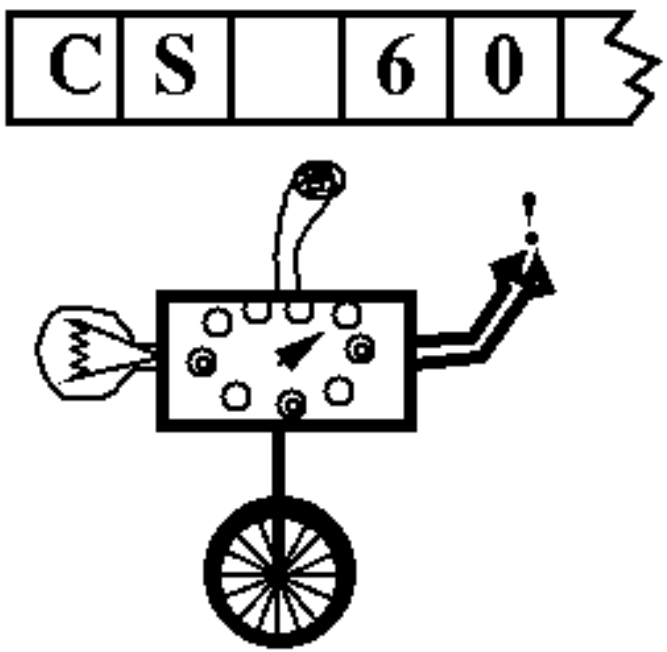
Welcome!

January 17, 2012

CS 60: Principles of Computer Science



Course Comparison

CS 5 Goals:

- ✓ Solving computational problems in *one* language
- ✓ Conveying some of CS's *breadth*

CS 60 Goals:

- ✓ Solving computational problems in *several* languages
- ✓ Conveying more of CS's *depth*
- ✓ More emphasis on clarity, efficiency, and limits of computation.
- ✓ Strengthening programming skills (prep for CS 70)

Syllabus, in brief

Lectures	MW or TR, 1:15–2:30pm, GA Pryne Covers key skills, topics, motivation Provides insight into HW problems Attendence is required.
Website	`http://www.cs.hmc.edu/cs60`
On-line Help	`http://piazza.com/hmc/spring2012/cs60`
Off-line Help	See Piazza for tutors and office hours
HW Submission	`http://www.cs.hmc.edu/submit`

Grading and Homeworks

Grades About 65% Homeworks, 30% Exams; 5% Participation

```
(define (calc-grade pct)
   (cond
      [(>= pct 0.95) "A"]
      [(>= pct 0.90) "A-"]
      ;; and so on...
```

