Automate & Chill

Sit back, relax, and automate your existence.

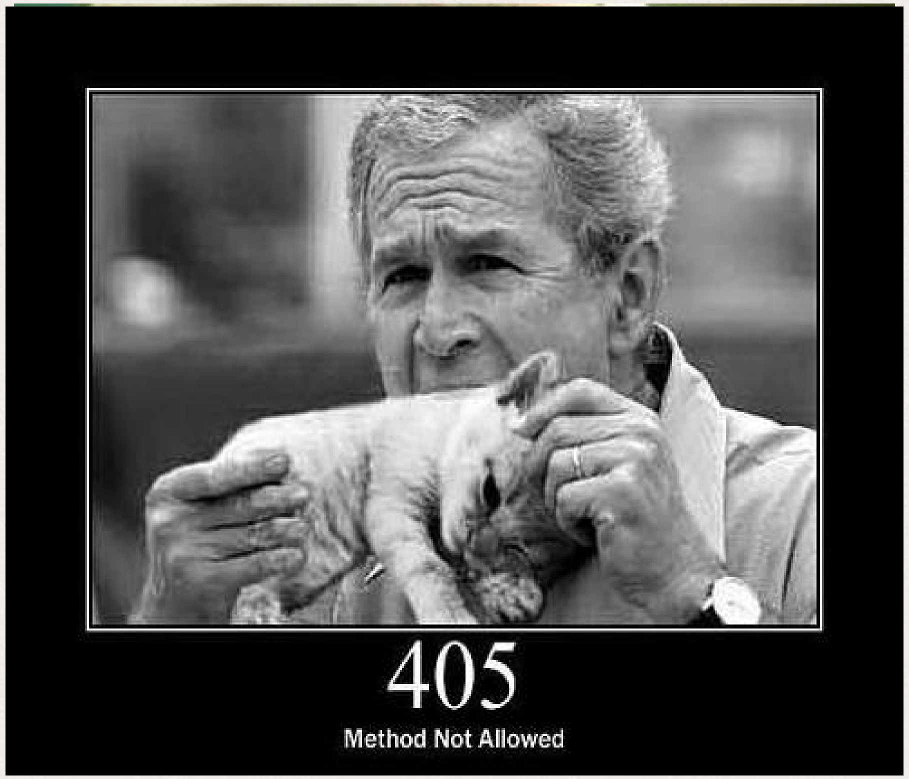
Redbrick
DCU's Networking Society

Automation

- Method of operating or controlling a process while reducing human intervention.
- Redbrick likes to automate stuff.

Why automate?

- Faster
- Consistent output
- Predictable
- Less room for human error



Redbrick
DCU's Networking Society

What you can automate?

- Social Media
- Emails
- Download notes
- Backups

Redbrick
DCU's Networking Society

Odd things to automate

Redbrick
DCU's Networking Society

IFTTT

- If This Then That
- Creates chains of simple conditional statements.
- No scripting.

Redbrick
DCU's Networking Society

Node-Red

- Didn't discover until this week.
- Painless, next level IFTTT.
- Can be used to capture IFTTT triggers.
- Check out at https://github.com/node-red/node-red

What we have automated

- User Registration and Management
- Student Registration
- Room Booking
- Voting

