

Objectives

- More on Functions
 - Scope, variable lifetime
- Modules

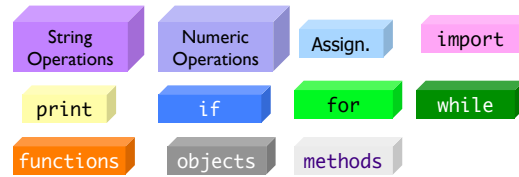
March 2, 2011

Sprenkle - CSCI1111

1

Lab Review

- Combinations of building blocks
- Faster recognition of how to solve
 - String without spaces → accumulate string!
- Iterative problem solving



March 2, 2011

Sprenkle - CSCI1111

2

Review: Functions

- What is the keyword to create a new function?
- What is the keyword to give output from a function?
- How do we give input to a function?
- Why write functions?

March 2, 2011

Sprenkle - CSCI1111

3

Review: 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

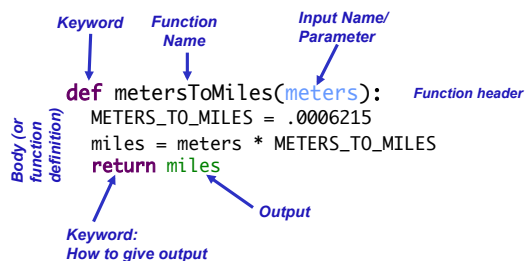


March 2, 2011

Sprenkle - CSCI1111

4

Review: Syntax of Function Definition



March 2, 2011

Sprenkle - CSCI1111

5

Review: 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.

Defined: **Formal** `def round(x, n) :` **Actual** `round(celc, 2)`
Use: `roundCelc = round(celc, 2)`

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

March 2, 2011

Sprenkle - CSCI1111

6

Review: Function Output

- When the code reaches a statement like
`return x`
the function stops executing and
`x` is the **output returned** to the place where
function was called
- 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`

March 2, 2011

Sprengle - CSCI111

7

Function Input and Output

- What is the input and output to this function?

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

March 2, 2011

Sprengle - CSCI111

8

Function Input and Output

- 1 **input**: `meters`
- 1 **output**: the converted miles

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

March 2, 2011

Sprengle - CSCI111

9

Function Input and Output

- Identify input and output

```
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
```

March 2, 2011

Sprengle - CSCI111

10

Function Input and Output

- 2 **inputs**: `animal` and `sound`
- 0 **outputs**
 - > Displays something but does not **return** anything

```
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?

```
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"
```

March 2, 2011

Sprengle - CSCI111

12

Function Input and Output

- 0 inputs and 0 outputs
 - Again, it *displays* something but does not *return* anything

```
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"
```

March 2, 2011

Sprenkle - CSCI1111

13

Alternative Approach to Development

- Create overview, define functions later
 - Top-down approach

```
def main():
    # get the binary number from the user, as a string
    binNum = raw_input("Please enter a binary number: ")
    isBinary = checkBinary(binNum)
    while not isBinary : # equivalent to isBinary == False
        print binNum, "is not a binary number."
        binNum = raw_input("Please enter a binary number: ")
        isBinary = checkBinary(binNum)

    decVal = binaryToDecimal(binNum)
    print binNum, "is", decVal
```

- More later...

Feb 28, 2011

Sprenkle - CSCI1111

14

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 OO Programming

March 2, 2011

Sprenkle - CSCI1111

15

VARIABLE LIFETIMES AND SCOPE

March 2, 2011

Sprenkle - CSCI1111

16

What does this program output?

```
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()
```

March 2, 2011

Sprenkle - CSCI1111

mystery.py

17

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()
```

Why can we name two different variables x?

March 2, 2011

Sprenkle - CSCI1111

mystery.py

18

Tracing through Execution

Defines functions

```
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()
```

When you call main(), that means you want to execute this function

March 2, 2011

Sprenkle - CSCI111

19

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()
```

The stack

main	x	10
------	---	----

Variable names are like first names

Function names are like last names

March 2, 2011

Sprenkle - CSCI111

20

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()
```

Called the function sumEvens
Add its parameters to the stack

sumEvens	limit	10
main	x	10

March 2, 2011

Sprenkle - CSCI111

21

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()
```

sumEvens	total	0	limit	10
main	x	10		

March 2, 2011

Sprenkle - CSCI111

22

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()
```

sumEvens	x	0	total	0	limit	10
main	x	10				

March 2, 2011

Sprenkle - CSCI111

23

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()
```

sumEvens	x	8	total	20	limit	10
main	x	10				

March 2, 2011

Sprenkle - CSCI111

24

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
```

Function `sumEvens` returned

- no longer have to keep track of its variables on stack
- lifetime of those variables is over

main()

main	sum	20
	x	10

March 2, 2011

Sprenkle - CSCI111

25

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()

main	x	10
	sum	20

March 2, 2011

Sprenkle - CSCI111

26

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

March 2, 2011

Sprenkle - CSCI111

27

Function Scope

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

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

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

March 2, 2011

Sprenkle - CSCI111

28

Variable Scope

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

March 2, 2011

Sprenkle - CSCI111

29

Practice

- What is the output of this program?
 - Example: user enters 4

```
def main():
    num = input("Enter a number to be squared: ")
    square = square(num)
    print "The square is:", square

def square(n):
    return n * n

main()
```

March 2, 2011

Sprenkle - CSCI111

practice1.py

30

Practice

- What is the output of this program?
 - Example: user enters 4

```
def main():
    num = input("Enter a number to be squared: ")
    squared = square(num)
    print "The square is:", squared
    print "The original num was:", n

def square(n):
    return n * n

main()
```

