* Course Overview
* Module Overview
* Prerequisites and Course Outline
* HTTP Requests and Responses
  + URL: resource locator
    - Allows you to uniquely identify where you want to get your content from
  + Browser makes http request to server to where resource is located
  + Server return http response
  + HTTP: hypertext transfer protocol
    - Simple ubiquitous text-based protocol used by browsers and apps to access web content
  + HTTP Clients
    - Client server protocol
    - Web browsers
    - Mobile apps
    - Programs
  + HTTP Servers
    - Host web pages and web content
    - Static or dynamic(javascript add interactivity to web page)
  + HTTP Requests
    - Clients make http requests
    - GET: fetch resources
    - POST: create/update resources
    - PUT: idempotently create/update
    - HEAD: to get only http header
    - DELETE: to delete resources
  + HTTP Responses
    - Servers are standing by to field requests
    - Send back http responses
      * Status line with code such as 404, 200, etc
      * Response header with metadata
      * Response body
* Web Scraping
  + Automated extraction of data from websites
  + Website content is first fetched(usually using HTTP) and then parsed to extract specific information
  + Web Pages
    - Websites are collections of web pages
    - Web pages consist of markup
    - This markup is understood and rendered by browsers
  + Fetching and Parsing
    - The same HTML markup can be accessed(fetched) via HTTP
    - Possesses an in-built hierarchical structure
    - Parsers can exploit this structure to extract information
* HTTP Client Libraries
  + Web scraping
    - Fetching content through http request through client library
    - Parsing content(HTML parsing, DOM Parsing, Computer Vision)
  + Fetching content
    - Urllib, Urlib2, Requests, Httplib, Httplib2
  + Requests
    - High level api
    - Easy to use
  + Httplib2
    - Fine-grained control of http request
  + Urllib
    - Part of python standard library
    - No need to download
  + Web servers make content available on HTTP endpoints
  + Browsers make HTTP requests under-the-hood to get web pages
  + Web scraping usually involves making such requests programmatically
  + Command-line HTTP requests
    - Curl
  + Python libraries for programmatic access
    - Requests
    - Httplib2
    - Urllib
* Making GET Requests Using h.
  + Httplib is an http client library which allows fine-grained access to HTTP requests
  + Httplib2 is goolge’s more powerful implementation
  + In ‘New’ Select python 3 as kernel
    - in the ‘In []’ slot type :!pip install httplib2
    - click run
  + in another ‘In []’
    - import httplib2
    - import webbrowser
    - from pprint import pprint
    - click run
  + in another
    - type url
    - ex) bin\_url = ‘https://httpbin.org/’
  + in another
    - invoke the web browser
    - type ‘ webbrowser.open(bin\_url)
    - will open page in new tab
  + in another
    - instantiate the http object to make http requests
    - type ‘http = httplib2.Http()
  + in another make a http request
    - resp, data = http.request(bin\_url)
    - returns response header(resp) and response body(data)
  + type(resp)
    - returns type of resp
  + len(resp)
    - returns length or response
  + resp.status: return status code
  + resp.reason: returns status code message
  + resp.version: returns version
  + resp.previous: contains original url incase url was redirected
    - if return empty, then there was no redirect
  + data
    - will contain the html markup
    - will be type of bytes
  + html = data.decode(“UTF-8”)
    - convert html to string format
  + http.request(url)
    - makes a get request to the url
  + different urls will have their data encoding in different formats
  + ex) data.decode(‘ISO-8859-1’), data.decode(‘UTF-8’)
* Making OPTIONS, POST, PUT ..
  + Can make different http requests to perform different types of actions
  + To know what type of requests is support at url, make http request with method = ‘OPTIONS’, then check the header
    - resp, data = http.request(bin\_url, method = ‘OPTIONS’)
    - pprint(resp)
  + method = ‘HEAD’ to only get the head and not the body
    - resp, data = http.request(bin\_url, method = ‘HEAD’)
  + when you don’t specify a method, it defaults to ‘GET’
  + POST: send data to server
    - post\_data = ‘{“name”: “Alice”, “college”: “Harvard”}’
    - resp, data = http.request(‘https://httpbin.org/post’, method = ‘POST’, body = post\_data, headers = {‘content-type’:’application/json’})
  + PUT: similar to post but idempotent
    - Can make the same put request multiple times and it will return the same result
* Handling Redirects with httpli..
  + Follow only safe redirects where a GET or a HEAD request is made
    - http.follow\_redirects
  + follow redirects for all kinds of requests
    - http.follow\_all\_redirects
  + resp.previous is not null if redirect happened
* Making HTTP Requests and P…
  + Urlib, urlib2 are high level interface for fetching data from URLS
  + Ex
    - import urllib
    - import webbrowser
    - from pprint import pprint
    - bin\_url = ‘https://httpbin.org/’
    - webbrowser.open(bin\_url)
  + urllib.request.urlopen(bin\_url)
    - resp = urllib.request.urlopen(bin\_url)
    - download contents of url
  + resp.url(): gets url
  + resp.getcode(): gets status code
  + resp.info(): gets the header of the response
  + resp.read(): to get content that have been downloaded
    - acts like a file handle
    - after reading the first time, it will return empty
  + use scraping tools such as beautiful soup to parse the response
  + resp.reason: to get status message
  + to make other type of http requests you need to instantiate a urllib request object
    - req = urllib.request.Request(bin\_url, method = ‘OPTIONS’
    - resp = urllib.request.urlopen(req)
  + to make a post request, you need to encode post data
  + ex)
    - post\_data = urllib.parse.urllencode({“name”: “Alice”, “college”: “Harvard”}).encode(‘ascii’)
    - req = urllib.request.Request(‘https://httpbin.org/post’, method = ‘POST’, data = post\_data, headers = {‘User-Agent’: ‘Mozilla/5.0 (Windows NT 6.1; x64)’})
    - resp = urllib.requst.urlopen(req)
  + good practice to make request with ‘with’ blocks
  + ex)
    - with urllib.request.urlopen(‘https://www.google.com/search?q=pluralsight’) as resp:
    - pprint(resp.read().decode(‘ISO-8859-1’))
  + With in python is used when working with unmanaged resources(like file streams)
    - It is similar to ‘using’ statement in C#
  + To handles errors
    - from urllib.error import URLError, HTTPError
    - try: …
    - except HTTPError as e: …
  + URLError is a more generic error
  + urllib.parse.urlparse
    - has specific functions that allows you to parse url
  + every url is made up of sub components which can be parsed
    - scheme, location, path
  + parsed\_data.scheme
  + parsed\_data.netloc
  + parsed.query
  + parsed.fragement
    - frag components is everything after #
    - often used as a bookmark in the page
* GET and POST Requests Using…