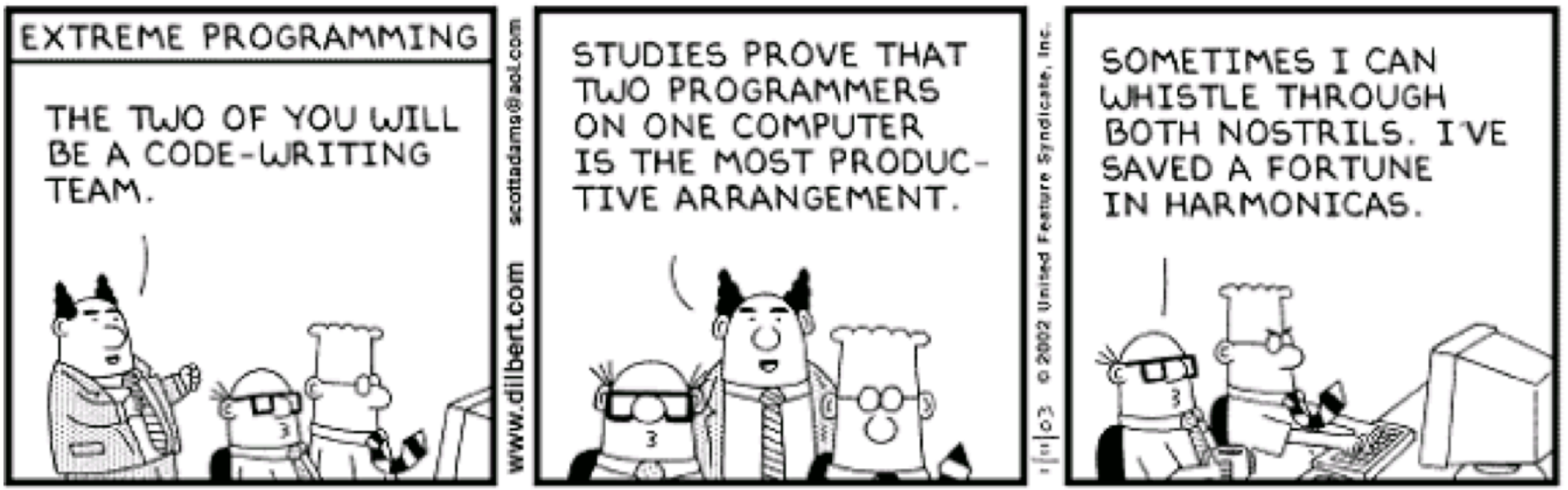
Homeworks Assigned weekly

Due Monday nights at 11:59pm

3 "CS 60 Euros" (extensions to Tue. 11:59pm)

Some parts individual, others pair-optional

Pair Programming



On *some* problems, you may "pair program"

- ✓ both people at *one* computer at the same time
- ✓ trade off "driver" and "navigator" every 30 mins
- ✓ both people contribute to & understand solution

Individual problems must be completed individually.
Read the Collaboration Policy!

Logging In: 1 name, 3 passwords

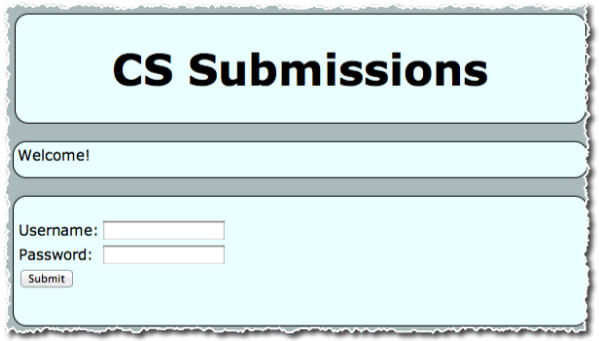
CIS Labs
(or your home college)

They reset passwords

CS Labs
(BK B102, BK B105)

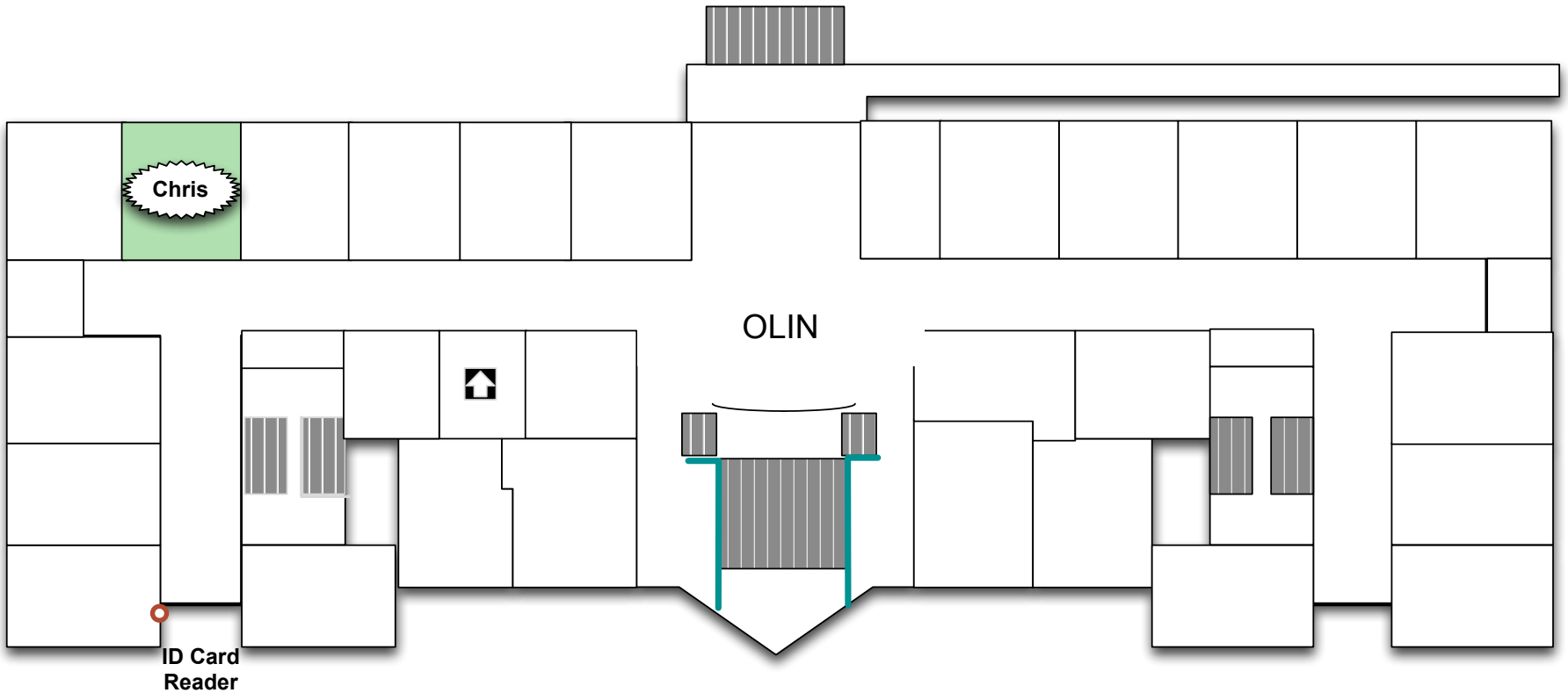
Tim (BK B101) resets passwords. Doors: 23-5-4

