* Introduction
  + WEB2PY: web development with python
    - Integrate numerous technologies
    - Python
    - Bootstrap
    - SQLite
    - Rocket web server
    - Browser base IDE
  + Full Stack Web Development
    - Front end
    - Server code
    - Database
* Course Overview
  + Download WEB2PY
  + MVW, SQLFORMS
  + Database Admin
* Download and Start WEB2PY
  + Download WEB2PY
  + Start web server
  + Model-View-Controller
  + Go to web2py website
  + In the application folder
    - Actual source code will be in there
  + To start the web server, click on the web2py executable
  + It will open a default welcome app
  + Admin on the welcome app
    - Browser based ide that allows us to access and edit code
* Model-View-Controller
  + controller
    - python code
  + view
    - html files
  + MVC Convention
    - software architecture
    - naming convention also
    - def index(), returns message back to the view
  + view name
    - [controller]/[method name].html
* Build Our First Web Page
  + create a controller
  + inside controller create a method
  + ex) create controller name basics
    - def helloworld():
    - msg = “Hello from the Controller!”
    - return locals()
  + return locals()
    - take all local variables defined in method and return it to view
  + then create a view
    - [controllername]/[methodname]
  + ex)
    - basics/helloworld
  + inside the view, you can mix python with html
  + ex)
    - {{extend ‘layout.html’}}
    - <h1>Our First Web Page</h1>
    - <h2> {{=msg}}</h2>
  + {{}} double curly bracket means python code
  + to go to webpage use url [local address]/[application name]/[controller name]/[view name]
* Request Object
  + web application use http protocol
  + request object and response object
  + request object: anything passed from client to web server
  + arguments: part of url
    - request.args(0), request.args(1), …
  + variables: part of html forms
    - form
    - associated with post
    - request.vars.form\_name
  + in side controller
    - def request\_args():
    - arg1 = float(request.args(0))
    - arg2 = float(request.args(1))
  + have to convert arguments to float because they come across as strings
  + then create a view
    - basics/request\_args
  + url is then [local address]/basics/request\_args/[parameter 1]/[parameter 2]
  + instead of passing in parameters via url, can pass variables in html form using request.vars
  + ex)
    - def request\_vars():
    - num1 = 0
    - num2 = 0
    - if request.post\_vars:
    - num1 = float(request.post\_vars.num1)
    - num2 = float(request.post\_vars.num2)
    - total = num1 + num2
    - return locals()
  + then create a view
    - <form method=’post’>
    - <input type=’text’ name=’num1’/>
    - <input type=’text’ name=’num2’/>
    - <input type=’submit’ value=’Add’/>
    - </form>
* Response Object
  + Serve sends response back to application
  + Ex)
    - response.flash = T(“The total is “ + str(total))
    - T(): indicates translation
    - Web2py has translation feature that translate information into multiple languages based on the client
  + response.flash displays the data into pop up
* Business Rules & Libraries
* Deploying Our App
  + Pythonanyhwere
    - Simply and easy to use to host, run and code in python
    - Create a beginner account
  + Click ‘Web’ Tab
    - Click ‘Add a new web app’
    - Click ‘web2py’
    - Create Admin password
    - This will launch a web server with a web2py welcome application
    - Make sure the python version of web server is same for the code you wrote
  + Take our existing code in local host and publish to website
    - In the local host go to the administrative interface
    - Click on ‘Manage’ -> ‘Pack all’
    - It will package it into simple folder with .w2p extension
    - On the web server login
    - In the administrative interface, go to the ‘Upload and install packed application’ section
    - Name the application and upload the files(or get from git repo)
    - Click ‘Overwrite installed app’
    - Click ‘Install’
* Build a Blog App
  + SQLForms: inserting record and updating in database
* Design the Application and Da..
