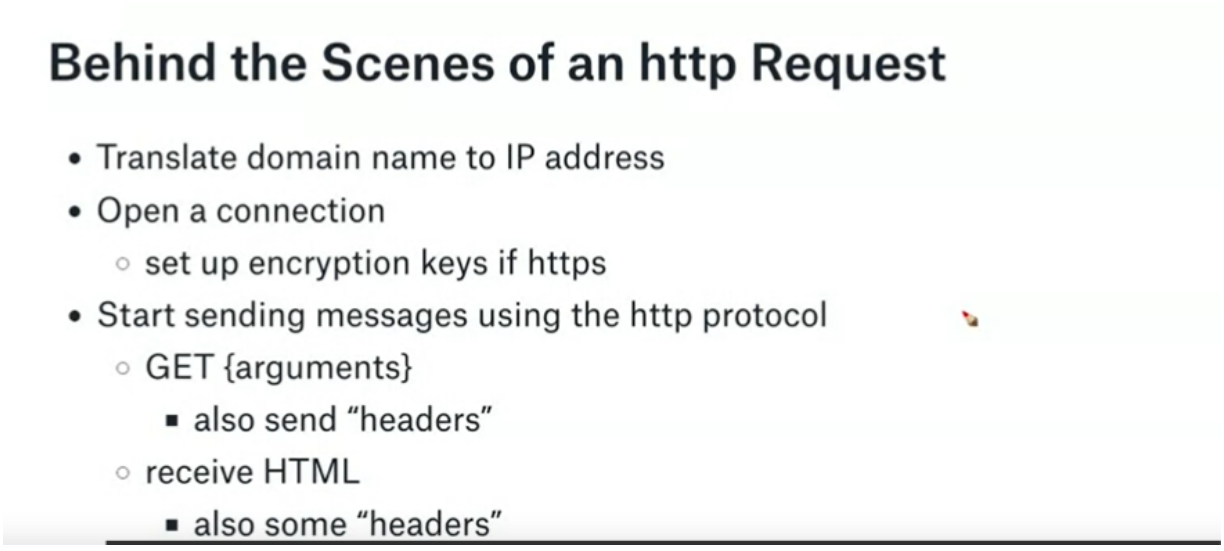
3.3 REST APIs

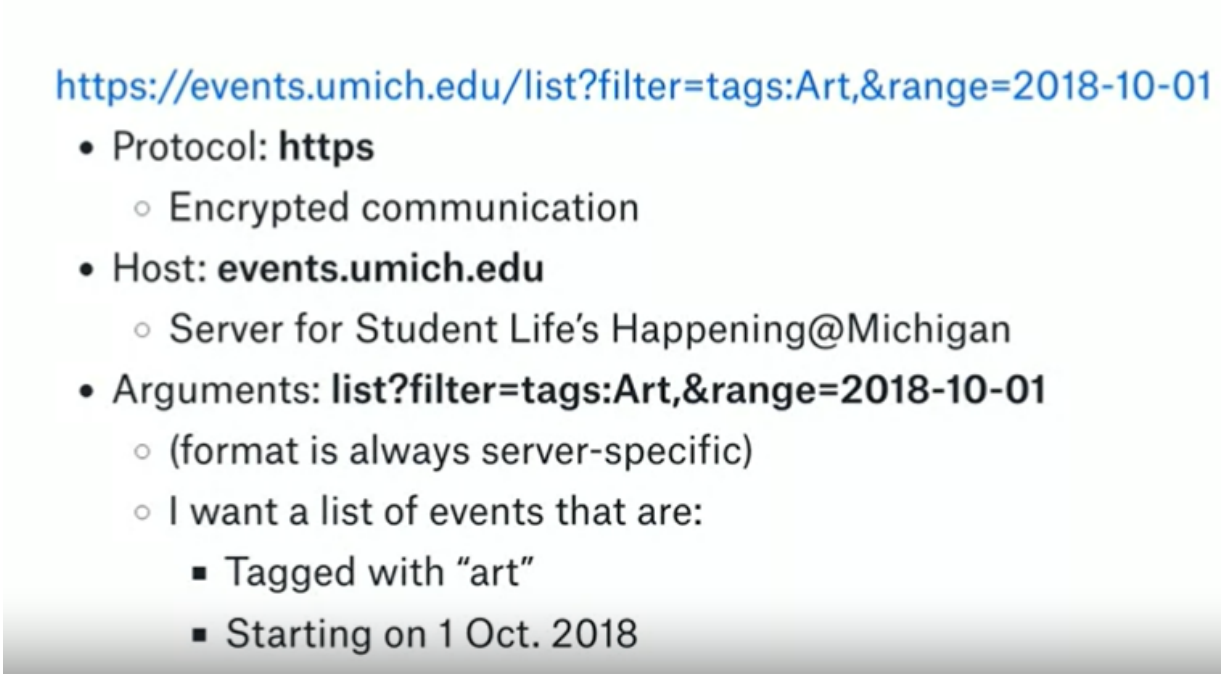
- URLs Domain Names IP Addresses
  - http://umich.edu/about 这个网址：http叫protocol协议 how to communicate...
  - {protocol}://{server}/{arguments}
  - DNS:domain name system 查找网址到ip地址的服务系统
  - ip地址每个点的数字不会超过255
  - URL的结构: <scheme>://<host>:<port>/<path>
- Routing
  - router路由器
  - 路径

```
~ % traceroute si.umich.edu
 1  100.68.0.5 (100.68.0.5)  71.102 ms  36.587 ms  58.474 ms
 2  172.19.242.241 (172.19.242.241)  72.073 ms  75.462 ms  2
83.510 ms
 3  10.255.255.253 (10.255.255.253)  288.951 ms  66.936 ms
76.281 ms
    ...
```