`http://www.cs.hmc.edu/submit`



Prof. resets passwords.
Never got one? `asdf`

Finding Me



Review

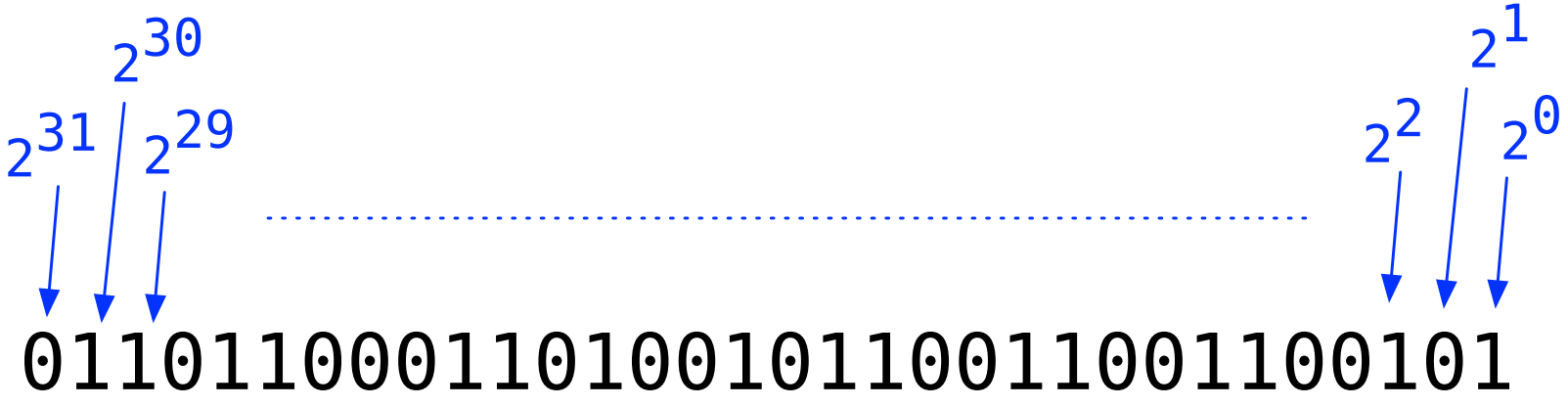
A Couple Big Ideas in Computer Science

- ✓ Everything's just bits
- ✓ Interfaces vs. Implementation
- ✓ Paradigms of Programming

What is This?

