Objectives

- Review
- Lab 2
  - Programming practice

Feedback on Lab 1

- Overall good
- Notes
  - Saved output from each program
    - With user input, try several different 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!

Review

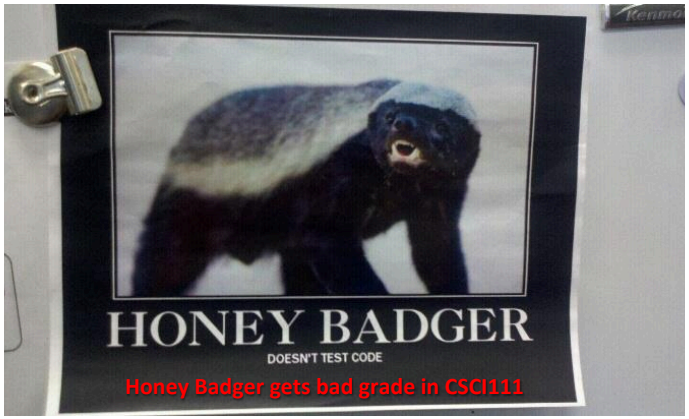
- What program do we use to develop programs?
  - What is the command you execute to start it?
- What is our process for developing programs?
- How can we make our program interactive with a user?

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

IDLE Review

- Run using `idle3 &`

Testing



Recommendation

- Get user input last – this is a fairly routine step
- Develop/test without getting input first
  - Speeds up process
- Then, add user input

Review

- What are the two types of division?
- How can we find the remainder of a division?
- How can we make something repeat a certain number of times?

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`

Review: Formalizing Process of Developing Computational Solutions

1. Think about the test cases
   a. Input, expected output
2. Create a sketch of how to solve the problem (the algorithm)
3. Fill in the details in Python
4. Test the Python program with *good* test cases
   a. If errors found, **debug** program
   b. Repeat step 3

Good Development Practices

- Design the algorithm
  - Break into pieces
- **Implement** *and* **Test** each piece *separately*
  - Identify the best pieces to make progress
  - Iterate over each step to improve it
- Write comments FIRST for each step
  - Elaborate on what you're doing in comments when necessary

Review

- How do we repeat code?

for Loop Syntax and Semantics

- Use when know how many times loop will execute
  - Repeat N times

Times to repeat

```
for i in range(10):
    statement_1
    statement_2
    …
    statement_n
```

"Body" of for loop
- Gets repeated
- Note indentation

for loop review

```
for x in range(5):
    # like assigning x values(0,1,2,3,4)
    # consecutively, each time through loop

    # rest of loop body …
```

- Note: when have range(5),
  - x gets values (0, 1, 2, 3, 4)
  - Which means that loop executes 5 times
- Optional: start and step parameters

Practicing for Loops

What is getting repeated? How many times?

- A)
  1
  2
  3
  4
  Tell me that you love me more
- C)
  10
  9
  8
  7
  …
  1
  Blast off
- B)
  I had the time of my life
  And I never felt this way before
  And I swear this is true
  And I owe it all to you

  3 times, followed by Dirty bit

Review: Programming Practice

- Add 5 numbers, inputted by the user
  - After implementing, simulate running on computer

Key questions:
- What is getting repeated?
- How many times?

sum5.py

Comparing Solutions

sum5.py

```
print("This program will add up 5
numbers given by the user.")

total = 0

for x in range(5):
    num = eval(input("Enter number:
"))
    total = total + num

print("The total of the inputted
numbers is ", total)
```

sum5_no_loop.py

```
print("This program will add up 5
numbers given by the user.")

num1 = eval(input("Enter number: "))
num2 = eval(input("Enter number: "))
num3 = eval(input("Enter number: "))
num4 = eval(input("Enter number: "))
num5 = eval(input("Enter number: "))

total = num1 + num2 + num3 + num4 +
num5

print("The total of the inputted
numbers is ", total)
```


Comparing Solutions

- Both are valid solutions
- sum5_no_loop.py is conceptually simpler
  - Don't need to understand what the loop does
- sum5.py has less repeated code
  - Makes it easier to change if we decide to change what gets repeated
- sum5.py is easier to change how many numbers are input
  - More on that on Wednesday

Generalizing Solution: Accumulator Design Pattern

1. Initialize accumulator variable
2. Loop until done
   - Update the value of the accumulator
3. Display result

How does this pattern relate to the sum5.py solution?

Generalizing Solution: Accumulator Design Pattern

1. Initialize accumulator variable
2. Loop until done
   - Update the value of the accumulator
3. Display result

`total` is the accumulator variable

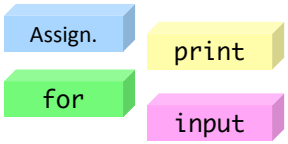
```
total = 0

for x in range(5):
    num = eval(input("Enter
number: "))
    total = total + num

print("The total of the inputted
numbers is ", total)
```

Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment
    - So far: Arithmetic, functions
  - Print
  - For
  - Input (also with assignment)



- We can combine them to create more complex programs
  - Solutions to problems
- When solving problems, think, "To solve this part of the problem, I need this building block."

Honor

- You may discuss programming assignments *informally* with other students
  - Sharing the **code** is an honor violation
  - Do **not** share your password
- You should know where to draw the line between legitimate outside assistance with course material and outright cheating
  - Students who obtain too much assistance without learning the material ultimately cheat themselves
- If you have any uncertainty about what this means, consult with me before you collaborate
- All written assignments should be done individually

Honors System: Rules of Thumb

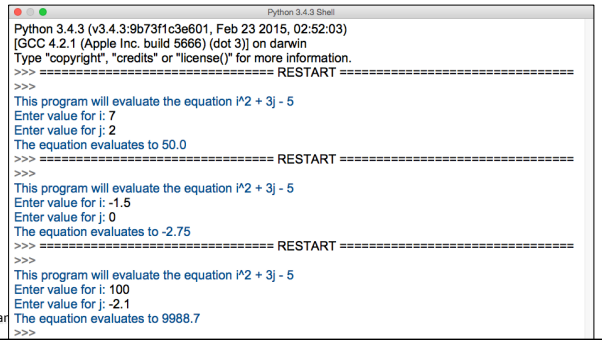
- Discussion of problems/programs - OK
  - Clarification questions
  - Algorithm discussion (on paper, board)
- Debugging help
  - Programmer always "owns" keyboard, mouse
  - Helper can read other's program/debug/help, up to 5 minutes
    - Ask TA or me or email me for problems that require more time

Lab 2 Expectations

- Comments in programs
  - High-level comments, author
  - Notes for your algorithms, implementation
- Testing programs
  - What are good test cases for your programs?
  - Show the output from those test cases
  - **But** don't go overboard by testing every possible number!
  - Don't need test things for which we can't handle well

Lab 2 Expectations: Example Output

- For programs that take user input, run multiple times to demonstrate that the program works.
- Example output that should be saved in the .out file



Lab 2 Expectations

- Nice, readable, understandable output
  - Think about if you were the user of the program: what would you want to see?
  - Don't show me any of your "scratch work" from earlier versions of the program that don't work.
- Honor System
  - Pledge the Honor Code on printed sheets