CPSC 301: Computing in the Life Sciences
Lecture Notes 5:
Python Functions

2017-01-24

Jessica Dawson
jqdawson@cs.ubc.ca
http://www.cs.ubc.ca/~jqdawson

University of British Columbia
Department of Computer Science

2016 W2
• Lab 01 grade reports posted
  – Access via handback in Connect (demoed in class)
  – Lab documents not handed back this time
    • If you lost a mark on the after lab portion it’s because one of the required files was missing (or corrupt)
  – Usually marks will be handed back within 1 week following after-lab due date

• Weird behaviour when running scripts / commands in Python on your personal computer?
  – See Piazza post on how to roll back to more stable version
Objectives

At the end of this unit, you will be able to:

• call a previously defined Python function
• define Python functions to perform certain tasks
• describe how a function is executed
• identify local variables and function parameters
• trace a function call and identify what value the function returns
• sketch out the state of memory before, during and after a function call
• design a function according to the suggested recipe
Functions

• A large program (procedure) for a task is usually broken down to subprograms each of which computes a subtask
  – easier to read understand and change the program
  – subtasks can be used in other programs
• A subtask is like a smaller task. It has its input values, its procedure and its results (output)
• In Python we use functions to define simple tasks which can be the subtasks of a problem
• That is, a complex program is split into a number of functions, each computing a subtask of the problem
Built-in Functions

Python comes with a number of useful functions already defined and available to use like:

- **General math functions**
  - `abs(number)` – absolute value of a number
    - `abs(5)` is 5, `abs(-5)` is 5
  - `pow(n1, n2)` - n1 to the power of n2
    - `pow(2, 3)` is 8
  - `round(number)` - rounds a floating-point number
    - `round(2.4999)` is 2.0, `round(2.5)` is 3.0, `round(2)` is 2.0
  - `min(number, number)` – returns the minimum of the arguments
    - `min(2, 4)` is 2
  - `max(number, number)` – returns the maximum of the arguments
    - `max(2, 4)` is 4,

- **Functions that convert a value of one type to another type:**
  - `int(number)` returns an integer
    - `int(2.9999)` is 2, `int(2.0001)` is 2
  - `float(number)` returns a floating-point number
    - `float(2)` is 2.0, `float(2.33455)` is 2.33455
  - `str(value)` returns a string
    - `str(2.25)` is '2.25', `str(123)` is '123', `str('abc')` is 'abc'
**Function Definition**

- A function definition looks like:

  ```python
def function-name( parameters ) :
    """ docstring for the function """
    block-of-statements
  
  where:
  - `def` is a Python keyword starting a function definition
  - `function-name` is the name we give to the function, like a variable name
  - `parameters` are variables that will provide the input values for the function
  - A `docstring` is a (usually multiline) string that appears immediately after the header line ("def...") of a function
  - the block-of-statements defines the tasks for the function; all these statements are indented by a tab stroke

- A return statement of the form

  ```python
  return some-value
  ```

  is used in the function statements to return the result
Function Definition (cont')

• