Review

Class Exercise

Functions Returning Values

Exercises

A Word on Testing

Exercise on Robustness
Programs Developed in Class...

<table>
<thead>
<tr>
<th>Topics</th>
<th>Lab/Hw</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monday:</strong> Video class today. The class is covered in 3 different videos. Click on the video icons below, from left to right, please!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lecture Notes</td>
<td>Lab 4</td>
<td>Read Chapter 6 in Zelle</td>
</tr>
<tr>
<td><strong>Wednesday:</strong> Review &amp; Q&amp;A, Strings are immutable, String methods, On-line documentation: docs.python.org, Exercises</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs developed in class</td>
<td>Homework 4, Solution programs can be found at the end of the lab and homework pages</td>
<td></td>
</tr>
<tr>
<td>Lecture Notes</td>
<td><strong>Friday:</strong> Split(), Functions, Lecture Notes</td>
<td></td>
</tr>
<tr>
<td>Programs developed in class</td>
<td>A video with some words about real numbers and why they behave strangely:</td>
<td>Lecture Notes</td>
</tr>
</tbody>
</table>

D. Thiebaut, Computer Science, Smith College
Class Example

Worker 0:
• clap!

Worker 1:
• get a number
• \textit{return} that number multiplied by 2, and then 1 added to the product.

Worker 2:
• get a number
• \textit{return} that number modulo 10
Class Example

Worker 3:
- get a number
- compute a new number equal to the number received multiplied by 2 and incremented by 1
- return the new number modulo 10
# Functions.py
# Example of functions returning values.

```python
def worker1( num1 ):
    num2 = num1 * 2 + 1
    return num2

def worker2( num1 ):
    num2 = num1 % 10
    return num2

def worker3( num1 ):
    num2 = worker1( num1 )
    num3 = worker2( num2 )
    return num3

def main():
    n = 12
    print("n = ", n, " worker1(n) = ", worker1( n ))
    print("n = ", n, " worker2(n) = ", worker2( n ))
    print("n = ", n, " worker3(n) = ", worker3( n ))

main()
```

Python Version

```python
# Functions.py
# Example of functions returning values.

def worker1( num1 ):
    num2 = num1 * 2 + 1
    return num2

def worker2( num1 ):
    num2 = num1 % 10
    return num2

def worker3( num1 ):
    num2 = worker1( num1 )
    num3 = worker2( num2 )
    return num3

def main():
    n = 12
    print("n = ", n, " worker1(n) = ", worker1( n ) )
    print("n = ", n, " worker2(n) = ", worker2( n ) )
    print("n = ", n, " worker3(n) = ", worker3( n ) )

main()
```

```python
>>> n = 12  worker1(n) = 25
n = 12  worker2(n) = 2
n = 12  worker3(n) = 5
```
Return and Print are very different statements!
Review

Class Exercise

Functions Returning Values

Exercises

A Word on Testing

Exercise on Robustness
Functions Boot Camp

Exercise 1:
• Write a function that gets a temperature in Celsius as a parameter and returns the equivalent Fahrenheit.

\[ F = C \times \frac{9}{5} + 32 \]

• Test the function from the Python shell
Exercise 2:
• Write a function that is given a string in the form dd MMM yyyy, and that **returns** it as mmddyyyy

*Example:* 7 Oct 2015  —>  10072015
Formatting Trick: Leading Zeros

```python
for n in [0, 1, 5, 10, 20, 99]:
    print( '{:02}'.format( n ) )
```

00 01 05 10 20 99
Happy Mountain Day!
Line up your Minions!

Testing

Passing Parameters, Returning Values

Exercises
Review
Line-up Your Minions

GOOD
Line-up Your Minions
def process(  ):  
  
    def fun1(  ):  
        x = input( "amount? " )  
        return x  

    def fun2( a, b ):  
        x = a // b  
        y = a % b  
        return x, y  

    amount = fun1()  
    No12, left = fun2( amount, 12 )  
    print( No12 )  

    def main():  
        print( "welcome" )  

        process()  

    main()
```python
def process( ):
    def fun1( ):
        x = input( "amount? " )
        return x

    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y

    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main( ):
    print( "welcome" )

    process()

main()
```
def process(  ):
    def fun1(  ):
        x = input( "amount? " )
        return x
    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )
    process()

main()
def process( ):
  
def fun1( ):
    x = input( "amount? " )
    return x
  
def fun2( a, b ):
    x = a // b
    y = a % b
    return x, y
  
amount = fun1()
No12, left = fun2( amount, 12 )
print( No12 )

def main():
  print( "welcome" )
  
  process()

main()
def process():
    def fun1():
        x = input( "amount? " )
        return x
    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )
    process()
    x = fun1()

main()
```python
def process():
    def fun1():
        x = input( "amount? " )
        return x
    def fun2( a, b ):
        x = a // b
        y = a % b
        return x, y
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )
    process()
    x = fun1()

main()
```

**Example**

**Why is it bad?**

Scope of `process()`

**ERROR!**
def process(  ):  
    def fun1(  ):  
        x = input( "amount? " )
        return x
    def fun2( a, b ):  
        x = a // b
        y = a % b
        return x, y
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():  
    print( "welcome" )
    process()
    x = fun1()
main()
```python
def fun1( ):
    x = input( "amount? " )
    return x

def fun2( a, b ):
    x = a // b
    y = a % b
    return x, y

def process( ):
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()
    x = fun1()

main()
```

The Right Way
def fun1( ):
    x = input( "amount? " )
    return x

def fun2( a, b ):
    x = a // b
    y = a % b
    return x, y

def process( ):
    amount = fun1()
    No12, left = fun2( amount, 12 )
    print( No12 )

def main():
    print( "welcome" )

    process()
    x = fun1()

main()
Line up your Minions!

Testing

Passing Parameters, Returning Values

Exercises
A Few Words On Testing…
Test #1
Test #1
Test #2
Test #1
Test #2
Test #3
Test #1
Test #2
Test #3
. .
. .
Test #n
Robustness
• **Inputting numbers:**
  - Test special values: 0, 1, -1, negative, very large.

• **Inputting lists:**
  - Test empty lists: [], or ()

• **Inputting strings:**
  - Mix upper- & lower-case: "HEllo"
  - Empty string: ""
  - Add spaces: " Hello Kitty! "
Line up your Minions!

Testing

Passing Parameters, Returning Values

Exercises
Calling Functions
Calling Functions

Do some work
Calling Functions

Return value(s)
def minion(a, b):
    x = a // b
    y = a % b
    return y

def Gru():
    x = minion(5, 3)
    print(x)

Gru()
def minion( a, b ):
    x = a // b
    y = a % b
    return y

def Gru():
    x = minion( 5, 3)
    print( x )

Gru()

Do some work
def minion(a, b):
    x = a // b
    y = a % b
    return y

def Gru():
    x = minion(5, 3)
    print(x)

Gru()
Functions Boot Camp

Exercise 1:
• Write a function that gets a sentence from the user and returns a list of words, all capitalized. Empty words "" are not allowed in the list.

Exercise 2:
• Write a function that gets a string of DNA symbols, and switches all the 'A's for 'T's, and all the 'C's for 'G's. The function returns the "flipped" string.
Functions Boot Camp

**Exercise 3:**
- Revisit the Teller Machine program
  - With one function that breaks down the amount in number of $-xx and left over
  - With one function that returns ALL the number of $-xx bills

**Exercise 4:**
- Get a list of numbers on one line
- Some numbers can be 0
- Print the average of all the numbers that are not 0.