public class Intro
{
    public static void main(String[] args)
    {
        System.out.println("Hello world!");
    }
}
Where We're Headed

- 2 lectures this week
- 2 lectures next week (MLK holiday)
- Sections begin Jan 25/26
Topic 01: Intro to Programming

- What is a computer?
- What can a computer do?
- What is pseudocode?
- What is a programming language?
- What is programming?
What is a Computer?

PDP-1
(early '60s)
What is a Computer?

the first Macintosh (1984)

What is a Computer?

a smartphone

"iPhone 5s top" by Calerusnak at English Wikipedia. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:IPhone_5s_top.jpg#/media/File:IPhone_5s_top.jpg
What is a Computer?

a Google datacenter (thousands of computers)

http://www.madeinalabama.com/2015/06/google-build-600-million-alabama-data-center/
What is a Computer?

All of these computers are fundamentally the same.

- Memory
- CPU
- Input/Output
What is a Computer?

Inside a computer, everything is a number.

- Letters: 'a' = 97, 'A' = 65
- Colors: red = (255,0,0), blue = (0,0,255)
- Feelings: sad = 75%, angry = 10%
What is a Computer?

Sometimes, we store lots of numbers in a group.

- Text: Array of letters
- File: Array of numbers
- Picture: 2D array of colors
- Sound: Array of loudness values
What is a Computer?

Related numbers can be organized into objects.

A 'Person' object might have:

- A name (text)
- An ID # (number)
- A photo
- Relatives (links to other Person objects)
Topic 01: Intro to Programming

• What is a computer?
• **What can a computer do?**
• What is pseudocode?
• What is a programming language?
• What is programming?
What Can a Computer Do?

Computers are far simpler than you think.

They do:

- Simple Operations
- In Order (one at a time)
- Very Quickly
Simple Operations

Basically, all a computer can do is:

- Perform a simple math operation
- Compare two values and make a choice
- Execute another program
  - We call this “calling a subroutine”
Simple Operations

Perform a simple math operation

\[ x = y + z \ ; \]

- Read \( y \)
- Read \( z \)
- Add them
- Write into \( x \)

Common Ops:
Add
Subtract
Multiply
Divide
not much else!
Simple Operations

Make a choice

```c
if (x < y)
{
    a = 1;
}
```

- Compare $x$ and $y$
- If $x$ is less than $y$:
  - Set $a$ to 1

If the comparison fails, then the computer will do **nothing**!
Simple Operations

Call a subroutine

System.out.println(“Hello world!”);

• Call the println() subroutine
  – Pass it “Hello world!” as a parameter
Subroutines?

But Wait!

Is `println()` really a “simple operation”? 

- `println()` puts things on the screen
- `println()` moves lots of data around
Subroutines?

println() is not simple.

But it is **made up of** simple operations.

Somebody figured out how to print letters to a file...using only add, subtract, multiply, divide.
Subroutines?

We can pretend that `println()` is simple

It's as if we have 5 simple operations:

- add
- subtract
- multiply
- divide

It's not just `println()`
There are millions of subroutines like this!
What Can a Computer Do?

Computers are far simpler than you think.

They do:

• Simple Operations

• In Order (one at a time)

• Very Quickly
In Order

• A computer always does **one thing at a time**
• It follows its instructions **in order**

If it looks like it's doing lots of things, it's just because it does them so quickly!
In Order

• A computer always does one thing at a time
• It follows its instructions in order

Let's run the following program:

1) Take 5 steps forward
2) If there is an obstruction
   a) Stop!
3) Take 5 steps forward

These aren't simple operations. Maybe they are subroutines?
In Order

• A computer always does one thing at a time
• It follows its instructions in order

What if we changed the order?

1) Take 5 steps forward
2) Take 5 steps forward
3) If there is an obstruction
   a) Stop!

Changing the order changes the outcome!
In Order

In English, we often write out of order:

“Take 5 steps forward, then take 5 more steps. But stop if there is an obstruction.”

Computers will require that you be more precise.
What Can a Computer Do?

Computers are far simpler than you think.

They do:

• Simple Operations
• In Order (one at a time)
• Very Quickly
Very Quickly!

How is it possible to do such complex things, using only simple operations?

- Subroutines
  - Subroutines
    - Subroutines
      - Subroutines
        - Subroutines
          - Subroutines
            - Subroutines
Very Quickly!

Wouldn't that require lots and lots of simple operations?

Yes

But you wouldn't believe how fast computers are.
Very Quickly!

A Typical Computer:

• 4 GHz
  – 4 billion clock ticks per second
• 2 ALUs per core
  – 2 adds per clock tick
• 6 cores
  =
  48 billion adds per second
  48 adds per nanosecond

We never get this full performance, for many reasons.

