**Twitter API**

**Step1**: Create a python program *twitter1.py.* Type the following program:

*import urllib*

*import twurl*

*import json*

*import sqlite3*

*TWITTER\_URL = 'https://api.twitter.com/1.1/friends/list.json'*

*conn = sqlite3.connect('spider.sqlite3')*

*cur = conn.cursor()*

*cur.execute('''*

*CREATE TABLE IF NOT EXISTS Twitter*

*(name TEXT, retrieved INTEGER, friends INTEGER)''')*

*while True:*

*acct = raw\_input('Enter a Twitter account, or quit: ')*

*if ( acct == 'quit' ) : break*

*if ( len(acct) < 1 ) :*

*cur.execute('SELECT name FROM Twitter WHERE retrieved = 0 LIMIT 1')*

*try:*

*acct = cur.fetchone()[0]*

*except:*

*print 'No unretrieved Twitter accounts found'*

*continue*

*url = twurl.augment(TWITTER\_URL,*

*{'screen\_name': acct, 'count': '20'} )*

*print 'Retrieving', url*

*connection = urllib.urlopen(url)*

*data = connection.read()*

*headers = connection.info().dict*

*# print 'Remaining', headers['x-rate-limit-remaining']*

*js = json.loads(data)*

*# print json.dumps(js, indent=4)*

*cur.execute('UPDATE Twitter SET retrieved=1 WHERE name = ?', (acct, ) )*

*countnew = 0*

*countold = 0*

*for u in js['users'] :*

*friend = u['screen\_name']*

*print friend*

*cur.execute('SELECT friends FROM Twitter WHERE name = ? LIMIT 1',*

*(friend, ) )*

*try:*

*count = cur.fetchone()[0]*

*cur.execute('UPDATE Twitter SET friends = ? WHERE name = ?',*

*(count+1, friend) )*

*countold = countold + 1*

*except:*

*cur.execute('''INSERT INTO Twitter (name, retrieved, friends)*

*VALUES ( ?, 0, 1 )''', ( friend, ) )*

*countnew = countnew + 1*

*print 'New accounts=',countnew,' revisited=',countold*

*conn.commit()*

*cur.close()*

**Step2**: Create another program *twurl.py*

*import urllib*

*import oauth*

*import hidden*

*def augment(url, parameters) :*

*secrets = hidden.oauth()*

*consumer = oauth.OAuthConsumer(secrets['consumer\_key'], secrets['consumer\_secret'])*

*token = oauth.OAuthToken(secrets['token\_key'],secrets['token\_secret'])*

*oauth\_request = oauth.OAuthRequest.from\_consumer\_and\_token(consumer,*

*token=token, http\_method='GET', http\_url=url, parameters=parameters)*

*oauth\_request.sign\_request(oauth.OAuthSignatureMethod\_HMAC\_SHA1(), consumer, token)*