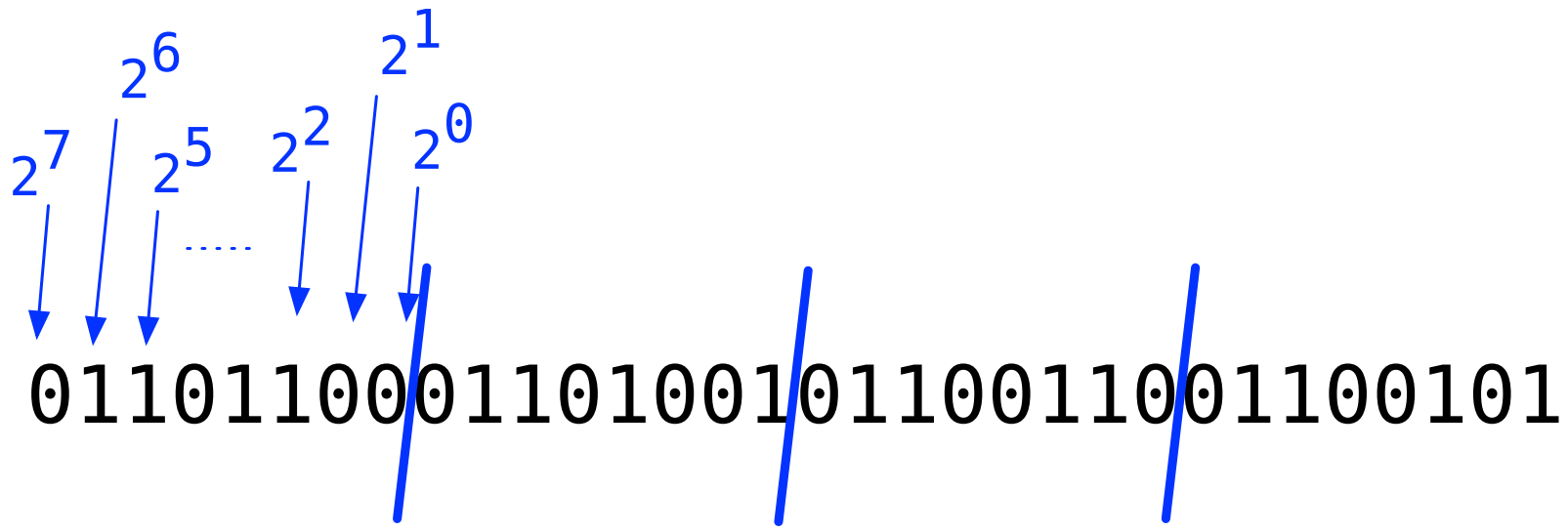
```
01101100011010010110011001100101
```

A 32-bit Integer?



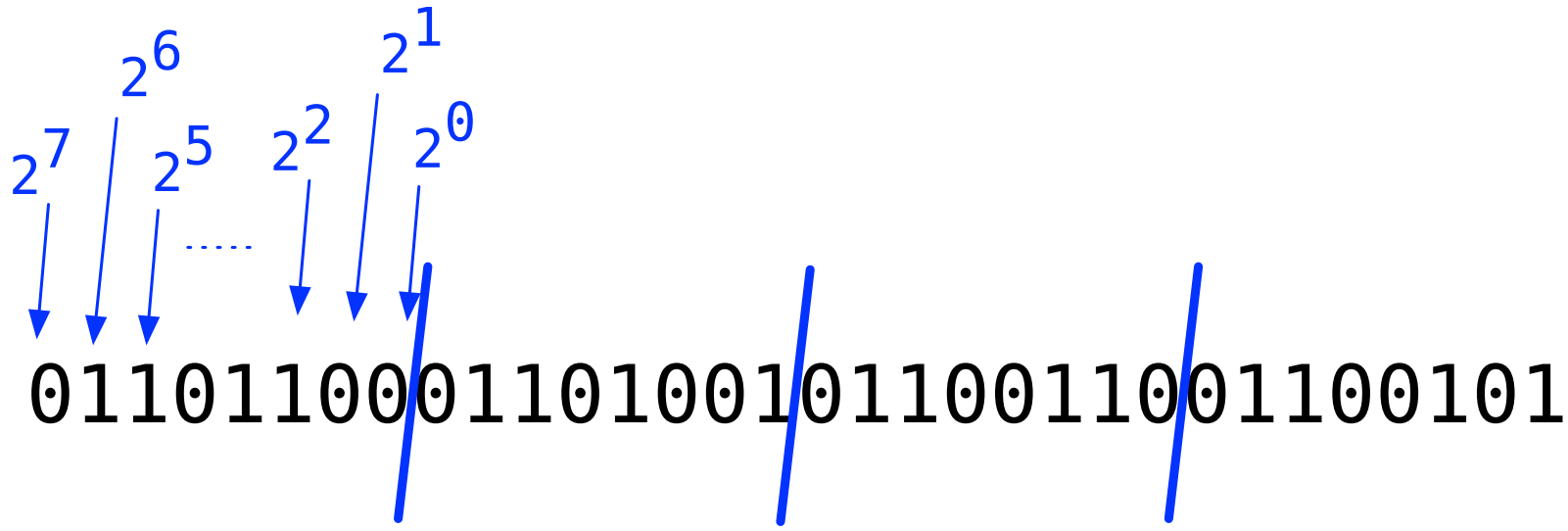
$$2^{30} + 2^{29} + \ldots + 2^{2} + 2^{0} = \underline{1818846821}$$

Four 8-bit Integers?



