Objectives

• More on Functions
  > Scope, variable lifetime
• Defining Modules

Functions

• In general, a function can have
  > 0 or more inputs (the parameters)
  > 0 or 1 outputs (what is returned)
• When we define a function, we know its inputs and if it has output

Syntax of Function Definition

Keyword: Function Name
Input Name/Parameter
Body (or function definition)
Keyword: How to give output

Parameters

• Formal Parameters are the variables named in the function definition
• Actual Parameters or Arguments are variables or literals that really get used when the function is called.

Formal & actual parameters must match in number, order, and type!

Passing Parameters

• Only copies of the actual parameters are given to the function for immutable data types
  > Immutable types: what we’ve talked about so far
    • Strings, integers, floats
• The actual parameters in the calling code do not change

Function Output

• When the code reaches a statement like
  return x
x is the output returned to the place where function called and the function stops
  > For functions that don’t have explicit output, return does not have a value with it, e.g.,
  • return
  • Optional: don’t need to have return
Function Input and Output

• What is the input and output to this function?

```python
def metersToMiles(meters):
    METERS_TO_MILES = .0006215
    miles = meters * METERS_TO_MILES
    return miles
```

Function Input and Output

• 1 input: `meters`
• 1 output: the converted miles

```python
def metersToMiles(meters):
    METERS_TO_MILES = .0006215
    miles = meters * METERS_TO_MILES
    return miles
```

Function Input and Output

• Identify input and output

```python
def printVerse(animal, sound):
    print BEGIN_END + EIEIO
    print "And on that farm he had a " + animal + EIEIO
    print "With a " + sound + ", " + sound + " here"
    print "And a " + sound + ", " + sound + " there"
    print "Here a", sound
    print "There a", sound
    print "Everywhere a " + sound + ", " + sound
    print BEGIN_END + EIEIO
    print
```

Function Input and Output

• 2 inputs: `animal` and `sound`
• 0 outputs

> Displays something but does not return anything

```python
def printVerse(animal, sound):
    print BEGIN_END + EIEIO
    print "And on that farm he had a " + animal + EIEIO
    print "With a " + sound + ", " + sound + " here"
    print "And a " + sound + ", " + sound + " there"
    print "Here a", sound
    print "There a", sound
    print "Everywhere a " + sound + ", " + sound
    print BEGIN_END + EIEIO
    print
```

Function Input and Output

• Input? Output?

```python
def printMenu():
    print "You have some options for what to do: "
    print "Enter an 'F' to find a song"
    print "Enter an 'S' to sort by Song title"
    print "Enter an 'A' to sort by Album"
    print "Enter an 'R' to sort by artist name"
    print "Enter a 'H' to list your options again"
    print "Enter a 'Q' to quit"
```

Function Input and Output

• 0 inputs and 0 outputs

> Again, it displays something but does not return anything

```python
def printMenu():
    print "You have some options for what to do: "
    print "Enter an 'F' to find a song"
    print "Enter an 'S' to sort by Song title"
    print "Enter an 'A' to sort by Album"
    print "Enter an 'R' to sort by artist name"
    print "Enter an 'H' to list your options again"
    print "Enter a 'Q' to quit"
Writing Comments for Functions

- Good style: Each function **must** have a comment
  - Describes functionality at a high-level
  - Include the **precondition**, **postcondition**
  - Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)

Example Comment

- Describes at high-level
- Describes parameters

```python
# prints a verse of Old MacDonald, plugging in the animal and sound parameters (which are strings), as appropriate
def printVerse(animal, sound):
    print BEGIN_END + EIEIO
    print "And on that farm he had a " + animal + EIEIO
```

Converting functionality into functions

- **binaryToDecimal.py**
  - Converting from binary to decimal
  - Checking if a string contains only binary numbers

- Write comments for the functions

