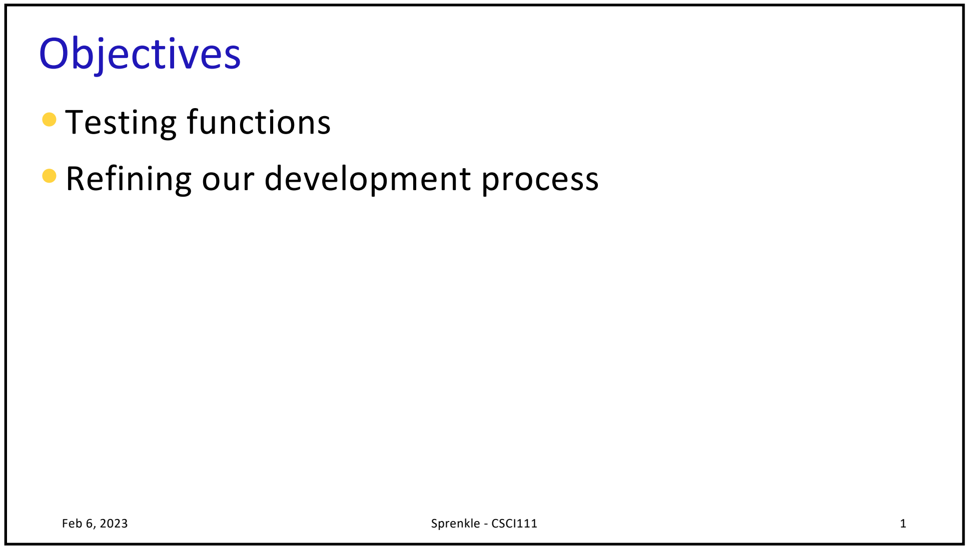
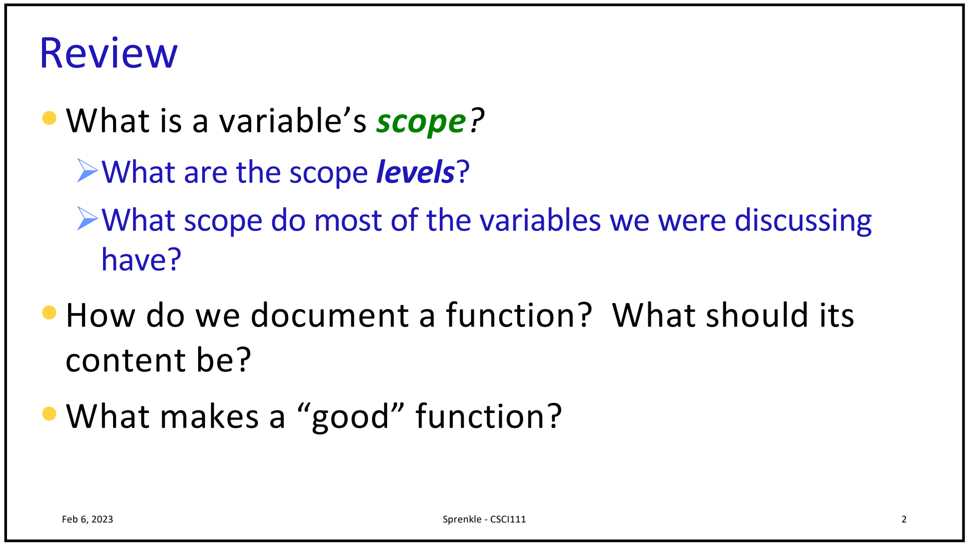


