CSC 111: Intro to Computer Science through Programming

Spring 2017
Prof. Sara Mathieson
Encouraged: Piazza for final review (no more final project help)

**Final project** extended til **Thursday at 5pm**

**TA review session** *tonight*: 7:30-9:30pm

Practice final during lab today/tomorrow

Self-scheduled **final exam** (similar style to the midterm)
Practice Final

- During each lab section (you don’t have to go to your assigned section)
- Just like the final: no technology, notes, etc (just 2 cheat sheets)
- Feel free to write on the backs, but no extra pages
- TAs will provide feedback
- I will add 3 points to your final exam score if you take the practice final during lab, give an honest effort, put your name on it, turn it in, and PICK IT UP after feedback has been given
Outline: 5/3

- Homework demos
- Finish 99 number activity
- Review **recursion**
- Review **classes**
Homework examples
Homework 7 – Rebecca
Homework 7 – Cai
Homework 7 – Maddie
Finish 99 number activity
Step 1: make the dictionary

- Find your random partner and introduce yourselves
- In main, write some code that will ask the user for their 99 number and their name (two questions)
- Use the first two and last two digits for speed and privacy
- Add the 99 number (key) and name (value) to a dictionary that will keep track of individuals using their 99 numbers

```
>>> Enter your 99 number: 9995
>>> Enter your name: Sara Mathieson
>>> {9995: 'Sara Mathieson'}
```
Step 2: use a while loop

- Create a while loop that will keep asking for more 99 numbers and student names (use you and your partner’s info)
- Create a way to stop the while loop (i.e. user enters -1 for their number or “stop” for their name)

```python
>>> Enter your 99 number: 9995
Enter your name: Sara Mathieson
Enter your 99 number: 9921
Enter your name: Alan Turing
Enter your 99 number: -1
{9921: 'Alan Turing', 9995: 'Sara Mathieson'}
```
Step 3: write dictionary data to a file

+ After the while loop is over, write each number and name to a file using a loop over the keys of the dictionary

```
9921  Alan Turing  
9995  Sara Mathieson
```
Informal quiz: discuss with a partner

1) What built-in types have we studied this semester?

2) Which of those types are mutable?

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"]), how many times will `binary_search(..)` be called?

4) See the constructor below. How many instance variables? What is going on?

```python
class Course:
    def __init__(self, title):
        tokens1 = title.split()
        self.dept = tokens1[0]
        self.number = int(tokens1[1][:-1])

        tokens2 = title.split(": ")
        self.name = tokens2[1]

        self.student_lst = []
        self.time = None

cs = Course("CSC 111: Introduction to Computer Science")
```
Informal quiz: discuss with a partner

1) What built-in types have we studied this semester?

int, str, list, bool, float, dict, tuple, “file”
Informal quiz: discuss with a partner

1) What built-in types have we studied this semester?

   int, str, list, bool, float, dict, tuple, “file”

2) Which of those types are mutable?

   list, dict, “file” (when using write)
Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w")],` how many times will `binary_search(..)` be called?

`binary_search("e", ["a", "c", "g", "i", "s", "t", "w")]`
Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`, how many times will `binary_search(..)` be called?

```
binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])

binary_search("e", ["a", "c", "g"])
```
Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"]), how many times will `binary_search(..)` be called?

```python
binary_search(“e”, [“a”, “c”, “g”, “i”, “s”, “t”, “w”])

binary_search(“e”, [“a”, “c”, “g”])

binary_search(“e”, [“c”, “g”])

binary_search(“e”, [“c”, “g”])
```
Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`, how many times will `binary_search(..)` be called?

```
binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])  # 1 call

binary_search("e", ["a", "c", "g"])                # 1 call

binary_search("e", ["c", "g"])                   # 1 call

binary_search("e", ["c"]                          # 1 call
```
Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`, how many times will `binary_search(..)` be called?

```
binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])  # 1 time
```

```
binary_search("e", ["a", "c", "g"])  # 2 times
```

```
binary_search("e", ["c", "g"])  # 2 times
```

```
binary_search("e", ["c"])  # 3 times
```

```
binary_search("e", ["c"])  # 4 times
```

4 times
Course class

4) See the constructor below. How many instance variables? What is going on?

