Filter same way as ONE-MANY: posts = Post.objects.filter(tag\_\_name=’food’)

**URLS**

* **urls.py** file**: url (r’^**hello**/(?P<**VarName**>\w+)$’, ‘**appname**.views.**hello**’, name=**hello**),** where:
  + **First ‘hello’**- url for the page
  + **Last ‘hello’** - name of the **FUNCTION** in the views.py file to pull from!
  + **VarName** – allows to capture variables in URL; must add variablename as an extra parameter in the views function:

*url(r'^genres/$', 'hollywood.views.genres', name='genres'),*

*url(r'^genres/new/$', 'hollywood.views.new\_genre', name='new\_genre'),*

*url(r'^genres/(?P<genre\_id>\w+)/$', 'hollywood.views.view\_genre', name='view\_genre'),*

*url(r'^genres/(?P<genre\_id>\w+)/edit/$', 'hollywood.views.edit\_genre', name='edit\_genre'),*

*url(r'^genres/(?P<genre\_id>\w+)/delete/$', 'hollywood.views.delete\_genre', name='delete\_genre'),*

* **Errors:**
  + **Error 404** – url doesn't exist ie add the link in urls.py
  + **View not found** – url.py updated but views.py isn’t!

**VIEWS:**(~templates in Ruby) ie. **FUNCTIONS** incl. **DB QUERIES** etc!

A python callable that takes a **request** object & returns a response  
*a) INDEX b) NEW c) SHOW (Detail) d) EDIT e) DELETE*

* **Import statements:  
  from django.shortcuts import render, render\_to\_response, redirect  
  from django.contrib import messages  
  from** appname.**models import** Model1, Model2…  
  **from** appname.**forms import** Model1Form, Model2Form …
  + **Def** hello **(request,** varname**):**
    - *Data = {“form”: form, “actor”: actor}*
    - **Return render** (request, “x.html”, data)  
      *Note: varname enters as a string; needs to be int() if needed*
  + **New or edit form  
    Def** signup\_form**(request,** signup\_id**):  
    signup = SignUp.objects.get(id=signup\_id)  
     If request.method == “POST”  
     form =** SignUpForm**(request.POST, instance**=signup**)**

**If form.is\_valid():  
 if form.save()** *if model form* *or*  
 email = **form.cleaned\_data**[‘email’] etc *for all elements in form* Author.**objects.create**(email=email, ….) to save the obj!  
 **messages.success**(request, “Thank you! We will be in touch”)  
 *In template view:  
 {% if messages %} {% for message in messages %}*

{{ message }} *{% endfor %} {% endif %}* **return redirect** (‘/…/’) or **return render(request, /…/, messages)**  
**else**:  
 form = SignUpForm(instance=signup)  
data = {“form”: form}  
return render(request, ‘signup.html’, data)

* **Delete records:  
  def** delete\_genre**(request,** genre\_id**):** genre = Genre.**objects.get(id=**genre\_id**)** genre.**delete()  
   return redirect**(“/genres”)
* **Generic Views:**
  + **class** XView**(generic.Detail/List/Form/Template/Create/ RedirectView)**
    - **ListView –** provides a view of a set of objects
  + **Create a view:**
    - **In views.py:** Import:
      * **from django.views.generic import** ListView
      * from appname.models import Model
    - Class ContactListView(**ListView):   
       model =** Contact  
       **template\_name** = ‘contact\_list.html’
    - **In url.py**: url(… , appname.views.ContactListView**.as\_view(),** name=…)

**TEMPLATES**

* **$mkdir templates** within **APP** folder
* Within templates, create pagename**.html** file with relevant html coding
* To link between pages **TEMPLATE TAGS**: **<a href =“{% url “pagename” %}”>**Text</a> or **<a href =”\pageurl”>**Text</a>
* **Display Forms**:  
  **<form method=”post”>  
  {% csrf\_token %}  
  *{{*** *form* ***}} {{ form.error }}****or* ***{% for field in form %}   
  <div style=”background-color: blue; …”>  
  {{ field.label\_tag}} {{field}} </div>  
  {% endfor %}*  
  <input type=”submit”, class = ‘btn btn-success’ value=”Update”>** or **<input type=”radio” name=”choice” idd =”choice{{forloop.counter}}/>  
  </form>**
* **IF ERROR**: Create search path for templates: In **settings.py**, add **TEMPLATE\_DIRS = (os.path.join(BASE\_DIR,** appname/**‘templates’),)** to base of the file; *base\_dir only in django 1.6+*
* **INCLUDE:**
  + To avoid duplicating html, can **include** snippets of same html in diff html templates
  + **Mkdir includes** inside templates directory which contains html files; don’t directly map to pages (like the templates in main templates directory)
  + **{% include ‘includes**/hand.html’ **with** cards = dealer\_cards%}