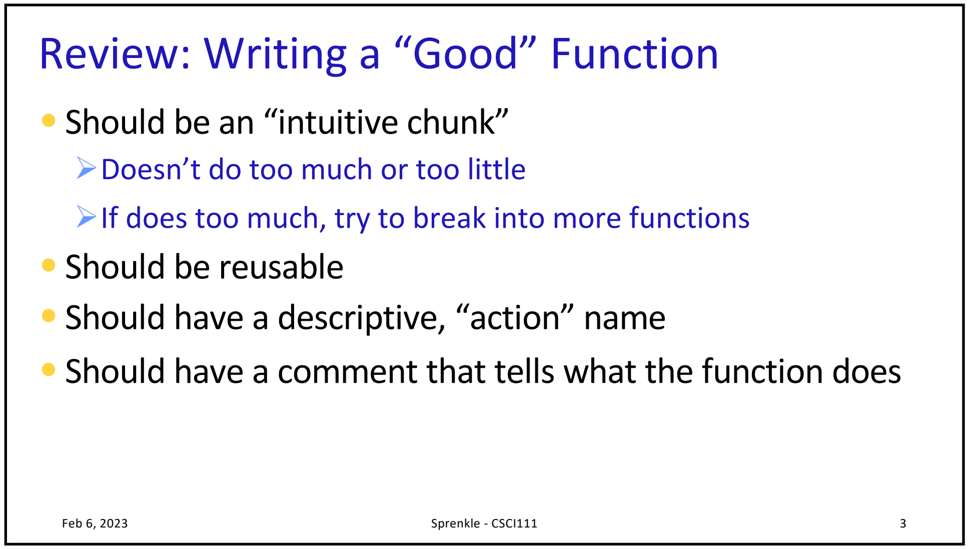
Objectives

- Testing functions
- Refining our development process



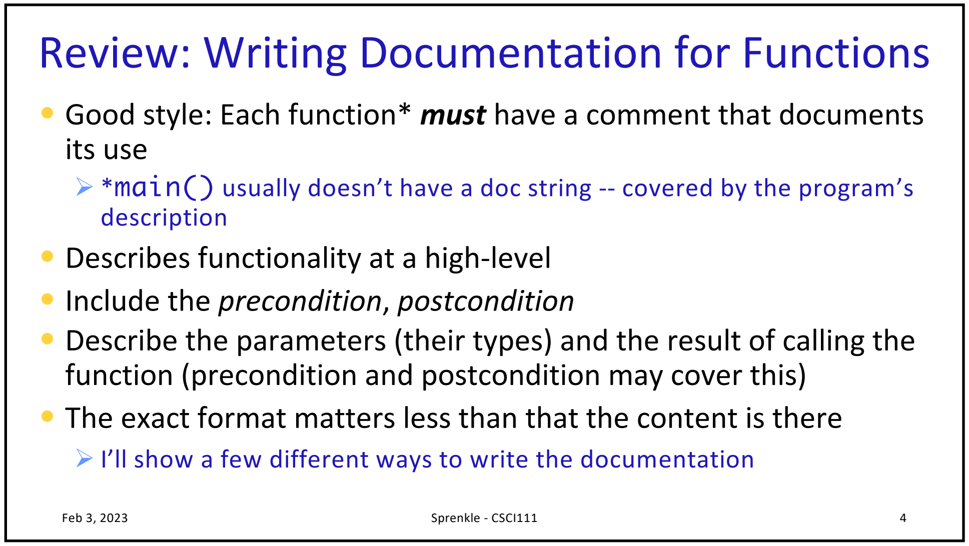
Review

- What is a variable's ***scope***?
  - What are the scope ***levels***?
  - What scope do most of the variables we were discussing have?
- How do we document a function? What should its content be?
- What makes a "good" function?



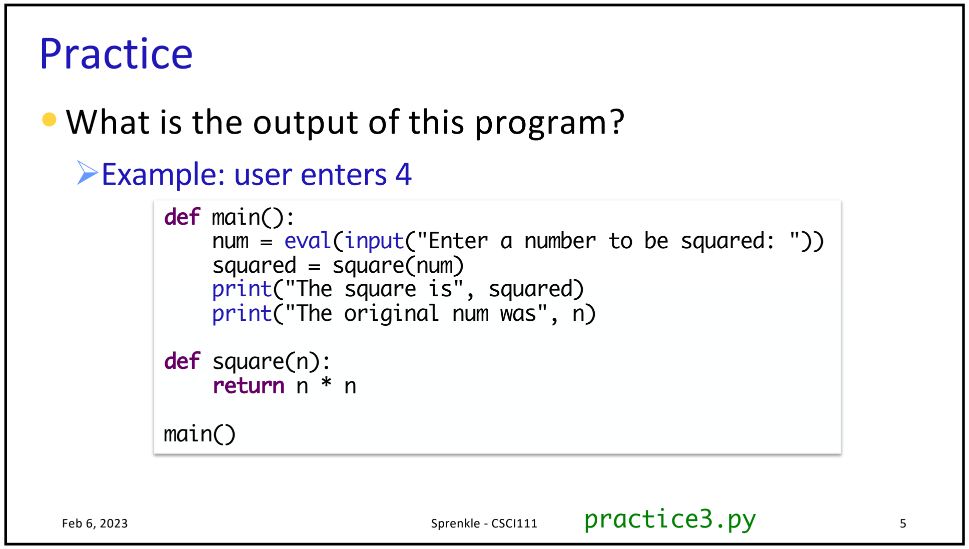
Review: Writing a "Good" Function

- 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 a descriptive, "action" name
- Should have a comment that tells what the function does



Review: Writing Documentation for Functions

- Good style: Each function* ***must*** have a comment that documents its use
  - ➢ *`main()` usually doesn't have a doc string -- covered by the program's description
- 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)
- The exact format matters less than that the content is there
  - ➢ I'll show a few different ways to write the documentation

