

Lab 4

- Review Lab 3
 - ▶ Run Animations!
- Function review

1

Lab 3

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

2

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!

Feb 7, 2023

Sprenkle - CSCI111

3

3

Program Organization

```
# high-level description
# author name

import statements

CONSTANT_DEFNS = ...

program_statements ...
program_statements ...
program_statements ...
```

Feb 7, 2023

Sprenkle - CSCI111

4

4

Program Organization

```
# high-level description
# author name

import statements

CONSTANT_DEFNS = ...

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

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

Feb 7, 2023

Sprenkle - CSC1111

5

5

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)
```

Feb 7, 2023

Sprenkle - CSC1111


6

6

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)
```

← Additional assignment each time through loop

Feb 7, 2023

Sprenkle - CSC1111

7

7

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)
```

Feb 7, 2023

Sprenkle - CSC1111


8

8

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

Feb 7, 2023

Sprenkle - CSC1111

9

9

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)
```

Feb 7, 2023

Sprenkle - CSC1111


10

10

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

Feb 7, 2023

Sprenkle - CSCI111

11

11

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)
```

Feb 7, 2023

Sprenkle - CSCI111

12

12

Run Animations

13

Review

- What are characteristics of a good function?
- How can we programmatically test functions?
- What are two development processes we have discussed?
- 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?

14

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

Feb 7, 2023

Sprenkle - CSC1111

15

15

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)

Feb 7, 2023

Sprenkle - CSC1111

16

16

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

Feb 7, 2023

Sprenkle - CSCI111

17

17

Refactoring: Converting 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
 - Comment out the other code temporarily
4. Call the function where appropriate
5. Create a `main` function that contains the "driver" for your program
 - Put at top of program
6. Call `main` at bottom of program
7. Write documentation for function

Feb 7, 2023

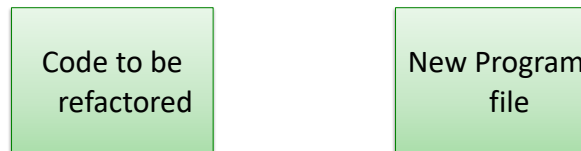
Sprenkle - CSCI111

18

18

Refactoring In Lab

- A little different than I showed you in class
- Refactoring in-line (modifying the code) is a little more sophisticated than you might be ready for
- Instead, copy code from the original file to the new file



Feb 7, 2023

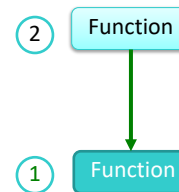
Sprenkle - CSCI111

19

19

Review: Bottom-Up Development

- Use the function in context/
call the function
- Define a function
 - Document
 - Test the function



Feb 7, 2023

Sprenkle - CSCI111

20

20

test module's testEqual function

- Example from yesterday

```
def testWinPercentage():
    test.testEqual( calculateWinPercentage(0, 1), 0 )
    test.testEqual( calculateWinPercentage(2, 2), .5 )
    test.testEqual( calculateWinPercentage(3, 7), .3 )
    test.testEqual( calculateWinPercentage(1, 0), 1 )

testWinPercentage() Could add a parameter for the
                    number of decimal places of precision
```

After confirming that the function works...

test module's testEqual function

- Example from yesterday

```
def testWinPercentage():
    test.testEqual( calculateWinPercentage(0, 1), 0 )
    test.testEqual( calculateWinPercentage(2, 2), .5 )
    test.testEqual( calculateWinPercentage(3, 7), .3 )
    test.testEqual( calculateWinPercentage(1, 0), 1 )

# testWinPercentage()
main()
```

Comment out call to test function.
Call main.

Docstring on Function

```
def calculateWinPercentage(wins, losses):  
    """  
    Calculates and returns a win percentage, based on the given wins  
    and losses.  
    Parameters:  
    - wins: a non-negative integer representing the number of wins  
    - losses: a non-negative integer representing the number of losses  
    Pre: either wins or losses must be greater than 0 or will throw a  
    divide by zero error  
    Post: returns the win percentage (between 0 and 1, inclusive)  
    """  
    ...
```

Good docstring because

- Describes parameters
- Describes what is return (is it a %?)
- Describes error cases

Development approach:

TOP-DOWN DEVELOPMENT

Top-Down Development

- I have a problem
- But, that problem can be broken into smaller problems
- Solution:
 - Problems → functions!
 - Divide and Conquer!

Example: Top-Down Development

- I want to calculate and then display a team's win/loss percentage based on user input

Example: Top-Down Development

- I want to calculate and then display a team's win/loss percentage based on user input
- Algorithm:
 - Get user input for number of wins and losses
 - Calculate the win percentage
 - Display the results

Feb 7, 2023

Sprenkle - CSC1111

27

27

Example: Top-Down Development – Design: Identify Functions

- I want to calculate and then display a team's win/loss percentage based on user input
 - Algorithm:
 - Get user input for number of wins and losses
 - Calculate the win percentage
 - Display the results
- main
- calculateWinPercentage
- Think about how the function will be used first → API!

Feb 7, 2023

Sprenkle - CSC1111

28

28

Example: Top-Down Development – Starting Implementation...

```
def main():
    # get user input

    winPct = calculateWinPercentage(wins, losses)

    # display results

def calculateWinPercentage( numWins, numLosses ):
    """
    Given the number of wins and losses,
    calculates and returns the win percentage
    """
    ...
    ...
    ...

main()
```

Think about how the function will be used first → API!

Feb 7, 2023

Sprenkle - CSCI111

29

29

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

Feb 7, 2023

Sprenkle - CSCI111

30

30

Debugging

- When you're debugging, a good mantra is

“I think I'm about to learn something”

31

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

32