CSCI 141
Computer Programming I

Filip Jagodzinski
From Last Time

```python
def printTheWordHello():
    print("Hello")

def generatePersonalizedGreetings(aName, someNumber):
    for x in range(0, someNumber):
        print("Hello", aName)

printTheWordHello()
generatePersonalizedGreetings("Filip", 4)
```

Task: Explain the above code

- Identify the functions
- Identify the variables
- How are the functions invoked?
- What is the output of the program?
From Last Time

A function might be defined as requiring **no parameters**, or more than 1 parameter

```python
def printTheWordHello():
    print("Hello")

def generatePersonalizedGreetings(aName, someNumber):
    for x in range(0, someNumber):
        print("Hello", aName)

printTheWordHello()
generatePersonalizedGreetings("Filip", 4)
```
When invoking a function, you write its name, and provide the correct number of arguments, and in the correct order.
From Last Time

```python
def printTheWordHello():
    print("Hello")

def generatePersonalizedGreetings(aName, someNumber):
    for x in range(0, someNumber):
        print("Hello", aName)

printTheWordHello()
generatePersonalizedGreetings("Filip", 4)
```

We say that this is an invocation of the function `printTheWordHello`.

Or you may say that this is a function call.

Whichever term you use, it means making that function run.
From Last Time

```python
def printTheWordHello():
    print("Hello")

def generatePersonalizedGreetings(aName, someNumber):
    for x in range(0, someNumber):
        print("Hello", aName)

printTheWordHello()
generatePersonalizedGreetings("Filip", 4)
```

Q: What is the difference between a parameter and an argument?

Q: Which of the above is an argument, and what of the above is a parameter?
From Last Time

```python
def printTheWordHello():
    print("Hello")

def generatePersonalizedGreetings(aName, someNumber):
    for x in range(0, someNumber):
        print("Hello", aName)

printTheWordHello()
generatePersonalizedGreetings("Filip", 4)
```

**Argument**: the value that is provided as input to a function

**Parameter**: in the declaration; providing the name(s) of the variable(s) that will refer to the inputs that the function receives
From Last Time

```python
def computeFourthPower(number):
    result = number ** 4
    return result

anOutput = computeFourthPower(23)
print("The result is", anOutput)
```
From Last Time

Functions can return values. The keyword `return` specifies that behavior.

```python
def computeFourthPower(number):
    result = number ** 4
    return result

anOutput = computeFourthPower(23)
print("The result is", anOutput)
```

Variables declared within a function (either declared as parameters, or declared and defined in the body of a function), are local, and have local scope.

EACH time that a function is invoked, a new set of local variables are created. Each time that a function ends, those variables are destroyed and cannot be used elsewhere.
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

(This is a sample of the type of question you can expect on the midterm)
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

First, make sure you understand each part
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
    aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

First, make sure you understand each part

The function `updateValue` receives a single parameter, declares a local variable `done` of type Boolean, includes a `while` loop that includes an `if` statement, and which returns the value of another local variable `aNumber`. 
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done :
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The python interpreter begins by “reading” and “executing” the file from top, proceeding down (sequential code execution)
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The function `updateValue` is declared. After this occurs, you can invoke this function in “code below”. When the function is declared, python does not execute the function.
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The variable `myFavoriteNumber` is declared and assigned the value 23

```
myFavoriteNumber = 23
```
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The assignment statement FIRST performs all calculations on the right hand side, thus the function `updateValue` is invoked with argument 23

```plaintext
myFavoriteNumber = 23
```
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The parameter (local variable) `aNumber` is declared, and assigned the value 23

```
myFavoriteNumber = 23     aNumber = 23
```

Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done :
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)

The local variable done is declared and assigned the value False

myFavoriteNumber = 23    aNumber = 23    done = False
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

While `done` is False ... Execute the body of the while loop
Warmup

Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)

Is \texttt{aNumber} modulo 8 equal to 0?

myFavoriteNumber = 23 \quad aNumber = 23 \quad done = False
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

No, therefore decrement `aNumber` by 1, which is the “end” of the body of the while loop, after which python goes to the “top” of the while body’s code block and continue as before …

```
myFavoriteNumber = 23  aNumber = 22  done = False
```
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

Is `aNumber` modulo 8 equal to 0?

```python
myFavoriteNumber = 23  # aNumber = 22  # done = False
```
Warmup

Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)

No ... thus, the while loop would iterate over and over again until ...
Q: Decrement aNumber ... until ... at which point does aNumber modulus 8 would equal 0?