*return oauth\_request.to\_url()*

**Step3**: Create 3rd program, name it *oauth.py*

*"""*

*The MIT License*

*Copyright (c) 2007 Leah Culver*

*Permission is hereby granted, free of charge, to any person obtaining a copy*

*of this software and associated documentation files (the "Software"), to deal*

*in the Software without restriction, including without limitation the rights*

*to use, copy, modify, merge, publish, distribute, sublicense, and/or sell*

*copies of the Software, and to permit persons to whom the Software is*

*furnished to do so, subject to the following conditions:*

*The above copyright notice and this permission notice shall be included in*

*all copies or substantial portions of the Software.*

*THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR*

*IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,*

*FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE*

*AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER*

*LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,*

*OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN*

*THE SOFTWARE.*

*"""*

*import cgi*

*import urllib*

*import time*

*import random*

*import urlparse*

*import hmac*

*import binascii*

*VERSION = '1.0' # Hi Blaine!*

*HTTP\_METHOD = 'GET'*

*SIGNATURE\_METHOD = 'PLAINTEXT'*

*class OAuthError(RuntimeError):*

*"""Generic exception class."""*

*def \_\_init\_\_(self, message='OAuth error occured.'):*

*self.mymessage = message*

*def build\_authenticate\_header(realm=''):*

*"""Optional WWW-Authenticate header (401 error)"""*

*return {'WWW-Authenticate': 'OAuth realm="%s"' % realm}*

*def escape(s):*

*"""Escape a URL including any /."""*

*return urllib.quote(s, safe='~')*

*def \_utf8\_str(s):*

*"""Convert unicode to utf-8."""*

*if isinstance(s, unicode):*

*return s.encode("utf-8")*

*else:*

*return str(s)*

*def generate\_timestamp():*

*"""Get seconds since epoch (UTC)."""*

*return int(time.time())*

*def generate\_nonce(length=8):*

*"""Generate pseudorandom number."""*

*return ''.join([str(random.randint(0, 9)) for i in range(length)])*

*class OAuthConsumer(object):*

*"""Consumer of OAuth authentication.*

*OAuthConsumer is a data type that represents the identity of the Consumer*

*via its shared secret with the Service Provider.*

*"""*

*key = None*

*secret = None*

*def \_\_init\_\_(self, key, secret):*

*self.key = key*

*self.secret = secret*

*class OAuthToken(object):*

*"""OAuthToken is a data type that represents an End User via either an access*

*or request token.*

*key -- the token*

*secret -- the token secret*

*"""*

*key = None*

*secret = None*

*def \_\_init\_\_(self, key, secret):*

*self.key = key*

*self.secret = secret*

*def to\_string(self):*

*return urllib.urlencode({'oauth\_token': self.key,*

*'oauth\_token\_secret': self.secret})*

*def from\_string(s):*

*""" Returns a token from something like:*

*oauth\_token\_secret=xxx&oauth\_token=xxx*

*"""*

*params = cgi.parse\_qs(s, keep\_blank\_values=False)*

*key = params['oauth\_token'][0]*

*secret = params['oauth\_token\_secret'][0]*

*return OAuthToken(key, secret)*

*from\_string = staticmethod(from\_string)*

*def \_\_str\_\_(self):*

*return self.to\_string()*

*class OAuthRequest(object):*

*"""OAuthRequest represents the request and can be serialized.*

*OAuth parameters:*

*- oauth\_consumer\_key*

*- oauth\_token*

*- oauth\_signature\_method*

*- oauth\_signature*

*- oauth\_timestamp*

*- oauth\_nonce*

*- oauth\_version*

*... any additional parameters, as defined by the Service Provider.*

*"""*

*parameters = None # OAuth parameters.*

*http\_method = HTTP\_METHOD*

*http\_url = None*

*version = VERSION*

*def \_\_init\_\_(self, http\_method=HTTP\_METHOD, http\_url=None, parameters=None):*