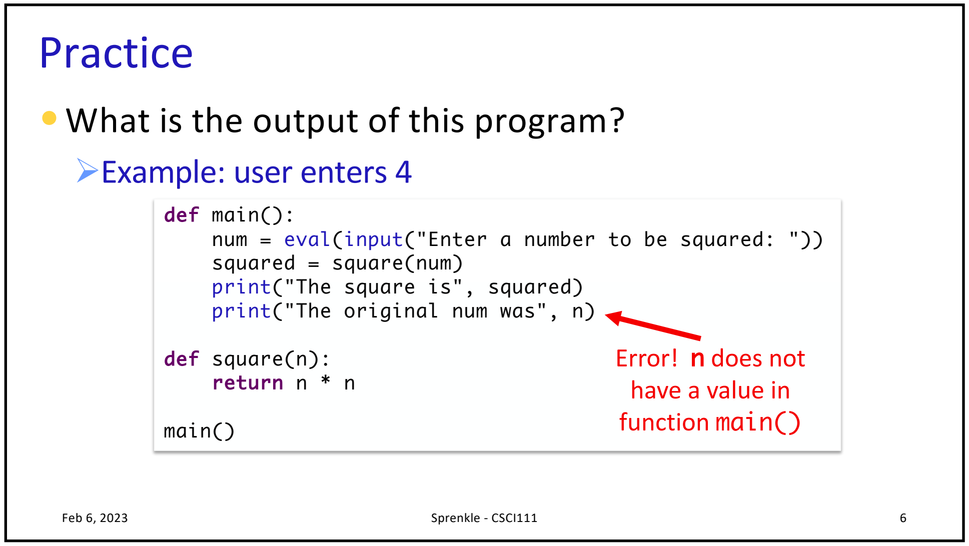


Practice

- What is the output of this program?
  - Example: user enters 4

```
def main():
    num = eval(input("Enter a number to be squared: "))
    squared = square(num)
    print("The square is", squared)
    print("The original num was", n)

def square(n):
    return n * n

main()
```

practice3.py

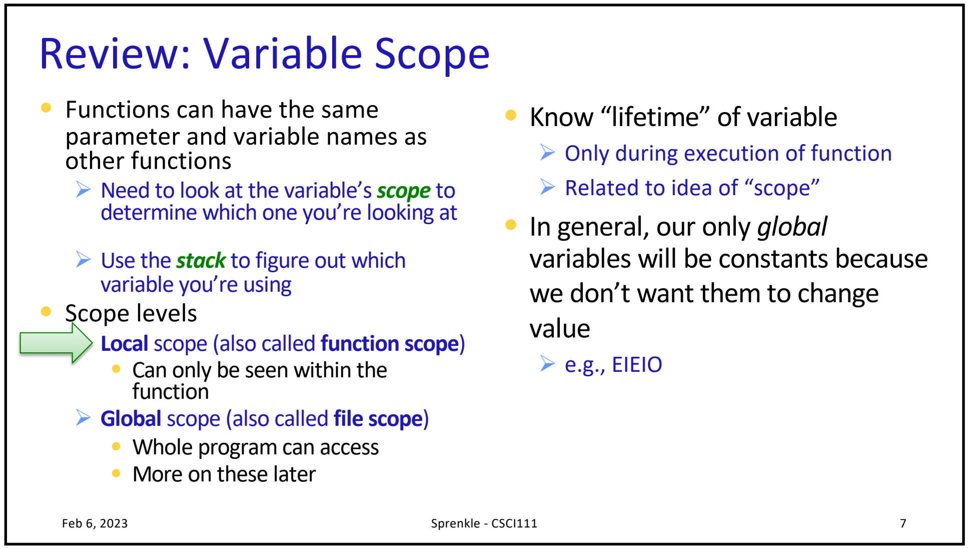


Practice

- What is the output of this program?
  - Example: user enters 4

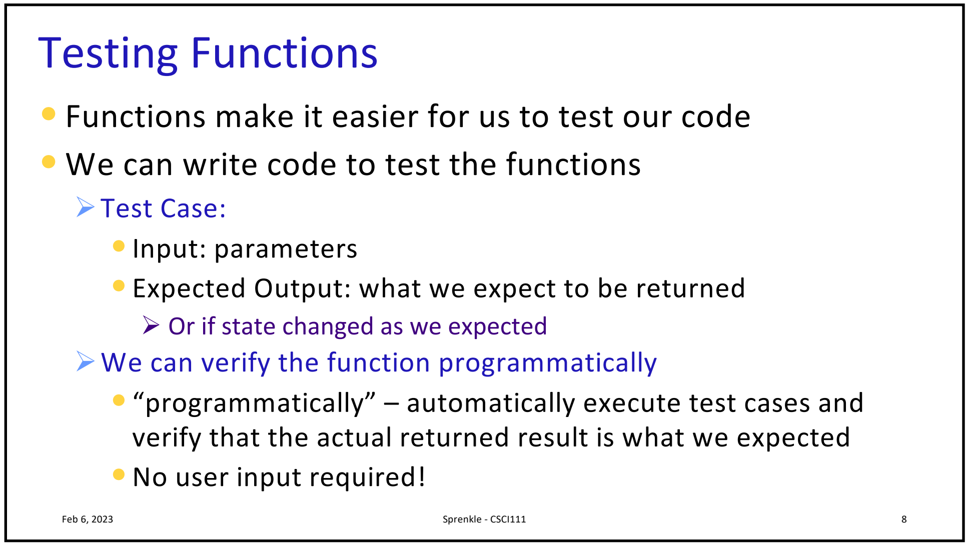
```
def main():
    num = eval(input("Enter a number to be squared: "))
    squared = square(num)
    print("The square is", squared)
    print("The original num was", n)

def square(n):
    return n * n

main()
```