myFavoriteNumber = 23  aNumber = 22  done = False
Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

At the next iteration of the while loop, does `aNumber` modulo 8 equal 0?

```python
myFavoriteNumber = 23  aNumber = 16  done = False
```
Warmup

Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)

Yes, therefore execute the body of the if statement, which updates done to True

myFavoriteNumber = 23  aNumber = 16  done = True
Q: What is the output of this program?

```
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The remainder of the current iteration of the while loop’s body completes, which decrements aNumber by 1
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done :
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

When the check of the next iteration of the while loop is performed, not done evaluates to False, so the while loop terminates.

myFavoriteNumber = 23  aNumber = 15  done = True
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The execution of the body of the function continues ...

- `aNumber` is decremented once more

```
myFavoriteNumber = 23   aNumber = 14   done = True
```
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The function returns the value of the variable `aNumber`

```
myFavoriteNumber = 23  aNumber = 14  done = True
```
Warmup

Q: What is the output of this program?

def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)

Thus this evaluates to 14

myFavoriteNumber = 23  aNumber = 14  done = True
Warmup

Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done:
        if aNumber % 8 == 0:
            done = True
        aNumber = aNumber - 1
    aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The value of the variable `myFavoriteNumber` is set to 14

myFavoriteNumber = 14  aNumber = 14  done = True
Q: What is the output of this program?

```python
def updateValue(aNumber):
    done = False
    while not done :
        if aNumber % 8 == 0:
            done = True
            aNumber = aNumber - 1
        aNumber = aNumber - 1
    return aNumber

myFavoriteNumber = 23
myFavoriteNumber = updateValue(myFavoriteNumber)
print("My new favorite number", myFavoriteNumber)
```

The output is thus:

My new favorite number 14

myFavoriteNumber = 14  aNumber = 14  done = True
Today

Functions with default parameters
Global variables
Accumulators
Functions that call other functions
The main function
Functions with default parameters

Recall that we’ve seen the following:

\[
\text{range}(0, 10) \\
\text{range}(0, 10, 2)
\]

Q: How would the range function declaration look like?
Q: How can we define a function to receive either two or three (or some other combination of) inputs?
Functions with default parameters

```python
def performCalc(aNumber, aSecondNumber, aThirdNumber = 22):
    calculation = aNumber + aSecondNumber + aThirdNumber
    return calculation

aNewNumber = performCalc(1,2)
print(aNewNumber)
aNewNumber = performCalc(1,2,3)
print(aNewNumber)
```

Q: What do you see that’s odd?
Q: What is the output of this program?
Functions with default parameters

```python
def performCalc(aNumber, aSecondNumber, aThirdNumber = 22):
    calculation = aNumber + aSecondNumber + aThirdNumber
    return calculation
```

```python
aNewNumber = performCalc(1, 2)
print(aNewNumber)
aNewNumber = performCalc(1, 2, 3)
print(aNewNumber)
```

There is an assignment in the declaration of the function `performCalc`

This means that IF the third argument is not provided when the function is invoked, the default value of the local variable `aThirdNumber` will be 22
Functions with default parameters

```python
def performCalc(aNumber, aSecondNumber, aThirdNumber = 22):
    calculation = aNumber + aSecondNumber + aThirdNumber
    return calculation

aNewNumber = performCalc(1, 2)
print(aNewNumber)
aNewNumber = performCalc(1, 2, 3)
print(aNewNumber)
```

These 2 invocations therefore are invoking the same function, but the first invocation does not explicitly provide an argument to the last parameter.

Q: What is the output of this program?

On the board explanation
Functions with default parameters

```python
def performCalc(aNumber, aSecondNumber, aThirdNumber = 22):
    calculation = aNumber + aSecondNumber + aThirdNumber
    return calculation
```

```python
aNewNumber = performCalc(1,2)
print(aNewNumber)
aNewNumber = performCalc(1,2,3)
print(aNewNumber)
```

But that’s not QUITE like `range`, because that function has different behavior depending on how many arguments are sent to it.

What would be the output of the following, assuming the above `performCalc`?

```plaintext
performCalc(1,2,22)
performCalc(1,2)
```

On the board explanation
On the board exercise

Task: Declare a function that receives either 1 or 2 parameters. If the function receives 1 arguments, it should print “Mary had a little lamb,” but if the function receives 2 arguments, it should print, “A lamb had a little Mary.” Write code that invokes the method twice, once with 1 argument and another with 2 arguments.
Q: Why does the following **NOT** achieve the task?