  + Full stack development is essentially connecting a database to a web application
    - Application design
    - Database design
  + Application design
    - Post
    - View
  + Database design
    - Data design
* Use Models and Validators to …
  + Create a model
    - Click edit on the db.py file
    - Scroll down to the defining tables section
  + By default web2py is a sql lite database
  + Define the table with fields
  + Ex)
    - db.define\_table(‘blog’, Field(‘blog\_title’), Field(‘blog\_details’), Field(‘blog\_image’), Field(‘blog\_url’), Field(‘blog\_category’), Field(‘blog\_date\_posted’))
  + Validators
    - Validate data before they go into database
    - Generate JS(jquery) code to validate data
  + Ex)
    - db.define\_table(‘blog’, Field(‘blog\_title’, requires=IS\_NOT\_EMPTY()), (Field(‘blog\_details’, type=’text’), …. Field(‘blog\_category’, requires=IS\_IN\_SET([‘News’, ‘Events’])), Field(‘blog\_date\_posted’, type=’date’, requires=IS\_DATE() )
* Build the Post Page with WEB…
  + Create a new controller
  + Method post to take form information an post it to database
  + Ex)
    - def post():
    - form = SQLFORM(db.blog).process()
    - return locals
  + db.blog references the blog table name from the model
  + define the view method to display return data
    - def view():
    - rows = db(db.blog).select(orderby=~db.blog.id)
  + Then build a view(blog/post)
    - {{extend ‘layout.html’}}
    - <h1>Post Blog</h1>
    - {{=form}}
* Update Records with SQLFOR…
  + accepted method allows us more control as we insert record into the database
  + ex)
    - def display\_form():
    - form = SQLFORM(db.blog)
    - if form.process().accepted:
    - session.flash = ‘form accepted’
    - redirect(URL(‘thanks’))
    - elif form.errors:
    - response.flash = ‘form has errors’
    - else:
    - response.flash = ‘please fil out the form’
    - return locals()
  + session.flash
    - need information available in the next page
    - moving to a new url
* Build the View Page
  + Ex)
    - {{extend ‘layout.html’}}
    - <h1> California Coast Blog </h1>
    - <hr />
    - <div class=’row’>
    - <img class=’…..’ src=’{{=x.blog\_image}}’ alt=’blog image’ />
    - <div class=’…’>
    - <p class=’..’>
    - <b><a href=’{{=x.blog\_url}}’ target=’\_new’>{{=x.blog\_title}}</a></b>
    - ….
  + Bootstrap formats the entire webpage into a 12 grid column
* Summary - Inserting and Up..
  + SQLForms are very powerful object in web2py framework
* Introduction to Database Adm..
  + import csv files
  + query database
  + inserts, updates, deletes
  + export csv, render json, xml
* Importing CSV Files
  + web2py has integrated database administration tool
  + sample data: <https://www.briandunning.com/sample-data>
  + create a new table
  + then go to main edit screen
    - click ‘database administration’
    - click on the newly created database
    - click ‘choose file’
    - click ‘import’
* Query Database Records
  + click on ‘database administration’
  + click on database
  + use the query
    - ex) db.contacts.first\_name==’Cory’
    - ex) (db.contact.state\_name==’CA’) | (db.contacts.state\_name==’LA’)
    - ex) ~(db.contacts.state\_name == ‘CA’)
    - ex) db.contacts.first\_name.startswith(‘A’)
* Insert, Update, and Delete Dat…
  + can insert, update and delete using the database administration tool
  + validators also work with database administration tool
  + can update individual records by click on them in the database and updating the info
  + or you can update multiple at once using the query and update fields
    - ex)
    - query: db.contacts.id > 495
    - update: company\_name=’General Mills’
  + have to check the box next to update
  + to delete have a query and click the delete checkbox
* Export CSV Files and Render …
  + create a new controller
    - def data():
    - rows = db(db.contacts).select()
    - return locals()
  + web2py gives us many options on how we want to render the data(html, json, xml, etc)
  + click on the database
    - at the bottom you can click ‘export as csv file’ to export the data
* Summary
* Introduction
  + filter, sorting, updating using code
  + CRUD(create, replace, update, delete)
* Sample Filters and Sorts
  + Ex) rows = db(db.tablename).select()
  + Special symbols
    - And(&)
    - Or(|)
    - Secondary Sort Order(|)
      * Sort by first condition then if same sort by second condition
    - Equalitities(==)
    - Inequalities(>, <, >=, <=)
    - Not or Descending Order(~)
  + Ex)
    - def filter():
    - rows1\_count = db(db.contacts).count()
    - rows2\_all\_sorted\_by\_name = db(db.contacts).select(orderby=~db.contacts.last\_name | db.contacts.first\_name)
    - rows3\_startswith = db(db.contacts.last\_name.startswith(‘M’)).select(orderby=db.contacts.state\_name | db.contacts.last\_name)
* Demo - Contacts App
  + Add methods to contacts controller to add and view data
  + Add views
* Introduction
  + Web2py has role based access control
* Account Registration
  + db.auth\_user: stores the account information
    - first name, last word, email etc
  + db.auth\_group & db.auth\_memberships are controlled by administrators
  + create a new record in db.auth\_group
  + then create a new record in db.auth\_membership to associate group with account
  + then write code to control access
* Access Control
  + Ex)
    - @auth.requires\_membership(‘blog\_poster’)
    - def post():
    - ….
