

Lab 4

- Review Lab 3
 - Run Animations!
- Function review

Lab 3

- Iterative Fibonacci Sequence has been a question on several students' internship or job interviews

Lab 3 Feedback

- Continuing to get tougher in grading
 - Paying more attention to style (e.g., variable names), efficiency, readability, good output
 - High-level descriptions
 - More strict on adhering to problem specification
 - Constants
 - Demonstrate program **more than once** if gets *input* from user or *outcome changes* when run again
 - Find errors before I do!

Program Organization

```
# high-level description  
# author name  
  
import statements  
  
CONSTANT_DEFNS = ...  
  
program_statements ...  
program_statements ...  
program_statements ...
```

Program Organization

```
# high-level description
# author name

import statements

CONSTANT_DEFNS = ...

def main():
    statements...
    statements...

def otherfunction():
    statement...
```

Lab 3 Feedback: Common Issues

Which solution is more efficient (i.e., requires the computer to do less “work”)?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

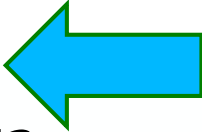
vs

```
for operand2 in range(1, 15):
    operand1=12
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

Lab 3 Feedback: Common Issues

Which solution is more efficient (i.e., requires the computer to do less “work”)?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```



vs

```
for operand2 in range(1, 15):
    operand1=12 ← Additional assignment each time through loop
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

Lab 3 Feedback: Common Issues

Which solution is simpler?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

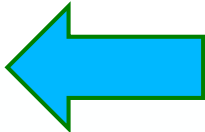
vs

```
operand1=12
operand2=0
for x in range(14):
    operand2 = x + 1
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```


Lab 3 Feedback: Common Issues

Which solution is simpler?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```



vs

```
operand1=12
operand2=0
for x in range(14):
    operand2 = x + 1
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

More code makes solution
more difficult to understand.

Let loop do the work of updating operand2.

Lab 3 Feedback: Common Issues

Which solution is simpler?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

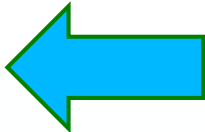
vs

```
operand1=12
for x in range(1, 15):
    operand2 = x
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

Lab 3 Feedback: Common Issues

Which solution is simpler?

```
operand1=12
for operand2 in range(1, 15):
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```



vs

```
operand1=12
for x in range(1, 15):
    operand2 = x
    result = operand1 % operand2
    print(operand1, "%", operand2, "=", result)
```

More code makes solution more difficult to understand.
Let loop do the work of updating operand2.

Animation Feedback

- If moving multiple objects together
 - Move *all* the objects, then sleep
 - Otherwise, animation looks choppy
- Could use a list with the `for` loop, as discussed in several sections in the textbook
 - Simplifies and reduces code

```
for object in [ myObj1, myObj2, myObj3 ]:  
    object.move()  
sleep(.001)
```

Run Animations

Review

- What are characteristics of a good function?
- What information should be in a function's docstring?
- How can we programmatically test functions?
- What is a variable's *scope*?
 - What are the scope *levels*?
 - What scope do most of the variables we were discussing have?
- What happens when a function reaches a *return* statement?
- Synthesis: Where do variables *implicitly* get assigned a value?
 - Provide examples where a variable's value is set, but there is no explicit *assignment* statement?

Review: Writing a “Good” Function

- Should be an “intuitive chunk”
 - Doesn’t do too much or too little
 - If does too much, try to break into more functions
- Should be reusable
- Should have an “action” name
- Should have a comment that tells what the function does

Review: Writing Comments for Functions

- Good style: Each function ***must*** have a comment
 - Describes functionality at a high-level
 - Include the *precondition*, *postcondition*
 - Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)

Review: Writing Comments for Functions

- Include the function's pre- and post- conditions
- **Precondition:** Things that must be true for function to work correctly
 - E.g., num must be even
- **Postcondition:** Things that will be true when function finishes (if precondition is true)
 - E.g., the returned value is the max

Review: Testing sumEvens

```
import test
...
def testSumEvens():
    actual = sumEvens( 10 )
    expected = 20
    test.testEqual( actual, expected )
    test.testEqual( sumEvens(12), 30)

def sumEvens(limit):
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total
```

This is the actual result from our function
This is what we expect the result to be

What are other good test cases?