Take CSc 252 to find out why!
Topic 01: Intro to Programming

- What is a computer?
- What can a computer do?
- **What is pseudocode?**
- What is a programming language?
- What is programming?
What is Pseudocode?

- English is easy to write and read, but imprecise
- Computer code is very precise, but sometimes hard to read

- Pseudocode is the middle-ground
  - Precise enough to communicate
  - Flexible enough to be easy to read
What is Pseudocode?

• Good pseudocode **MUST:**
  – Use simple steps
  – Give the steps in the proper order
  – Use indentation for if() statements and loops

• Good pseudocode **SHOULD:**
  – Use English descriptions of operations
  – Use symbols & code only if that is simpler
Pseudocode Example #1

You are going to walk to the store, but you don't want to get wet. Your program has subroutines for the following operations:

- Checking to see if it is raining
- Opening your umbrella
- Stepping outside

In groups, write pseudocode which will keep you dry.
My Solution:

if (it is raining)
  Open umbrella
Step outside

Let's compare this to some not-as-good versions.
Good Pseudocode:

if (it is raining)
  Open umbrella
Step outside

Bad Pseudocode (no clear steps):

Don't get wet
Pseudocode Example #1

**Good Pseudocode:**

if (it is raining)
    Open umbrella
Step outside

**Bad Pseudocode (out of order):**

Step outside
if (it is raining)
    Open umbrella
Pseudocode Example #1

Good Pseudocode:

if (it is raining)
    Open umbrella
Step outside

Bad Pseudocode (no indentation):

if (it is raining)
    Open umbrella
Step outside
Pseudocode Example #1

**Good Pseudocode:**

```plaintext
if (it is raining)
    Open umbrella
Step outside
```

**Bad Pseudocode (wrong indentation):**

```plaintext
if (it is raining)
    Open umbrella
Step outside
```
Pseudocode Example #1

**Good Pseudocode:**

if (it is raining)
    Open umbrella
Step outside

**Bad Pseudocode (forgot the if check):**

Open umbrella
Step outside
You are writing a program for a bank, which will transfer money from one account to another. Each has a variable, which contains the current account balance: \textit{amt1} and \textit{amt2}.

Write pseudocode which does:

- Transfers 100 dollars from \textit{amt1} to \textit{amt2}.
  - But report an error if there is an insufficient amount.
- Call the \texttt{sendWarning()} subroutine if \textit{amt1} falls below 50 dollars.

Work on this in your groups!
Pseudocode Example #2

**My Solution:**

if (amt1 < 100)
    Error: amt1 too low
else
    Decrease amt1 by 100
    Increase amt2 by 100
    if (amt1 < 50)
        sendWarning()
Pseudocode Example #2

My Solution:

if (amt1 < 100)
    Error: amt1 too low
else
    Decrease amt1 by 100
    Increase amt2 by 100
if (amt1 < 50)
    sendWarning()
Pseudocode Example #2

My Solution:

if (amt1 < 100)
    Error: amt1 too low
else
    Decrease amt1 by 100
    Increase amt2 by 100
    if (amt1 < 50)
        sendWarning()

This if() is “nested” inside the other if().
Pseudocode Example #2

**My Solution:**

if (amt1 < 100)

    Error: amt1 too low

else

    amt1 = amt1 - 100
    amt2 = amt2 + 100

if (amt1 < 50)

    sendWarning()
Pseudocode Example #2

**My Solution:**

if (amt1 < 100)

    *Error: amt1 too low*

else

    amt1 = amt1 - 100
    amt2 = amt2 + 100

if (amt1 < 50)

    sendWarning()

This line reports an error.

It doesn't detail how to report the error, but the programmer can figure that out later.

Note that **nothing else happens** if this error occurs.

Why? Because everything else is inside the **else** block.
Pseudocode Example #3

• Input:
  – 3 integers, called \( a, b, c \)

• Output:
  – The minimum value with its name
  – If there is a tie, then always prefer \( a \) over \( b \), and \( b \) over \( c \)

**Example:** 1,3,2
\[ a: 1 \]

**Example:** 0,-10,0
\[ b: -10 \]
Pseudocode Example #3

if (a is minimum)
    output a
else
    if (b is minimum)
        output b
    else
        output c
Pseudocode Example #3

```plaintext
if (a is minimum)
    output a
else
    if (b is minimum)
        output b
    else
        output c
```

This line checks to see if \( a \) is the minimum of the three values.

(In a moment, we'll think about how exactly to do that.)
Pseudocode Example #3

if (a is minimum)
   output a
else
   if (b is minimum)
      output b
   else
      output c

This code runs if a is the minimum.
Pseudocode Example #3

if (a is minimum)
    output a
else
    if (b is minimum)
        output b
    else
        output c

This code runs if \( a \) is not the minimum.