Error! **n** does not have a value in function `main()`



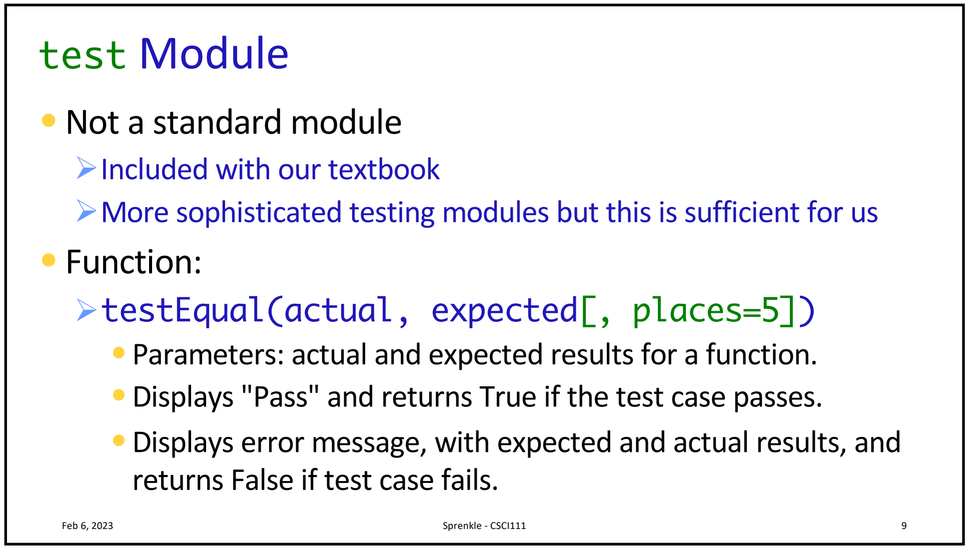
Review: Variable Scope

- 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



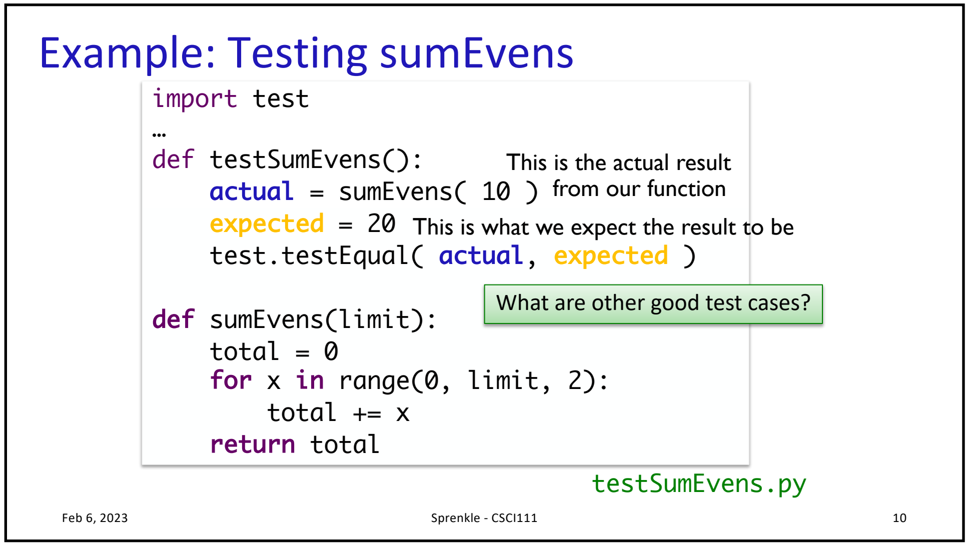
Testing Functions

- Functions make it easier for us to test our code
- We can write code to test the functions
  - Test Case:
    - Input: parameters
    - Expected Output: what we expect to be returned
      - Or if state changed as we expected
  - We can verify the function programmatically
    - "programmatically" – automatically execute test cases and verify that the actual returned result is what we expected
    - No user input required!



test Module

- Not a standard module
  - Included with our textbook
  - More sophisticated testing modules but this is sufficient for us
- Function:
  - `testEqual(actual, expected[, places=5])`
    - Parameters: actual and expected results for a function.
    - Displays "Pass" and returns True if the test case passes.
    - Displays error message, with expected and actual results, and returns False if test case fails.



