Objectives

- Introduction to Functions
- Broader Issues: Excel 2007 Bug

Lab 5 Work

Functions

- We’ve used functions
  - Built-in functions: `len`, `input`, `raw_input`
  - Functions from modules, e.g., `math` and `random`
- Today, we’ll learn how to define our own functions!

In general, a function can have
- 0 or more inputs
- 0 or 1 outputs

When we define a function, we know its inputs and if it has output

Example Functions

- `spread(<condiment>, <amount_in_TB>, <sandwich>)`
  - Spreads given amount of condiment on sandwich
  - Input: ?
  - Output: ?
- `userPBPref(<username>)`
  - For the given user, returns the amount of PB they want on their sandwich
  - Input: ?
  - Output: ?
Example Functions

- `spread(<condiment>, <amount_in_TB>, <sandwich>)`
  - Spreads given amount of condiment on sandwich
  - Input: condiment, amount_in_TB, sandwich
  - Output: no output
    - State of sandwich changes → now has condiment on it
- `userPBPref(<username>)`
  - For the given user, returns the amount of PB they want on their sandwich
  - Input: username
  - Output: the user’s PB preference

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

Comparison of Code Using Functions

- Without functions:
  - `menu_withoutfunc.py`
- With functions
  - `menu_withfunctions.py`

Example Program

- Lab 2, Problem 4
  - Any place to make a function?
  - Any place that has some useful code that we may want to reuse?

Convert meters to miles

- Input: meters
- Output: miles

Syntax of Function Definition

```
def metersToMiles(meters):
    METERS_TO_MILES = .0006215
    miles = meters * METERS_TO_MILES
    return miles
```
Functions: Similarity to Math

• In math, function definition looks like:
  \[ f(x) = x^2 + 2 \]
• Plug values in for \( x \)
  \[ f(3) = 3^2 + 2 = 11 \]
  \( 3 \) is your input, assigned to \( x \)
  \( 11 \) is output

Parameters

• The inputs to a function are called parameters or arguments
• When calling/using functions, arguments must appear in same order as in the function header
  \[ \text{Example: } \text{round}(x, n) \]
  \( x \) is the float to round
  \( n \) is integer of decimal places to round to

Parameters

• Formal Parameters are the variables named in the function definition
• Actual Parameters or Arguments are the variables or literals that really get used when the function is called.
  
  Defined: \texttt{def round}(x, n) :
  
  Use: \texttt{roundCelc = round(celc, 2)}

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

Passing Parameters

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

Function Output

• When the code reaches a statement like \texttt{return} \( x \)
  \( x \) is the output returned to the place where function called and the function stops
  \[ \text{For functions that don’t have explicit output, } \texttt{return} \text{ does not have a value with it, e.g.,} \]
  \[ \texttt{return} \]
  \[ \text{Optional: don’t need to have return (see menu.py)} \]

Calling your own functions

\[ \text{miles} = \text{metersToMiles}(100) \]
\[ \text{miles2} = \text{metersToMiles}(200) \]
\[ \text{miles3} = \text{metersToMiles}(400) \]
\[ \text{miles4} = \text{metersToMiles}(800) \]

Output is assigned to \texttt{miles4}

Function Name

Input
Flow of Control

• When you call the function, the computer jumps to the function and executes it
• When it is done executing the function, the computer returns to the same place in the first code where it left off

#Make conversions
dist1 = 100
miles1 = metersToMiles(dist1)

Flow of Control

def max(num1, num2):
    result = 0
    if num1 >= num2:
        result = num1
    else:
        result = num2
    return result

x = 2
y = input("Enter a number")
z = max(x, y)
print "The max is", z
Where are Functions Defined?

- Functions can go inside of program script
  - Defined before use/called (if no `main()` function)
  - Example from lab2.4.py

Where are Functions Defined?

- 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():`

Using a `main` Function

- Call `main()` at the bottom of your program

- Side effects:
  - Do not need to define functions before `main` function
  - `main` can “see” other functions
  - Note that `main` is a function that calls other functions
    - Any function can call other functions

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)
```

Presents overview of what program does (hides details)

Example program with a `main()`

```python
oldmac.py
```
Broader Issues in Computer Science

- Testing isn’t a broader issue
  - Glad you noticed lots of the issues with testing
  - We’ll keep talking about it because I love it!

Mike, Sara, Kevin, Ben
Greg, Charles, Dylan, David

Notes from a Keynote Speech about Testing Microsoft Vista

- Users are “trained” to not use buggy features
  - After user encounters a certain bug when doing something enough times, eventually, the user stops trying to do that

\[
\text{User’s Loss in Confidence} = \text{Disruption Frequency} \times \text{Recovery Time} \times \text{Lost data} \times \text{Uncertainty}
\]

- Only ship fixes that affect many users

Mike, Sara, Kevin, Ben
Greg, Charles, Dylan, David

Relation to Our Class

- When do you stop testing?

Mike, Sara, Kevin, Ben
Camille, Carrie, Aaron, Russ
Greg, Charles, Dylan, David
Michelle, Taylor, Thomas, Chen

Status from Official Excel Blog

- Post on 9/25
  - We’ve come up with a fix for this issue and are in the final phases of a broad test pass in order to ensure that the fix works and doesn’t introduce any additional issues - especially any other calculation issues. This fix then needs to make its way through our official build lab and onto a download site - which we expect to happen very soon.

- Post on 10/9
  - As of today, fixes for this issue in Excel 2007 and Excel Services 2007 are available for download ...
  - We are in the process of adding this fix to Microsoft Update so that it will get automatically pushed to users running Excel 2007 or Excel Services 2007. Additionally, the fix will also be contained in the first service pack of Office 2007 when it is released (the release date for SP1 of Office 2007 has not been finalized).