- Behind the scenes
  - 看



- URL Query Parameters
  - edu/后面的就是query parameter了 也叫argument 其中包含两个key filter 和range



- fetching a page
  - request module
    - 有这些功能：
      - text
      - .url
      - .json()
      - .status_code (not available in Runestone implementation)
      - .headers (not available in Runestone implementation)
      - .history (not available in Runestone implementation)
    - request.loads function
    - request.get function
      - 您不需要使用浏览器来获取页面的内容。在Python中，有一个可用的模块，称为请求。您可以使用请求模块中的get函数来获取页面的内容。
      - indent=2的地方是想要输出美观一点 会自动跳行

```python
import requests
import json

page = requests.get("https://api.datamuse.com/words?rel_rhy=funny")
print(type(page))
print(page.text[:150]) # print the first 150 characters
print(page.url) # print the url that was fetched
print("------")
x = page.json() # turn page.text into a python object
print(type(x))
print("---first item in the list---")
print(x[0])
print("---the whole list, pretty printed---")
print(json.dumps(x, indent=2)) # pretty print the results

```

- Using requests.get to encode URL parameters格式
  - d = {'q': '"violins and guitars"', 'tbm': 'isch'}
  - results = requests.get("https://google.com/search", params=d)
  - print(results.url)
  - 所有的结果就是url后面加?问号 然后加q=...加tbm=...再用&隔开 因为dic没有order所以链接可能有不同的写法

    Below are more examples of urls, outlining the base part of the url - which would be the first argument when calling `request.get()` - and the parameters - which would be written as a dictionary and passed into the params argument when calling `request.get()`.

base URL	parameters
https://www.youtube.com/watch	?v=Eq9CSdI7Mdo
http://services.faa.gov/airport/status/DTW	?format=json
https://google.com/	?q=university+of+michigan+news
https://itunes.apple.com/lookup	?id=909253&entity=album
http://baseurl.com/some/path	?key1=val1&key2=val2&key3=val3

    Here's an executable sample, using the optional params parameter of `requests.get`. It gets the same data from the datamus api that we saw previously. Here, however, the full url is built inside the call to `requests.get`; we can see what url was built by printing it out, on line 5.
  - 这里其实就是 用dic去requests.get一个page和直接复制粘贴page得到的东西是一样的

```python
import requests

# page = requests.get("https://api.datamuse.com/words?rel_rhy=funny")
kval_pairs = {'rel_rhy': 'funny'}
page = requests.get("https://api.datamuse.com/words", params=kval_pairs)
print(page.text[:150]) # print the first 150 characters
print(page.url) # print the url that was fetched