Example: Testing sumEvens

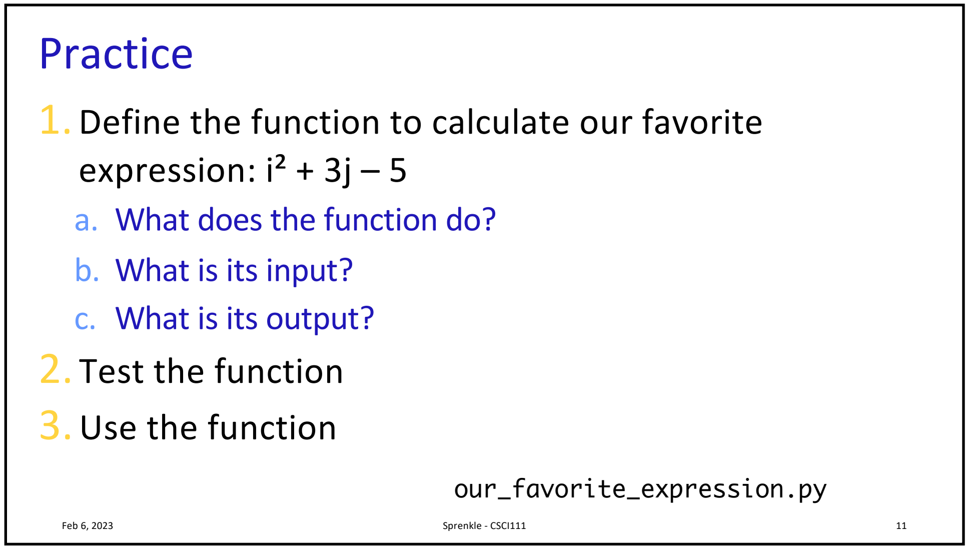
```
import test
…
def testSumEvens():
    actual = sumEvens( 10 )
    expected = 20
    test.testEqual( actual, expected )

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?

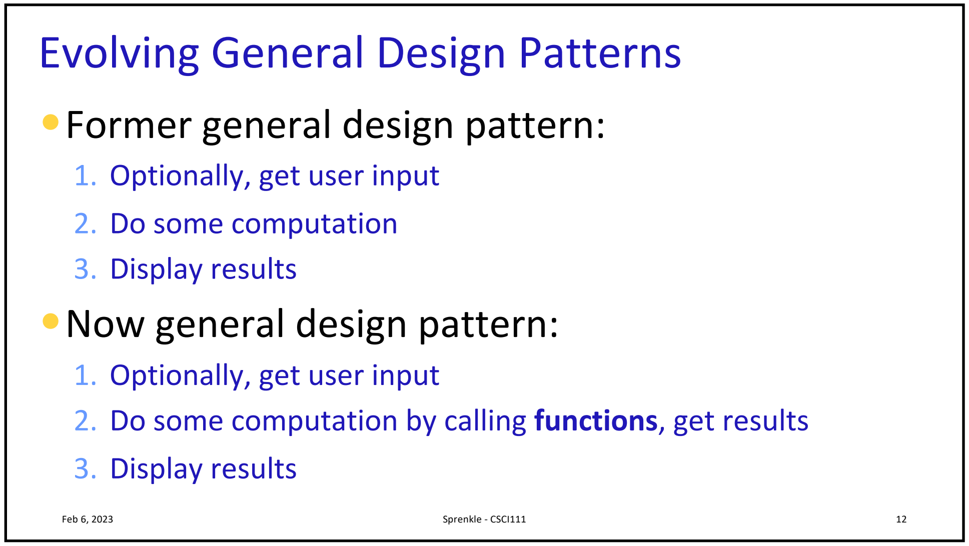
testSumEvens.py



Practice

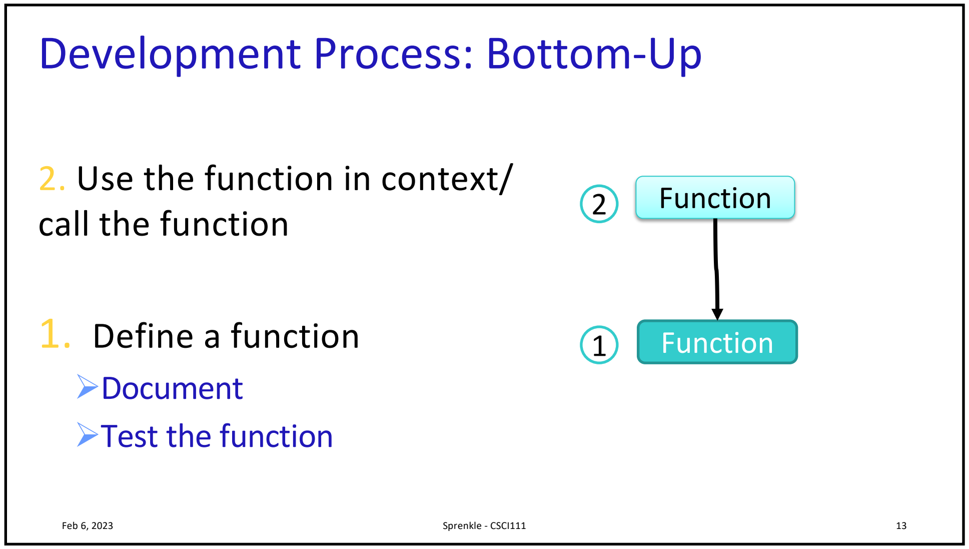
1. Define the function to calculate our favorite expression: $i^2 + 3j – 5$
   a. What does the function do?
   b. What is its input?
   c. What is its output?
2. Test the function
3. Use the function

`our_favorite_expression.py`



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



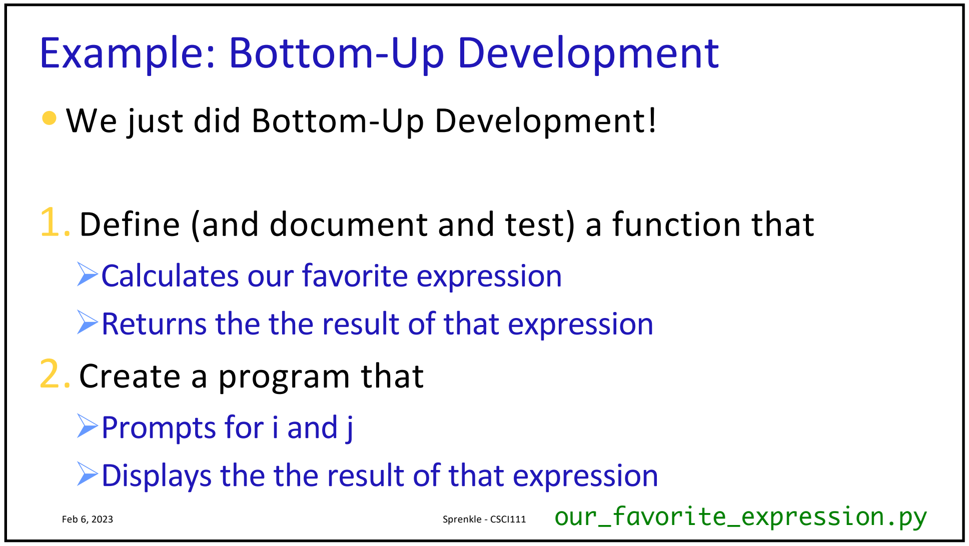
Development Process: Bottom-Up

2. Use the function in context/ call the function

1. Define a function
   - Document
   - Test the function

2
Function
1
Function

13



Example: Bottom-Up Development

- We just did Bottom-Up Development!

