Recap: Reduction

<table>
<thead>
<tr>
<th>Level</th>
<th>Examples</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>software libraries</td>
<td>graphics, animation, robotics</td>
<td>networking, security, mathematics</td>
</tr>
<tr>
<td>high-level language</td>
<td>Python</td>
<td>C, Java, C++, Logo, LISP, Fortran, ML</td>
</tr>
<tr>
<td>machine language</td>
<td>ML³</td>
<td>x86, CARDIAC, Z80</td>
</tr>
<tr>
<td>logic gates</td>
<td>equal, ifthenelse, add</td>
<td>memlookup, memwrite</td>
</tr>
<tr>
<td>basic logic gates</td>
<td>and, or, not</td>
<td>nor, nand, xor</td>
</tr>
<tr>
<td>physical bits</td>
<td>0,1 via high/low voltage</td>
<td>water pressure, kinetic energy</td>
</tr>
</tbody>
</table>
Does It End There?

- Of course, not.
- We can continue to build sophisticated programs out of simpler programs.
- The idea of **subroutines** (procedures, functions) makes this work.

Subroutines

- A **lot** of research in computer science is about designing and creating just the right set of subroutines, sometimes called **libraries**.
- You don’t have enough background yet to weigh in on these problems.
- But, there is an analogous set of problems where you are already an expert...
Sing-A-Long Programs

Gilligan's Island Theme

Just sit right back and you'll hear a tale, a tale of a fateful trip.
That started from this tropic port, aboard this tiny ship.
The mate was a mighty sailin' man, the skipper brave and sure.
Five passengers set sail that day, for a three hour tour, a three hour tour…
The weather started getting rough, the tiny ship was tossed.
If not for the courage of the fearless crew,
the Minnow would be lost; the Minnow would be lost.
The ship took ground on the shore of this uncharted desert isle,
with Gilligan, the Skipper too, the Millionaire, and his Wife,
the Movie Star, the Professor and Mary Ann, here on Gilligan's Isle.

Python Code

def GilligansIsland():
    print “Just sit right back and you'll hear a tale, a tale of a fateful trip.”
    print “That started from this tropic port, aboard this tiny ship.”
    print “The mate was a mighty sailin' man, the skipper brave and sure.”
    print “Five passengers set sail that day, for a three hour tour, a three hour tour.”
    print “The weather started getting rough, the tiny ship was tossed.”
    print “If not for the courage of the fearless crew,”
    print “    the Minnow would be lost; the Minnow would be lost.”
    print “The ship took ground on the shore of this uncharted desert isle,”
    print “    with Gilligan, the Skipper too, the Millionaire, and his Wife,”
    print “    the Movie Star, the Professor and Mary Ann, here on Gilligan's Isle.”

GilligansIsland()
Chorus Structure

Clementine

In a cavern, in a canyon,
Excavating for a mine,
Dwell a miner forty niner,
And his daughter Clementine.

Oh my darling, oh my darling,
Oh my darling, Clementine!
Thou art lost and gone forever
Dreadful sorry, Clementine.

Light she was and like a fairy,
And her shoes were number nine,
Herring boxes, without topses,
Sandals were for Clementine.

Drove she ducklings to the water
E\'vry morning just at nine,
Hit her foot against a splinter,
Fell into the foaming brine.

Ev\'ry morning just at nine,
Drove she ducklings to the water
E\'vry morning just at nine,
Hit her foot against a splinter,
Fell into the foaming brine.

Ruby lips above the water,
Blowing bubbles, soft and fine,
But, alas, I was no swimmer,
So I lost my Clementine.

Oh my darling, oh my darling,
Oh my darling, Clementine!
Thou art lost and gone forever
Dreadful sorry, Clementine.

How I missed her! How I missed her,
How I missed my Clementine,
But I kissed her little sister,
I forgot my Clementine.

Oh my darling, oh my darling,
Oh my darling, Clementine!
Thou art lost and gone forever
Dreadful sorry, Clementine.

