CSC104 fall 2013

Computational thinking

week 11

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/104/F12/
416-978-5899

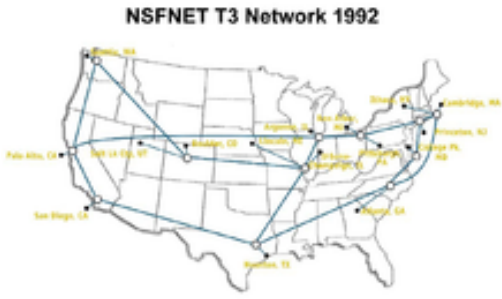
Text: Picturing Programs

Outline

privacy

Notes

share, but don't share



information wants
to be shared...
how much?
with whom?

privacy, pro or con?

make your lists...

concentric circles of privacy

who fits where?

privacy leaks

know privacy's plumbing

buyer loyalty plans what do they want for those "deals"?

surveys not just an interruption

credit information who knows you paid late?

recorded viewing check the agreement

black boxes not just for crashes

911 where's waldo?

rfid bar codes, shopping history, drugs?

computer use what's your admin see?

cookies where have you been browsing?

required leaks

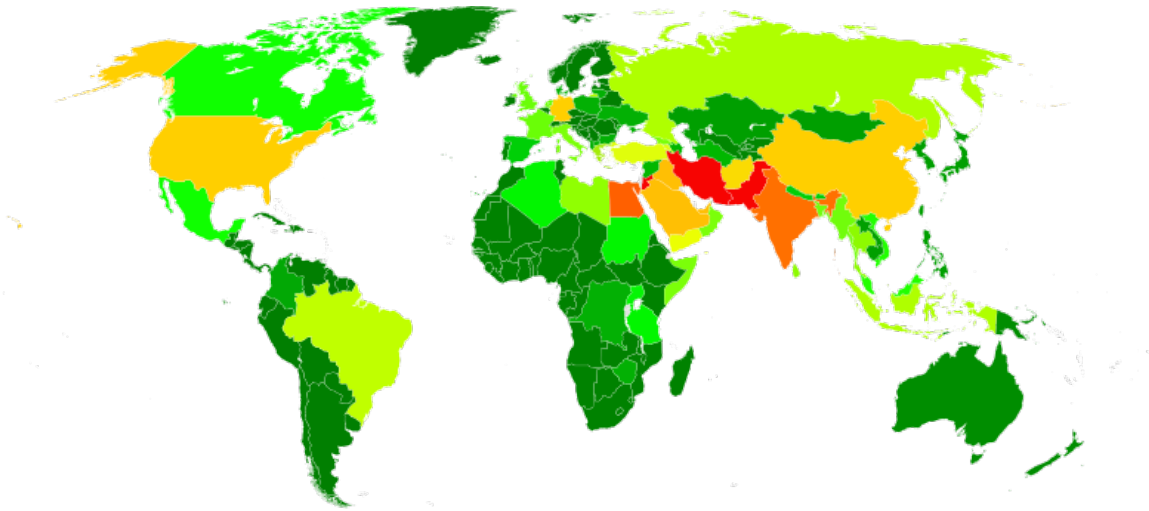
can lose your identity

Being born, working, imprisoned, or paying taxes, can generate information about you that you aren't allowed to keep private. How many of these do you think are necessary to identify an American with more than 80% accuracy (according to Dr Latanya Sweeny)?

- hospital of birth
- date of birth
- gender
- postal code
- blood type

prism program

view from one set of eyes:



where are Canada's eyes?

what about U of T?

student email has been contracted out to Microsoft. The administration says this will:

- allow your, possibly personal, data to be shared with the U.S. government
- perhaps increase the chance that your name will appear on a no-fly list
- not infringe on your privacy rights (!)

what can you do?

increase privacy and security

- passwords and pins
- know something about contacts
- update security software, OSs
- careful when you click
- **speak up!**

For important stuff **encrypt**

flatten

```
; flatten : list -> list
(define (flatten L)
  (cond
    [(cons? L) (apply append (map flatten L))]
    [else (list L)]))

; predict what (flatten 3) does

; predict what (flatten (list 3)) does

; predict what (flatten (list 1 2 (list 3))) does
```

depth

```
; depth : list -> number
(define (depth L)
  (cond
    [(cons? L) (+ 1 (apply max (map depth L)))]
    [else 0]))

; predict what (depth 3) does

; predict what (depth (list 3 4)) does

; predict what (depth (list 3 4 (list 5 6))) does
```

Notes