March 2, 2011

Sprengle - CSCI1111

practice2.py

31

Practice

- What is the output of this program?
 - Example: user enters 4

```
def main():
    num = input("Enter a number to be squared: ")
    squared = square(num)
    print "The square is:", squared
    print "The original num was:", n

def square(n):
    return n * n

main()
```

Error! **n** does not have a value in function main()

March 2, 2011

Sprengle - CSCI1111

32

Variable Scope

- Know "lifetime" of variable
 - Only during execution of function
 - Related to idea of "scope"
- What about variables outside of functions?
 - Example: `non_function_vars.py`

March 2, 2011

Sprengle - CSCI1111

33

Passing Parameters

- Only **copies** of the actual parameters are given to the function
 - For **immutable** data types (which are what we've talked about so far)
- The **actual** parameters in the calling code do not change
- Swap example:**
 - Swap two values in script
 - Then, put into a function

```
x = 5
y = 7
```

→

```
x = 7
y = 5
```

March 2, 2011

Sprengle - CSCI1111

swap.py

34

Swapping Characters

- Had this team:
 
- Wanted this team (temporarily):
 

Mar 5, 2010

Sprengle - CSCI1111

35

WHAT MAKES A FUNCTION GOOD?

March 2, 2011

Sprengle - CSCI1111

36

Writing a “Good” Function

- Should be an “intuitive chunk”
 - Doesn't do too much or too little
- Should be reusable
- Always have comment that tells what the function does

March 2, 2011

Sprenkle - CSCI111

37

Good vs. Bad Functions

- Bad: Does too little

```
def getUserInput():  
    input = input("Enter a number")  
    return input
```

- Good: Validates the input

```
def getUserInput():  
    input = input("Enter a number")  
    while input <= 0:  
        print "Number must be positive"  
        input = input("Enter a number")  
    return input
```

March 2, 2011

Sprenkle - CSCI111

38

Debugging Advice

- Build up your program in steps
 - Always write only small pieces of code
 - Test, debug. **Repeat**
- Write function body as part of **main**, test
 - Then, separate out into its own function
 - Similar to process using in lab probs
- Test function separately from other code
 - Comment out irrelevant code to make sure that the function behaves as expected

March 2, 2011

Sprenkle - CSCI111

39

CREATING MODULES

March 2, 2011

Sprenkle - CSCI111

40

Where are Functions Defined?

- Functions can go inside of program script
 - Defined before use/called (if no **main()** function)
 - Or, below the **main()** function
- Functions can go inside a separate **module**

March 2, 2011

Sprenkle - CSCI111

41

Benefits of Defining Functions in Separate Module

- Reduces code in primary driver script
- Easier to reuse by importing from a module
- Maintains the “black box”
 - **Abstraction**
- Isolates testing of function
- Write “test driver” scripts to test functions separately from use in script

March 2, 2011

Sprenkle - CSCI111

menu.py

42

Creating Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
 - A module is named by its filename

Just a Python file!

- Example, oldmac.py
 - In Python shell: `import oldmac`
 - Explain what happened

March 2, 2011

Sprenkle - CSCI1111

43

Creating Modules

- So that our program doesn't execute when it is `imported` in a program, at bottom, add

```
if __name__ == '__main__':  
    main()
```

Not important how this works;
just know when to use

- Then, to call `main` function
 - `oldmac.main()`
- Note the files now listed in the directory

March 2, 2011

Sprenkle - CSCI1111

44

Creating Modules

- Then, to call `main` function
 - `oldmac.main()`
- Why would you want to call a module's `main` function?
 - Automation
 - Use `main` function as driver to test functions in module
- To access one of the defined constants
 - `oldmac.EIEIO`

March 2, 2011

Sprenkle - CSCI1111

45

Defining Constants in Modules

- Constant in `menu.py`
 - `STOP_OPTION`
- Show use in `menu_withfunctions.py`

March 2, 2011

Sprenkle - CSCI1111

46

Summary: Program Organization

- Larger programs require `functions` to maintain readability
 - Use `main()` and other functions to break up program into *smaller, more manageable* chunks
 - “Abstract away” the details
- As before, can still write smaller scripts without any functions
 - Can try out functions using smaller scripts
- Need the `main()` function when using other functions to keep “driver” at top
 - Otherwise, functions need to be defined **before** use

March 2, 2011

Sprenkle - CSCI1111

47

This Week

- Lab 6 due Friday
- SSA – Friday
 - Opportunities for Extra Credit
 - *Truth Values* play requires tickets—Free!

March 2, 2011

Sprenkle - CSCI1111

48

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)

March 2, 2011

Sprenkle - CSCI1111

49

Writing Comments for Functions

- Include the function's pre- and post-conditions
- **Precondition**: Things that must be true for function to work correctly
 - E.g., num must be even
- **Postcondition**: Things that will be true when function finishes (if precondition is true)
 - E.g., the returned value is the max

March 2, 2011

Sprenkle - CSCI1111

50

Example Comment

- Describes at high-level
- Describes parameters

```
# 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
    ...
```

March 2, 2011

Sprenkle - CSCI1111

51

Writing a Function

- Write a function that determines if a string is a binary string
- Write comments for that function

March 2, 2011

Sprenkle - CSCI1111

52

Pre/Post Conditions

```
# 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
```

March 2, 2011

Sprenkle - CSCI1111

53