**FORMS**:

* + Create **forms.py file** in APP
  + Have **WIDGETS** (HTML output to page), **FIELDS** (render/validate input), **FORMS** (collection of fields), **FORM ASSETS** (CSS/JS to render form)
  + **Import statements:  
    From django import forms  
    From** appname**.models import** Model1, Model2… or \*

**MODEL FORMS:  
class** Genre**Form(ModelForm):   
 class Meta:   
 model=**Genre  
**FORM OBJECTS:**  
**class** CommentForm**(forms.Form):** author = **forms.ModelChoiceField(queryset=**Author.**obects.all())** text = **forms.CharField(max\_length=20, required=False,**

**validators**=[no\_trash\_talk], **attrs={‘placeholder’:** ‘Text’) email = **forms.EmailField(EmailField/URLField** *– inbuild validation*)  
**Custom Validators:  
def** no\_trash\_talk(**value**):  
 **if re.match**(“crap|shady|…”, **value**):  
 **raise ValidationError** (“No trash talk allowed”)

* Forms are CLASSES and can be SUBCLASSES i.e. class Form(ParentForm)…

**NOTES ccbv.co.uk *(Classy Class-Based Views)*:**

* + **Get (request), Post (request)**
  + **Form\_invalid (form)**
  + From django.contrib.auth import **Authenticate, login, logout**
  + **Utils.py** (in app) for helper functions!

**STATIC FILES:**

* **Set up in app:**
  + **Mkdir static** with folders **img, js, css**
  + **Pip install pillow**
  + In **urls.py:** 
    - From django.conf import settings
    - **from django.conf.urls.static import static**
    - **if settings.DEBUG: urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)**
  + Before file that imports local\_settings in **settings.py:  
    PROJECT\_ROOT = os.path.abspath(os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), '..'))**

**MEDIA\_URL = "/media/"**

**MEDIA\_ROOT = os.path.join(PROJECT\_ROOT, "static", \*MEDIA\_URL.strip("/").split("/"))**

* In **base.html**
* **{% load staticfiles %}** *//top of template*
* <img id="joker" width="40px" src="{% **static** "img/red\_joker.jpg" %}">

[**USERS**](http://rocketu.yetihq.com/week4/2_am)**:**

* **Notes:**
  + **Is\_staff –** allows admin access (BooleanField)
  + **Username** –(unique=”True”) (vs. email, need to make unique)
  + *In views:* **USER = REQUEST.USER** and then **USER.NAME** etc
  + *In templates*: **USER.NAME** directly
  + Only **1 USER model** per project!!
* **REGISTER** needs a url, a view (create using **(Updated)User CreationForm**, isvalid/save/redirect etc.), a template
  + **From django.contrib.auth.forms import** **UserCreationForm**def register(request):  
     if request.medthod == “POST” ….  
     …. If form.is\_valid(): **user=form.save user.email\_user(‘subj’, ‘message’)** return redirect(“profile”)
  + To override form with new custom fields, create forms.py
    - **Class UpdatedUserCreationForm(UserCreationForm):** email = forms.EmailField(required=True)  
      class Meta:  
       model = User  
       fields = (“first\_name”, “last\_name”, “username”, “email”, “password1”, “password2”)
* **LOG IN** has a view; need to create url (r’^login/$’, **‘django.contrib.auth.views.login**’, name=’login’), template
  + Update **settings.py** with **LOGIN\_REDIRECT\_URL = 'profile'** (normally “/accounts/profile” assuming have a ‘profile’ page!)
* **LOG OUT** create url **(url(r'^logout/$', 'django.contrib.auth.views.logout', name='logout'),** and template
  + ***Note****: if LogOut leads to admin logout, ensure appnames in settings are right at the top of INSTALLED\_APPS*
* **Request.user** 🡪 can use anytime! Always gives you current logged in user
* *In view: call request.user (?)*
* In template: call **{{user.username}}** or **{{user.last\_login}} {{user.date\_joined | timesince }}**
* **Update logout/login links** based on user’s logged in/out status in base.html:<span style="float:right;"> **{% if user.is\_authenticated %}**

<a href="{% url **admin:index** %}">Admin</a> | <a href="{% url 'logout' %}">Logout</a>

{% else %} <a href="{% url 'login' %}">Login</a> | <a href="{% url 'register' %}">Register</a> {% endif %} </span>