*self.http\_method = http\_method*

*self.http\_url = http\_url*

*self.parameters = parameters or {}*

*def set\_parameter(self, parameter, value):*

*self.parameters[parameter] = value*

*def get\_parameter(self, parameter):*

*try:*

*return self.parameters[parameter]*

*except:*

*if parameter == "oauth\_token" : return None*

*raise OAuthError('Parameter not found: %s' % parameter)*

*def \_get\_timestamp\_nonce(self):*

*return self.get\_parameter('oauth\_timestamp'), self.get\_parameter(*

*'oauth\_nonce')*

*def get\_nonoauth\_parameters(self):*

*"""Get any non-OAuth parameters."""*

*parameters = {}*

*for k, v in self.parameters.iteritems():*

*# Ignore oauth parameters.*

*if k.find('oauth\_') < 0:*

*parameters[k] = v*

*return parameters*

*def to\_header(self, realm=''):*

*"""Serialize as a header for an HTTPAuth request."""*

*auth\_header = 'OAuth realm="%s"' % realm*

*# Add the oauth parameters.*

*if self.parameters:*

*for k, v in self.parameters.iteritems():*

*if k[:6] == 'oauth\_':*

*auth\_header += ', %s="%s"' % (k, escape(str(v)))*

*return {'Authorization': auth\_header}*

*def to\_postdata(self):*

*"""Serialize as post data for a POST request."""*

*return '&'.join(['%s=%s' % (escape(str(k)), escape(str(v))) \*

*for k, v in self.parameters.iteritems()])*

*def to\_url(self):*

*"""Serialize as a URL for a GET request."""*

*return '%s?%s' % (self.get\_normalized\_http\_url(), self.to\_postdata())*

*def get\_normalized\_parameters(self):*

*"""Return a string that contains the parameters that must be signed."""*

*# Chuck - Make a copy of the parameters so we can modify them*

*params = dict(self.parameters)*

*try:*

*# Exclude the signature if it exists.*

*del params['oauth\_signature']*

*except:*

*pass*

*# Escape key values before sorting.*

*key\_values = [(escape(\_utf8\_str(k)), escape(\_utf8\_str(v))) \*

*for k,v in params.items()]*

*# Sort lexicographically, first after key, then after value.*

*key\_values.sort()*

*# Combine key value pairs into a string.*

*return '&'.join(['%s=%s' % (k, v) for k, v in key\_values])*

*def get\_normalized\_http\_method(self):*

*"""Uppercases the http method."""*

*return self.http\_method.upper()*

*def get\_normalized\_http\_url(self):*

*"""Parses the URL and rebuilds it to be scheme://host/path."""*

*parts = urlparse.urlparse(self.http\_url)*

*scheme, netloc, path = parts[:3]*

*# Exclude default port numbers.*

*if scheme == 'http' and netloc[-3:] == ':80':*

*netloc = netloc[:-3]*

*elif scheme == 'https' and netloc[-4:] == ':443':*

*netloc = netloc[:-4]*

*return '%s://%s%s' % (scheme, netloc, path)*

*def sign\_request(self, signature\_method, consumer, token):*

*"""Set the signature parameter to the result of build\_signature."""*

*# Set the signature method.*

*self.set\_parameter('oauth\_signature\_method',*

*signature\_method.get\_name())*

*# Set the signature.*

*self.set\_parameter('oauth\_signature',*

*self.build\_signature(signature\_method, consumer, token))*

*def build\_signature(self, signature\_method, consumer, token):*

*"""Calls the build signature method within the signature method."""*

*return signature\_method.build\_signature(self, consumer, token)*

*def from\_request(http\_method, http\_url, headers=None, parameters=None,*

*query\_string=None):*

*"""Combines multiple parameter sources."""*

*if parameters is None:*

*parameters = {}*

*# Headers*

*if headers and 'Authorization' in headers:*

*auth\_header = headers['Authorization']*

*# Check that the authorization header is OAuth.*

*if auth\_header.index('OAuth') > -1:*

*auth\_header = auth\_header.lstrip('OAuth ')*

*try:*

*# Get the parameters from the header.*

*header\_params = OAuthRequest.\_split\_header(auth\_header)*

*parameters.update(header\_params)*

*except:*

*raise OAuthError('Unable to parse OAuth parameters from '*

*'Authorization header.')*

*# GET or POST query string.*

*if query\_string:*

*query\_params = OAuthRequest.\_split\_url\_string(query\_string)*

*parameters.update(query\_params)*

*# URL parameters.*

*param\_str = urlparse.urlparse(http\_url)[4] # query*

*url\_params = OAuthRequest.\_split\_url\_string(param\_str)*

*parameters.update(url\_params)*

*if parameters:*

*return OAuthRequest(http\_method, http\_url, parameters)*

*return None*

*from\_request = staticmethod(from\_request)*

*def from\_consumer\_and\_token(oauth\_consumer, token=None,*

*http\_method=HTTP\_METHOD, http\_url=None, parameters=None):*

*if not parameters:*

*parameters = {}*

*defaults = {*

*'oauth\_consumer\_key': oauth\_consumer.key,*

*'oauth\_timestamp': generate\_timestamp(),*

*'oauth\_nonce': generate\_nonce(),*

*'oauth\_version': OAuthRequest.version,*

*}*

*defaults.update(parameters)*

*parameters = defaults*

*if token:*

*parameters['oauth\_token'] = token.key*

*return OAuthRequest(http\_method, http\_url, parameters)*

*from\_consumer\_and\_token = staticmethod(from\_consumer\_and\_token)*

*def from\_token\_and\_callback(token, callback=None, http\_method=HTTP\_METHOD,*

