Objectives

- Review
- Lab 2
  - Programming practice

Feedback on Lab 1

- Overall good
- Notes
  - Saved output from each program
    - With user input, try several different good test cases
  - Want *good* output
    - think about what the user wants to see
  - High-level comments
    - Describes what the program does
      - Helps for quick overview when reviewing
  - Electronic submission
    - In directory – looked good!
  - Fix problems in web pages today so that you can build on them today

“Good” Output

- Depends on context
- Not necessarily showing how computation was performed

50 vs When i = 7 and j = 2, i^2+3*j-5 = 50

```
Rickey Henderson's Stealing %: 80.75818495117748
Lou Brock's Stealing %: 75.34136546184739
Henderson was 5.416819489330095 % more successful at stealing than Brock.
```

(we can reduce the number of decimal places soon)

Review: Linux Commands

- What is the command to…
  - Determine which directory you're in?
  - View the contents of a directory?
  - Create a directory?
  - Copy a file?
  - Delete a file?
- How do you refer to … your home directory? The current directory? The parent directory?

Linux Command: mv

- Used to *move* or *rename* a file
- `mv <sourcefile> <destination>`
- Example usage:
  - Renames `file.py` to `newfilename.py`

```
mv file.py newfile.py
```

  - Moves `~/cs111/file.py` to *current* directory with a new name

```
mv ~/cs111/file.py newfilename.py
```

  - If `<destination>` is a *directory*, keeps the original source file's name

```
mv ~/cs111/file.py ~/cs111/lab1/
```

    - File `file.py` will now be in `cs111/lab1` directory instead of `cs111/`

Linux Command: `rm`

- Used to *delete* or ***rem****ove* a file
- `rm <filename>`
- Example usage:
  - Deletes `file.py` in the current directory

```
rm file.py
```

  - Deletes `~/cs111/lab1/file.py`

```
rm ~/cs111/lab1/file.py
```

Review

- What program/application do we use to develop programs?
  - What is the command to execute the application?
- What are the expectations for complete programs/submissions in this class?
- What is our *process* for developing programs?
  - In general and for lab (e.g., what do you need to submit for your programs?)

IDLE Review

- Run using `idle &`

You can install Python/IDLE on your own computer to practice between labs.

Development Process for Lab

- Develop in **IDLE**

1. Create a new file
2. Develop the program
3. Close the shell
4. Run the program again
5. Save output from program

Submission Expectations

- Code should be easy to read/understand (for someone familiar with code)
- Executed program should be easy/intuitive for user
  - Descriptive clear output
- Demonstrate program running in .out file

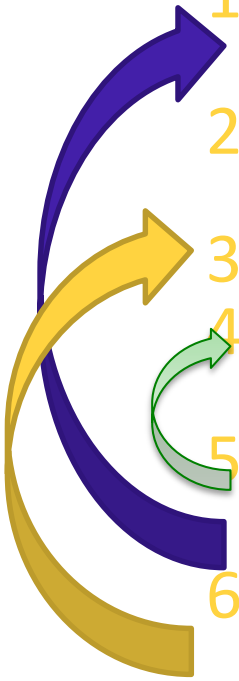
Review

- How can we make our program interactive with a user?
- What are the two types of division?
- How can we find the remainder from a division?

Goals of a Good Development Process

- Produces high-quality code
  - Correct behavior
  - Easy to use
  - Easy to read and understand by another programmer
- Programmer is productive
  - Doesn't waste time on manual/unnecessary tasks (e.g., user input until needed)
    - Reduces time spent debugging

Formalizing Process of Developing Computational Solutions



1. Think about expectations/test cases
   - "When user enters these values, this should happen."
2. Create a sketch of how to solve the problem (the algorithm)
3. Fill in the details in Python
4. Execute the program ***with good, varied test cases*** *to try to* ***reveal errors***
5. If output doesn't match your expectation, debug the program
   - (Where is the problem? How do I fix it?)
6. Iterate to improve your program
   - Better variable names, better input/output, more efficient, …