class Course:
    def __init__(self, title):
        tokens1 = title.split()
        self.dept = tokens1[0]
        self.number = int(tokens1[1][:-1])

        tokens2 = title.split(":")
        self.name = tokens2[1]

        self.student_lst = []
        self.time = None

    cs = Course("CSC 111: Introduction to Computer Science")
Recursion
Recursion

- A recursive function must call itself
- Usually it will call itself on a simpler/smaller/different version of the problem
- Must have a way of stopping the recursion (base case)
- If we return in the base case, we must return in the recursive call
- Going “down” to the base case, passing the answer back “up” through the recursive calls
Fibonacci Function Stack

fib(5)

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
Fibonacci Function Stack

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

```
A
fib(3)
A
fib(4)
A
fib(5)
```
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```
Fibonacci Function Stack

fib(5)
fib(4)
fib(3)
fib(2)
fib(1)

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

Line A
Line B
Line C

Function Stack

fib(1)
fib(2)
fib(3)
fib(4)
fib(5)
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack:
- Line A: fib(5)
- Line B: fib(4)
- Line C: fib(3)
- Line A: fib(2)
- Line C: fib(2)

Current line: out1=1
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

fib(5)

fib(4)

fib(3)

fib(2)

1 1

return 2

Function Stack

fib(2)
A
A
A
fib(3)
fib(4)
fib(5)
**Fibonacci Function Stack**

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

<table>
<thead>
<tr>
<th>Line A</th>
<th>Line B</th>
<th>Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td>out1=2</td>
<td>fib(3)</td>
<td>fib(4)</td>
</tr>
<tr>
<td></td>
<td>fib(5)</td>
<td></td>
</tr>
</tbody>
</table>

```
out1=2
```
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

```
return 1
out1=2
```

|   |   |   
|---|---|---|
| A | B |   |
|   |   | fib(5) |
|   | A | fib(4) |
|   | B | fib(3) |
|   |   | fib(1) |
| 2 |   |   |
| 1 | 1 |   |
Fibonacci Function Stack

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

Fibonacci Function Stack

out1=2
out2=1

fib(5)
fib(4)
fib(3)

2 1
1 1
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

<table>
<thead>
<tr>
<th>Line A</th>
<th>Line B</th>
<th>Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib(5)</td>
<td>fib(4)</td>
<td>fib(3)</td>
</tr>
</tbody>
</table>

return 3
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack:
- **A**: `fib(5)`
- **B**: `fib(4)`
- **C**: `fib(3)`

Current line:
- `out1 = 3`
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

```
<table>
<thead>
<tr>
<th>Line A</th>
<th>Line B</th>
<th>Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib(2)</td>
<td>fib(4)</td>
<td>fib(5)</td>
</tr>
<tr>
<td>out1=3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Current line:

```
A
```

Function Stack:

```
fib(5)
fib(4)
fib(2)
3
2
1
1
1
```
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

<table>
<thead>
<tr>
<th>Function Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

```
fib(5)
fib(4)
fib(2)
fib(1)
```

```
3
2
1
1
```

```
out1=3
```

```
return 1
```

Line A
Line B
Line C
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

fib(5)

Fibonacci Function Stack

fib(4)

3

fib(2)

2 1 1

1 1

Line A
Line B
Line C

Function Stack

B
fib(2)
out1=1

B
fib(4)
out1=3

A
fib(5)
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack:

<table>
<thead>
<tr>
<th></th>
<th>fib(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>fib(4)</td>
</tr>
<tr>
<td></td>
<td>fib(2)</td>
</tr>
<tr>
<td></td>
<td>fib(0)</td>
</tr>
<tr>
<td></td>
<td>out1=3</td>
</tr>
<tr>
<td></td>
<td>return 1</td>
</tr>
</tbody>
</table>

|   | fib(2) |
| B | fib(0) |
|   | out1=1 |
|   | return 1 |

|   | fib(4) |
| B | fib(5) |
|   | out1=3 |
|   | return 1 |
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

- **A**
  - Line A
    - `fib(5)`

- **B**
  - Line B
    - `fib(4)`
    - `out1=3`

- **C**
  - Line C
    - `fib(2)`
    - `out1=1`
    - `out2=1`
    - `out1=3`
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

