Announcements

- First lab is tomorrow (Prelab is due before 10am)
- Office hours today 1:30 - 3:30
Examples

```python
1  print "Hello"
2  print
3  print "Hello," first
4  print first, last  # automatic space
5  print total
6  print str(total) + "."  # no space
7  print "Wait...",  # space; no newline
8  print "Done"
```

```cpp
1  cout << "Hello" << endl;
2  cout << endl;
3  cout << "Hello, " << first << endl;
4  cout << first << " " << last << endl;
5  cout << total << endl;
6  cout << total << "." << endl;
7  cout << "Wait... ";  // no newline
8  cout << "Done" << endl;
```

Figure 7: Demonstration of console output in Python and C++. We assume that variables first and last have previously been defined as strings, and that total is an integer.
Formatting output

Unfortunately, %d output is not really available (\# of digits)

(Inherited from C, so these but can't be used with C++ objects like strings.)

Python

```python
print '{s: ranked %d of %d teams}' % (team, rank, total)
```

C++

```cpp
cout << team << ": ranked " << rank << " of " << total << " teams" << endl;
```
Setting precision is harder:

```c++
print 'pi is %0.3f', pi;
output?
```

```
pi is 3.141
```

In C++:

```c++
cout << "pi is " << fixed << setprecision(3) << pi << endl;
```

**Note:** Precision stays set to 3.
Cin: Other data types (not strings)

Python:

```python
number = int(input('Enter a number from 1 to 10: '))
```

C++:
```
int number;
cout << "Enter a number from 1 to 10: ";
cin >> number;
```

**Note:** `cin` looks for white space. `cin << a << b;` is the same as `cout << 10; cin >> a; cout << 20; cin >> b;`.

```cpp
10 20
```

10
20
Input: Strings

Python: raw_input

```
person = raw_input('What is your name?')
```

C++: getline

```
string person;
cout << "What is your name? ";
getline(cin, person);
```

Note (for getline):
- inputs a string
- stores up to the newline, but strips the newline off
Some other differences with cin:

Chaining multiple inputs

```cpp
int a, b;
cout << "Enter two integers: ";
cin >> a >> b;
cout << "Their sum is " << a + b << "." << endl;
```

Note: - different types are allowed
      (but must match the variable)
      - separated by any whitespace!
A word of caution:

Ex:

```cpp
string person;
cout << "What is your name? ";
cin >> person;
```

I type "Erin Wolf Chambers /n".

What happens?

```cpp
person = "Erin"
```

use getline!
Another caution:

```cpp
int age;
string food;
cout << "How old are you? ";
cin >> age;
cout << "What would you like to eat? ";
getline(cin, food);
```

```
30 40 50 60 2
```

```
\sqrt{30} \, \text{pepperoni pizza}
age = 30
food = "pep"
```
File Streams: Input

If file name is known:

    ifstream mydata("scores.txt");  // declares and opens an input file

If file name is unknown:

    ifstream mydata;
    string filename;
    cout << "What file? ";
    cin >> filename;
    mydata.open(filename.c_str());

Converts to a C-type string (historical legacy)
Output:

By default, opening ofstream overwrites an existing file!

(just like "w" option in Python)

To append:

```cpp
ofstream datastream("scores.txt", ios::app);
```

normally "a" in Python get deleted
fstream

There is also an "fstream" object which allows both input & output.

Much more confusing.

(Whenever possible, much safer to keep input & output separate.)
String Streams
Casting from numbers to strings is not straightforward.

```cpp
int age(40);
string displayedAge;
stringstream ss;
ss << age; // Insert the integer representation into the stream
ss >> displayedAge; // extract the resulting string from the stream
```
Classes

What is a class?

- Way to store information in your own objects

Ex: Credit Card
- related collection of data
- to make life easy
- limit (define) functionality
- access control
Classes

Creating an instance of a class

```cpp
string s;
string greeting("Hello");
```

**NEVER:**

```cpp
string s();
```

Why? Create a function called `s` that returns a string

**NEVER:**

```cpp
string("Hello") greeting;
```

Why? Compile error
Defining a class: Remember the Point class?

class Point {
private:
    double x;
    double y;

public:
    Point() : x(0), y(0) {}    // constructor
    double getX() const {       // accessor
        return x;
    }
    void setX(double val) {      // mutator
        x = val;
    }
    double getY() const {       // accessor
        return y;
    }
    void setY(double val) {      // mutator
        y = val;
    }
};

// explicit declaration of data members

// instance of the class
Point myPoint;

// access the Point.x member
cout << myPoint.x << endl;

// ERROR: myPoint.getX()