* **Ensure view based on logged in or not**: In view:
  + First update settings.py with **LOGIN\_URL = ‘login’**
  + **Import** login\_required decorator and add decorator over a view  
    **@login\_required**  
    def profile(request): return render(request, ‘profile.html’)
* **RESET PW***:*
  + **Add 4 Urls.py**:
    - url(r'^password\_reset/$', 'django.contrib.auth.views.password\_reset', name='password\_reset'),
    - url(r'^password\_reset/done/$', 'django.contrib.auth.views.password\_reset\_done', name='password\_reset\_done'),

*# Support old style base36 password reset links; remove in Django 1.7*

* + - url(r'^reset/(?P<uidb36>[0-9A-Za-z]{1,13})-(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$', 'django.contrib.auth.views.password\_reset\_confirm\_uidb36'),
    - url(r'^reset/(?P<uidb64>[0-9A-Za-z\_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$','django.contrib.auth.views.password\_reset\_confirm', name='password\_reset\_confirm'),
    - url(r'^reset/done/$', 'django.contrib.auth.views.password\_reset\_complete', name='password\_reset\_complete'),
  + [**Templates**](http://rocketu.yetihq.com/week4/2_pm/#/1/2)**:**
    - password\_reset\_form.html (with form)
    - password\_reset\_done.html
    - password\_reset\_confirm.html
    - password\_reset\_complete.html *(Yay! + link to login)*
  + **SENDING AN EMAIL THROUGH DJANGO:** *(usually from* ***views*** *or models)*
  + In **settings.py** (for gmail): **EMAIL\_USE\_TLS** = True **EMAIL\_HOST** = ‘smtp.gmail.com’ **EMAIL\_HOST\_USER** = ‘your\_email@gmail.com’ **EMAIL\_HOST\_PASSWORD** = ‘your\_password’ **EMAIL\_PORT** = 587 **DEFAULT\_FROM\_EMAIL =** ‘your\_email@gm..’
    - *For scale, use Mailgun or Mandrill*
  + ***In views.py****:* 
    - **from django.core.mail import EmailMultiAlternatives**

from **django.conf** import settings

from projectname import **settings**

* + - *In specific view: where* ***user=form.save()*** *or* ***user=request.user***
      * **User.email\_user**(‘subject’, ‘text’)   
        *or*
      * **text\_content** = 'Thank you for signing up for our website, {}'.format(user.username)
      * **html\_content** = '<h2>Thanks {} for signing up!</h2> <div>I hope you enjoy using our site</div>'.format(user.username)
      * **msg** = EmailMultiAlternatives("Welcome!", text\_content, settings.DEFAULT\_FROM\_EMAIL, [user.email])
      * **msg.attach\_alternative**(html\_content, "text/html")
      * **msg.send()**

**DJANGO DEBUG TOOLBAR:**

* **Pip install django-debug-toolbar**
* Add debug\_toolbar in INSTALLED\_APPs
* Add INTERNAL\_IPS = (“127.0.0.1”, “10.0.2.2”) in local\_settings.py
* Reload page, and RHS of page should have debugging tool – use “Templates”, SQL (for length to run queries) etc.

[**DJANGO PACKAGES**](djangopackages.com)**:**

* + - **To install:**
      * **Pip install** xxx; add in **INSTALLED\_APPS**
    - **Mezzanine** – robust CMS package; allows to put backend on to edit content, write blogposts etc
    - **E-Commerce**: payments, SEO etc.
    - Calendar, Messaging,
    - **Gallery** – for images