Redbrick
DCU's Networking Society

useradm

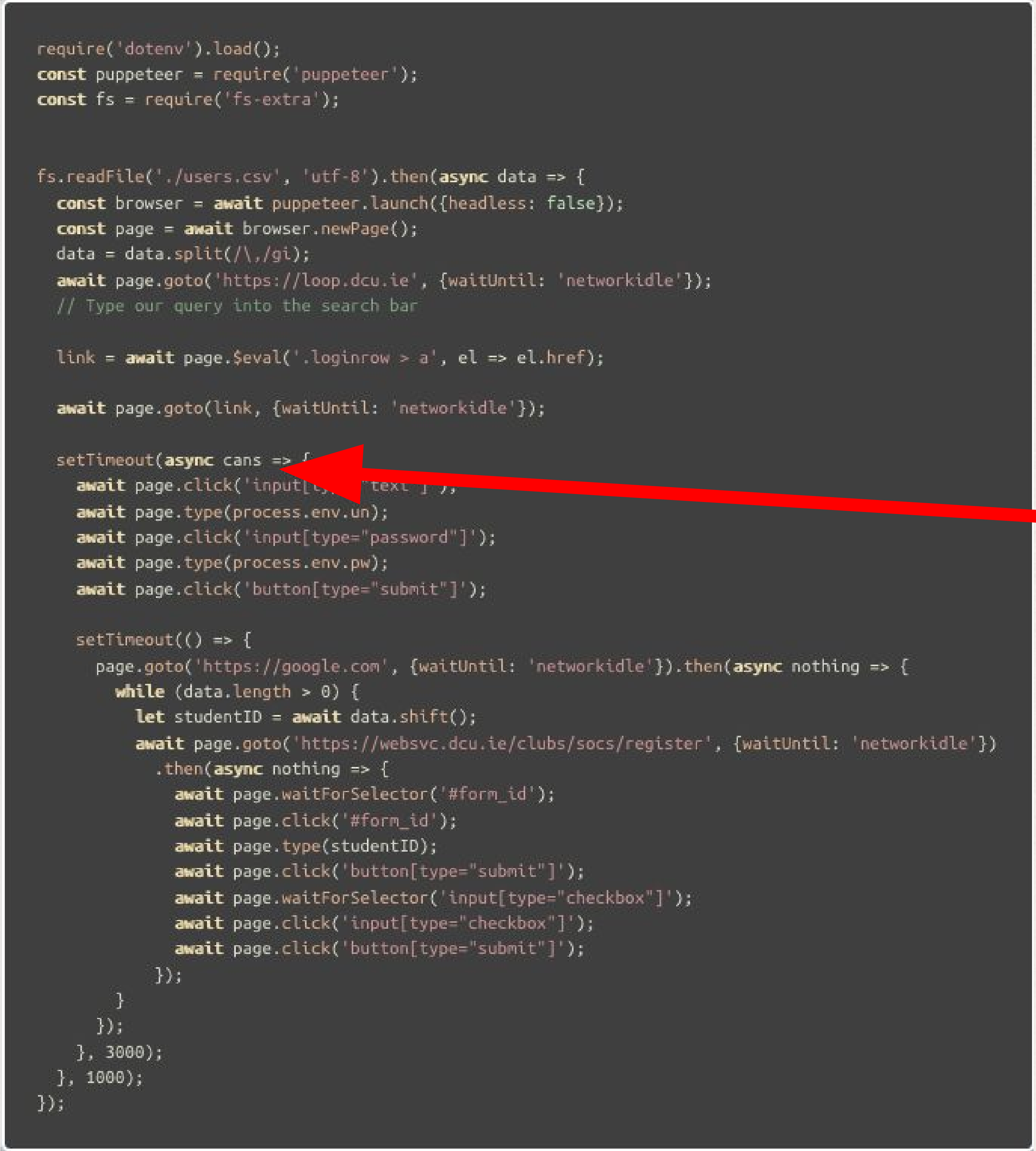
- Helps us manage Redbrick's membership.
- Creates new users, renews accounts.
- Check it out at - https://github.com/redbrick/useradm

Redbrick
DCU's Networking Society

Student Register

- Written in NodeJS.
- Uses puppeteer and reads a csv.
- Submits the info on our membership.
- Much faster than the alternative.

Redbrick
DCU's Networking Society



```
require('dotenv').load();
const puppeteer = require('puppeteer');
const fs = require('fs-extra');


fs.readFile('./users.csv', 'utf-8').then(async data => {
  const browser = await puppeteer.launch({headless: false});
  const page = await browser.newPage();
  data = data.split(/\,/gi);
  await page.goto('https://loop.dcu.ie', {waitUntil: 'networkidle'});
  // Type our query into the search bar

  link = await page.$eval('.loginrow > a', el => el.href);

  await page.goto(link, {waitUntil: 'networkidle'});

  setTimeout(async cans => {
    await page.click('input[type="text"]');
    await page.type(process.env.un);
    await page.click('input[type="password"]');
    await page.type(process.env.pw);
    await page.click('button[type="submit"]');

    setTimeout(() => {
      page.goto('https://google.com', {waitUntil: 'networkidle'}).then(async nothing => {
        while (data.length > 0) {
          let studentID = await data.shift();
          await page.goto('https://websvc.dcu.ie/clubs/socs/register', {waitUntil: 'networkidle'})
            .then(async nothing => {
              await page.waitForSelector('#form_id');
              await page.click('#form_id');
              await page.type(studentID);
              await page.click('button[type="submit"]');
              await page.waitForSelector('input[type="checkbox"]');
              await page.click('input[type="checkbox"]');
              await page.click('button[type="submit"]');
            });
        }
      });
    }, 3000);
  }, 1000);
});
```

Reads a csv of student numbers.