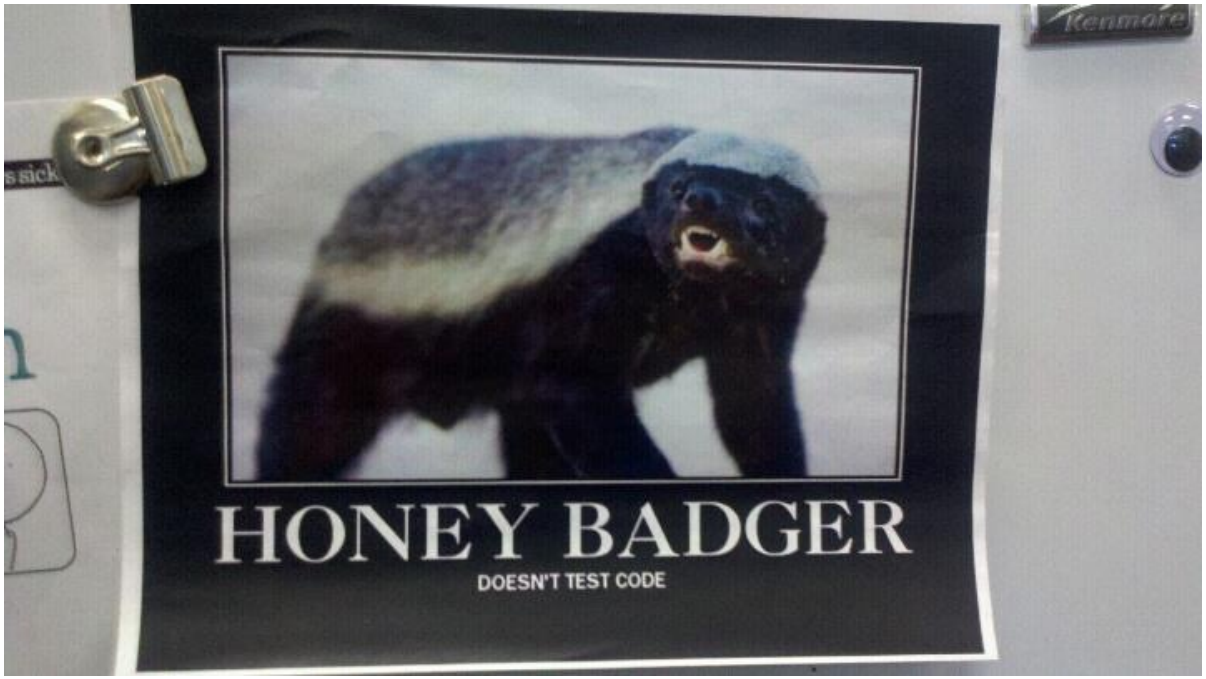
Review: Suggested Approach to Development

- Input is going to become fairly routine.
- Wait to get user input until you have figured out the rest of the program/algorithm.
- Develop/test without getting input first
  - Hardcode values
  - Speeds up process
- Then, add user input

Good Development Process

- Working in small chunks helps isolate problems in the code
  - Easier debugging!
- Iterative process encourages refinement, which yields higher quality
- Entering user input takes time. When you hardcode values, you can focus on the code working on one case, and then generalize with user input.
- Making your code easier to read makes it easier to maintain your mental model as code grows

Testing



Honey Badger gets a bad grade in CSCI111

Submission Expectations

- When have user input, your output file contains multiple runs, demonstrating that your program works for a variety of test cases
- Suggestion
  - Demonstrate an easy-to-validate test case
  - Demonstrate some “tricky” cases to show that your code works as expected
- Don’t need to test things that we can’t handle
  - Example: user enters a string instead of a number

Review: Arithmetic Operations

Symbol	Meaning	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder (“mod”)	Left
**	Exponentiation (power)	Right

Precedence rules: P E - DM% AS

negation

Associativity matters when you have the same operation multiple times

Review: Two Division Operators

/ Float Division

- Result is a `float`
- Examples:
  - `6/3 → 2.0`
  - `10/3 → 3.3333333333333335`
  - `3.0/6.0 → 0.5`
  - `10/9 → 1.9`

// Integer Division

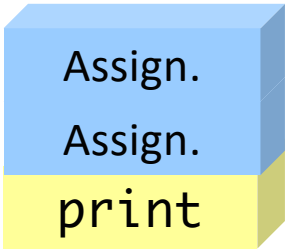
- Result is an `int`
- Examples:
  - `6//3 → 2`
  - `10//3 → 3`
  - `3.0//6.0 → 0`
  - `10//9 → 1`

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution
- Example (Standard Algorithm)
  - Get input from user
  - Do some computation
  - Display output



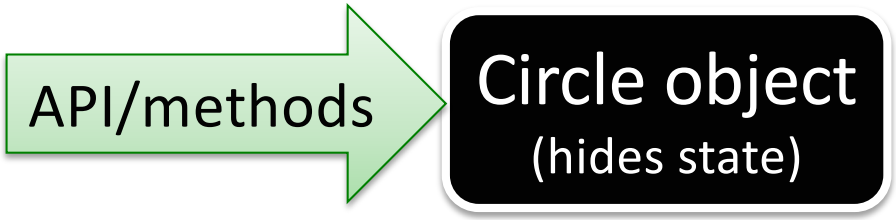

```
x = input("…")
ans = …
print(ans)
```

Review: Object-Oriented Programming

- What is the term for how we create a new object?
  - What is the syntax for that?
- What is the term for how we give commands to/do operations on objects?
  - What is the syntax for that?
  - What are two types of those operations we talked about?
    - What is the difference between the methods?
    - How does that difference affect how we use them?
- How do we get access to the code in `graphics.py` in our code?
- How can we find out what we can do to an object?
- Consider a Circle
  - What can we do to a circle?
  - What state do you think the Circle has?
- Create a *design pattern* for graphics programs
  - (What is the template? What should it contain?)

