

Objectives

- More on Functions
 - Scope, variable lifetime
- Defining Modules

Feb 27, 2008

Sprenkle - CS111

1

Quick Review

- The data type of the loop variable depends on what's after **in**

```
string = "some string"

for x in xrange(len(string)):
    # loop body ...

for x in string:
    # loop body ...
```

What is the data type the loop variable **x**?

Feb 27, 2008

Sprenkle - CS111

2

Quick Review

- The data type of the loop variable depends on what's after **in**

```
string = "some string"

for x in xrange(len(string)):
    # loop body ...

for x in string:
    # loop body ...
```

Integer

String

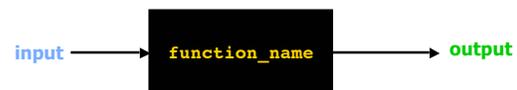
Feb 27, 2008

Sprenkle - CS111

3

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**



Feb 27, 2008

Sprenkle - CS111

4

Syntax of Function Definition

```
Keyword      Function Name      Input Name/
  ↓           ↓                Parameter
def metersToMiles(meters) :
    METERS_TO_MILES = .0006215
    miles = meters * METERS_TO_MILES
    return miles
```

Function header

Body (or function definition)

Keyword: How to give output

Output

Feb 27, 2008

Sprenkle - CS111

5

Parameters

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

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

Formal

Actual

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

Feb 27, 2008

Sprenkle - CS111

6

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

Feb 27, 2008

Sprenkle - CS111

7

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 (see `menu.py`)

Feb 27, 2008

Sprenkle - CS111

8

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

Feb 27, 2008

Sprenkle - CS111

9

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

Feb 27, 2008

Sprenkle - CS111

10

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():`

Feb 27, 2008

Sprenkle - CS111

11

Program With **main()** & Functions

```
def main():
    print "This program converts binary numbers to decimal numbers."
    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()
```

Presents overview of what program does (hides details)

Feb 27, 2008

Sprenkle - CS111

12

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()
```

Feb 27, 2008

Sprengle - CS111

13

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 variables x?

Feb 27, 2008

Sprengle - CS111

14

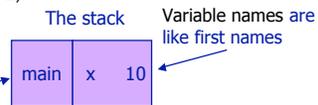
Function Variables

```
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 names are like last names



Feb 27, 2008

Sprengle - CS111

15

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

Feb 27, 2008

Sprengle - CS111

16

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	limit	10
	total	0
main	x	10

Feb 27, 2008

Sprengle - CS111

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()
```

sumEvens	limit	10
	total	0
	x	0
main	x	10

Feb 27, 2008

Sprengle - CS111

18

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()

sum	limit	10
Evens	total	20
	x	8
main	x	10

Feb 27, 2008

Sprengle - CS111

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()

Function max returned, so we no longer have to keep track of its variables on the stack.

The lifetime of those variables is over.

main	sum	20
	x	10

Feb 27, 2008

Sprengle - CS111

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()

main	x	10
	sum	20

Feb 27, 2008

Sprengle - CS111

21

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

Feb 27, 2008

Sprengle - CS111

22

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()
    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
```

...

Feb 27, 2008

Sprengle - CS111

23

Variable Scope

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

Feb 27, 2008

Sprengle - CS111

24

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()
```

Feb 27, 2008

Sprengle - CS111

practice1.py

25

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()
```

Feb 27, 2008

Sprengle - CS111

practice2.py

26

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
```

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

```
def square(n):
    return n * n
```

```
main()
```

Feb 27, 2008

Sprengle - CS111

27

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`

Feb 27, 2008

Sprengle - CS111

28

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

Feb 27, 2008

Sprengle - CS111

29

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

Feb 27, 2008

Sprengle - CS111

30

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
```

Feb 27, 2008

Sprenkle - CS111

31

Creating Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
 - Modules are named by the filename
 - Example, oldmac.py
 - In Python shell: `import oldmac`
 - Explain what happened

Feb 27, 2008

Sprenkle - CS111

32

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

Feb 27, 2008

Sprenkle - CS111

33

Creating Modules

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

Feb 27, 2008

Sprenkle - CS111

34

Defining Constants in Modules

- Add constant to `menu.py`
 - `STOP_OPTION`
- Show use in `menu_withfunctions.py`

Feb 27, 2008

Sprenkle - CS111

35

Program Organization

- Larger programs require functions to maintain readability
 - Use `main()` and other functions to break up your program into smaller, more manageable chunks
 - "Abstract away" the details
- As before, you 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

Feb 27, 2008

Sprenkle - CS111

36

Broader Issue Reading

- Two articles about Microsoft Excel 2007 Bug
 - Just write one summary