CIS 210 Review

What have we learned, anyway?
Schedule reminder

Monday (today): Review

Wednesday:
   Midterm re-do here

Friday: Review
   also optional assignment due

Thursday next (12/08):
   Final exam, 15:15-17:15

Course evaluations available in Duckweb
Computer “Science”

Not a conventional science.

Neither is mathematics. That isn’t bad.

In 10 years, all current computers will be obsolete.

In 10 years, or 20, or 30, very little of the computer science you learn will be obsolete ... because of universality.
Goals for CIS 210

Learn computer science concepts
Problem solving with computation

General programming skills

• includes designing programs to be understood and modified by humans
• includes testing, debugging

Expressing programs in Python and Java

• but the programming concepts apply to other languages
We design programs for humans

Brilliant, creative.  
But bad with details.

Stupid, but very fast
Programming was tedious

These cables are the program
Two big ideas ...

1. Instead of connecting cables, let’s store the program in memory!
   (stored program machine)

2. Let’s write a program that simulates a more convenient computer!
   (Universal Turing Machine)
01100010\textsubscript{2}

Means 62\textsubscript{16} (interpreted as an integer)
Or 98\textsubscript{10}
Or ‘b’ (interpreted as an ASCII character)
Or BOUND (interpreted as an x86 instruction)

Or ... it doesn’t “\textit{mean}” anything, but we can interpret it several ways, as data or as a program instruction.
Computers are Universal (all the same)

Smart phones can execute old video games by *emulating* the original video game computers.

Any (universal) computer can mimic any other computer.

Question for thought: Including meat computers?
Java is a “virtual computer”

A machine language program can mimic an imaginary computer that executes Java

- Same principle as a cell phone program emulating a video game console

Multiple levels possible

Java program emulating video console

JVM emulating Java computer

Machine language on cell phone processor
Two steps to run Java program

**source code**

```java
class Foo {
    public Foo() {
        ...
    }
    ...
}
```

**byte code**

`Foo.java` -> `javar` -> `Foo.class` -> `java`
The `javac` compiler translates from source code to byte code or machine code.
```c
int maxOfTwo( int x, int y) {
    int choice;
    choice = x;
    if (y > x) {
        choice = y;
    }
    return choice;
}
```

```
0: iload_1
1: istore_3
2: iload_2
3: iload_1
4: if_icmple 9
7: iload_2
8: istore_3
9: iload_3
10: ireturn
```

**Activation Stack**
- `x`: 17
- `y`: 34
- `choice`: 17

**Evaluation Stack**
- `this`
- `Top`

**Activation Stack** (arguments and variables)
**Evaluation Stack** (expression evaluation)
What’s a variable?

It names a location in memory

It has a type, which determines how the bits in memory are interpreted, and what operations we can apply to the value

- In “dynamic” languages like Python, values have types;
  in “static typed” languages like Java, variables do

It has a scope (the narrower the better)

Its name should be descriptive (enough) and follow naming conventions
What’s memory, really?

Memory cells are one big list, numbered from zero.
The computer access them by “address” (number).

Variable ‘x’ might mean cell #6.
x = 33 might mean:
put 100001 in cell #6

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>010110101110</td>
</tr>
<tr>
<td>1</td>
<td>010101101110</td>
</tr>
<tr>
<td>2</td>
<td>010110100000</td>
</tr>
<tr>
<td>3</td>
<td>110110101100</td>
</tr>
<tr>
<td>4</td>
<td>010110111110</td>
</tr>
<tr>
<td>5</td>
<td>011010101110</td>
</tr>
<tr>
<td>6</td>
<td>011110101110</td>
</tr>
<tr>
<td>7</td>
<td>010111101110</td>
</tr>
<tr>
<td>...</td>
<td>.........</td>
</tr>
</tbody>
</table>
Local variables live in activation records

```c
void bar(int x) {
    y: int = 13;
    if ( .. ) {
        bar (y-3);
    }
}

foo ( ... ) {
    m: ...
    ...
    bar(12);
}
```
How to think about scope ...

```java
int x = 5;
{
    int y = 7;
    ...
    y = x;
}
for (int y = 3; y < 7; ++y) {
    ...
}
```
How to think about scope ...

```c
int x = 5;
{
    int y = 7;
    ...
    y = x;
}
for (int y = 3; y < 7; ++y) {
    ...
}
```
How to think about scope ...

```java
int x = 5;
{
    int y = 7;
    ...
    y = x;
}
for (int y = 3; y < 7; ++y) {
    ...
}
```
How to think about scope ...

```java
int x = 5;
{
    int y = 7;
    ...
    y = x;
}
for (int y = 3; y < 7; ++y) {
    ...
}
```
Some reasons for scope

