Announcements

- **Reading**
  - Read Chapter 4 (functions)

- **Project1**
  - Will be posted Friday, September 7
  - Due Saturday, September 15th 11:58pm
  - We will post instructions on how to turn in from home
  - Note: you can always turn in from the lab machines

- **Codelab:**
  - Sign up for CodeLab; instructions on home page
  - Practice, practice, practice
Python Boot Camp Recap

- Variables name values: \texttt{X\_12, \_par, greeting, \ldots}
- All variables can be assigned: \texttt{x = 15}
- All variables can be “read”: \texttt{y = x}

- Values have types: \texttt{int, float, str, list, \ldots}
- The type determines what additional operations can be performed on the values, and what they mean
- Some values can be converted to another type: \texttt{chr(66), ord(“B”), float(5), int(3.2)}
Python Boot Camp Recap

- Functions name programs
  - Functions can be defined:
    ```python
def myFunction(<parameter names>):
    function body (indented)
```
  - Functions can be called:
    ```python
    myFunction(<parameter values>)
    ```
- The number of arguments is fixed; could be zero
- The function returns a value; could be `None`
Function Output

- The return statement specifies what value the function returns to the call site:
  
  `return <expression>`

- Return Statement
  - Passing a value out of a function (output)
  - Last statement executed
  - Multiple conditional exit possible
Function Call as Substitution

- When a function is called
  - The argument values are bound (assigned) to the parameter names
  - The program of the function is executed as if those program statements were substituted for the call
  - Upon execution of the return statement, the return value is substituted for the function call
  - If no return statement is executed, and the program of the function body ends, the function returns \texttt{None}, indicating that no value is returned to the calling place.
Example

x = 12
y = 4

def mySum(a,b):
    u = a+b
    x = 1
    return u

print(x,y)
print(mySum(x,y))
print(x,y)

Unless explicitly declared global, x is a local variable
More on Returns

- Returns are the last statement executed by a function
  - The function is considered to be “done” at this point
- There may be other statements “after” the return, but these are not executed (but see conditionals later-on

```python
def sumOfTwo(a,b):
    return a+b
    print(a+b)
    print(a+b)
    print(sumOfTwo(4,5))
```
Stringing together functions

- We can build up complex expressions from functions

```python
def sumOfTwo(a,b):
    return a+b
print(sumOfTwo(sumOfTwo(4,5), sumOfTwo(1,2)))
```

OR

```python
a = sumOfTwo(4,5)
b = sumOfTwo(1,2)
c = sumOfTwo(a,b)
print(c)
```
A “Sample” Program

def task1(a,b):
    ...
    y = ....
    return y
def task2(a,b,c):
    ...
    z = ...
    return z
def main():
    input1, input2 = eval(input(....))
    output1 = task1(input1, input2)
    output2 = task2(input1, input2, output1)
    print(output2)
Local Variables

- Function parameter names act as *local variables*:
  - They are available as variables for the execution of the function body
  - They get their initial value from the values at the call site (value binding)
  - When the function ends, they are no longer available

- You can define additional variables in the function body
  - They will be local variables by default
  - When the function ends, they no longer exist
Local Variables

a = 3
y = 10
def myFun(a):
    print(a)
y = 1
myFun(4)
print(a)
print(y)
Clicker Question: does this program print 3 or 4?

```python
x = 3
def myFun():
    print (x)
x = 4
myFun()
```

A: 3
B: 4
Clicker Question:
Are these programs equivalent?

A: yes  
B: no
Variables and Functions

- Variables defined outside of the function are *global* variables
  - Global variables can be changed inside a function body
  - Using global variables inside functions is dangerous:
    - Multiple calls to a function may yield different results if the program “rebinds” such variables
- At minimum, declare global variables as *global* in the function body and document their use.
Variables and Functions

x = 3
def myFun():
    print (x)
x = 4
myFun()
x = 5
myFun()
You must be careful!

x = 3
def myFun():
    print (x)
    x = 1
x = 4
myFun()  # ERROR!
x = 5
myFun()  # ERROR!
Global Variables

- How can we get the example code we saw earlier to work?
  - Python is not sure if we want $x$ to be a local variable or if it should refer to the $x$ we defined outside of the function
  - We can inform python if we want $x$ to refer to the variable outside of the function
    - New keyword `global`
This works!

```python
x = 3
def myFun():
    global x
    print (x)
    x = 1
x = 4
myFun()
myFun()
```
Global or Local?

- If the global keyword is used, the variable is global
- If the first use of the variable is a ‘read’ (reference), the variable is global
  - NOTE: We cannot assign to such a variable later
  - Function arguments are always local
- If the first use of the variable is a write (assignment), the variable is local
  - Unless the variable is defined global
Clicker Question: Is x global or local?

