Lab Trends

- Think time is increasing!
  - Not *that* much more code than, say, in Lab 4
  - BUT, **much** more ***think*** time
- Give yourself time to think
  - Write out comments if you're not ready for the code
    - I *still* do that in my code, especially when I'm feeling overwhelmed by a problem.
    - Commenting gives me an outline/todo list that I can then tackle more easily

Best Use of Student Assistants

While it's comforting to have a student assistant stand with you while you write your program, that is not an effective use of their time.

- Think (but don't overthink)
- Try to write some code
  - Find small chunks that you can solve.

No Wednesday evening hours
→ Ethiopia and Sarah Anne on Thursday

Lab 7 Feedback

- Lots of practice with functions
- Goal:
  - Good, descriptive names for parameters, functions
  - Descriptive comments
    - Others need to know *how to use* the function

Comment Example

```
def encodeLetter(char, key):
    """Encodes a single character.
    PRE: Input parameters are a single, lowercase
    character string (char) and an integer key
    (between -25 and 25, inclusive)
    POST: returns the encoded character"""
```

- Does not say *who* called function, where parameters came from, or where returned to
  - *Any* code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, …)
- Does not say variable name returned

Commenting Exercise

- Write a comment for this function:

```
def sumList(listOfNumbers):
```


Commenting Exercise

- Write a comment for this function:

```
def sumList(listOfNumbers):
    """Returns the sum of the values in a list,
    called listOfNumbers.
    PRE: listOfNumbers must be a list of numbers
    POST: returns the sum of the numbers"""
```

Function that you can use in the Olympic scores problem!

Commenting Notes

- Well-named parameters make documentation easier
- I'm not strict on the pre/post format.
- Just need to be clear on
  - what the function does (at high level)
  - types of parameters
  - type of the output

→ The caller knows what to pass to the function and if they should assign the output to a variable

Caesar Cipher w/Functions

```
def main():
   text = input("Enter some text: ")
   key = int(input("Enter an integer key (between -25 and 25): "))
    # make sure it's a valid key
    if key < -KEY_BOUND or key > KEY_BOUND:
        print("Invalid key!")
        sys.exit(1)

    message = encoder(text,key)
    print("The encoded message is", message)

def encoder(text,key):
    message=""
    for ch in text:
        if ch == " ":
            encode= " "
            message+=encode
        else:
            message += translateLetter(ch,key)
    return message
```

More efficient: constants *not* defined in function

Note: no "side effects" e.g., no printing

Showing example without reading in file → less code

Partial Gymnastics Code

```
def main():
    scores = getScoresFromFile(filename)
    avgDiffScore = scores.pop(0)
                                Returns and deletes first item in list
    avgExecScore = calculateAverageExecScore(scores)
    …

def calculateAverageExecScore(listOfScores):
    listOfScores.sort()
    totalExecScore = sumList(listOfScores[1:-1])
    average = totalExecScore/len(listOfScores)-2
    return average

…
```

For space, no comments, partial solution

Writing Encodings to the File

- With functions, writing to the file became simpler
  - Called a function that returned a string
  - Could write that string to a file
- Additional code was essentially
  - 1) open file for writing
  - 2) write the encoding to the file
  - 3) close the file

Review: Test Functions

- Review: What are our goals for a test function?
  - What are the benefits?
  - What are the alternatives?

```
def testBinaryToDecimal():
    paramInputs = ["0", "1", "10", "1001", "10000", "1101", "10110"]
    expectedResults = [0, 1, 2, 9, 16, 13, 22]
    for index in range(len(paramInputs)):
        paramInput = paramInputs[index]
        expectedResult = expectedResults[index]
        actualResult = binaryToDecimal(paramInput)
        if actualResult != expectedResult:
            print("**ERROR! **", paramInput, "should be",
expectedResult)
            print("Instead, got", actualResult)
        else:
            print("Success on binary to decimal conversion for",
paramInput, "-->", actualResult)
```


Test Functions

- Designing test function
  - Pick good test cases
  - Automatically (i.e., program) checks results so it's easy to spot problems
  - Report input/test cases that cause the problems
- Benefits:
  - Quickly and automatically test functions
  - Quickly add new test cases
  - Can rerun test cases quickly if function implementation changes
  - If tested well, you can use the function in other programs with confidence

DEFINING MODULES

Where are Functions Defined?

- Functions can go inside of program script
  - Defined before use/called (if no `main`() function)
  - Or, below the `main`() function (***preferred***)
- Functions can go inside a separate **module**

Creating Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
  - A module is named by its filename

Just a Python file!

- Example, `oldmac.py`
  - In Python shell: `import oldmac`
  - Explain what happened

Defining Constants in Modules

- Constant in `oldmac.py`
  - `EIEIO`

Creating Modules

- So that our program doesn't execute when it is **imported** in a program, at bottom, add

```
if __name__ == '__main__' :
    main()
```

Not important how this works; just know when to use

- Then, to call `main` function
  - `oldmac.main()`
- Note the sub-directories now listed in the directory

Creating Modules

- Then, to call `main` function
  - `oldmac.main()`
- Why would you want to call a module's `main` function?
  - Automation
  - Use `main` function as driver to test functions in module
- To access one of the defined constants
  - `oldmac.EIEIO`

Benefits of Defining Functions in Separate Module

- Reduces code in ***primary*** driver script
- Easier to reuse by importing from a module
- Maintains the "black box"
  - ***Abstraction***
- Isolates testing of function
- Write "test driver" scripts to test functions separately from use in script

Lab 8 Overview

- Creating and using a module
- Indefinite Loops