Review: Variable Scope

- Functions can have the same parameter and variable names as other functions
 - Need to look at the variable's **scope** to determine which one you're looking at
 - Use the **stack** to figure out which variable you're using
- Scope levels
 - **Local scope (also called function scope)**
 - Can only be seen within the function
 - **Global scope (also called file scope)**
 - Whole program can access
 - More on these later
- Know "lifetime" of variable
 - Only during execution of function
 - Related to idea of "scope"
- In general, our only *global* variables will be constants because we don't want them to change value
 - e.g., EIEIO



Evolving General Design Patterns

- Former general design pattern:
 1. Optionally, get user input
 2. Do some computation
 3. Display results
- Now general design pattern:
 1. Optionally, get user input
 2. Do some computation by calling **functions**, get results
 3. Display results

Development Process: Bottom-Up

1. Define a function

➤ Document

➤ Test the function

1

Function

Focus on just a part of the larger problem

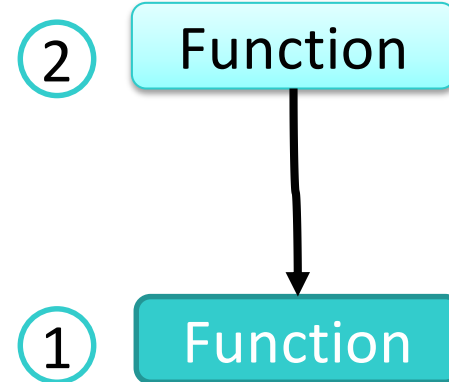
Development Process: Bottom-Up

2. Use the function in context/ call the function

1. Define a function

➤ Document

➤ Test the function



Bottom-Up Development Example

1. Define (and document and test) a function that
 - Given the number of successes and failures
 - Returns the success rate
 - Could be used for a win/loss percentage or for a player's stealing percentage
2. Create a program that
 - Prompts for a team's wins and losses
 - Displays the team's win percentage

Review: Refactoring

- Refactoring is the process of changing your code to improve maintainability, reusability, quality, etc. without significantly changing its functionality
- Examples: renaming variables to be more descriptive, creating a variable for a “magic number”, ...

Refactoring into Functions

- Symptom: you note that there is some functionality that would benefit from being a function
- Motivation: improve readability and reusability of your programs

Development Process:

Refactoring Functionality into Functions

1. Identify functionality that should be put into a function
 - What should the function do?
 - What is the function's input?
 - What is the function's output (i.e., what is returned)?
2. Define the function
3. Test the function programmatically
4. Call the function where appropriate, replacing the former non-function-ified code
5. Test

Example: PB & J

1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread
7. Close bread
8. Clean knife
9. Put away materials

- Which of these are the “core” part of making a PB & J sandwich?
- How would you describe the rest of the parts?

Example: PB & J

1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread
7. Close bread
8. Clean knife
9. Put away materials

Example: PB & J as Functions

1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread
7. Close bread
8. Clean knife
9. Put away materials

```
def main():  
    prepare()  
    makePBJSandwich()  
    cleanUpSupplies()  
main()
```

Example: PB & J as Functions, 10 x

1. Gather materials (bread, PB, J, knives, plate)
2. Open bread

3. Put 2 pieces of bread on plate
4. Spread PB on one side of bread
5. Spread Jelly on other side of bread
6. Place PB-side facing up

7. Close bread
8. Clean knife
9. Put away materials

```
def main():  
    prepare()  
    for sandwich in range(10):  
        makePBJSandwich()  
    cleanUpSupplies()  
main()
```

Refactoring in Practice

Original file with code
to be refactored

New (empty)
program file

1. Copy relevant code from original file into the new file
2. Convert that code into a function in the new file
3. Test the function programmatically
4. Copy other code from original function into new file, replacing the functionality with a call to the newly defined function

Refactoring: An Iterative Process

- As you refactor, you'll often note new places to refactor
- Example: after extracting functionality into a function, you'll realize that it would be helpful to put the rest of your code in a `main` function

Summary: Development Approaches

- There are several development approaches
- Not mutually exclusive
- Often will switch between them, depending on circumstances
- As programs grow in size, there is no “one way” to write code
 - But there may be better ways to make progress
 - If you’re stuck, step back and reassess your approach

Default Values for Parameters

- Can assign a default value to parameters
- We've seen this with other functions
 - Example: range has a default start of 0 and step of 1 when called as range(stop)

```
def rollDie(sides=6):  
    """  
    Given the number of sides on the die (a positive integer),  
    simulates rolling a die by returning the rolled value,  
    between 1 and sides, inclusive.  
    If no parameter passed, the number of sides defaults to 6.  
    """
```

Debugging Mantra

- When you're debugging, a good mantra is

“I think I'm about to learn something”

Lab 4 Overview

- Calling functions defined in the same program
- Refactoring code
- Modifying function definitions
- Testing functions
- Creating a module
- Writing a program with a function from scratch