```python
def nurseryRhymeMixUp(firstInput, secondInput = 2):
    if (secondInput == 2):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
```

**Task**: Declare a function that receives either 1 or 2 parameters. If the function receives 1 argument, it should print “Mary had a little lamb,” but if the function receives 2 arguments, it should print, “A lamb had a little Mary.” Write code that invokes the method twice, once with 1 argument and another with 2 arguments.
Functions with default parameters

Q: Why does the following **NOT** achieve the task?

```python
def nurseryRhymeMixUp(firstInput, secondInput = 2):
    if (secondInput == 2):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

What does the above code print to the screen?
Functions with default parameters

Q: Why does the following **NOT** achieve the task?

```python
def nurseryRhymeMixUp(firstInput, secondInput = 2):
    if (secondInput == 2):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

What does the above code print to the screen?

```
A lamb had a little Mary
A lamb had a little Mary
```

Q: Is that what we wanted this function to do?
Functions with default parameters

Q: Why does the following **NOT** achieve the task?

```python
def nurseryRhymeMixUp(firstInput, secondInput = 2):
    if (secondInput == 2):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

What does the above code print to the screen?

A lamb had a little Mary  
A lamb had a little Mary

Q: Is that what we wanted this function to do?

What we want is the ability for a function to perform a specific task if the last input argument is the value **nothing**.
A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else :
        print("Mary had a little lamb")

nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

Q: What does this code output, and why?
Functions with default parameters

A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")

nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

Q: What does this code output, and why?

None is the keyword that is python’s equivalent of null, which means “does not refer to anything”
Functions with default parameters

A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

Q: What does this code output, and why?

None is the keyword that is python’s equivalent of null, which means “does not refer to anything”

When the function is invoked with no arguments, the local variable secondInput refers to ...
Functions with default parameters

A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else :
        print("Mary had a little lamb")

nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

**Q:** What does this code output, and why?

`None` is the keyword that is python’s equivalent of null, which means “does not refer to anything”

The selection statement then checks if `secondInput` is “anything”, which in this case it is not, and the functions prints ...
Functions with default parameters

A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1,2)
```

Q: What does this code output, and why?

None is the keyword that is python’s equivalent of null, which means “does not refer to anything”

The selection statement then checks if `secondInput` is “anything”, which in this case it is not, and the functions prints ...

A lamb had a little Mary
Functions with default parameters

A “sample” solution

```python
def nurseryRhymeMixUp(firstInput, secondInput = None):
    if (secondInput == None):
        print("A lamb had a little Mary")
    else:
        print("Mary had a little lamb")
nurseryRhymeMixUp(1)
nurseryRhymeMixUp(1, 2)
```

Q: What does this code output, and why?

None is the keyword that is python’s equivalent of null, which means “does not refer to anything”

At the next invocation of the function, secondInput is assigned the value 2, and the if statement’s else code block is executed

A lamb had a little Mary
Mary had a little lamb
Global Variables

Q: What is the opposite of a variable that has local scope?
Q: What is the opposite of a variable that has local scope?

A variable that can be used anywhere. Such a variable is called a global variable.

There are sometimes very specific reasons why you’d want to use a global variable, but you shouldn’t use them at this stage of your coding career. Global variables are strongly discouraged in the coding world. Unless there’s a good reason to use a global variable (and NONE of the code that you’ll see or write for this course will require you to use global variables), you shouldn’t use global variables.
Global Variables

Q: What is the opposite of a variable that has local scope?

A variable that can be used anywhere. Such a variable is called a **global** variable.

There are sometimes very specific reasons why you’d want to use a global variable, **but you shouldn’t use them at this stage of your coding career**. Global variables are strongly discouraged in the coding world. Unless there’s a good reason to use a global variable (and NONE of the code that you’ll see or write for this course will require you to use global variables), you shouldn’t use global variables.

Q: Does the code on the right use a global variable?

```python
def calcSum(aNumber):
    print (aNumber + aSecondNumber)

aSecondNumber = 34
calcSum(6)
```
Global Variables

Q: What is the opposite of a variable that has local scope?

A variable that can be used anywhere. Such a variable is called a **global** variable.

There are sometimes very specific reasons why you’d want to use a global variable, but you shouldn’t use them at this stage of your coding career. Global variables are strongly discouraged in the coding world. Unless there’s a good reason to use a global variable (and NONE of the code that you’ll see or write for this course will require you to use global variables), you shouldn’t use global variables.

