CS131

Saketh Kasibatla

Today

- Scheme

- Python

This Class

- homework 5 is due **Monday, November 20**
- This homework will be graded with automated scripts
  - not compiling → no credit
  - code should behave exactly according to spec
  - check Piazza for clarifications

Any Questions?

Installing Scheme

- we'll be using racket for this class

- http://racket-lang.org/

Scheme Resources

- The Racket Guide
    - https://docs.racket-lang.org/guide/index.html
- The Racket Reference
    - https://docs.racket-lang.org/reference/index.html
- Racket Cheat Sheet
    - https://docs.racket-lang.org/racket-cheat/index.html

Scheme

- another functional programming language
- dynamic typing - types are attached to values, not variables
    - can check types at runtime
- blurs lines between program and data

Basics

- identifiers
  - string with any chars except `( ) [ ] { } " , ' ` ; # | \`
  - `hello`, `string->int`, `number?`
- numbers
  - `1`, `1/2`, `0.5`
  - `number?`, `integer?`, `rational?`, `real?`
- booleans
  - `#t`, `#f`
  - anything other than `#f` is considered true
  - `boolean?`

Basics

- strings
  - “hello world!”
  - (display “hello world!”)
  - string?

Useful Functions

- and, or, not
    - `(and a b c)` returns c if all 3 are true
    - `(or a b c)` returns the first value that isn't #f
- equal?, eq?
- +, -, *, /, modulo

Define

- `(define <id> <expr>)`
  - defines a variable
  - `(define PI 3.14159)`
- `(define (<id> <id>*) <expr>+)`
  - defines a function with 0 or more arguments
  - final expr in body is return value
  - `(define (timesTwo x) (* x 2))`
  - sugar for `(define timesTwo (lambda (x) (* x 2))`

Calling a function

- `(<expr> <expr>*)`
  - first expr evaluates to the function
  - `(+ 1 2 3 4 5)`
  - `(even? 2)`
- conditionals
  - `(if <expr> <expr> <expr>)`
  - `(cond {[ <expr> <expr> ]}*)`

Cond

```
(define (evenorodd x)
  (cond
    [(equal? (modulo x 2) 0) “even”]
    [else “odd”]))
```

Exercise

- Write the dotwice function

```
(dotwice (lambda (x) (* x 2)) 2)
8
```

Let and Scoping

- `(let ( {[ <id> <expr> ]}* ) <expr>+)`
  - `(let ([x 5] [y 6]) (+ x y))`
- `(let* ( {[ <id> <expr> ]}* ) <expr>+)`

- Scope: where a variable can be used
- `(let ([x 5]) (* y (let ([y 6]) y)))`
  - is y in scope?

Lists

```
(list “red” 2 “blue”)
‘(“red” 2 “blue”)

(list 1 2 3 4 5)
‘(1 2 3 4 5)

empty
‘()
```

- some useful functions: `length`, `append`, `reverse`, `member`, `list?`, `empty?`

Lists = Pairs

```
(cons 1 2)
(1 . 2)

(pair? empty)
#f

(pair? (cons 1 2))
#t
```

```
(pair? (list 1 2 3))
#t

(cons 1 (list 2 3))
‘(1 2 3)

(cons 1 (cons 2 3))
‘(1 2 . 3)
```

Pairs

```
(car (cons 1 2))
1

(cdr (cons 1 2))
2

(cdr (list 1 2 3))
‘(2 3)

(cdr (list 1))
‘()
```

Improper Lists

- lists that don’t end in ‘()

- e.g. `(cdr (cons 1 2))` returns 2

Exercise

- Write map, which takes a function f, and a list l, and returns a new list that has the results of applying f to each element in l.
- Can we do this using tail recursion? Is your solution tail recursive?

```
(map (lambda (x) x*2) (list 1 2 3))
‘(2 4 6)
```

Quote

- lists and programs are the same thing!
- the only difference is the way they are treated

```
(list 1 2 3)
‘(1 2 3)

(quote (list 1 2 3))
‘(list 1 2 3)


‘(list 1 2 3)
‘(list 1 2 3)
```

Eval

- opposite of quote
- take a list and treat it like a program

```
(define ns (make-base-namespace))
(eval '(cons 1 2) ns) ; must pass in a namespace
```

Installing Python

- https://www.python.org/

- we will be using python 3.6 in this class

Python

- Dynamically Typed
- Scripting Language
- Meant to have easy syntax
    - not context free. Whitespace matters

Cheat Sheet

```
print(“Hello World”)

bikes = [‘trek’, ‘redline’, ‘giant’]
for bike in bikes:
  print(bike)
bikes.append(‘roadbike’)

if x < 2:
  print(‘less than 2’)
elif x > 3:
  print(‘greater than 3’)
else:
  print(‘2 or 3’)
```

```
alien = {
  ‘color’: ‘green’,
  ‘points’: 5
}
alien[‘x_position’] = 0
print(alien[‘color’])
```

```
class Dog():
  """Represent a dog."""
  def __init__(self, name):
    """Initialize dog object."""
    self.name = name

  def sit(self):
    """Simulate sitting."""
    print(self.name + “sits.")
```

Variadic Arguments

```
def printArguments(*args, **kwargs):
  print(‘args’, args)
  print(‘kwards’, kwargs)

printArguments(1, 2, “hi”, r=3, f=4)
```