1. Define (and document and test) a function that
   - Calculates our favorite expression
   - Returns the the result of that expression
2. Create a program that
   - Prompts for i and j
   - Displays the the result of that expression

`our_favorite_expression.py`

14

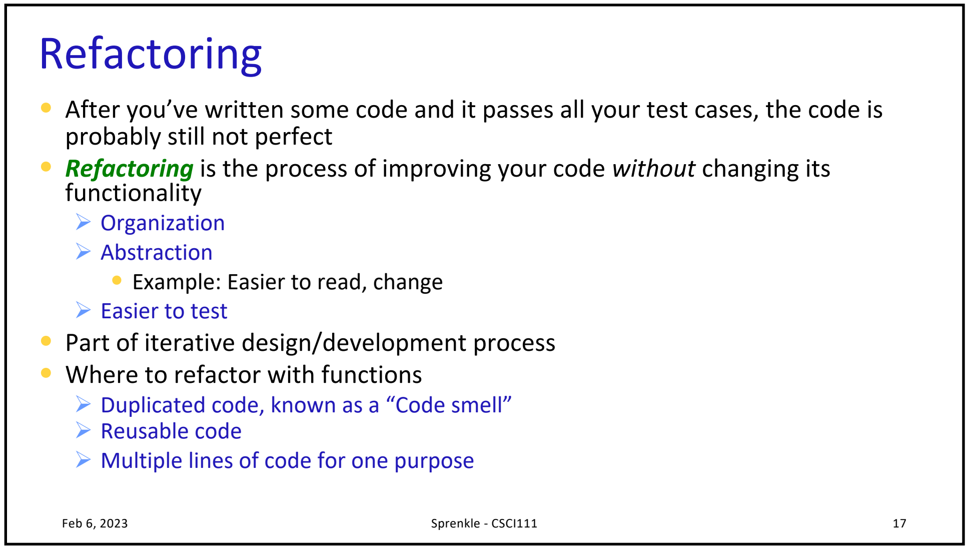
Practice: Finding a Team’s Winning Percentage

- There are lots of ways to develop programs
- Let’s go back to the way we originally developed programs
- Problem:
  - Prompt the user for a team’s wins and losses and display the team’s win percentage

`winpercent.py`

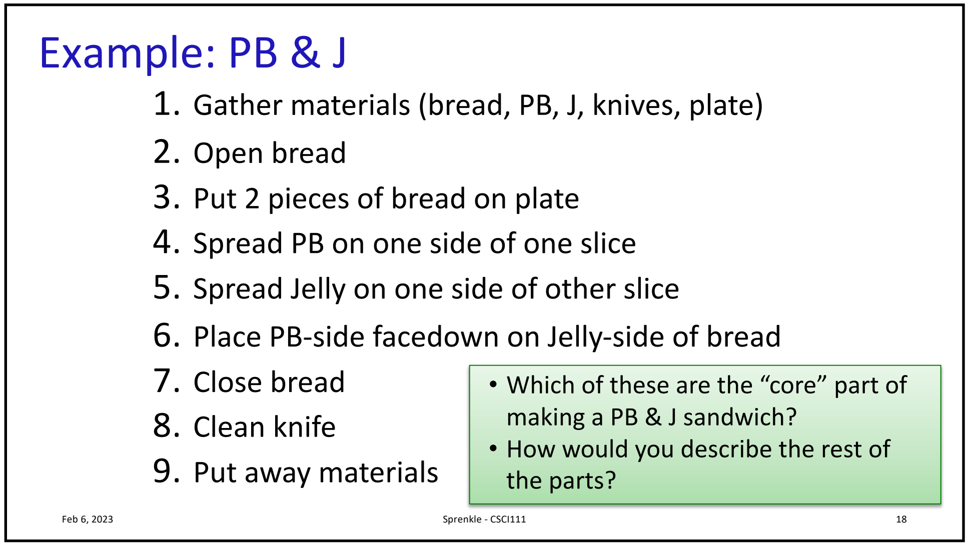
Another development approach

REFACTORING



Refactoring

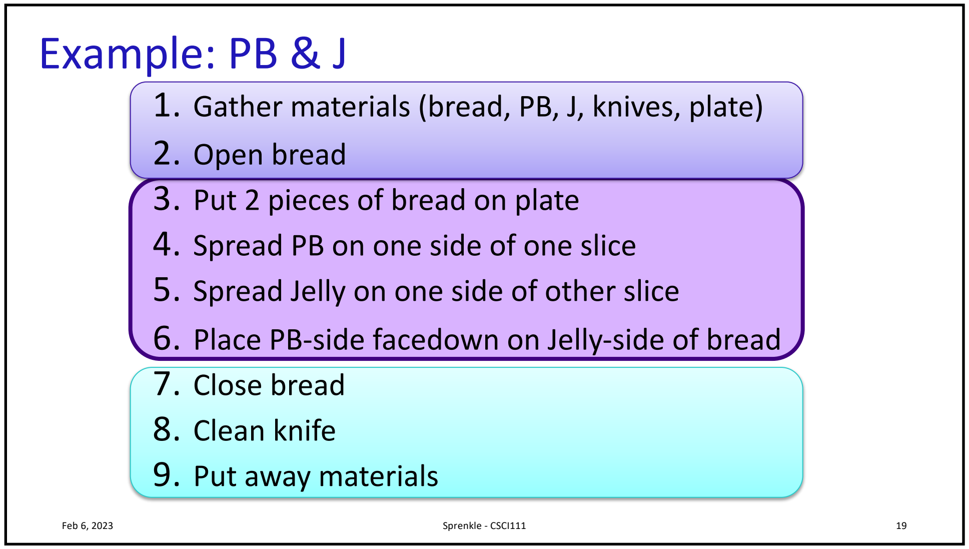
- After you’ve written some code and it passes all your test cases, the code is probably still not perfect
- ***Refactoring*** is the process of improving your code *without* changing its functionality
