Objectives

- Refining our development process
- Passing parameters

Review

- With respect to functions, what are options for how we organize our program?
- How do we document a function? What should its content be?
- How do we define a test case?
    - How can we test functions easily?
    - What do we need to test functions?

Practice: Trace through the Program's Execution

- 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)

def square(n):
    return n * n

main()
```

practice1.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()
```

practice2.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()`

5

Review: Variable Scope

- Know "lifetime" of variable
  - Only during execution of function
  - Related to idea of "scope"
- Consider: how many functions probably use a variable like x or i? What would the impact be on our programs if all variables had global scope?
  - Example: round(x, n)
- In general, our only *global* variables will be constants because we don't want them to change value
  - e.g., EIEIO

6

Review: 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!

Review: `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

9

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

10

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

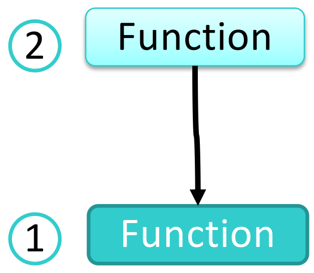
Evolving General Design Patterns

- 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



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`

15

Another development approach

REFACTORING

16

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()
```

Feb 7, 2022 Sprenkle - CSCI111 21

21

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, 2022 Sprenkle - CSCI111 22

22

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

Refactoring: Finding a Team's Winning Percentage

- Problem:
  - Prompt the user for a team's wins and losses and display the team's win percentage
- What code can we convert into a function?

`winpercent.py`

Refactoring: Finding a Team’s Winning Percentage

- Problem:
  - Prompt the user for a team’s wins and losses and display the team’s win percentage
- What code can we convert into a function?
  - Generalize: a success percentage
  - Calculates a success percentage, given the successes and failures

`winpercent.py`