Python Code

def Clementine():
    print "In a cavern, in a canyon,"
    print "Excavating for a mine,"
    print "Dwell a miner forty niner,"
    print "And his daughter Clementine."
    print "Oh my darling, oh my darling,"
    print "Oh my darling, Clementine!"
    print "Thou art lost and gone forever"
    print "Dreadful sorry, Clementine."
    print "Light she was and like a fairy,"
    print "And her shoes were number nine,"
    print "Herring boxes, without topses,"
    print "Sandals were for Clementine."
    print "Drove she ducklings to the water"
    print "$\text{Ev'}vry$ morning just at nine,"
    print "$\text{Hit her foot against a splinter,}$"
    print "$\text{Fell into the foaming brine.}$"

    print "How I missed her! How I missed her,"
    print "How I missed my Clementine,"
    print "But I kissed her little sister,"
    print "I forgot my Clementine."
    print "Oh my darling, oh my darling,"
    print "Oh my darling, Clementine!"
    print "Thou art lost and gone forever"
    print "Dreadful sorry, Clementine."

ClementineChorus():
    print "Dreadful sorry"
    print "Thou art lost and gone forever"
    print "Oh my darling, Clementine!"
    print "Thou art lost and gone forever"
    print "Dreadful sorry, Clementine."
Using A Subroutine

def ClementineChorus():
    print "Oh my darling, oh my darling,"
    print "Oh my darling, oh my darling, ... then the program resumes from where the call took place.
• Simplifies the program, easier to maintain, extend, fix.

Shared Structure

All My Loving
Lennon/McCartney

Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

I'll pretend I am kissing,
The lips I am missing,
And hope that my dreams will come true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

All my loving, I will send to you,
All my loving, darling, I'll be true,
All my loving, I will send to you.

What subroutines would you define?

def closeYourEyes():
    print "Close your eyes and I'll kiss you" 
    print "Tomorrow I'll miss you"
    print "Remember I'll always be true"
    print "And then while I'm away"
    print "I'll write home every day"
    print "And I'll send all my loving to you"
    print

def allMyLoving():
    closeYourEyes()
    print "I'll pretend that I'm kissing"
    print "The lips I am missing"
    print "And hope that my dreams will come true"
    print "And then while I'm away"
    print "I'll write home every day"
    print "And I'll send all my loving to you"
    print "All my loving, I will send to you"
    print "All my loving, darling, I'll be true" 
    print
    closeYourEyes()
    print "All my loving, I will send to you"
    print "All my loving, darling, I'll be true"
    print "All my loving, All my loving"
    print "Woo, all my loving, I will send to you"
Deeper Nesting?

All My Loving
Lennon/McCartney

Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.
I'll pretend I am missing,
The lips I am missing,
And hope that my dreams will come true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.
All my loving, I will send to you,
All my loving, darling, I'll be true,
All my loving, I will send to you.

This subroutine called its own subroutine.

What subroutines would you define?

def Away():
    print "And then while I'm away"
    print "I'll write home every day"
    print "And I'll send all my loving to you"
def closeYourEyes():
    print "Close your eyes and I'll kiss you"
    print "Tomorrow I'll miss you"
    print "Remember I'll always be true"

def allMyLoving():
    closeYourEyes()
    print "I'll pretend that I'm kissing"
    print "The lips I am missing"
    print "And hope that my dreams will come true"
    Away()
    print "All my loving, I will send to you"
    print "All my loving, darling I'll be true"
    print "All my loving, All my loving"
    print "Woo, all my loving, I will send to you"
**Python Code**

```python
def B():
    print "Ooh I need your love babe"
    print "Guess you know it's true"
    print "Hope you need my love babe"
    print "Just like I need you"

def R():
    print "Hold me, love me"
    print "Hold me, love me"
    print "Ain't got nothin' but love babe"
    print "Eight days a week"

def Y():
    print "Love you every day girl"
    print "Always on my mind"
    print "One thing I can say girl"
    print "Love you all the time"

def G():
    print "Eight days a week"
    print "I love you"
    print "Eight days a week"
    print "Is not enough to show I care"

def eightDaysAWeek():
    B(); R(); Y(); R(); G()
    B(); R(); G(); Y(); R()
    print "Eight days a week"
    print "Eight days a week"
```