0x6c = 108 0x69 = 105 0x66 = 102 0x65 = 101

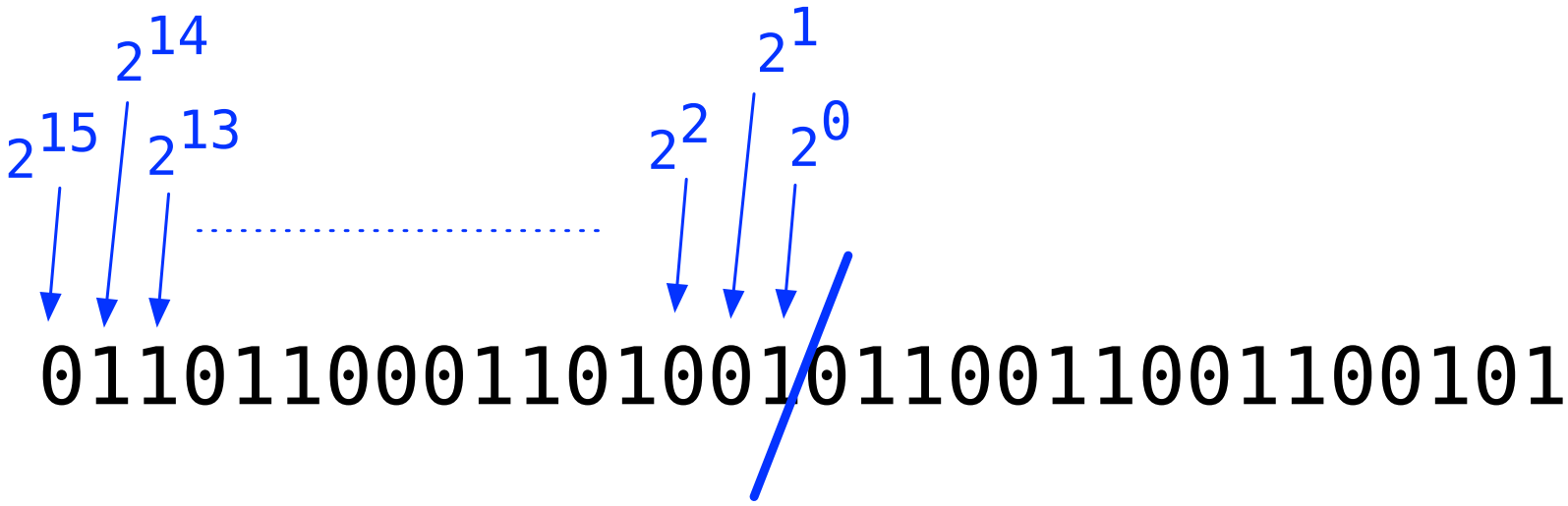
Four ASCII Characters?



0x6c = 108 = 'l' 0x69 = 105 = 'i' 0x66 = 102 = 'f' 0x65 = 101 = 'e'