**DJANGO HTML**

* **{# … #} - COMMENT**
* **{{ variable }}** 🡪 evaluates variable and replaces with the result
* **FILTERS - PIPE: {{ object | filter }**
  + **{{** name **|** **LOWER/RANDOM** **}}** 🡪 displays name.lower()/random!
    - In views.py – Card**.objects.order\_by(‘?’)[:1]**
  + {{value|**LENGTH**}} 🡪 length of the value
  + {{ value|**STRIPTAGS** }} 🡪 strips all html tags <b> Jo </b> 🡪 Jo
  + **Chained Filters:** {{ text|escape|linebreaks }} 🡪 escape text contents and then convert breaks to <p> tags
  + **Filters w/ args:** {{ bio|**truncatewords**:30 }} 🡪 1st 30 words of bio var or {{ list|**JOIN: “, “** }} 🡪 join a list with commas and a space
  + {{ value|**DEFAULT: “nothing” }}** 🡪 if variable is false, use default
* **TAGS**: **{%** tag **%}** …tag contents … **{%** endtag **%}**🡪 ~calling a function
  + **FOR**: **{% for** x **in** x\_list **%}** <li> {{ athlete.name }} </li> **{% endfor %}**
    - **index.counter** 🡪 gives you index of each item in x\_list
  + **IF/ELIF/ELSE:** **{% if** x\_list **%}** Num is: {{ x\_list | length }} **{% endif %}**
  + **Others: {% != < > not and or in %}**
  + **TEMPLATE TAGS:**
    - **{% now** “SHORT\_DATETIME\_FORMAT” **%}** –current timestamp
    - <p class = **{% cycle** “even” “odd” **%}** –current timestamp
      * *Customize class***: <style> .odd {color: blue;} </style>**
* **BUILD OWN FILTER/TEMPLATE TAG:**
  + **Mkdir templatetags** in app
    - Create **\_\_init\_\_.py** file *(makes it a PACKAGE, not just a folder)*
    - Create **list\_filters.py**
      * + **@register.filter  
          def** suit(list, suit\_type):  
           return [item for item in list if item.get\_suit\_display() == suit\_type]
    - Call it in templatename.html:
      * **{% load list\_filters %}**{% for card in cards|suit:”diamond” %} …. {% endfor %}
* **BLOCKS** & **EXTENDS:**
  + **{%** **EXTENDS “base.html” %}** at top of child template
    - Where you want it, **{% block blockname %} {% endblock %}**
  + In child template: **{% BLOCK** blockname %} *Content* **{% endblock %}**
* To use **VI**  (Visual Editor) (or emacs) – text editors built into terminal:
  + **Create** file: $ **vi** filename.txt
  + Quit vi: $ZZ (when in ‘Command mode)
  + Move into edit mode (**a**) or command mode (**ESC**)

# **4. TERMINAL, GIT, HEROKU etc.**

**GIT:**

* **Distributed Version Control System**, a series of snapshots (commits) of your code. You see a path of these snapshots, in which order they where created. You can make branches to experiment and come back to snapshots you took. Stays local unless you commit your code to GitHub (or..)
* **GIT IN LOCAL** (on your own computer); **Inside MASTER folder**:
  + **$ git init** (once for new project)
  + **$ git status** 🡪 Keeps track of files, lists “untracked” files
  + Every time you save: (stores locally)
    - **$ git add .** *or*folder/filename eg. js or css/style.css 🡪 add everything or specific file/folder
    - **$ git commit** **–m** “First commit” 🡪 Saves/commits to git with message “ xx”
  + **$ git log** 🡪 tells you list of changes and who made them on specific branch
* **GIT ON REMOTE** (work accessible to other people online)
  + **$ git remote**
  + Go to [**GITHUB**](#_GIT_HUB), make a new repository in your account
  + Then under SSH key, copy-paste lines under “Push an existing repo” in terminal (if already initialized i.e git init) else lines under “Create new repo”
  + Enter SSH pw (StillyGandu)
* **GIT BRANCHES:**
  + **$ git push *origin master*** (or *branchname*)🡪 pushes code to origin, branchname (main branch is *master*)
  + **$ git branch** 🡪 lists the branches of your project
  + **$ git checkout –b** *branch\_name* 🡪 creates a new branch and switches to it
  + **$ git checkout** *branch\_name* 🡪 switches to working on branch\_name
    - In finder, git folder shows files specific to branch you’ve “checked out”
    - If *checkout new\_branch* and make changes and then *checkout master* $ git log won’t show those changes!
  + **$Git push origin** *new\_branch*🡪 pushes new\_branch (or master branch) ***from local to origin (remote)***
  + $Checkout new\_branch:: **$ git pull origin *master*** 🡪 pulls changes ***from remote (online) master to local*** (not in new\_branch on github until you push them)
    - Opens up vi 🡪 **:wq** to quit vi
  + $Checkout master: **$git merge new\_branch**🡪 merges changes/files from new\_branch into master on LOCAL
* **CONFLICTS**
  + If try to merge BRANCHES and there’s a **conflict** (eg. same file says diff things in diff branches) - MUST IMMEDIATELY FIX:
    - OPTION 1: $open file.txt, fix what you want, git.add, git.commit
    - OPTION 2**: $git mergetool,** Enter, choose version you want, save!
* **$ subl .gitignore** 🡪 create ignore file in root project dir, edit ignore file in Sublime for files you don't want tracked– add \***.pyc, db.sqlite3,** .\*swp
* **Notes**:
  + If don’t want a folder to be a git repo:
    - Within the folder: **ls –la**
    - Should show a hidden **.git** file
    - Remove.git and it’s not longer a git repo! **rm –rf .git**
  + Each project should have OWN repository!

### GIT HUB

* A webpage on which you can publish your Git repositories and collaborate with other people.