**Building Structure**

> Old Macdonald had a farm, E-I-E-I-O
> And on his farm he had a cow, E-I-E-I-O
> With a "moo-moo" here and a "moo-moo" there
> Here a "moo" there a "moo"
> Everywhere a "moo-moo"

> Old Macdonald had a farm, E-I-E-I-O
> And on his farm he had a pig, E-I-E-I-O
> With a (snort) here and a (snort) there
> Here a (snort) there a (snort)
> Everywhere a (snort-snort)

> Old Macdonald had a farm, E-I-E-I-O
> And on his farm he had a horse, E-I-E-I-O
> With a "neigh, neigh" here and a "neigh, neigh" there
> Here a "neigh" there a "neigh"
> Everywhere a "neigh-neigh"

> Old Macdonald had a farm, E-I-E-I-O
> And on his farm he had a chick, E-I-E-I-O
> With a "cluck, cluck" here and a "cluck, cluck" there
> Here a "cluck" there a "cluck"
> Everywhere a "cluck-cluck"

> Old Macdonald had a farm, E-I-E-I-O
> And on his farm he had a farm, E-I-E-I-O
> With a "moo-moo" here and a "moo-moo" there
> Here a "moo" there a "moo"
> Everywhere a "moo-moo"
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a pig, E-I-E-I-O'
    snort()
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a horse, E-I-E-I-O'
    neigh()
    snort()
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    neigh()
    snort()

def moo():
    print 'With a "moo-moo" here and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'

def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)'
    print 'Everywhere a (snort-snort)'

def neigh():
    print 'With a "neigh, neigh" here and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'

def cluck():
    print 'With a "cluck, cluck" here and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"'
    print 'Everywhere a "cluck-cluck"'

But Notice...

- Each time we “snort”, we follow it with “moo”.
- Each time we “neigh”, we follow it with “snort”.
- Each time we “cluck”, we follow it with “neigh”.
- Let’s move the earlier noises into the later subroutines...
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a pig, E-I-E-I-O'
    snort()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a horse, E-I-E-I-O'
    neigh()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print

def moo():
    print 'With a "moo-moo" here and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'

def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)'
    print 'Everywhere a (snort-snort)'
    moo()

def neigh():
    print 'With a "neigh, neigh" here and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"
    print 'Everywhere a "neigh-neigh"'
    snort()

def cluck():
    print 'With a "cluck, cluck" here and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"
    print 'Everywhere a "cluck-cluck"
    neigh()
Subroutine Stack

```python
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, ...'
    print '  and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'
```

```python
def moo():
    print 'With a "moo-moo" here'
    print '  and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'
```

```python
def neigh():
    print 'With a "neigh, neigh" here'
    print '     and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'
```

```python
def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)'
    print 'Everywhere a (snort-snort)'
```

Subroutine Stack

```python
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print ''

    def moo():
        print 'With a "moo-moo" here'
        print '  and a "moo-moo" there'
        print 'Here a "moo" there a "moo"'
        print 'Everywhere a "moo-moo"'
    
    def neigh():
        print 'With a "neigh-neigh" here and a "neigh-neigh" there'
        print 'Here a "neigh" there a "neigh"'
        print 'Everywhere a "neigh-neigh"'
    
    def snort():
        print 'With a (snort) here and a (snort) there'
        print 'Here a (snort) there a (snort)'
        print 'Everywhere a (snort-snort)'
    moo()
```

Subroutine Stack

```python
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print ''
```

Subroutine Stack
Subroutines So Far

- Can be used to gather up repeated code in one place.
- Can be deeply nested to capture structure when necessary.
- Next, parameters greatly increase the power of the subroutine idea...

Near Miss...

**Magical Mystery Tour** (Beatles)
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is _____ to take you away, _____ to take you away.

Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is _____ to take you away, _____ to take you away.

Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is _____ to take you away, _____ to take you away.