Maybe \( b \) is the minimum; or maybe it is \( c \).
If (a is minimum)
    output a
else
    if (b is minimum)
        output b
    else
        output c

If a was not the minimum, then check if b is.
Pseudocode Example #3

if (a is minimum)
  output a
else
  if (b is minimum)
    output b
  else
    output c

If neither a nor b was the minimum, then c was.

Note that we don't need a 3rd if() statement here. This automatically runs if everything else fails.
Pseudocode Example #3

if (a is minimum)
    output a
else
    if (b is minimum)
        output b
    else
        output c

How do we figure out if a was the minimum?
Pseudocode Example #3

if (a <= b && a <= c)
    output a
else
    if (b is minimum)
        output b
    else
        output c
Pseudocode Example #3

if (a <= b && a <= c)
    output a
else
    if (b <= a && b <= c)
        output b
    else
        output c

Doing the same check here works.
But it's possible to simplify this a bit.
Pseudocode Example #3

\[
\text{if (a \leq b \text{ and } a \leq c)} \\
\quad \text{output a} \\
\text{else} \\
\quad \text{if (b \leq c)} \\
\quad \quad \text{output b} \\
\quad \text{else} \\
\quad \quad \text{output c}
\]

Since \(a\) is \textbf{not} the minimum, we know that \(b\) or \(c\) (or both) are less than \(a\).

We only need to know which of them is smaller.
Pseudocode Example #3

```python
if (a <= b && a <= c)
    output a
else
    if (b <= c)
        output b
    else
        output c
```

Now, we need to decide what exactly to output.

Pseudocode is not standard; you can write whatever communicates well to your users.
if (a <= b && a <= c)
    print “a: “+a
else
    if (b <= c)
        print “b: “+b
    else
        print “c: “+c

This is the style I like.
Make sure that, whatever you do, it is easy to understand.
Pseudocode Example #4

- **Input:**
  - 3 integers, called *a*, *b*, *c*

- **Output:**
  - All three integers in order, with their names
  - If there is a tie, then always list them in name order: *a*, *b*, *c*

**Example:** 1, 3, 2
- a: 1
- c: 2
- b: 3

**Example:** 0, -10, 0
- b: -10
- a: 0
- c: 0
Pseudocode Example #4

if (a <= b && a <= c)
    do all printouts
else
    if (b <= c)
        do all printouts
    else
        do all printouts

This is the solution for the previous example … we've just adapted it a bit.

In this new program, instead of printing one value, we'll do something more complex.
Pseudocode Example #4

if (a <= b && a <= c)
   do all printouts
else ...

Unfortunately, we won't be able to fit all of the code onto one slide.

So from here on, we're just going to focus on how to do this one step.
Pseudocode Example #4

if (a <= b && a <= c)
    do all printouts
else ...

There are 2 possible cases when a is the minimum: a, b, c and a, c, b.

How do we choose which one to print?
if (a <= b && a <= c)
    if (b <= c)
        answer is a,b,c
    else
        answer is a,c,b
else ...

We need to compare b and c.
Pseudocode Example #4

if (a <= b && a <= c)
    if (b <= c)
        print "a: " + a
        print "b: " + b
        print "c: " + c
    else
        print "a: " + a
        print "c: " + c
else ...

Let's get a little more detailed about what we print out.
Pseudocode Example #4

```plaintext
if (a <= b && a <= c)
    print "a: " + a
    if (b <= c)
        print "b: " + b
        print "c: " + c
    else
        print "c: " + c
        print "b: " + b
else ...
```

Why not simplify things by getting rid of duplicate code?

---

This is my final version of (part) of the pseudocode.

Can you fill in the next two blocks? (They will be similar.)
Topic 01: Intro to Programming

- What is a computer?
- What can a computer do?
- What is pseudocode?
- **What is a programming language?**
- What is programming?
What is a Programming Language?

- A programming language expresses your **algorithm** in a clear, unambiguous form
  - No flexibility
  - 100% clarity about all the details

- It's like pseudocode, but with more rules
Programming Languages

- There are lots of programming languages
  - C
  - C++
  - Java
  - Python
  - Fortran
  - Lisp
  - Haskell

100s of others!
Programming Languages

• All languages are equivalent (can do the same things) but approach the problems differently

• Most famous family: C and its descendants
  C       C++       Java       C#       Objective-C
Machine Language

- All programming languages eventually are converted to **machine code** – which is the actual code which runs on the CPU
  - All in binary (ones and zeroes)
  - Very hard to read by hand!

- The **purpose of a programming language** is to make it easy to generate machine code.
A compiler is a program which turns a programming language into machine code.
Topic 01: Intro to Programming

• What is a computer?
• What can a computer do?
• What is pseudocode?
• What is a programming language?
• What is programming?
What is Programming?