*http\_url=None, parameters=None):*

*if not parameters:*

*parameters = {}*

*parameters['oauth\_token'] = token.key*

*if callback:*

*parameters['oauth\_callback'] = callback*

*return OAuthRequest(http\_method, http\_url, parameters)*

*from\_token\_and\_callback = staticmethod(from\_token\_and\_callback)*

*def \_split\_header(header):*

*"""Turn Authorization: header into parameters."""*

*params = {}*

*parts = header.split(',')*

*for param in parts:*

*# Ignore realm parameter.*

*if param.find('realm') > -1:*

*continue*

*# Remove whitespace.*

*param = param.strip()*

*# Split key-value.*

*param\_parts = param.split('=', 1)*

*# Remove quotes and unescape the value.*

*params[param\_parts[0]] = urllib.unquote(param\_parts[1].strip('\"'))*

*return params*

*\_split\_header = staticmethod(\_split\_header)*

*def \_split\_url\_string(param\_str):*

*"""Turn URL string into parameters."""*

*parameters = cgi.parse\_qs(param\_str, keep\_blank\_values=False)*

*for k, v in parameters.iteritems():*

*parameters[k] = urllib.unquote(v[0])*

*return parameters*

*\_split\_url\_string = staticmethod(\_split\_url\_string)*

*class OAuthServer(object):*

*"""A worker to check the validity of a request against a data store."""*

*timestamp\_threshold = 300 # In seconds, five minutes.*

*version = VERSION*

*signature\_methods = None*

*data\_store = None*

*def \_\_init\_\_(self, data\_store=None, signature\_methods=None):*

*self.data\_store = data\_store*

*self.signature\_methods = signature\_methods or {}*

*def set\_data\_store(self, data\_store):*

*self.data\_store = data\_store*

*def get\_data\_store(self):*

*return self.data\_store*

*def add\_signature\_method(self, signature\_method):*

*self.signature\_methods[signature\_method.get\_name()] = signature\_method*

*return self.signature\_methods*

*def fetch\_request\_token(self, oauth\_request):*

*"""Processes a request\_token request and returns the*

*request token on success.*

*"""*

*try:*

*# Get the request token for authorization.*

*token = self.\_get\_token(oauth\_request, 'request')*

*except OAuthError:*

*# No token required for the initial token request.*

*version = self.\_get\_version(oauth\_request)*

*consumer = self.\_get\_consumer(oauth\_request)*

*self.\_check\_signature(oauth\_request, consumer, None)*

*# Fetch a new token.*

*token = self.data\_store.fetch\_request\_token(consumer)*

*return token*

*def fetch\_access\_token(self, oauth\_request):*

*"""Processes an access\_token request and returns the*

*access token on success.*

*"""*

*version = self.\_get\_version(oauth\_request)*

*consumer = self.\_get\_consumer(oauth\_request)*

*# Get the request token.*

*token = self.\_get\_token(oauth\_request, 'request')*

*self.\_check\_signature(oauth\_request, consumer, token)*

*new\_token = self.data\_store.fetch\_access\_token(consumer, token)*

*return new\_token*

*def verify\_request(self, oauth\_request):*

*"""Verifies an api call and checks all the parameters."""*

*# -> consumer and token*

*version = self.\_get\_version(oauth\_request)*

*consumer = self.\_get\_consumer(oauth\_request)*

*# Get the access token.*

*token = self.\_get\_token(oauth\_request, 'access')*

*self.\_check\_signature(oauth\_request, consumer, token)*

*parameters = oauth\_request.get\_nonoauth\_parameters()*

*return consumer, token, parameters*

*def authorize\_token(self, token, user):*

*"""Authorize a request token."""*

*return self.data\_store.authorize\_request\_token(token, user)*

*def get\_callback(self, oauth\_request):*

*"""Get the callback URL."""*

*return oauth\_request.get\_parameter('oauth\_callback')*

*def build\_authenticate\_header(self, realm=''):*

*"""Optional support for the authenticate header."""*

*return {'WWW-Authenticate': 'OAuth realm="%s"' % realm}*

*def \_get\_version(self, oauth\_request):*

*"""Verify the correct version request for this server."""*

*try:*

*version = oauth\_request.get\_parameter('oauth\_version')*

*except:*

*version = VERSION*

*if version and version != self.version:*

*raise OAuthError('OAuth version %s not supported.' % str(version))*