Project

- Flooding algorithm
  - servers talk to clients
  - `IAMAT kiwi.cs.ucla.edu +34.068930-118.445127 1479413884.392014450`
    - `AT Alford +0.263873386 kiwi.cs.ucla.edu +34.068930-118.445127 1479413884.392014450`
  - servers share info with each other, using TCP
  - can use ‘AT’ messages or your own format
- ask any server about interesting stuff around a particular client
- `WHATSAT kiwi.cs.ucla.edu 10 5`

Asyncio

- Python library for asynchronous computation
- used to handle events that don't come in any particular order (like TCP connections)
- includes a library for TCP messaging
  - https://docs.python.org/3/library/asyncio.html
  - https://github.com/python/cpython/tree/3.6/Lib/asyncio/

Google Places API

- Used to query for places around a location
- Server responds using JSON format. use `json` library in python
- request url: https://maps.googleapis.com/maps/api/place/textsearch/json?<*parameters*>
  - parameters: location=-33.86,151.19&radius=50&key=API_KEY
  - more info at: https://developers.google.com/places/webservice/search

Google Places API

- visit https://console.developers.google.com to create an API project/key

- turn on service at: https://code.google.com/apis/console

Google Places API

```
import urllib2
import json

PLACES = “…”
response = urllib2.urlopen(PLACES)
data = json.load(response)
print(data)
__
```


{u'status': u'OK', u'next_page_token':
u'CvQB7AAAAPYRQizjo2phtNQiK6XTdOP0IAcJApBKJQOIvnMar3Q2c9Z3Pw9IshfiuinBWd4PPHHx2bIbDAHe1SUYbUF6Rmzlr2mO5QfJf0xVjkEbnZZPrChkiTn58rSvFGVEX0FK
Mau1tDrSkHCf8yTW4CDJlcKWdy-QmgmNEqb0PZSqIYrGEMbDm3UCFZXFL05Rju-_eRYdaqXhbJHAHsvfJ9qg3Pq0MbQF8vxZjZeNCSCykG6IZTfs54eKS8mBkvgYreQemf0myz
CcdxgYfyV8ygdRHWswEH_6EiG_Mn_A-4uZnNH0eGppSCxJ9vvzZ21YCZYUkhIQbCabDSgJrMSbNqli6x1V3RoUKUCt8SIO6AGUyc54qbL3VRt0OG4', u'html_attributions':
[], u'results': [{u'name': u'Surry Hills', u'reference':
u'CoQBfAAAAAoLb5QWBPT_xOFCyETDIm1E0S2IaOW6K5crT2PWJnlKNMiADUvw7om7IQJlgaEnKD3CGXH8uYDK1jQHD69IDCyKyeaNujQFx-xOns5dmtRKskng4MSO309bcsr
BJNySMZc5dCkVWKqT09ezX1GLCqHaPfMgy1k-Y1FjkS4BiKLTEhCyT6LUNlXPpzzhIbSyuFWbGhQnjrYHE2snDML3d8ywSaYYuW5iLw', u'geometry': {u'location':
{u'lat': -33.8906466, u'lng': 151.2129254}, u'viewport': {u'northeast': {u'lat': -33.8768894, u'lng': 151.2180916}, u'southwest': {u'lat':
-33.8921154, u'lng': 151.201598}}}, u'place_id': u'ChIJW7w1-SGuEmsRkMwyFmh9AQU', u'vicinity': u'Surry Hills', u'scope': u'GOOGLE', u'id':
u'ecf153a93d24ae472f6f26d8f0e54f6a0e0debca', u'types': [u'locality', u'political'], u'icon': u'http://maps.gstatic.com/mapfiles/place_api/
icons/geocode-71.png'}, {u'name': u'Organic Bread Bar', u'reference':
u'CnRkAAAAG8s0_jTiV1rJ28iPxRfDpHZBmByUZvY0jNkwgQU4u0BNHhZiJ1x5uuUlYthUaY1IAs5i24-h70aXmazLbaY1UfW3sDlYZyMyBSdmbBKVfswQR9ameLdWhqvznUib-
Ehth cf3qrzGgqtwGyxCY8cnyxIQnuA723GHV-1so4v6NDJVpBoU_ak1m8ui1o8477-_9xJCKc4w2AM', ...

Google Places API

- BUT, we're using Asyncio, and urllib2 is synchronous
- You'll have to implement an HTTP GET request using TCP
- https://tools.ietf.org/html/rfc7230#section-3
    - message format
- https://tools.ietf.org/html/rfc7231#section-4.3.1
    - definition of a GET request

What is a GET request

- specially formatted TCP message sent to a URL

```
GET uri HTTP/1.1\r\n
Host: host\r\n
Content-Type: text/plain; charset=utf-8\r\n
\r\n
```

- Use urllib.parse to divide a string up into its respective pieces
    - https://maps.googleapis.com/maps/api/place/nearbysearch/json?…
    - hostname: maps.googleapis.com
    - uri: /maps/api/place/nearbysearch/json?…
- Need to parse raw response as well.'\r\n' separates headers from bodies

Tips

- Places API uses HTTPS. Be sure to use SSL instead of plain TCP when talking to Google servers

- Use telnet to test your protocol. Telnet generates TCP messages

- Servers should use the most recent location of a client in their places requests. Check the timestamp of different AT messages.