```
<table>
<thead>
<tr>
<th>Line A</th>
<th>Line B</th>
<th>Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib(5)</td>
<td>fib(4)</td>
<td>fib(2)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>out1=3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>return</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

Function Stack

A: fib(5)
B: fib(4)
C: fib(2)
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack:
- Line A: `fib(5)`
- Line B: `fib(4)`
- Line C: `fib(4)
  out1=3
  out2=2`
fib(5)

fib(4)

3

2

2

1

1

1

1

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

A

current line

Function Stack

return 5
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A
Line B
Line C

B  fib(5)  out1 = 5

current
line

Function Stack
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

out1 = fib(5)
Fibonacci Function Stack

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

out1=5
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack:

- fib(1)
- fib(2)
- fib(3)
- fib(5)

Out1 = 5
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

<table>
<thead>
<tr>
<th>Stack Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line A</td>
</tr>
<tr>
<td>Line B</td>
</tr>
<tr>
<td>Line C</td>
</tr>
</tbody>
</table>

current line

<table>
<thead>
<tr>
<th>Stack Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
</tbody>
</table>

fib(5)
fib(3)
fib(2)
out1=5
out1=1
out1=1

1 1 1
1 1
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

```
Function Stack

<table>
<thead>
<tr>
<th></th>
<th>fib(0)</th>
<th>fib(2)</th>
<th>fib(3)</th>
<th>fib(5)</th>
<th>out1=1</th>
<th>return 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current line

- Line A
- Line B
- Line C
Fibonacci Function Stack

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

fib(5)

5

fib(3)

3 2

fib(2)

2 1

1 1

out1=5

return 2
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

```
B
<table>
<thead>
<tr>
<th>fib(5)</th>
</tr>
</thead>
</table>
```

```
B
<table>
<thead>
<tr>
<th>fib(3)</th>
</tr>
</thead>
</table>
```

```
out1=5
out1=2
```

Current line

```
out1=5
```

```
out1=2
```

```
A
```

```
B
```

```
C
```

```
A
```

```
B
```

```
C
```

```
A
```

```
B
```

```
C
```
Fibonacci Function Stack

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

fib(5)

5

3 2

2 1

1 1

fib(3)

2

fib(1)

2

fib(1)

out1=5

out1=2

return 1

Line A

Line B

Line C

Function Stack
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

<table>
<thead>
<tr>
<th>Line A</th>
<th>Line B</th>
<th>Line C</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib(5)</td>
<td>fib(3)</td>
<td>fib(3)</td>
</tr>
<tr>
<td>out1=5</td>
<td>out1=2</td>
<td>out1=5</td>
</tr>
<tr>
<td></td>
<td>out2=1</td>
<td></td>
</tr>
</tbody>
</table>
Fibonacci Function Stack

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Function Stack

```
B
fib(5)
out1=5
```

Current Line

```
return 3
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

fib(5)

out1=5
out2=3

Fibonacci Function Stack

C
current line
Function Stack
Fibonacci Function Stack

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A
- `if n == 0 or n == 1:`
  - return 1

Line B
- `out1 = fib(n-1)`
- `out2 = fib(n-2)`

Line C
- `return out1 + out2`
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2

Function Stack

empty!
Fibonacci Tree with Function Calls

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```
Lab 11 (sort and search)

def binary_search(item, lst):
    #print(lst)

    # base case
    if len(lst) == 1:
        return lst[0]

    index = int(len(lst)/2)
    middle = lst[index]

    # recursion
    if item < middle:
        return binary_search(item, lst[:index])
    else:
        return binary_search(item, lst[index:])
Classes
Goal of classes: encapsulate data (instance variables) and operations on data (methods) in one structure we can represent with single variable names.
Classes

- Goal of classes: encapsulate data (instance variables) and operations on data (methods) in one structure we can represent with single variable names

- list, str, dict are also [special] classes, and instances of these classes have methods associated with them
Classes

- Goal of classes: encapsulate data (instance variables) and operations on data (methods) in one structure we can represent with single variable names

- `list, str, dict` are also [special] classes, and instances of these classes have methods associated with them

- When we say `x = [3,5,6]` and then say `print(x)`, calling the `list __str__` method internally
Classes

+ Goal of classes: encapsulate data (instance variables) and operations on data (methods) in one structure we can represent with single variable names

+ list, str, dict are also [special] classes, and instances of these classes have methods associated with them