  - Organization
  - Abstraction
    - Example: Easier to read, change
  - Easier to test
- Part of iterative design/development process
- Where to refactor with functions
  - Duplicated code, known as a “Code smell”
  - Reusable code
  - Multiple lines of code for one purpose



Example: PB & J

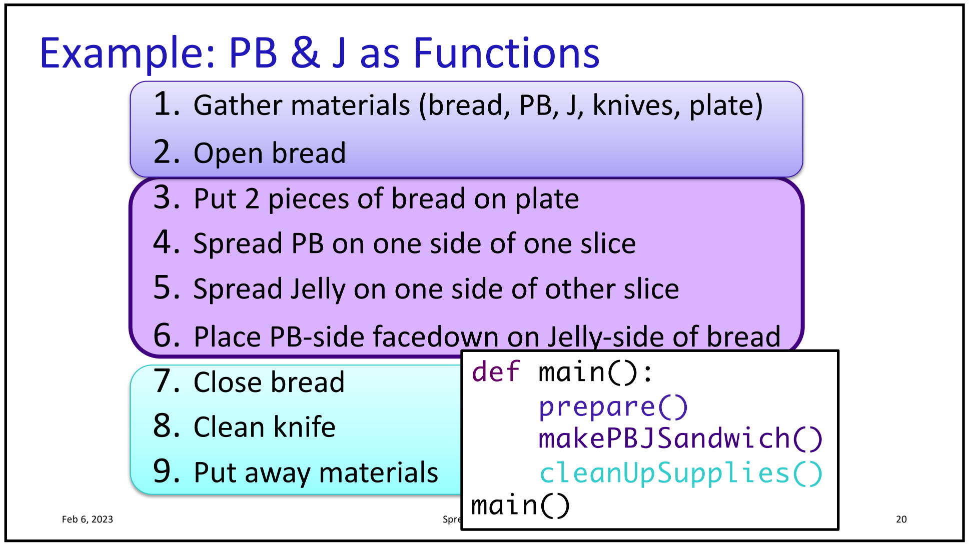
1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread
7. Close bread
8. Clean knife
9. Put away materials

- 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

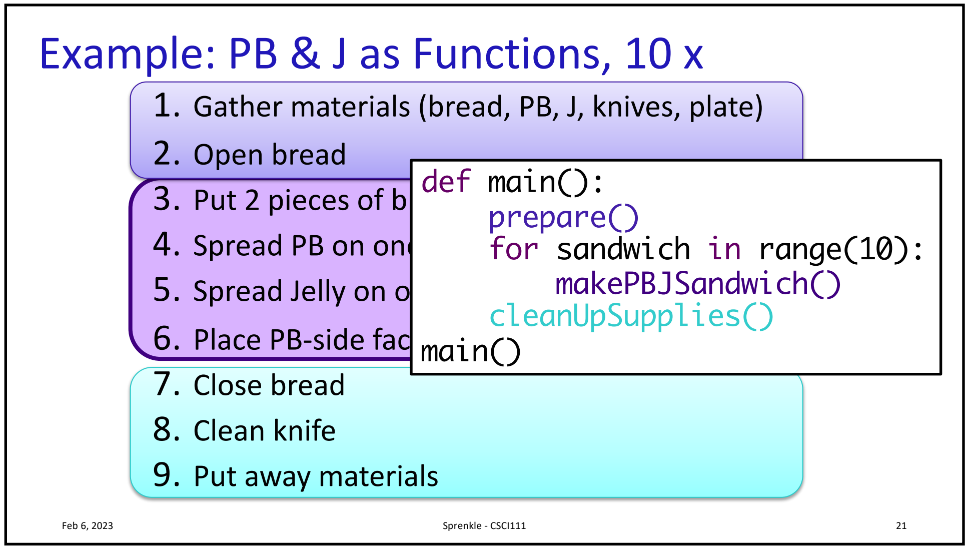
1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of bread on plate
4. Spread PB on one side of one slice
5. Spread Jelly on one side of other slice
6. Place PB-side facedown on Jelly-side of bread
7. Close bread
8. Clean knife
9. Put away materials



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

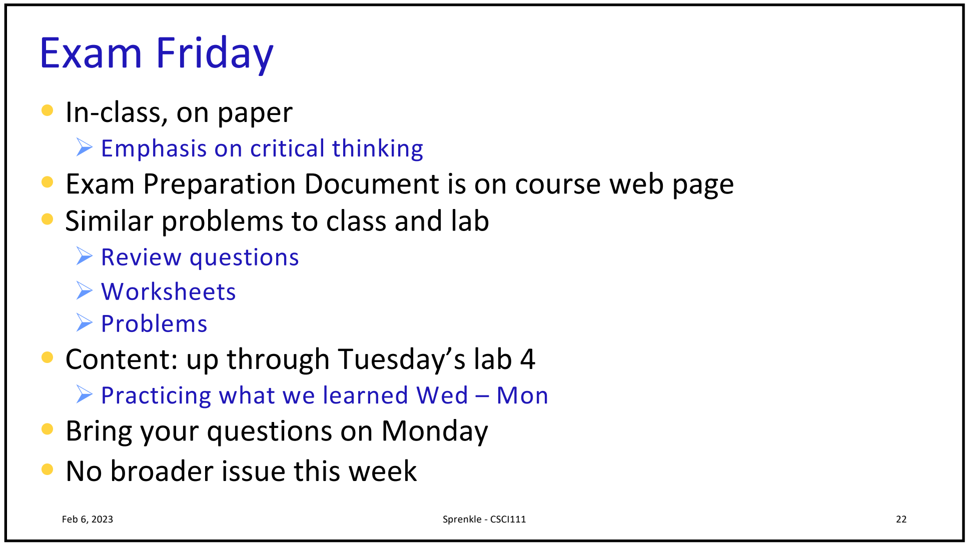