```

```
[{"word":"money","score":4417,"numSyllables":2},{"word":"honey","score":1208,"numSyllables":2},{"word":"sunny","score":720,"numSyllables":2},{"word":"
https://api.datamuse.com/words?rel_rhy=funny
```

- 题目！

Check Your Understanding

requests-6-1: How would you request the URL `http://bar.com/goodstuff?greet=hi+there&frosted=no` using the requests module?

- A. requests.get("http://bar.com/goodstuff", '?', {'greet': 'hi there'}, '&', {'frosted':'no'})
- B. requests.get("http://bar.com/", params = {'goodstuff':'?', 'greet':'hi there', 'frosted':'no'})
- C. requests.get("http://bar.com/goodstuff", params = ['greet', 'hi', 'there', 'frosted', 'no'])
- D. requests.get("http://bar.com/goodstuff", params = {'greet': 'hi there', 'frosted':'no'})

Check me | Compare me

✔ The ? and & are added automatically, and the space in hi there is automatically encoded as %3A.

Activity: 3 -- Multiple Choice (question27_1_1)

requests-6-2: If `resp` is a Response object returned by a call to `requests.get()`, which of the following is a way to extract the contents into a python dictionary or list?

- [x] A. resp.json()
- [ ] B. resp.json
- [ ] C. json.dumps(resp.text)
- [x] D. json.loads(resp.text)
- [ ] E. json.loads(resp.url)

Check me | Compare me

✔ Correct.
A. .json() invokes the json method
D. loads turns a json-formatted string into a list or dictionary

- reading API Documentation： Datamuse
  - Figuring Out How to Use a REST API
    - endpoints 端点 就是像/？word or sug这样的 https://api.datamuse.com/words or https://api.datamuse.com/sug.
    - ml=funny就会输出一些跟funny相关的次 如果像rel_cns=book的话就说输出辅音一样的词 比如bike back这种
    - 左边的像ml rel_cns rel_rhy 就是keys 右边的像bike这样的就是value
    - 一定要是dic格式！而且value格式一定是string 不然可能generate不了
    - 还有就是json的txt和url是否转化

```python
# import statements for necessary Python modules
import requests

def get_rhymes(word):
    baseurl = "https://api.datamuse.com/words"
    params_diction = {} # Set up an empty dictionary for query parameters
    params_diction["rel_rhy"] = word
    params_diction["max"] = "3" # get at most 3 results
    resp = requests.get(baseurl, params=params_diction)
    # return the top three words
    word_ds = resp.json()
    return [d['word'] for d in word_ds]
    return resp.json() # Return a python object (a list of dictionaries in this ca

print(get_rhymes("funny"))