```
x = 3
def myFun():
    y = 4
    z = x + y
myFun()
```

A: global

B: local
Let’s Review

- Functions take input and produce output
- Output is provided by the “return” statement
  - Otherwise the function does not provide output
- At the call site of the function the arguments get bound
  - The arguments can rebind variables that have already been defined for the duration of the call
- You can use global variables, defined outside the function, but you must be careful!
Advice

- Unless absolutely necessary avoid naming parameters and local variables the same as global variables
Argument Typing

- Functions typically assume something important about the arguments

```python
def sumOfTwo(a,b):
    return a+b
```

- Will this work no matter what we provide as arguments?
Function Arguments

- Consider the following three cases:

  \[ \text{res} = \text{sumOfTwo}(1, 2) \]
  \[ \text{res} = \text{sumOfTwo}(\text{"Hello", "World"}) \]
  \[ \text{res} = \text{sumOfTwo}(\text{"Hello", 1}) \]

- One of these cases will throw an error. This behavior is defined by the code inside the function.
Function Arguments

- There are two ways to handle this difficulty
  1. Tell everyone what the function expects
  2. Include checks inside the function to ensure the arguments are what is expected

- A combination of both techniques should be used
Function Documentation

- This solution uses comments and if-statements.
- We will revisit this in later slides

```python
# This function expects two integers
# and returns -1 otherwise

def sumOfTwo(a,b):
    if type(a) == int and type(b) == int :
        return a+b
    return -1
```
Announcements

- Reading
  - Read Chapter 4 (functions)
  - Start on reading Chapter 5 (conditionals)
- Project 1
  - Will be posted Friday, September 7
  - Due Saturday, September 15th 11:58pm
  - We will post instructions on how to turn in from home
  - Note: you can always turn in from the lab machines
- Codelab:
  - Sign up for CodeLab; instructions on home page
  - Practice, practice, practice
Functions that Modify Parameters

- Suppose you are writing a program that manages bank accounts. One function we would need to do is to accumulate interest on the account. Let’s look at a first-cut at the function.

- def addInterest(balance, rate):
  
  newBalance = balance * (1 + rate)
  
  balance = newBalance
Functions that Modify Parameters

- The intent is to set the balance of the account to a new value that includes the interest amount.

- Let’s write a main program to test this:
  ```python
def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
  ```
Functions that Modify Parameters

- We hope that the 5% will be added to the amount, returning 1050.
- >>> test()
  1000
- What went wrong? Nothing!
The first two lines of the test function create two local variables called amount and rate which are given the initial values of 1000 and 0.05, respectively.

def addInterest(balance, rate):
    newBal = balance*(1+rate)
    balance = newBal

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)

    test()
Functions that Modify Parameters

- The first two lines of the test function create two local variables called amount and rate which are given the initial values of 1000 and 0.05, respectively.

```python
def addInterest(balance, rate):
    newBal = balance*(1+rate)
    balance = newBal

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)

test()
```
Functions that Modify Parameters

def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)

    test()
Functions that Modify Parameters

def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance

def test():
    amount = 1000
    rate = 0.05
    amount = addInterest(amount, rate)
    print(amount)

test()
Functions that Modify Parameters

- Why is the following solution inferior?

```python
amount = 1000
rate = 0.05
def addInterest():
    global amount
    global rate
    amount = amount * (1 + rate)
addInterest()
pdint(amount)
```
Project 1: The Angry Birds Lost

- Found at homepage >> projects
- To be posted by tomorrow, Sep 7 (Friday)
- Due Sep. 15, just before midnight
  - Submit early in case there is a problem
Project 1

The angry birds tried their level best but in the end lost. Now the pigs are celebrating by going on a cruise, and their favorite entertainment is to jump into the pool. Realizing your valuable help, the pigs retain you as consultant and have you calculate the necessary speed to hit the pool squarely in the middle…
First scenario: The pigs can’t jump up, so they run off a horizontal stage for their splash-down.
Scenario 2: The pigs gained a lot of weight, so the stage collapsed. Lucky for the pigs, it can still be used as a ramp, and they need the exercise, but the calculation is a bit more complex…
Boolean Values

- True, False
- Result of comparison, e.g. $x < y$
- Boolean operations:
  - And: True and False $\equiv$ False
    $(x \geq 5) \text{ and } (x \leq 10)$
  - Or: True or False $\equiv$ True
    $(x < 5) \text{ or } (x > 10)$
  - Not: not True $\equiv$ False
    not $(3 == 4)$
Precedence

Clicker Question

- Consider $a*b+c*d$; which of the three is it equal to?

  A. $(a*b) + (c*d)$
  B. $a * (b+c) * d$
  C. $((a * b) + c) * d$
Precedence

- Multiplication and division take precedence over addition and subtraction.

- Consider $a*b+c*d$; which of the three is it equal to?
  A. $(a*b) + (c*d)$
  B. $a * (b+c) * d$
  C. $((a * b) + c) * d$

- What about comparison ($<, <=, ==, >=, >$) vs. Boolean ops (and, or, not)?

- Comparisons take precedence …
IF Statement

- For the conditional expression <exp>, evaluating to True or False, the simple IF statement is

```python
if <exp>:
    <body>
```

```
x = 7
if x > 10:
    print x
x = x + 1
```

```
x = True
if x:
    print x
```
if x > 10:
    print x
else:
    x = x + 1
x = 7

End
Conditionals

- It is also possible to specify what to do if the condition is **False**. Contrast these two examples.

```python
x = 7
if x > 10:
    print x
x = x + 1
```

```python
x = 7
if x > 10:
    print x
else:
    x = x + 1
```
CQ: Are these programs equivalent?

A: yes
B: no
x = 7
if x > 10:
    print x
x = x + 1
if $x > 10$:
    print $x$
else:
    $x = x + 1$
Homework

- Work on Project 1
- Do the pre lab
- Review chapter 4 (functions)
- Start on chapter 5 (conditionals)