*return version*

*def \_get\_signature\_method(self, oauth\_request):*

*"""Figure out the signature with some defaults."""*

*try:*

*signature\_method = oauth\_request.get\_parameter(*

*'oauth\_signature\_method')*

*except:*

*signature\_method = SIGNATURE\_METHOD*

*try:*

*# Get the signature method object.*

*signature\_method = self.signature\_methods[signature\_method]*

*except:*

*signature\_method\_names = ', '.join(self.signature\_methods.keys())*

*raise OAuthError('Signature method %s not supported try one of the '*

*'following: %s' % (signature\_method, signature\_method\_names))*

*return signature\_method*

*def \_get\_consumer(self, oauth\_request):*

*consumer\_key = oauth\_request.get\_parameter('oauth\_consumer\_key')*

*consumer = self.data\_store.lookup\_consumer(consumer\_key)*

*if not consumer:*

*raise OAuthError('Invalid consumer.')*

*return consumer*

*def \_get\_token(self, oauth\_request, token\_type='access'):*

*"""Try to find the token for the provided request token key."""*

*token\_field = oauth\_request.get\_parameter('oauth\_token')*

*token = self.data\_store.lookup\_token(token\_type, token\_field)*

*if not token:*

*raise OAuthError('Invalid %s token: %s' % (token\_type, token\_field))*

*return token*

*def \_check\_signature(self, oauth\_request, consumer, token):*

*timestamp, nonce = oauth\_request.\_get\_timestamp\_nonce()*

*self.\_check\_timestamp(timestamp)*

*self.\_check\_nonce(consumer, token, nonce)*

*signature\_method = self.\_get\_signature\_method(oauth\_request)*

*try:*

*signature = oauth\_request.get\_parameter('oauth\_signature')*

*except:*

*raise OAuthError('Missing signature.')*

*# Validate the signature.*

*valid\_sig = signature\_method.check\_signature(oauth\_request, consumer,*

*token, signature)*

*if not valid\_sig:*

*key, base = signature\_method.build\_signature\_base\_string(*

*oauth\_request, consumer, token)*

*raise OAuthError('Invalid signature. Expected signature base '*

*'string: %s' % base)*

*built = signature\_method.build\_signature(oauth\_request, consumer, token)*

*def \_check\_timestamp(self, timestamp):*

*"""Verify that timestamp is recentish."""*

*timestamp = int(timestamp)*

*now = int(time.time())*

*lapsed = now - timestamp*

*if lapsed > self.timestamp\_threshold:*

*raise OAuthError('Expired timestamp: given %d and now %s has a '*

*'greater difference than threshold %d' %*

*(timestamp, now, self.timestamp\_threshold))*

*def \_check\_nonce(self, consumer, token, nonce):*

*"""Verify that the nonce is uniqueish."""*

*nonce = self.data\_store.lookup\_nonce(consumer, token, nonce)*

*if nonce:*

*raise OAuthError('Nonce already used: %s' % str(nonce))*

*class OAuthClient(object):*

*"""OAuthClient is a worker to attempt to execute a request."""*

*consumer = None*

*token = None*

*def \_\_init\_\_(self, oauth\_consumer, oauth\_token):*

*self.consumer = oauth\_consumer*

*self.token = oauth\_token*

*def get\_consumer(self):*

*return self.consumer*

*def get\_token(self):*

*return self.token*

*def fetch\_request\_token(self, oauth\_request):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def fetch\_access\_token(self, oauth\_request):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def access\_resource(self, oauth\_request):*

*"""-> Some protected resource."""*

*raise NotImplementedError*

*class OAuthDataStore(object):*

*"""A database abstraction used to lookup consumers and tokens."""*

*def lookup\_consumer(self, key):*

*"""-> OAuthConsumer."""*

*raise NotImplementedError*

*def lookup\_token(self, oauth\_consumer, token\_type, token\_token):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def lookup\_nonce(self, oauth\_consumer, oauth\_token, nonce):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def fetch\_request\_token(self, oauth\_consumer):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def fetch\_access\_token(self, oauth\_consumer, oauth\_token):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*def authorize\_request\_token(self, oauth\_token, user):*