Launches a headless browser using puppeteer.

Yes that does say “async cans”

Logs in and goes to registry page.

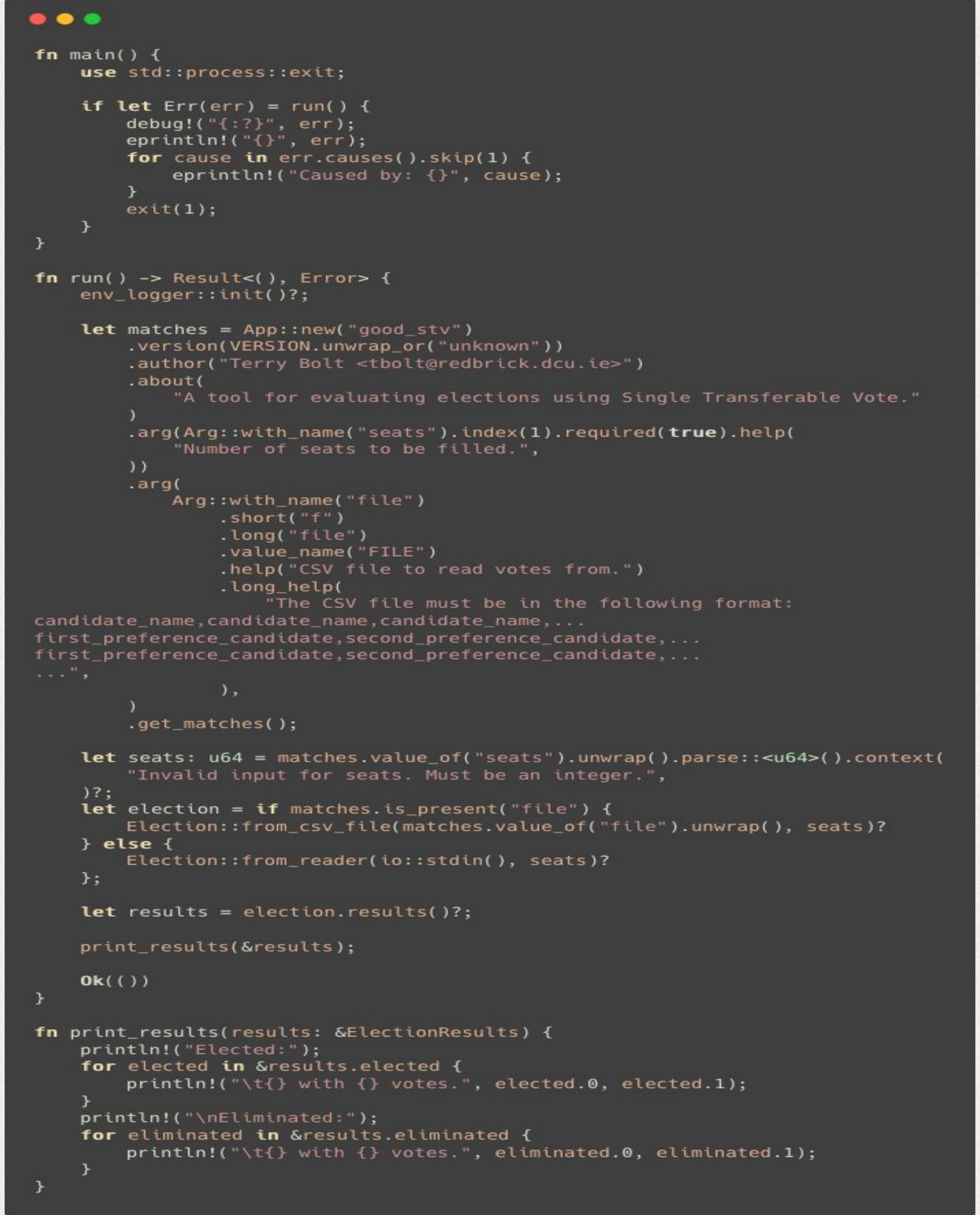
Enters student number one by one.