```python
# pre: binary_string is a string that contains only 0s and 1s
# post: returns the decimal value for the binary string
def binaryToDecimal(binary_string):
    exponent = len(binary_string) - 1
    dec_value = 0
    for bit in binary_string:
        bit = int(bit)
        # print bit,"* 2^%d" % exponent
        dec_value += bit * (2 ** exponent)
        exponent -= 1
    return dec_value
```

Review: Why write functions?
Review: Why write functions?

- Allows you to break up a hard problem into smaller, more manageable parts
- Makes your code easier to understand
- Hides implementation details (abstraction)
  - Provides interface (input, output)
- Makes part of the code reusable so that you:
  - Only have to write function code once
  - Can debug it all at once
  - Isolates errors
  - Can make changes in one function (maintainability)
- Similar to benefits of classes in OO Programming

Where are Functions Defined?

- Functions can go inside of program script
  - Defined before use/called (if no `main()` function)
- Functions can go inside a separate module
  - Reduces code in main script
  - Easier to reuse by importing from a module
  - Maintains the “black box”
  - Isolates testing of function
  - Write “test driver” scripts to test functions separately from use in script

Program Organization: main function

- In many languages, you put the “driver” for your program in a main function
  - You can (and should) do this in Python as well
- Typically main functions are defined at the top of your program
  - Readers can quickly see what program does
- main usually takes no arguments
  - Example: `def main():`

Program With main() & Functions

```python
def main():
    print "This program converts from binary to decimal numbers."
    print
    binary_string = raw_input("Enter a number in binary: ")
    while not isBinary(binary_string):
        print "Sorry, that is not a binary string"
        binary_string = raw_input("Enter a number in binary: ")
    print "The decimal value is", binaryToDecimal(binary_string)

def isBinary(binary_string):
    ...

def binaryToDecimal(binary_string):
    ...

main()
```

What does this program output?

```python
def main():
    x = 10
    sum = sumEvens(x)
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit):
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total

main()
```

Function Variables

```python
def main():
    x = 10
    sum = sumEvens(x)
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit):
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total

main()
```
Tracing through Execution

When you call `main()`, that means you want me to do this:

```
def main():
    x = 10
    sum = sumEvens(x)
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit):
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total
```

Where does program "doing stuff"?

Function Variables

```
def main():
x = 10
sum = sumEvens(x)
print "The sum of even #s up to", x, "is", sum
def sumEvens(limit):
total = 0
for x in xrange(0, limit, 2):
total += x
return total
main()
```

Where does program "doing stuff"?

Function Variables

```
def main():
x = 10
sum = sumEvens(x)
print "The sum of even #s up to", x, "is", sum
def sumEvens(limit):
total = 0
for x in xrange(0, limit, 2):
total += x
return total
main()
```

Function Variables

```
def main():
    x = 10
    sum = sumEvens(x)
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit):
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total
main()
```

Function Variables

```
def main():
x = 10
sum = sumEvens(x)
print "The sum of even #s up to", x, "is", sum
def sumEvens(limit):
total = 0
for x in xrange(0, limit, 2):
total += x
return total
main()
```
Function Variables

```python
def main() :
    x=10
    sum = sumEvens( x )
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit) :
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total

main()
```

Function Variables

```python
def main() :
    x=10
    sum = sumEvens( x )
    print "The sum of even #s up to", x, "is", sum

def sumEvens(limit) :
    total = 0
    for x in xrange(0, limit, 2):
        total += x
    return total

main()
```

Variable Scope

- Functions can have the same parameter and variable names as other functions
  - Need to look at the variable’s scope to determine which one you’re looking at
  - Use the stack to figure out which variable you’re using
- Scope levels
  - Local scope (also called function scope)
    - Can only be seen within the function
  - Global scope (also called file scope)
    - Whole program can access
    - More on these later

Variable Scope

- Practice: scope.py
  - Trace through program–what does it do?
  - Answer questions in program…

Function Scope

- What variables can we “see” (i.e., use)?

```python
def main() :
    binary_string = raw_input("Enter a binary #: ")
    if not isBinary(binary_string):
        print "That is not a binary string"
        sys.exit() 
    print "The decimal value is", binaryToDecimal(binary_string)

def isBinary(string):
    for bit in string:
        if bit != "0" and bit != "1":
            return False
    return True
```

This Week

- Lab 6
- No Broader Issue
  - SSA
    - Extra credit opportunity
- Reminder: Next Friday, Mar 6, Midterm