```python
def calcSum(aNumber):
    print(aNumber + aSecondNumber)

aSecondNumber = 34
calcSum(6)
```

Q: Does the code on the right use a global variable?

The variable `aSecondNumber` is not declared in the function `calcSum`, but it is used in its body. Thus `aSecondNumber` is a global variable.
Global Variables

Not only does it make code a bit more difficult to read, but using global variables in functions doesn’t allow us to think of functions as black boxes anymore.

Q: What is the output of the program shown on the right?

A. 40 34 50 60
B. 56 50 34 88
C. 50 44 50 94
Global Variables

Not only does it make code a bit more difficult to read, but using global variables in functions doesn’t allow us to think of functions as black boxes anymore.

Q: What is the output of the program shown on the right?

A. 40 34 50 60
B. 56 50 34 88
C. 50 44 50 94

We want functions to have predictable behavior, and using global variables prevents that. **lesson learned: do not use global variables**
Q: Why do we like to use functions?

Now we are actually at a point where you can write fairly complex functions. For example:

Task: Write a function that receives a single integer argument and which calculates the square of that argument. For demonstration purposes, you cannot use the ** operator.

How might you achieve this?
Q: Why do we like to use functions?

Now we are actually at a point where you can write fairly complex functions. For example:

Task: Write a function that receives a single integer argument and which calculates the square of that argument. For demonstration purposes, you cannot use the ** operator.

How might you achieve this?

\[
\begin{align*}
4 \text{ squared} &= 16 & 4 + 4 + 4 + 4 &= 16 \\
6^2 &= 36 & 6 + 6 + 6 + 6 + 6 + 6 &= 36 \\
5^2 &= 25 & 5 + 5 + 5 + 5 + 5 &= 25
\end{align*}
\]
The Accumulator Pattern

```python
def square(upperNum):
    accumulate = 0
    for x in range(0, upperNum):
        accumulate = accumulate + upperNum
    return accumulate
output = square(5)
print(output)
```

Notice

- `x` is NOT used in the calculation. It is used ONLY to determine how many times the body of the for loop should iterate.
- The variable `accumulate` is initialized to 0

**Q: What is the output of the above program?**

On the board explanation
The Accumulator Pattern

Notice

- $x$ is NOT used in the calculation. It is used ONLY to determine how many times the body of the for loop should iterate.
- The variable `accumulate` is initialized to $0$.

Q: What is the output of the above program?

Q: What would happen if the `accumulate = 0` were moved inside the for loop?
The Accumulator Pattern

```python
def square(upperNum):
    accumulate = 0
    for x in range(0, upperNum):
        accumulate = accumulate + upperNum
    return accumulate

output = square(5)
print(output)
```

Q: What does the program print to the screen?

A. 0  
B. 5  
C. 10 
D. 15 
E. 20 
F. 25 
G. 30
The Accumulator Pattern

```python
def square(upperNum):
    for x in range(0, upperNum):
        accumulate = 0
        accumulate = accumulate + upperNum
    return accumulate

output = square(5)
print(output)
```

Q: What does the program print to the screen?

A. 0
B. 5
C. 10
D. 15
E. 20
F. 25
G. 30

During each iteration of the for loop, the variable `accumulate` is assigned the value 0. Because `upperNumber = 5`, `range`'s last output is 4 but `accumulate` is immediately reset to 0, and the return value is the same as the received argument.
Midterm exam

Everything up to this point (lecture slides) is fair game for the midterm exam.
A function can call another function

Q: Can a function call (or invoke) another function?
A function can call another function

Q: Can a function call (or invoke) another function?

```python
def addTwoNumbers(aNum, aSecondNum):
    return aNum + aSecondNum

def add7(number, anotherNumber):
    output = addTwoNumbers(number, anotherNumber)
    return output + 7

calculation = add7(4, 11)
print(calculation)
```

Q: What does the above code output?

(walk through exercise on the board)
A function can call another function

So the big question ... why use functions?
A function can call another function

They make code easier to read, and allow us to break up complicated calculations in smaller manageable chunks

\[ \text{calculation} = X + Y + Z \]

Where \( X, Y \) and \( Z \) are themselves complicated calculations

Write these function \( \text{calcX}, \text{calcY}, \) and \( \text{calcZ} \), and invoke THOSE functions INSIDE of another function
main is itself a function, that calls OTHER functions.

Make sure you remember to invoke the function main

```python
def calcX():
    # do some calculation

def calcY():
    # do some calculation

def calcZ():
    # do some calculation

def main():
    calculation = calcX() + calcY() + calcZ()
    print(calculation)

main()
```
Function composition
Strings