*"""-> OAuthToken."""*

*raise NotImplementedError*

*class OAuthSignatureMethod(object):*

*"""A strategy class that implements a signature method."""*

*def get\_name(self):*

*"""-> str."""*

*raise NotImplementedError*

*def build\_signature\_base\_string(self, oauth\_request, oauth\_consumer, oauth\_token):*

*"""-> str key, str raw."""*

*raise NotImplementedError*

*def build\_signature(self, oauth\_request, oauth\_consumer, oauth\_token):*

*"""-> str."""*

*raise NotImplementedError*

*def check\_signature(self, oauth\_request, consumer, token, signature):*

*built = self.build\_signature(oauth\_request, consumer, token)*

*return built == signature*

*class OAuthSignatureMethod\_HMAC\_SHA1(OAuthSignatureMethod):*

*def get\_name(self):*

*return 'HMAC-SHA1'*

*def build\_signature\_base\_string(self, oauth\_request, consumer, token):*

*sig = (*

*escape(oauth\_request.get\_normalized\_http\_method()),*

*escape(oauth\_request.get\_normalized\_http\_url()),*

*escape(oauth\_request.get\_normalized\_parameters()),*

*)*

*key = '%s&' % escape(consumer.secret)*

*if token and token.secret:*

*key += escape(token.secret)*

*raw = '&'.join(sig)*

*return key, raw*

*def build\_signature(self, oauth\_request, consumer, token):*

*"""Builds the base signature string."""*

*key, raw = self.build\_signature\_base\_string(oauth\_request, consumer,*

*token)*

*# HMAC object.*

*try:*

*import hashlib # 2.5*

*hashed = hmac.new(key, raw, hashlib.sha1)*

*except:*

*import sha # Deprecated*

*hashed = hmac.new(key, raw, sha)*

*# Calculate the digest base 64.*

*return binascii.b2a\_base64(hashed.digest())[:-1]*

*class OAuthSignatureMethod\_PLAINTEXT(OAuthSignatureMethod):*

*def get\_name(self):*

*return 'PLAINTEXT'*

*def build\_signature\_base\_string(self, oauth\_request, consumer, token):*

*"""Concatenates the consumer key and secret."""*

*sig = '%s&' % escape(consumer.secret)*

*if token:*

*sig = sig + escape(token.secret)*

*return sig, sig*

*def build\_signature(self, oauth\_request, consumer, token):*

*key, raw = self.build\_signature\_base\_string(oauth\_request, consumer,*

*token)*

*return key*

**Step4**: Create a Twitter account. (Not necessary if one already exists) Or open your Twitter account.

**Step5**: Create a simple application from the following link.

<https://apps.twitter.com/>

**Step6**: Create another python program *hidden.py*

*# Keep this file separate*

*def oauth() :*

*return { "consumer\_key" : "hKNW…GqiV",*

*"consumer\_secret" : "5IO…pYR",*

*"token\_key" : "426704466-uNp5…hHx",*

*"token\_secret" : "vsUUL…4k4i" }*

The positions with “…” in the above program should be modified which is listed in the following step.

**Step7**: Open the application that you’ve created from the link.

<https://apps.twitter.com/>

That should open a web page with 4 tabs.

[Details](https://apps.twitter.com/app/13113513) [Settings](https://apps.twitter.com/app/13113513/settings) [Keys and Access Tokens](https://apps.twitter.com/app/13113513/keys) [Permissions](https://apps.twitter.com/app/13113513/permissions)

Select the [Keys and Access Tokens](https://apps.twitter.com/app/13113513/keys) tab.

Under “Application Settings”:

Copy the Consumer Key (API Key) , Consumer Secret (API Secret) values

Under “Your Access Token Settings”:

Copy the Access Token, Access Token Secret values

And place it in the appropriate modules (“…”) in the “hidden.py” program.

**NOTE** : When creating a Repository, do not publish your keys in public.

**Step8**: Create another python program hidden.py

# Keep this file separate

def oauth() :

return { "consumer\_key" : "hKNW…GqiV",

"consumer\_secret" : "5IO…pYR",

"token\_key" : "426704466-uNp5…hHx",

"token\_secret" : "vsUUL…4k4i" }

The positions with “…” in the above program should be modified.