- Verse structure is nearly repetitive. Can’t quite define a single subroutine that covers all three.
- If we could fill in the blank, we could reuse the same routine...
Naive Python Code

def mysteryVerseWaiting():
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print "waiting to take you away, "
    print "waiting to take you away."
    print

def mysteryVerseHoping():
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print "hoping to take you away, "
    print "hoping to take you away."
    print

def mysteryVerseComming():
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print "coming to take you away, "
    print "coming to take you away."
    print

def magicalMysteryTour():
    mysteryVerseWaiting()
    mysteryVerseHoping()
    mysteryVerseComing()
    print "The magical mystery tour is "
    print "dying to take you away, "
    print "dying to take you away."

Formal Parameters

def mysteryVerse(x):
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print " + x + " to take you away, "
    print " + x + " to take you away."
    print

- When mysteryVerse("waiting") is executed, the caller temporarily assigns the variable “x” the value “waiting”.
- The subroutine does its thing, using the value “waiting” whenever “x” appears. When finished, it returns to the caller, and “x” is forgotten.
- x is called the formal parameter, “waiting” is the actual parameter.
def OldMacdonald(intro):
    moo(True)
    snort(True)
    neigh(True)
    cluck(True)
def doIntro(animal, noise, intro):
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print 'And on his farm he had a ' + animal + ', E-I-E-I-O'
        makeNoise(noise)
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print ''
def makeNoise(noise):
    print 'With a "' + noise + '-' + noise + '  here and a '
    print ' "' + noise + '-' + noise + '  there'
    print 'Here a "' + noise + ' there a "+ noise + "'
    print 'Everywhere a "+ noise + "+ noise + "'
def moo(intro):
    doIntro("cow", "moo", intro)
def snort(intro):
    doIntro("pig", "(snort)", intro)
moo(False)
def neigh(intro):
    doIntro("horse", "neigh", intro)
snort(False)
def cluck(intro):
    doIntro("chick", "cluck", intro)
neigh(False)

• Writes out the “here-a-there-a” part with the given noise.

• If “intro” is False, just makes the given animal noise. If “intro” is True, also includes that animal’s introduction (“And on this farm...”).
The Verses

def moo(intro):
    doIntro("cow", "moo", intro)

def snort(intro):
    doIntro("pig", "(snort)", intro)
moo(False)

def neigh(intro):
    doIntro("horse", "neigh", intro)
    snort(False)

def cluck(intro):
    doIntro("chick", "cluck", intro)
    neigh(False)

• If “intro” is True, sings the cow verse, otherwise, just makes the cow noise.
• If “intro” is True, sings the pig verse (including the cow noises), otherwise, just makes the pig noise, and the cow noise.
• If “intro” is True, sings the horse verse (including the pig/cow noises), otherwise, just makes the horse/pig/cow noises.
• If “intro” is True, sings the chick verse (including all noises), otherwise, just makes all noises.

The Song

def OldMacdonald():
    moo(True)
    snort(True)
    neigh(True)
    cluck(True)

• Sings all four complete verses.

• Important thing to notice:
  • snort(True) means “set intro to True and begin executing the snort subroutine”.
  • While executing snort(True), it makes a call to moo(False)
  • moo(False) means “set intro to False and begin executing the moo subroutine.
  • No confusion---each subroutine has its own copy of “intro”!
def OldMacdonald():
    moo(True)
    snort(True)
    neigh(True)
    cluck(True)

def snort(intro = True):
    doIntro("pig", "(snort)", intro)
    moo(False)

def moo(intro = False):
    doIntro("cow", "moo", intro)

Local Variables

• Unless otherwise specified, variable names refer to their values within the subroutine itself only.
• Changes to these local variables do not impact any of the calling subroutines.
• “What happens in LV stays in LV.”
Recursion...

• Since each subroutine call has its own context (local variables, “red arrow”), a subroutine can call itself.
• This is called recursion.
• First, some song examples, then more general...