Redbrick
DCU's Networking Society

good_stv

- Yes there was a bad_stv.
- Fast and quite robust in my opinion.
- good_stv is written in Rust.

Redbrick
DCU's Networking Society



```rust
fn main() {
    use std::process::exit;

    if let Err(err) = run() {
        debug!("{:?}", err);
        eprintln!("{}", err);
        for cause in err.causes().skip(1) {
            eprintln!("Caused by: {}", cause);
        }
        exit(1);
    }
}

fn run() -> Result<(), Error> {
    env_logger::init()?;

    let matches = App::new("good_stv")
        .version(VERSION.unwrap_or("unknown"))
        .author("Terry Bolt <tbolt@redbrick.dcu.ie>")
        .about(
            "A tool for evaluating elections using Single Transferable Vote."
        )
        .arg(Arg::with_name("seats").index(1).required(true).help(
            "Number of seats to be filled.",
        ))
        .arg(
            Arg::with_name("file")
                .short("f")
                .long("file")
                .value_name("FILE")
                .help("CSV file to read votes from.")
                .long_help(
                    "The CSV file must be in the following format:
candidate_name,candidate_name,candidate_name,...
first_preference_candidate,second_preference_candidate,...
first_preference_candidate,second_preference_candidate,...
...",
                ),
        )
        .get_matches();

    let seats: u64 = matches.value_of("seats").unwrap().parse::<u64>().context(
        "Invalid input for seats. Must be an integer.",
    )?;
    let election = if matches.is_present("file") {
        Election::from_csv_file(matches.value_of("file").unwrap(), seats)?
    } else {
        Election::from_reader(io::stdin(), seats)?
    };

    let results = election.results()?;

    print_results(&results);

    Ok(())
}

fn print_results(results: &ElectionResults) {
    println!("Elected:");
    for elected in &results.elected {
        println!("\t{} with {} votes.", elected.0, elected.1);
    }
    println!("\nEliminated:");
    for eliminated in &results.eliminated {
        println!("\t{} with {} votes.", eliminated.0, eliminated.1);
    }
}
```

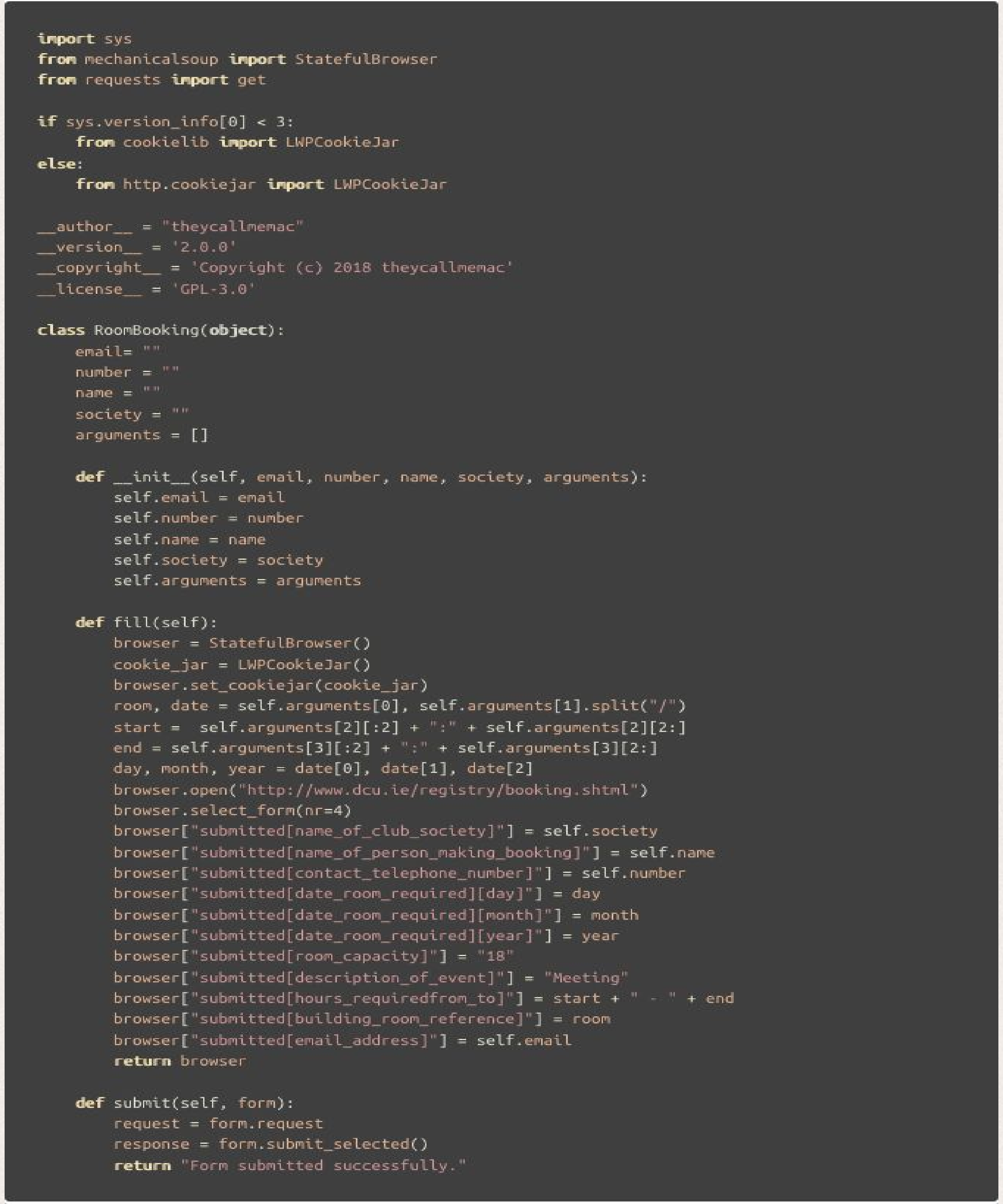
This is just some of the main.rs script.