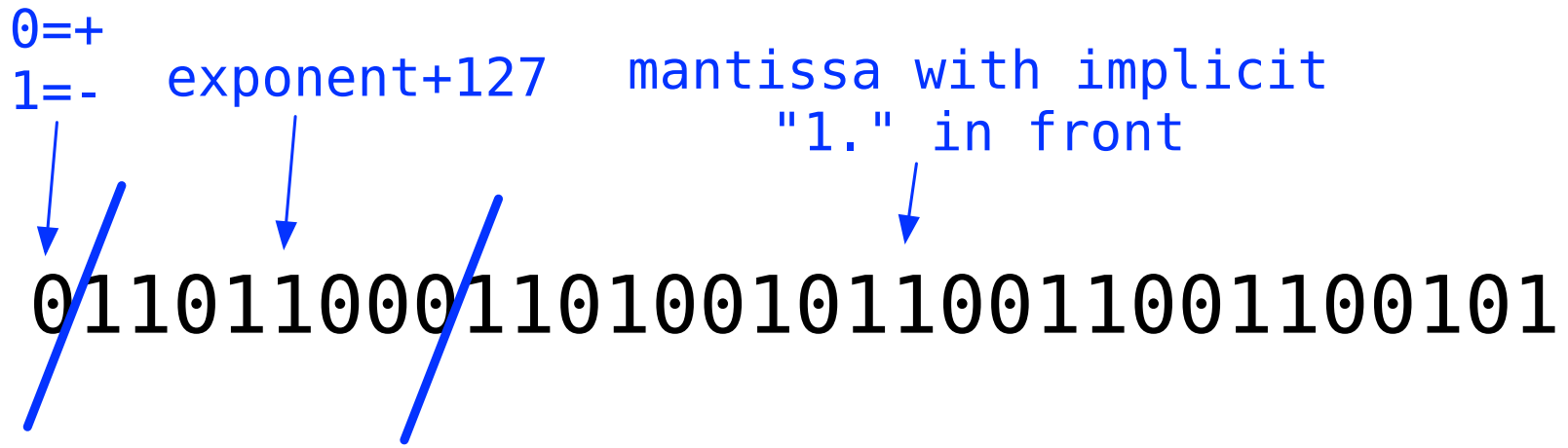
(ASCII Character Set)

Two Unicode Characters?



(Unicode Character Set)

A Floating-Point Number?



$$+\ 1.110100...101_2 \times 2^{(216-127)}$$

$$\approx \underline{1.12865304 \times 10^{27}}$$

Part of a Picture?

One brightness number per pixel
(0 = pure black, 255 = pure white)

Arrow points to 4 consecutive pixels
with brightness levels 108, 105, 102, 101

A Bit of Abstraction

There won't be that many bits in CS 60!

Why?

Abstraction

We rarely care how our numbers, strings, pictures, etc. are represented as bits!

(Similarly, engineers rarely worry about with individual atoms.)

If we know the *interface*, we can ignore the *implementation*!

- ✓ **Interface:** How can it be used?
- ✓ **Implementation:** How does it actually work?

Layers of Abstraction

Abstraction is the only way to build really big software systems!

- ✓ To compare two strings: need string representation
- ✓ To sort many strings: need comparison operator
  (but don't care how it works)
- ✓ To display mp3 files: need to sort strings
  (but don't care how that works)

Warning: although we can use code without knowing *how* it works, we might worry *how efficient* it is.

Sapir-Whorf Hypothesis

(Not this Worf)

"The language we use determines the way in which we view and think about the world" —Sapir and Whorf (1950's)

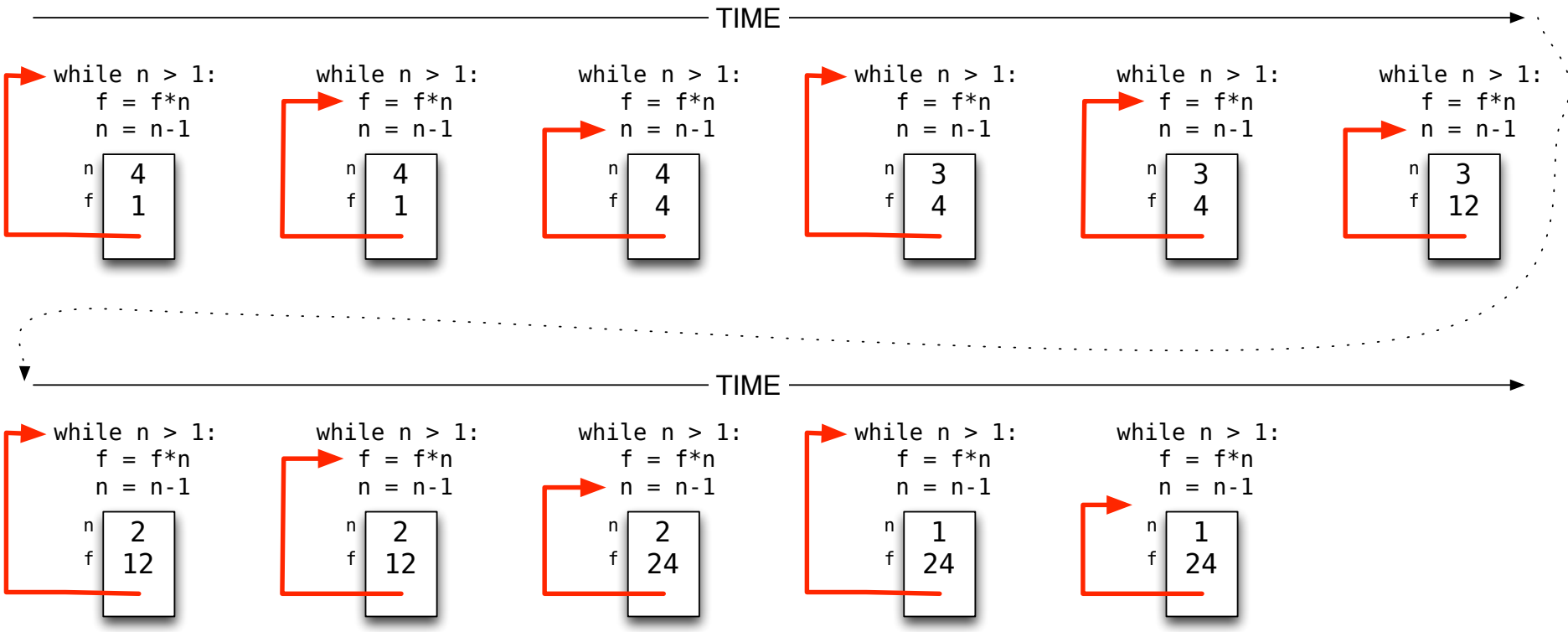
"A language that doesn't affect how you think about programming isn't worth knowing." — Alan Perlis