I can (safely) reuse variable names,
  (so my names can be shorter)

I know what I can forget

I can get more error messages from the compiler
  (yes, really that’s a good thing)
boolean values

Typically from comparisons:

```java
int i=7; int j = 8;
boolean b = ( i == j );  // false

boolean c = (i < j);

b = b && (i > j) || c;
```

Combine with “and” (&&), “or” (||), “not” (!)
True or False

boolean type represents logical values
just two values, true and false
comparisons create boolean values
i < j, “Ben” < “Jerry”
create expressions with &&, ||, !
like arithmetic with numbers (almost)
use in “if”, “if/else”, and loop statements
control what the program does
# Boolean Operators

<table>
<thead>
<tr>
<th>Java</th>
<th>logic, math</th>
<th>electronics</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and, $\land$, $\cdot$</td>
<td><img src="image" alt="AND" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not, $\neg$</td>
<td><img src="image" alt="NOT" /></td>
</tr>
</tbody>
</table>
if ( ... ) { ... } else { ... }

int limit = 1000;
int balance = 5000;
int cost = 1001;
if (cost > limit || cost > balance) {
    System.out.println("Sorry, out of money.");
} else {
    balance = balance - cost;
}
A couple of things I see ...

```java
boolean p;
...
if (p == true) {
    ...
}

if (x > y) {
    return true;
}
```
Using boolean values directly

```java
boolean p;
...
if (p == true) {
    ...
}
if (x > y) {
    return true;
}
```

```java
boolean p;
...
if (p) {
    ...
}
return x > y;
```
Computers are stupid, but fast
We write loops to take advantage of their speed

LOOPS
for (int i=1; 
i <= nStars; 
++i)
{
    System.out.print('*');
}

Some Typical Loop Patterns

Do to each ...
  • Example: Print each integer from 1 to n

Accumulate ...
  • Example: Sum a set of integers

Select
  • Example: Print longest word in dictionary

Filter
  • Example: Print dictionary words with length > 7
While loops: Two parts

Basis (base case)
One or more small, simple cases, which can be done in one step without looping.

Progress
A case that requires a simple step, and then solution of a smaller problem.
Inductive *while* loop structure

While ( work to be done ) {
  if (*simplest case*) {
    *just do it*;
  } else {
    *do one step*;
    *prepare for next step*;
  }
}
But is it correct?

Argument for correctness follows case breakdown

**Basis:** Should handle all problems up to some size, e.g., “words of up to 1 character”

**Progress:** Should handle all other problems, and the “prepare for next step” part should produce a smaller problem (e.g., a shorter substring between left and right finger)
METHODS (FUNCTIONS, SUBROUTINES, ETC)
Chunking

Very limited working capacity

Nearly unlimited complexity
static int diff (int a, int b) {
    return a - b;
}

... then in main ...
int a = 5; int b = 7;

int c = diff(b, a);

The called method gets its own copies of the inputs, by position (not by name).
"Pass by value"

```java
static void foo( int a, int b, double c, String d ) {
    ...
}

foo( x, y, 37.489, "e tu, Brute?" );
```

The "actual arguments" x, y, 37.489, "et tu, Brute" are copied into the "formal arguments" of the method. The copies become distinct, local variables.
What makes a good method?

Simplifies the code that calls it
Isolates a design decision (easier to change)
Used more than once
Can be tested separately

...  

A good method may have only some of these properties. Few have all.
Javadoc as method contract

/**
 * Serves a specified amount of ice cream.
 * @param scoops number of units to serve
 * @param scoopSize grams per scoop
 */
static void serve(int scoops, double scoopSize) {
    ... body that does what the javadoc promises ...

Goals for CIS 210

Learn computer science concepts
Problem solving with computation

General programming skills
  • includes designing programs to be understood and modified by humans
  • includes testing, debugging

Expressing programs in Python and Java
  • but the programming concepts apply to other languages