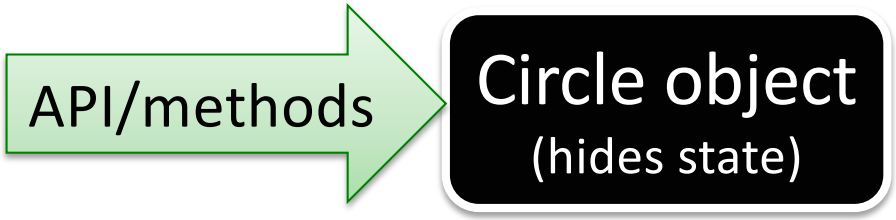
Circle Object

- Methods
  - getCenter()
  - getRadius()
  - setFill(<color>)
  - …
- State
  - A center
  - A radius
  - If it was drawn
  - Fill color
  - …



Circle Object

- Methods
  - getCenter()
  - getRadius()
  - setFill(<color>)
  - …
- State
  - A center
  - A radius
  - If it was drawn
  - Fill color
  - …




```
circle = Circle(…)
```

If state not hidden, could change incorrectly

```
circle.fill_color = “frog”
```

Method will fail and fill color will not change

```
circle.setFillColor(“frog”)
```

Our Graphics Programming Design Pattern

- Import the Graphics Library
- Create the GraphWin
- Repeat:
  - Construct an object
    - May need to construct the objects it needs first
    - Set up its color, width, …
  - Draw the object
- At the end of program
  - Call getMouse to make the window stay open until the user clicks
  - Then, call close on the window

Programming with the Graphics Library

- Algorithm for our program
  - Create an instance of a 50x100 Rectangle
  - Draw the rectangle
  - Shift the instance of the Rectangle class to the right 10 pixels
  - Display (print) the x- and y- coordinates of the upper-left corner of the Rectangle
- Now, implement it!
  - Draw on paper to help you think it through
  - Refer back to example program

Post-mortem:
Analyzing Problem-Solving Process

- There were gaps in our algorithm
  - We needed a GraphWin
  - We needed to import graphics.py
  - Don't forget to wait for the mouse click and then close
- We didn't necessarily work linearly
  - Iteration often involves working backwards or in circles or …

Designing for Change

- Sometimes there are “magic numbers” in our code
  - Example: 200 in board
- Humans have more trouble understanding numbers than understanding words
- Give our magic numbers meaning by assigning them to variables, called ***constants***
  - Example: PI = 3.14159…
  - Name constants with all capital letters (and maybe underscores)
  - Put constants at the top of programs
  - Conventions makes them easier to identify and change
    - Software is ***soft***

Example: Designing for Change

- First, define the constant
- Base later values on constants

```
WIDTH=200
window = GraphWin(WIDTH, WIDTH*2)
upperRightPoint = Point(0, WIDTH)
```

- Why is this a better design?
  - If want to change the width and keep rest of code working, update the constant (in one place)
- Using all caps is an indication that this is something that won't change during the program's execution

Example: Designing for Change

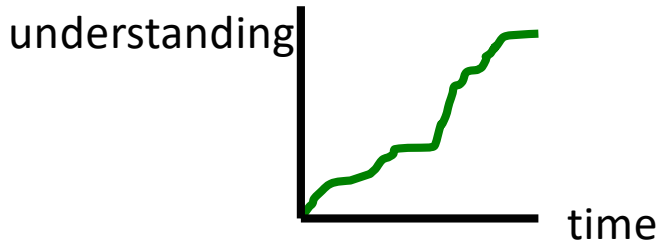
- Example with a non-integer data type
- Consider a *color theme* for your image

```
MAIN_COLOR=rgb_color(135, 206, 235)
HIGHLIGHT_COLOR=rgb_color(255, 219, 0)
```

- Later…

```
rect = Rectangle(…)
rect.setFill(MAIN_COLOR)
rect.setOutline(HIGHLIGHT_COLOR)
```

Lessons from Lab



- Look at examples!
  - "We were able to do this in that other program. How did we do that?"
  - On the course schedule page
- Explore!
  - Try things out in interactive mode
  - Then, put the ones that work into a script/program
- Testing!
  - Start with smaller and easy-to-verify tests
  - Test a variety of inputs
- Follow all of the directions!

Lab Overview

- Arithmetic problems
- Graphics API Problems
  - Update web page

Sprenkle office hours for Wed
are changed to 12 p.m. to 2 p.m.