"For application software, you want to be using the most powerful (reasonably efficient) language you can get. . . [yet programmers are] satisfied with whatever language they happen to use, because it dictates the way they think about programs." — Paul Graham

Imperative Programming

Step-by-step instructions for updating memory (data)

```
while n > 1:
    f = f*n
    n = n-1
```



Functional Programming

Calculating answers (in terms of sub-calculations)

$$\text{fact}(n) := \begin{cases} 1 & \text{if } n = 0 \\ n \times \text{fact}(n-1) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \therefore \quad \text{fact}(4) &= 4 \times \text{fact}(3) \\ &= 4 \times (3 \times \text{fact}(2)) \\ &= 4 \times (3 \times (2 \times \text{fact}(1))) \\ &= 4 \times (3 \times (2 \times (1 \times \text{fact}(0)))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \\ &= 4 \times (3 \times 2) \\ &= 4 \times 6 \\ &= 24 \end{aligned}$$

A Quick Python Comparison

Imperative

```
>>> mySet = set([2,4,6,8])

>>> mySet
set([8, 2, 4, 6])

>>> mySet.add(3)

>>> mySet
set([8, 2, 3, 4, 6])
```

Functional

```
>>> mySet = set([2,4,6,8])

>>> mySet
set([8, 2, 4, 6])

>>> newSet = mySet.union([3]))

>>> newSet
set([8, 2, 3, 4, 6])

>>> mySet
set([8, 2, 4, 6])
```

Declarative Programming

Describe the solution, not the process.

```
# Prolog Example
#   Assumes permutation and increasing
#   were previously defined

sort(In,Out) :- permutation(In,Out), increasing(Out).

# Then we can ask...

?- sort([3,6,2,1], Answer).
Answer = [1, 2, 3, 6].
```

Racket: A Functional Language

- ✓ Racket is a variant of Scheme
- ✓ Scheme is a "cleaned up" version of LISP
  (one of the very first programming languages!)
- ✓ Key characteristics
  - lists are the main data structure
  - almost everything is a list, including your program!
  - closely related to Alonzo Church's λ-Calculus, which predates computers (and even Turing Machines)!

Questions When Learning a New Language

- ✓ What kinds of data are supported? (booleans, integers, . . .)
- ✓ How do I give names to data?
- ✓ How do I call functions and built-in operations?
- ✓ How do I define my own functions?
- ✓ How do I make choices?

Racket: What Kinds of Data?