+ When we say \( x = [3, 5, 6] \) and then say \texttt{print(x)}, calling the \texttt{list.__str__} method internally

+ Fish, Car, etc are no different or less special than Circle, Point
class MyClass:
    def __init__(self, param1, param2, param3):
        self.
        self.

    def method1(self, param):
        self.

    def method2(self, param1, param2, param3):
        self.

    def function1(self, param1, param2, param3):
        
    def function2(self, param1, param2):
        
    def main():
        

Template for classes
Template for classes

class MyClass:
    def __init__(self, , ,):
        self.
        self.
    def method1(self, ):
        self.
    def method2(self, , ,):
        self.
    def function1(self, , ,):
    def function2(self, , ,):
    def main():
class MyClass:

    def __init__(self, arg1, arg2, arg3):
        self.arg1 = arg2
        self.arg3 = arg1

    def method1(self, arg1):
        self.arg1 = arg2

    def method2(self, arg1, arg2, arg3):
        self.arg1 = arg2

    def function1(self, arg1, arg2, arg3):
        pass

    def function2(self, arg1, arg2):
        pass

    def main(self):
        pass
Template for classes

class MyClass:

    def __init__(self, arg1, arg2):
        self.arg1 = arg1
        self.arg2 = arg2

    def method1(self, arg3):
        self.arg3 = arg3

    def method2(self, arg4, arg5):
        self.arg4 = arg4
        self.arg5 = arg5

    def function1(self, arg6, arg7, arg8):

    def function2(self, arg9, arg10):

    def main():

class MyClass:

def __init__(self, x, y):
    self.x = x
    self.y = y

def method1(self, arg):
    self.x = arg

def method2(self, arg1, arg2):
    self.x = arg1
    self.y = arg2

def function1(x, y, z):

def function2(x, y):

def main():
# use CamelCase for class names (no underscores)
class Projectile:

    def __init__(self, height, velocity, angle):
        # two instance variables to keep track of position
        self.x = 0
        self.y = height

        # two instance variables to keep track of velocity
        theta = angle*math.pi/180 # OR math.radians(angle)
        self.xvel = velocity * math.cos(theta)
        self.yvel = velocity * math.sin(theta)

    def update(self, dt):
        """Update the position and velocity based on:
        dt, a small amount of time in seconds.""

        # update the (x,y) position
        self.x += (self.xvel * dt)
        self.y += (self.yvel * dt)

        # update y-velocity (don't need to update x-velocity)
        self.yvel -= (9.8 * dt) # gravity: 9.8 m/(s^2)

    def getX(self):
        """Return the x position of the projectile.""
        return self.x

    def getY(self):
        """Return the y position of the projectile.""
        return self.y

Methods:
__init__(..)
update(..)
getX()
getY()

Constructor:
Special method

update(dt) is kind of like move(..) for graphics objects

Instance variables:
x
y
xvel
yvel

Getters
def main():

    # either try out values or ask the user
    height = eval(input("Enter height in meters: "))  # example: 1.6
    velocity = eval(input("Enter velocity in m/s: "))  # example: 5
    angle = eval(input("Enter angle in degrees: "))  # example: 40
    dt = 0.01  # seconds

    # create our projectile using the constructor
    shotput = Projectile(height, velocity, angle)

    # update the position while the projectile is above ground
    while shotput.getY() >= 0:
        shotput.update(dt)
        print(shotput.getX(), shotput.getY())  # print current position
Lab 10 (with __str__)
List Comprehensions
List comprehensions

```python
>>> # list comprehensions (ways of operating on a list without a for loop)
>>> lst1 = ["1","2","3","4"]

>>> # how to turn all elements to ints?
>>> lst2 = []
>>> for s in lst1:
    lst2.append(int(s))

>>> lst2
[1, 2, 3, 4]

>>> # OR
>>> lst3 = [int(s) for s in lst1]
>>> lst3
[1, 2, 3, 4]

>>> lst4 = [x+10 for x in lst3]
>>> lst4
[11, 12, 13, 14]

>>> [fish.move() for fish in fish_lst]
```
Feedback
We are interested in thoughts about...

- Capping enrollments for 111
- What do you think about random partners during lab?
- The textbook: make optional in the future?
- Use of class time
Final Thoughts on CS