Lab 4 Feedback

- We need some work on functions
- Follow examples and instructions

Refactoring: Displaying Fibonacci Sequence

- What part of this code needs to go into the function that displays the first 20 Fib numbers?
- What is the input to the function?
- What is the output from the function?

```
print("Displays the first 20 Fib nums…")

prevNum2 = 0
prevNum = 1

print(prevNum2)
print(prevNum)

for i in range(18) :
   fibNum = prevNum + prevNum2
   print(fibNum)
   prevNum2 = prevNum
   prevNum = fibNum
```

Refactoring: Displaying Fibonacci Sequence

```
print("Displays the first 20 Fib nums…")

prevNum2 = 0
prevNum = 1

print(prevNum2)
print(prevNum)

for i in range(18) :
    fibNum = prevNum + prevNum2
    print(fibNum)
    prevNum2 = prevNum
    prevNum = fibNum
```

Unintended side effect

This will go into `main`

Code that displays the Fibonacci sequence

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """


    """
```

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """
    Pre: numInSequence must be an integer greater than 2
    Post: returns the numInSequence value
          in the Fibonacci sequence
    """
```

Does not mention user input – does not require user input.

Doc String for Fibonacci Sequence Function

- How should we describe this function?
  - What is a good precondition for the function?
    - What info does a good precondition include?

```
def generateFibonacciNumber(numInSequence):
    """
    Pre: numInSequence must be an integer greater than 2
    Post: returns the numInSequence value
          in the Fibonacci sequence
    """
```

Does not mention user input – does not require user input.

```
for x in range( 3, 10, 2):
    print( generateFibonacciNumber(x) )
```

Molecular Weight

- Given a non-negative integer of hydrogen, oxygen, carbon atoms, return the molecular weight

```
def calcMolecularWeight( hAtoms, oAtoms, mAtoms ):
    ... # calculation ...

    return weight
```

Rounding should not be done in here
→ Reduces the reusability of the function

Molecular Weight

- Given a non-negative integer of hydrogen, oxygen, carbon atoms, return the molecular weight

```
def main():
    # get user input …
    weight = calcMolecularWeight(...)
    print("The weight is", round(weight, 6))
```

Would still only round to 3 places if rounding performed in function

Testing

- Testing this way won't always be possible, but it works well in certain situations:

```
def testCalculateMolecularWeight():
    test.testEqual(calculateMolecularWeight(1,1,1),
                   H_WEIGHT + C_WEIGHT + O_WEIGHT)
    test.testEqual(calculateMolecularWeight(0,0,1),
                   O_WEIGHT)
    …
```

General Reminders

- Read instructions carefully
  - Example: **Write a test function** that tests that your function works correctly. After you have verified that your tests work, **comment out the *call* to your test function**. Now, modify the **`main`** function to prompt a user for which Fibonacci number they want and then **display that Fibonacci number**.
  - Example 2: After verifying that your function works, create a main function. Your program should prompt the user for the number of atoms of each type and **display** the total weight with the appropriate units, **rounded** to 3 decimal places.
- Review example programs on the course web site

Review

- How can we make our code make [good] decisions?
  - What variations are available to us?

More Complex Conditions

- Boolean
  - Two logical values: True and False
- Combine conditions with Boolean operators
  - **and** – `True` only if **both** operands are `True`
  - **or** – `True` if **at least one** operand is `True`
  - **not** – `True` if the operand is not `True`
- English examples
  - If it is raining **and** it is cold
  - If it is Saturday **or** it is Sunday
  - If the shirt is on sale **or** the shirt is purple

What is the output?

Focus: how operations work
Not good variable names

```
x = 2
y = 3
z = 4

b = x==2
c = not b
d = (y<4) and (z<3)
print("d=",d)
d = (y<4) or (z<3)
print("d=",d)
d = not(y<4 or z<3)
print("d=",d)
d = not d
print(b, c, d)
```

Because of precedence, we don't *need* parentheses

eval_cond.py

Truth Tables

operands

A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T						
T	F						
F	T						
F	F						

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T				
T	F	F	T				
F	T	F	T				
F	F	F	F				

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T	F	F		
T	F	F	T	F	T		
F	T	F	T	T	F		
F	F	F	F	T	T		

Truth Tables

operands

A	B	A and B	A or B	not A	notB	not A and B	A or not B
T	T	T	T	F	F	F	T
T	F	F	T	F	T	F	T
F	T	F	T	T	F	T	F
F	F	F	F	T	T	F	T

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - In Python, we can't (always) write a condition like 0 <= num_grade <= 100, so we need to break it into two conditions
- Write an appropriate condition for this check on the numeric grade
  - Using **and**
  - Using **or**

Focus on the ***condition***
Then, we'll block out the code

Practice: Numeric Grade Input Range

- Enforce that user must input a numeric grade between 0 and 100
  - Using **and**

```
if num_grade >= 0 and num_grade <= 100:
    computation
else:
    print error message
```

  - Using **or**

```
if num_grade < 0 or num_grade > 100:
    print error message
else:
    computation
```

Lab 5 Overview

- “only” two non-exam class periods since last lab, so…
- Focus on conditionals
  - Table functions for a week
- More building blocks to draw from
  - More tests we can “handle nicely”
  - Break problem into smaller pieces
  - Think, write your algorithm outline, write a few lines of code, then try them out.