

Lab 5

- Review Lab 4
- Prepare for Lab 5

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

This should go into main

```
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
```

If in function,
it is an unintended side effect

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 explain how to call the function and what function does
Does **not** mention user input – does not require user input.
Does **not** mention where called *from* (could be called from anywhere)

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) )
```

Giving Parameters Default Values

- Can assign a default value to parameters
- We've seen this with other functions
 - Example: range has a default start of 0 and step of 1 when called as range(stop)

```
def rollDie(sides=6):  
    """  
    Given the number of sides on the die (a positive integer),  
    simulates rolling a die by returning the rolled value,  
    between 1 and sides, inclusive.  
    If no parameter passed, the number of sides defaults to 6.  
    """  
    return randint(1, sides)
```

Finding Areas of Shapes


- Given a non-negative radius and height, returns the area of a cylinder

```
def calculateCylinderArea( radius, height ):
    ... # calculation ...
    return area
```

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

Function in main

```
def main():  
    # get user input ...  
    area = calculateCylinderArea(...)  
    print("The area is", round(area, 2))
```



If rounding already performed
in function, might be rounded
more than we want

Discussion

- Why do we need to test/run our program multiple times if we already tested our function programmatically?

Discussion

- Why do we need to test/run our program multiple times if we already tested our function programmatically?

Need to test the user interface too!

➤ More tests → more bugs found

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**.

```
def testGenerateFibonacciNumber():
    test.testEqual( generateFibonacciNumber(2), 1 )
    test.testEqual( generateFibonacciNumber(3), 2 )
    test.testEqual( generateFibonacciNumber(6), 5 )
    test.testEqual( generateFibonacciNumber(20), 4181 )

# testGenerateFibonacciNumber()
main()
```

- Review example programs on the course web site

Review

- How can we make our code make [good] decisions?
 - What variations are available to us?
 - What are they good for?
- From text book reading: What are the Logical/Boolean operators?
 - How do they work?

Review: 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

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

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

Truth Tables

operands

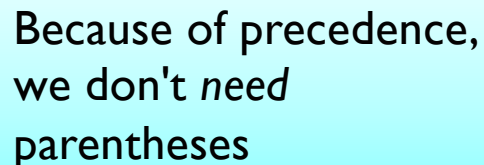
A	B	A and B	A or B	not A	not B	not A and B	A or not B
T	T	T	T	F	F	F	T
T	F	F	T	F	T	F	T
F	T	F	T	T	F	T	F
F	F	F	F	T	T	F	T

What is the output?

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

Focus: how operations work
Not good variable names

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



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

```
d = not d  
print(b, c, d)
```

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 $0 \leq \text{num_grade} \leq 100$, 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
```

Short-circuit Evaluation

- Don't necessarily need to evaluate all expressions in a compound expression
- A **and** B
 - If A is **False**, compound expression is **False**
- A **or** B
 - If A is **True**, compound expression is **True**
- No need to evaluate B
 - Put more important/limiting expression first
 - Example:

```
if count != 0 and sum/count > 10:  
    do something
```


Testing the Game Functions

```
def testRollMultipleDice():
    numTests = 0
    numSuccesses = 0
    for numDie in range(1, 5):
        for sides in range(1, 13):
            numTests += 1
            roll = rollMultipleDice( numDie, sides)
            if roll < numDie or roll > numDie * sides:
                print("Error rolling", numDie, "dice with", sides,
                    "sides. Got", roll)
            else:
                numSuccesses += 1
    print("Test passed", numSuccesses, "out of", numTests,
```

Now you know what this does!

- Why could I write a test of your function?
 - Emphasizing **abstraction**
 - The code I wrote has **no** knowledge of your code, e.g., your variable names
 - Only knows what the code *should* return

Determining Multiples

Original Code:
Emphasized mutually
exclusive behavior

```
if x % 2 == 0 :  
    print(x, "is a multiple of 2")  
elif x % 3 == 0 :  
    print(x, "is a multiple of 3")  
else :  
    print(x, "is not a multiple of 2 or 3")
```

Implementation of the likely
expected behavior:

```
isNotDivisibleBy2Or3 = True  
  
if x % 2 == 0:  
    print(x, "is a multiple of 2")  
    isNotDivisibleBy2Or3 = False  
  
if x % 3 == 0:  
    print(x, "is a multiple of 3")  
    isNotDivisibleBy2Or3 = False  
  
if isNotDivisibleBy2Or3:  
    print(x, "is not a multiple of 2 or 3")
```

Compare control flow diagrams

Determining Multiples

Original Code:
Emphasized mutually
exclusive behavior

```
if x % 2 == 0 :  
    print(x, "is a multiple of 2")  
elif x % 3 == 0 :  
    print(x, "is a multiple of 3")  
else :  
    print(x, "is not a multiple of 2 or 3")
```

What statements execute
when x is 4, 5, 6, and 9?

```
isNotDivisibleBy2Or3 = True
```

- 1 `if x % 2 == 0:`
- 2 `print(x, "is a multiple of 2")`
`isNotDivisibleBy2Or3 = False`
- 3 `if x % 3 == 0:`
- 4 `print(x, "is a multiple of 3")`
`isNotDivisibleBy2Or3 = False`
- 5 `if isNotDivisibleBy2Or3:`
- 6 `print(x, "is not a multiple of 2 or 3")`

Lab 5 Overview

- Focus on conditionals
 - **Functions only in last problem**
- More building blocks to draw from
 - More use cases we can “handle nicely”
 - More tests for you to think of/write/pass!
 - Think about if you’ve covered all execution paths
 - Break problems into smaller pieces
 - Think, write your algorithm outline, write a few lines of code, then try them out.