Programming is:

• Start with a complex idea
• Break it into simpler steps
• Tell the computer in what order to do them
What is Programming?

Programming is:

- Start with a complex idea
- Break it into simpler steps
- Tell the computer in what order to do them

You've done this with trivial problems above.

How would you do it with a more complex program?
Example: PacMan

I wrote this version of PacMan before class in Fall 15.

It's not complete, but it has a lot of the pieces.
Example: PacMan

- What tasks must this program do?

I'm not looking for pseudocode (yet).

I just want a list.
Example: PacMan

- What tasks must this program do?
  - Respond to keyboard
  - Run ghost AI routines
  - Move PacMan and ghosts around
  - Notice if PacMan eats a dot
  - Notice if PacMan eats a power pill
  - Notice if PacMan eats a ghost
  - Notice if a ghost eats PacMan
  - Redraw screen
  (and more!)
Example: PacMan

• What is the structure of this program?
  – A loop?
  – Or a line?
Example: PacMan

● What is the structure of the program?
● This program, like most games, uses a loop. It does the same thing many times:
  – Check the keyboard
  – Do ghost AI
  – Check for events
  – Draw the board again
Example: PacMan

We can express this as pseudo-code:

- until (out of lives)
  - Check the keyboard
  - Do ghost AI
  - Check for events
  - Draw the board again
Example: PacMan

Here's the real Java code from PacMan:

```java
while (lives > 0) {
    drawScreen(false);
    checkForKeyboard();
    ghostNavigation();
    checkForPacmanEaten();
    checkForDotsEaten();
    moveEntities();
    StdDraw.show(10);
}
```
Example: PacMan

Here's the real Java code from PacMan:

```java
while (lives > 0) {
    drawScreen(false);
    checkForKeyboard();
    ghostNavigation();
    checkForPacmanEaten();
    checkForDotsEaten();
    moveEntities();
    StdDraw.show(10);
}
```

Java executes these operations in order, and one at a time.

But how does Java know what to do for each subroutine?
Example: PacMan

You must **write the implementation** for each subroutine.

- Let's consider the `checkForKeyboard()` method. What does it do?
Example: PacMan

checkForKeyboard():

• if (user pushes up)
  - set PacMan to move North
• if (user pushes left)
  - set PacMan to move West

... and so on ...
Example: PacMan

checkForKeyboard():
• if (user pushes up)
  – set PacMan to move North
• if (user pushes left)
  – set PacMan to move West

... and so on ...

This is an improvement, but still too complex.

We need to make it even simpler. How do we check to see what the user is pressing? And how do we get PacMan to move?
Example: PacMan

Here's the **real Java code** from PacMan:

```java
checkForKeyboard():
  if (StdDraw.isKeyPressed('W'))
  {
    changeDirection(move, 0, true);
  }
  if (StdDraw.isKeyPressed('A'))
  {
    changeDirection(move, 3, true);
  }
```
Example: PacMan

Here's the real Java code from PacMan:

```java
checkForKeyboard():
    if (StdDraw.isKeyPressed('W'))
    {
        changeDirection(move, 0, true);
    }
    if (StdDraw.isKeyPressed('A'))
    {
        changeDirection(move, 3, true);
    }
```

`isKeyPressed()` is a subroutine which asks if a certain key is currently being pressed.

It was written by somebody else, and I can just used it.
Example: PacMan

Here's the _real Java code_ from PacMan:

```java
checkForKeyboard():
    if (StdDraw.isKeyPressed('W'))
    {
        changeDirection(move, 0, true);
    }
    if (StdDraw.isKeyPressed('A'))
    {
        changeDirection(move, 3, true);
    }
```

`changeDirection()` is a subroutine that I wrote for PacMan. It sends PacMan in a new direction.
What Have We Learned?

Programming is:
- Start with a complex idea
- Break it into simpler steps
- Tell the computer in what order to do them

Often, we have to do this several times before our steps are simple enough.
Levels of Complexity

Remember this?

• Subroutines
  – Subroutines
    • Subroutines
      – Subroutines
      • Subroutines
      • Subroutines
      • Subroutines
      • Subroutines
Levels of Complexity

- Subroutines
  - Subroutines
    - Subroutines
      - Subroutines
        - Subroutines
          - Subroutines

Sometimes, we start at the top, and break the program into smaller parts.

This is known as top-down design.
Levels of Complexity

- Subroutines
  - Subroutines
    - Subroutines
      - Subroutines
        - Subroutines
          - Subroutines

Sometimes, we start at the bottom, and build progressively more complex things.

This is known as **bottom-up design**.

Both strategies work well.
Topic 01: Intro to Programming

- What is a computer?
- What can a computer do?
- What is pseudocode?
- What is a programming language?
- What is programming?

Summary