Classes
March 21, Ch 10.1 - 10.3

Okay, any ideas on how to make women more interested in us?

Make more exceptions?
Redefine our methods?
Stop treating them like objects?

C++
Python
C
Java
ANSI C
Highlights

- public/private
- constructor

class myName
{
    public:
        myName();
        myName(int x);
        char takeThis();
    private:
        char itsASecretToEveryone;
};
struct/class vs array

Arrays group together similar data types (any amount you want)

Classes and structs group together dissimilar types that are logically similar
A class is functionally the same as a struct (creates a new data type)

However, the notation is slightly different (contains functions)
public vs private

class date
{
    private:
        int day;
        int month;
        int year;
    public:
        void print();
        void setDate(int day, int month, int year);
};
public vs private

The **public** keyword allows anyone anywhere to access the variable/method

The **private** keyword only allows access by/in the class where the variable/method is defined (i.e. only variables of this type can access this within itself)
public vs private

All variables should be private

While this means you need methods to set variables, users do not need to know how the class works.

This allows an easier interface for the user (also easier to modify/update code)

(See: datePrivate.cpp)
public vs private
public vs private

Creating interfaces with public allows users to not worry about the private implementation.

So... more work for you (programmer) less work for everyone else.
The date class has two functions: `setDate()` and `print()`.

As we need to run `setDate()` on a variable before it is useful anyways.

In fact, such a thing exists and is called a constructor (run every time you create a variable).
Constructors

The class name and the constructor must be identical
(constructors also have no return type)

```cpp
class date
{
private:
    int day;
    int month;
    int year;
public:
    date(int day, int month, int year);
    // ^^ constructor has same name as class
    void print();
};
```

(See: dateConstructor.cpp)
Constructors

If you don't put a constructor, C++ will make a default constructor for you (no arguments)

```cpp
date();  // default constructor

date(int day, int month, int year);
```

To use the default constructor say this:

```cpp
date never;
```

... not this:

```cpp
date notWhatYouWant();
// ^ function declaration
```
Constructors

If you declared constructors you must use one of those. Only if you declare no constructors, does C++ make one for you (the default).

Note: our dateConstructor.cpp has no way to change the value of the date after it is created (thus gives control over how to use class).
#include

Just as writing very long main() functions can start to get confusing...

... writing very long .cpp files can also get confusing

Classes are a good way to split up code among different files
You can `#include` your class back in at the top or link to it at compile time

You have to be careful as `#include` basically copies/pastes text for you

Will not compile if class declared twice (used in two different classes you `#include`)
To get around this, you can use compiler commands in your file:

“if not defined”

“define”

This ensures you only have declarations once

(See: dateClass.hpp, dateClass.cpp, runDate.cpp)