    - @auth.requires\_login()
    - def view():
    - …..
* Miscellaneous Topics
  + Role Based Access Control = DB Tables
  + Third Party Authentication extensions
* Summary
* Relational Database Concepts
* Execute SQL
  + Click on ‘db.py’
  + Create a model
  + Input data
  + Write code to extract data
  + Create a new controller
    - def admin\_view():
    - sql = “SELECT products.id, product\_name, orders.order\_price, orders.order\_date “
    - sql = sql + “FROM products inner join orders on products.id = orders.product\_id;”
    - rows = db.executesql(sql)
    - return locals()
  + Create a view
    - <table class=’table’>
    - {{ for x in rows: }}
    - <tr>
    - <td>{{=x[0])) </td>
    - …..
    - {{ pass }}
    - </table>
* Dictionaries
  + ex)
    - def admin\_view1():
    - userdict = {}
    - userrows = db(db.auth\_user).select()
    - for x in userrows:
    - userdict[x.id] = x.first\_name + “ “ + x.last\_name
    - sql = “SELECT products.id, products. product\_name, orders.order\_price “
    - sql = sql + “orders.order\_date, orders.orderer\_id “
    - sql = sql + “FROM products inner join orders on products.id = orders.product\_id;”
    - rows = db.executesql(sql)
    - return locals()
* Reference Integrity
  + mechanism to ensure data integrity in relational databases
  + ex) in the db.py file
    - db.define\_table(‘orders’,
    - …..
    - Field(‘product\_id’, ‘reference products’ ),
    - …..
    - Field(‘orderer\_id’, ‘reference auth\_user’)
    - db.orders.product\_id.requires = IS\_IN\_DB(db, db.products.id, ‘%(product\_name)s’)
    - db.orders.orderer\_id.requires = IS\_IN\_DB(db, db.auth\_user.id, ‘%(last\_name)s’)
  + by using ‘reference products’ and ‘reference auth\_user’ we are clearly defining they are foreign keys and what table they are associated with
  + web2py generates front end code that enforce reference integrity through code
  + another way to write it is
    - …..
    - Field(‘orderer\_id’, db.auth\_user)
    - and omit (db.orders.orderer\_id.requires = IS\_IN\_DB(.....)
  + auth\_user is connected to auth\_user table
  + it will display alot more information from the table
* Summary
  + Web2py is MVC framework
  + Relational database
  + Web2py automatically generates an identify field for each table
* Introduction
  + Sessions
  + SQL Form Factories
  + Session and form variables
* Sessions
  + Part of http protocol
  + Stateless
  + Login
    - To control stateless
    - Server sets a session(you have been authenticated)
  + Set and view session variables
  + View login sessions
  + Ex)
    - def set():
    - session.name = ‘Sam’;
    - session.date=”8/1/2015”
    - session.membership=”family”
    - return locals()
    - def view():
    - return locals()
  + Then in the view
    - <h2>{{=session.auth.user.id}}
    - <h3> {{=session.name}}</h3>
    - <h3>{{=session.date}}</h3>
    - <h3>{{=session.membership}}</h3>
  + session.auth.user\_id
    - when a user login, this variable is automatically set
* SQL Form Factories
  + HTML forms with data validation
  + Validation without model
  + Store forms data in session variables
  + Create a controller
    - def form():
    - form = SQLFORM.factory(
    - Field(‘your\_name’, requires=IS\_NOT\_EMPTY()),
    - …..
    - if form.process().accepted:
    - response.flash = ‘form accepted’
    - session.your\_name = form.vars.your\_name
    - ….
  + Create the view
  + SQL form factories gives validation without connecting to database
* Demo - Prime Numbers
* Summary
* Introduction
* Demo - Farm Fresh App
* Demo - Model Code
  + auth.signature gives use 4 additional fields
    - who is entering data and who is updating it
    - also time stamps
  + ex)
    - db.define\_table(‘profile’,
    - Field(‘farm\_name’),
    - Field(‘farm\_address’),
    - Field(‘farm\_website’),
    - auth.signature
    - )
* Demo - Controller Code