```

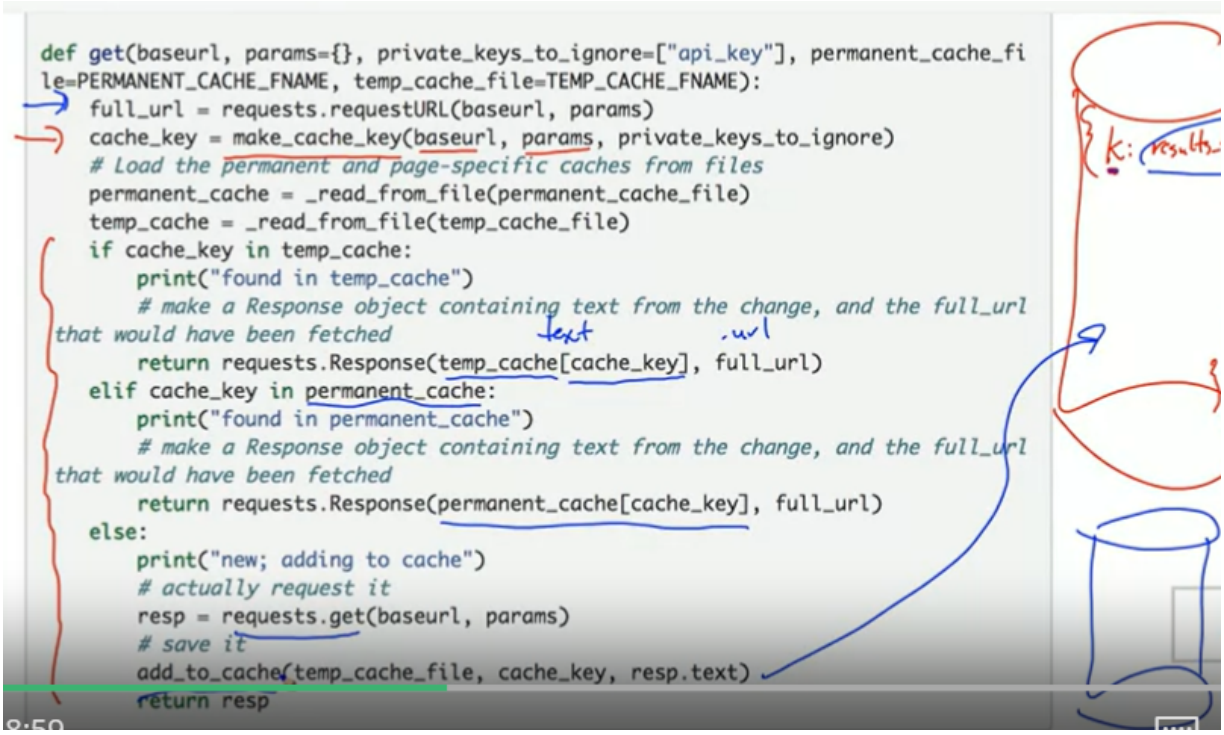
```
['money', 'honey', 'sunny']
```

- caching response content
  - cache 发音是cash 意思就有点像是储藏柜？藏东西的地方 分成permanent和temporary 要用的话就是先import request_with_caching 这是个module 之后另某变量 =requests_with_coaching.get(网址要求) 真实的python情况下这个module加载不出来
  - 有三个结果
    - found in permanent cache：可以使用但是不能添加
    - found in page-specific cache
    - new; adding to cache
  - temporary的在刷新页面之后就会没有掉
  - 代码的大概想法就是 存在cache里当成一个dic 然后有keys和values store in a file，就有funcition叫做_write_to_file和read_to_file 而且也要区别是不是permenant
  - 1

```
import requests
import json

PERMANENT_CACHE_FNAME = "permanent_cache.txt"
TEMP_CACHE_FNAME = "this_page_cache.txt"

def _write_to_file(cache, fname):
    with open(fname, 'w') as outfile:
        outfile.write(json.dumps(cache, indent=2))

def _read_from_file(fname):
    try:
        with open(fname, 'r') as infile:
            res = infile.read()
            return json.loads(res)
    except:
        return {}

def add_to_cache(cache_file, cache_key, cache_value):
    temp_cache = _read_from_file(cache_file)
    temp_cache[cache_key] = cache_value
    _write_to_file(temp_cache, cache_file)

def clear_cache(cache_file=TEMP_CACHE_FNAME):
    _write_to_file({}, cache_file)
```

- 2

```
def make_cache_key(baseurl, params_d, private_keys=["api_key"]):
    """Makes a long string representing the query.
    Alphabetize the keys from the params dictionary so we get the same order each
 time.
    Omit keys with private info."""
    alphabetized_keys = sorted(params_d.keys())
    res = []
    for k in alphabetized_keys:
        if k not in private_keys:
            res.append("{}-{}".format(k, params_d[k]))
    return baseurl + "_".join(res)
```



- 有个问题是：为什么要用make_cache_key而不是直接用full url当作key 是因为当requests.get去encodes url的时候 keys不是按照一定的order去排的 那排列组合如果keys很多的话就会越来越多种 增加负担 而且这个可能下次还要用但是顺序却不一样了 会很麻烦