```
def main():
    prepare()
    makePBJSandwich()
    cleanUpSupplies()
main()
```



Example: PB & J as Functions, 10 x

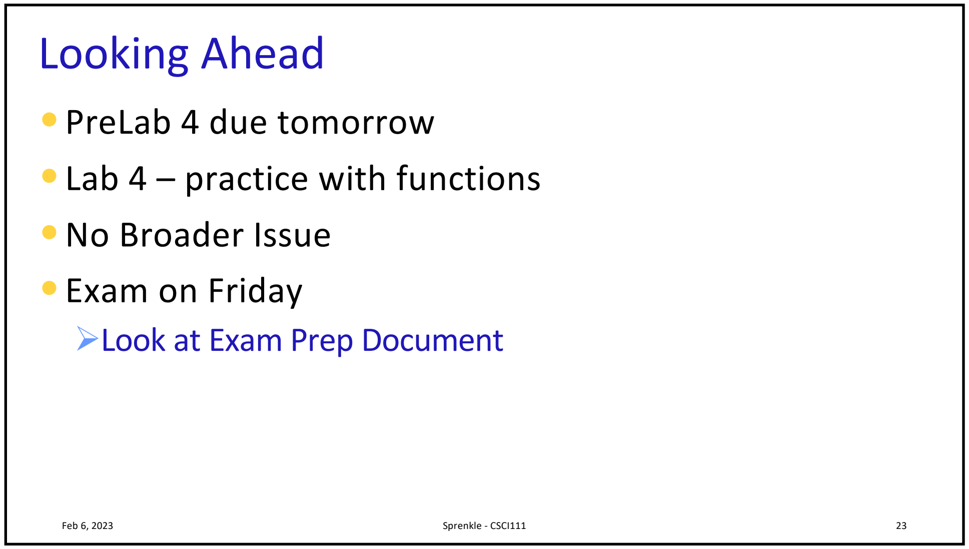
1. Gather materials (bread, PB, J, knives, plate)
2. Open bread
3. Put 2 pieces of b
4. Spread PB on one
5. Spread Jelly on o
6. Place PB-side fac
7. Close bread
8. Clean knife
9. Put away materials

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



Exam Friday

- In-class, on paper
  - Emphasis on critical thinking
- Exam Preparation Document is on course web page
- Similar problems to class and lab
  - Review questions
  - Worksheets
  - Problems
- Content: up through Tuesday’s lab 4
  - Practicing what we learned Wed – Mon
- Bring your questions on Monday
- No broader issue this week



Looking Ahead

- PreLab 4 due tomorrow
- Lab 4 – practice with functions
- No Broader Issue
- Exam on Friday
  - Look at Exam Prep Document