`#f  #t`	booleans
`42`	integers
`42.0`	floating-point
`"this is a string"`	strings
`#\c`	characters
`year  x`	symbols
`(a b c)  (a "hi" (42 19.0))`	lists

How Do I Give Names To Data?

```
(define answer 42)

(define howdy "hello world!")
```

How do I use built-in operations?

```
(+ 20 22)              ;; Everything in *prefix* notation
(+ 60 (- 18))
(zero? (- 27 19))
```

Other Operators

`and` `or` `not`	*boolean operators*
`+` `-` `*` `/` `modulo` `quotient`	*arithmetic*
`max` `min` `expt` `sqrt` `sin` `cos` …	*mathematical functions*

How Do I Define My Own Functions?

```
(define (f N)
   (* N (+ N 1)))

(define (sum-upto N)
   (/ (f N) 2))
```

How Do I Make Choices?

```
(if b
    true-branch
    false-branch)
```

```
(cond
    ( test1   result1 )
    ( test2   result2 )
      ⋮
    ( else    result3 )
)
```

Example: add42

```
;; example scheme function
;; add42: adds 42
;;   inputs: an integer, N
;;   outputs: 42 more than N

(define (add42 N)
  (+ 42 N))
```

Example: is42

```
;; is42: is it Douglas Adams's
;;       answer?
;;   inputs: an integer, N
;;   outputs: true if N is 42,
;;            false otherwise

(define (is42 N)
  (if (equal? 42 N)
      #t
      #f
      ))
```

```
;; Or...

(define (is42 N)
   (equal? 42 N))
```

Example: sign

```
;; sign: returns -1, 0, or 1
;;   inputs: an integer, N
;;   outputs: -1 if N < 0;
;;             1 if N > 0;
;;             0 otherwise
(define (sign N)
  (cond
    ( (> N 0)  1 )
    ( (< N 0) -1 )
    (  else    0 )
    ))
```

Example: is-odd

Python

```
def is-odd(N):
  """Determines whether
     the given N is odd"""
  if N % 2 == 1 :
     return True
  else:
     return False
```

Racket

```
;; returns #t when given
;;   an odd number, and
;;   #f otherwise.
(define (is-odd N)



)
```

Example: fac

Python

```
def fac(N):
  """Returns the factorial
     of the given number"""
  if N < 2:
    return 1
  else:
    return N * fac(N-1)
```

Racket

```
;; computes the factorial
;;   of the given number
(define (fac N)




)
```

Example: halve-count

Racket

```
;; Returns the number of
;; times we can divide the
;; given integer by 2
;; (rounding down) until 1.
(define (halve-count N)




)
```

Examples

```
> (halve-count 8)
3

> (halve-count 11)
3

> (halve-count 1)
0
```

What's another name for this function?

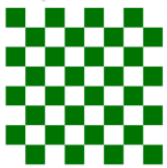
Assignment 0 Due Monday 11:59pm

Question 0: Using Piazza

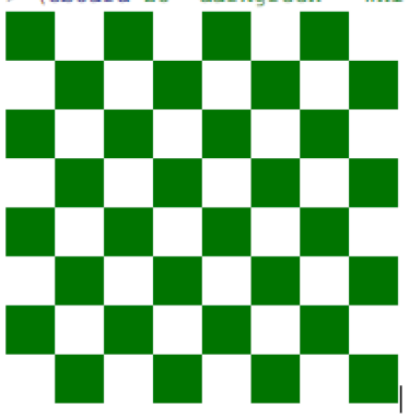
Question 1: Creating Simple Pictures

```
> (cboard 10 "gold" "black")
```

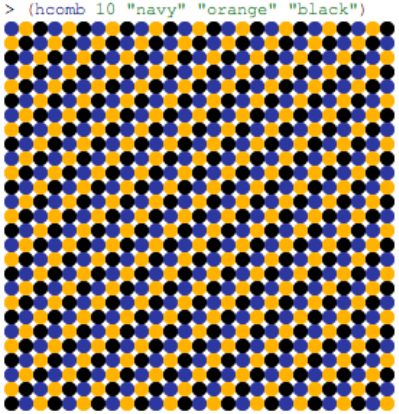
```
> (cboard 10 "darkgreen" "white")
```



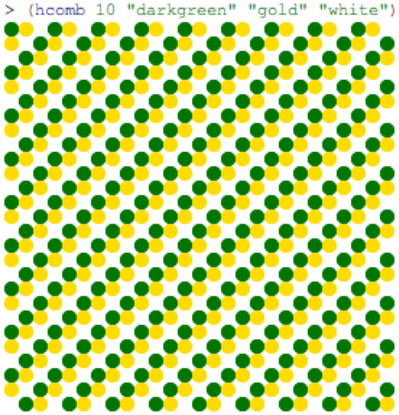
```
> (cboard 25 "darkgreen" "white")
```



```
> (hcomb 10 "navy" "orange" "black")
```



```
> (hcomb 10 "darkgreen" "gold" "white")
```



Question 2: Recursive functions of integers

See you next week!