- iTunes的例子
  - import requests_with_caching
  - import json
  - parameters = {"term": "Ann Arbor", "entity": "podcast"}
  - iTunes_response = requests_with_caching.get("https://itunes.apple.com/search", params = parameters,permanent_cache_file="itunes_cache.txt")
  - py_data = json.loads(iTunes_response.text)
  - 输出是found in permanent_cache
  - 如果想要干点什么 比如 print(iTunes_response.url)
  - 会输出https://itunes.apple.com/search?term=Ann+Arbor&entity=podcast这个网址
  - 继续做些什么的话就是
  - import requests_with_caching
  - import json
  - parameters = {"term": "Ann Arbor", "entity": "podcast"}
  - iTunes_response = requests_with_caching.get("https://itunes.apple.com/search", params = parameters, permanent_cache_file="itunes_cache.txt")
  - py_data = json.loads(iTunes_response.text)
  - for r in py_data['results']:
  - print(r['trackName']) 最灵性的是这句 从我的results的keys的value 其实是个list of dic 然后里面再去输出他们的key value

- flickr API的例子 是个图像共享的网站
  - 可以使用API使应用程序更容易地从站点获取数据并将数据发布到站点
  - baseurl是https://api.flickr.com/services/rest/
  - endpoint是?
  - method=flickr.photos.search 命名method variable是找photo的
  - 这个网站因为results不是json format的 所以用format=json去return results
  - per_page=5是 return 5results at a time
  - tags=mountains,river. 就是return things that are tagged with “mountains” and “river”.
  - tag_mode=all. This says to return things that are tagged with both mountains and river.两个都
  - media=photos. This says to return photos
  - api_key=... 因为这个网站需要有授权才能down数据
  - nojsoncallback=1 这表示返回原始JSON结果，而不需要对JSON响应进行函数包装。
  - 如果真正python的环境中还可以用webbrowser.open()去打开这些url 自动的

```
# import statements
import requests_with_caching
import json
# import webbrowser

# apply for a flickr authentication key at http://www.flickr.com/services/apps/create/apply/?
# paste the key (not the secret) as the value of the variable flickr_key
flickr_key = 'yourkeyhere'

def get_flickr_data(tags_string):
    baseurl = "https://api.flickr.com/services/rest/"
    params_diction = {}
    params_diction["api_key"] = flickr_key # from the above global variable
    params_diction["tags"] = tags_string # must be a comma separated string to work correctly
    params_diction["tag_mode"] = "all"
    params_diction["method"] = "flickr.photos.search"
    params_diction["per_page"] = 5
    params_diction["media"] = "photos"
    params_diction["format"] = "json"
    params_diction["nojsoncallback"] = 1
    flickr_resp = requests_with_caching.get(baseurl, params = params_diction, permanent_cache_file="flickr_cache.txt")
    # Useful for debugging: print the url! Uncomment the below line to do so.
    print(flickr_resp.url) # Paste the result into the browser to check it out...
    return flickr_resp.json()
```

```
# Some code to open up a few photos that are tagged with the mountains and river t

photos = result_river_mts['photos']['photo']
for photo in photos:
    owner = photo['owner']
    photo_id = photo['id']
    url = 'https://www.flickr.com/photos/{}/{}'.format(owner, photo_id)
    print(url)
    # webbrowser.open(url)
```

```
found in permanent_cache
https://api.flickr.com/services/rest/?api_key=yourkeyhere&tags=river%2Cmountains&tag_mode=all&meth
od=flickr.photos.search&per_page=5&media=photos&format=json&nojsoncallback=1
https://www.flickr.com/photos/45934971@N07/44858440865
https://www.flickr.com/photos/145056248@N07/43953569330
https://www.flickr.com/photos/145056248@N07/43953448610
https://www.flickr.com/photos/131540074@N08/44857602655
https://www.flickr.com/photos/145056248@N07/44857423045
```

- 可爱的老人家的网址是 www.py4e.com/code3/