

COMP140: Creative Computing — Hacking  
**Twitter bots**

# Today's class

- ▶ RESTful web APIs
- ▶ Tutorial / live coding: a simple Twitter bot in Python
- ▶ OAuth
- ▶ API hacking: sprint review and general support
- ▶ At 5pm in the Chapel: guest lecture by Barry Caudill, Firaxis Games

# The Twitter API

# REST

- ▶ REST = **Representational State Transfer**
- ▶ Many web services provide a **REST API**
  - ▶ Including your COMP110 client/server coding task!
- ▶ An API is **RESTful** if it is:
  - ▶ Based on a **client-server model**: clients send requests to a server
  - ▶ **Stateless**: each request from the client is self-contained
  - ▶ **Cacheable**: the API makes clear which responses can and cannot be cached
  - ▶ **Layered**: the “server” may actually be a cluster of machines
  - ▶ Uses a **uniform interface**: e.g. HTTP requests, URLs, XML, JSON, ...

# Twitter API

- ▶ Twitter provides a REST API
- ▶ Example: to post a tweet
  - ▶ (see documentation at <https://dev.twitter.com/rest/reference/post/statuses/update>)
  - ▶ The client makes an **HTTP POST request** to `https://api.twitter.com/1.1/statuses/update.json?status=Hello+world`
  - ▶ The **HTTP request header** contains authentication information for the app and for the user
  - ▶ The **response** from the server is a JSON document containing information about the posted tweet

# Libraries

- ▶ Working with REST APIs directly through HTTP requests can be cumbersome
- ▶ For most popular web services, there are many **libraries** (official and third party) which wrap the REST APIs in a more programmer-friendly interface
- ▶ For Twitter: `https://dev.twitter.com/overview/api/twitter-libraries`
- ▶ For today's live coding, I will be using the **Tweepy** library for Python

**Your first Twitter bot**

# Account set up

- ▶ Go to `https://www.twitter.com` and **either**
  - ▶ Create an account, **or**
  - ▶ Sign in to your existing account
- ▶ NB: Twitter **requires** app developers to add a **mobile phone number** to their accounts (don't ask me why...)



# Application set up

- ▶ Go to `https://apps.twitter.com`
- ▶ Click on **Create New App**
- ▶ Fill in the required details and agree to the license agreement

# Project set up

- ▶ Open **PyCharm** and create a **new project**
- ▶ Go to **File** → **Settings** → **Project** → **Project Interpreter**
- ▶ Click the + button next to the list of packages
- ▶ Search for and install the `tweepy` package

# Your first bot

- Enter the following code, but **don't** run it yet

```
import tweepy

CONSUMER_KEY = '...'
CONSUMER_SECRET = '...'
ACCESS_KEY = '...'
ACCESS_SECRET = '...'

auth = tweepy.OAuthHandler(CONSUMER_KEY, ↵
    CONSUMER_SECRET)
auth.set_access_token(ACCESS_KEY, ACCESS_SECRET)

api = tweepy.API(auth)
api.update_status("Hello, world!")
```

# Adding your API keys

- ▶ Go to `https://apps.twitter.com` and click on your app
- ▶ Click on **Keys and Access Tokens**
- ▶ Copy and paste the **Consumer Key** and **Consumer Secret** into the code, replacing the . . .
  - ▶ Do this **carefully** – ensure there are no extraneous spaces or other characters between the ' quotes
- ▶ Click on **Create my access token**
- ▶ Copy and paste the **Access Token** and **Access Token Secret** into the code, replacing the . . .

# API keys: best practices

- ▶ Many web APIs require an **API key**
- ▶ This is like a **password** and should be kept **secret**
- ▶ Code in a world-readable GitHub repository is **not secret!**
- ▶ Easy solution: put your API keys in a separate source / header / configuration file, and use `.gitignore` to keep that file out of the repository
  - ▶ NB: For assignment submissions via LearningSpace, please **do** include your API keys so that we can test your code!
- ▶ Other solutions: `http://programmers.stackexchange.com/q/205606`

# Fin

- ▶ Run the code
- ▶ Open the twitter app or website and admire your bot's first tweet!
- ▶ You now know how to tweet any text string — integrate this into your Python code as you see fit
- ▶ Refer to the docs on <http://www.tweepy.org> for how to do more interesting things, e.g. reading and replying to other people's tweets

# Further reading

- ▶ Twitter, “Automation rules and best practices”.  
<https://support.twitter.com/articles/76915>
- ▶ Darius Kazemi, “Basic Twitter bot etiquette”.  
<http://tinysubversions.com/2013/03/basic-twitter-bot-etiquette/>

# **User authentication with OAuth**



# User authentication

```
ACCESS_KEY = '...'  
ACCESS_SECRET = '...'  
  
auth = tweepy.OAuthHandler(CONSUMER_KEY, ↵  
    CONSUMER_SECRET)  
auth.set_access_token(ACCESS_KEY, ACCESS_SECRET)
```

- ▶ OK for creating a bot that only ever tweets on its own account
- ▶ Not suitable for writing a game component that allows users to use their own Twitter accounts, i.e. tweets on the user's behalf

# OAuth

- ▶ Twitter (and many other web services) use **OAuth**
- ▶ Allows you (the user) to use a third-party app without giving it your account password
- ▶ OAuth is yet another API to get to grips with...
- ▶ Tweepy has a Twitter-specific wrapper for OAuth

# Using OAuth

```
import tweepy
import webbrowser

CONSUMER_KEY = '...'
CONSUMER_SECRET = '...'

auth = tweepy.OAuthHandler(CONSUMER_KEY, ←
                             CONSUMER_SECRET)

print "Opening Twitter website -- please log in"
webbrowser.open(auth.get_authorization_url())
verifier = raw_input("Verification code:")
auth.get_access_token(verifier)

api = tweepy.API(auth)
api.update_status("Hello, world!")
```

# Staying logged in

- ▶ Store `auth.access_token` and `auth.access_token_secret` along with your application's saved data
- ▶ Now these can be reloaded and passed to `auth.set_access_token`, just like in our first bot example

**Sprint reviews and general support**