Recursive Algorithm

```python
def bottlesOfBeer(n):
    if n > 0:
        print str(n) + " bottles of beer on the wall."
        print str(n) + " bottles of beer."
        print "If one of those bottles should happen to fall."
        print str(n-1) + " bottles of beer on the wall."
        print bottlesOfBeer(n-1)
```

• bottlesOfBeer(99) prints a message, then calls
• bottlesOfBeer(98), which prints a message, then calls
• bottlesOfBeer(97), which prints a message, then calls...
• bottlesOfBeer(0), which does nothing.
def songNeverEnds():
    print "This is 'The Song That Doesn't End"
    print "Yes it goes on, and on my friend"
    print "Some people, started singing it"
    print "not knowing what it was"
    print "Now they'll continue singing it"
    print "forever just because..."
    songNeverEnds()

- From “Lambchop’s Play-along”
- Canceled.
Noun-Phrase Recursion

• A noun phrase can be a determiner followed by a noun, as in:
  • NP = “a fly-swatter”.

• A noun phrase can also be a determiner followed by a noun followed by “that” (or “who” if the noun is a person) followed by a verb, followed by a noun phrase:
  • NP = “a beetle that spied the fly-swatter”.

• A sentence can be a noun phrase followed by a transitive verb followed by two noun phrases.

Example

• The scale gave the frog a ride.
• The scale that weighed the snail gave the frog a ride.
• The scale that weighed the snail gave the frog that learned a song a ride.
• The scale that weighed the snail gave the frog that learned a song that stunned the nation a ride that lasted a minute.
• The scale that weighed the snail gave the frog that learned a song that stunned the nation a ride that lasted a minute that resembled an hour.

It’s not easy being green...
Time Expressions

- Normal:
  - 3 (3:00)
  - half past noon (12:30)
  - 10 of 6 (5:50)
  - quarter past 4 (4:15)
  - 2 30 (2:30)

- Extended:
  - 3 past quarter of 7 (6:48)
  - 5 of 5 of 5 of 5 (4:45)
  - half past 9 past quarter of 7 14 (7:38)
  - 2 past 2 of 2 past 2 of 1 1 1 (11:11)

Grammatical Definition

- **Time** can be an hour (1-12).
  - **Time** can be an hour and a minute (1-12 0-59).
  - **Time** can be “noon” (ie, 12).
  - **Time** can be “midnight” (ie, 12).

- **Time** can be an interval (1-59) followed by “of” followed by a **Time**.
  - **Time** can be an interval (1-59) followed by “past” followed by a **Time**.
Recursive Call

• To decode the expression, need to see if there’s a subexpression. If not, return the time.

• If so, decode the subexpression, then add or subtract.

```python
def decodeTimeExpr(TimeExpr):
    if len(TimeExpr) == 1:
        return [decodeHourExpr(TimeExpr[0]), 0]
    if len(TimeExpr) == 2:
        return [decodeHourExpr(TimeExpr[0]), TimeExpr[1]]
    if TimeExpr[1] == "of":
        return subtractTime(decodeIntervalExpr(TimeExpr[0]),
                            decodeTimeExpr(TimeExpr[2:]))
    if TimeExpr[1] == "past":
        return addTime(decodeIntervalExpr(TimeExpr[0]),
                        decodeTimeExpr(TimeExpr[2:]))
```

Example / Results

• decodeTimeExpr([3]) ⇒ [3,0]
• decodeTimeExpr(["half", "past", "noon"]) ⇒ [12,30]
• decodeTimeExpr([10, "of", 6]) ⇒ [5,50]
• decodeTimeExpr(["quarter", "past", 4]) ⇒ [4,15]
• decodeTimeExpr([2, 30]) ⇒ [2,30]
• decodeTimeExpr([3, "past", "quarter", "of", 7]) ⇒ [6,48]
• decodeTimeExpr([5, "of", 5, "of", 5, "of", 5]) ⇒ [4,45]
• decodeTimeExpr(["half", "past", 9, "past", "quarter", "of", 7, 14]) ⇒ [7,38]
• decodeTimeExpr([2, "past", 2, "of", 2, "past", 2, "of", 2]) ⇒ [2,0]
Next Time

- Now, we know what a computer is.
- Next, what is computation?
- We’ll look at “algorithms” and why some are better than others.
- Read: Hillis, Chapter 4, Sections ?