Takes in a csv file and number of seats to be filled.

Handles almost anything you throw at it.

Room Booking and Lookup

- I wrote this. It is not good.
- Looks up and books rooms
- Uses BeautifulSoup, and MechanicalSoup
- I have automated myself out of a job.



```python
import sys
from mechanicalsoup import StatefulBrowser
from requests import get

if sys.version_info[0] < 3:
    from cookielib import LWPCookieJar
else:
    from http.cookiejar import LWPCookieJar

__author__ = "theycallmemac"
__version__ = '2.0.0'
__copyright__ = 'Copyright (c) 2018 theycallmemac'
__license__ = 'GPL-3.0'

class RoomBooking(object):
    email= ""
    number = ""
    name = ""
    society = ""
    arguments = []

    def __init__(self, email, number, name, society, arguments):
        self.email = email
        self.number = number
        self.name = name
        self.society = society
        self.arguments = arguments

    def fill(self):
        browser = StatefulBrowser()
        cookie_jar = LWPCookieJar()
        browser.set_cookiejar(cookie_jar)
        room, date = self.arguments[0], self.arguments[1].split("/")
        start =  self.arguments[2][:2] + ":" + self.arguments[2][2:]
        end = self.arguments[3][:2] + ":" + self.arguments[3][2:]
        day, month, year = date[0], date[1], date[2]
        browser.open("http://www.dcu.ie/registry/booking.shtml")
        browser.select_form(nr=4)
        browser["submitted[name_of_club_society]"] = self.society
        browser["submitted[name_of_person_making_booking]"] = self.name
        browser["submitted[contact_telephone_number]"] = self.number
        browser["submitted[date_room_required][day]"] = day
        browser["submitted[date_room_required][month]"] = month
        browser["submitted[date_room_required][year]"] = year
        browser["submitted[room_capacity]"] = "18"
        browser["submitted[description_of_event]"] = "Meeting"
        browser["submitted[hours_requiredfrom_to]"] = start + " - " + end
        browser["submitted[building_room_reference]"] = room
        browser["submitted[email_address]"] = self.email
        return browser

    def submit(self, form):
        request = form.request
        response = form.submit_selected()
        return "Form submitted successfully."
```

Lab Booking object from my room booking program.

Yes, the __init__ takes too many variables

“fill” interacts directly with https://www.dcu.ie/registry/booking.shtml

“submit” just confirms submission of the form.

Redbrick
DCU's Networking Society

Questions? (feedback)

Redbrick
DCU's Networking Society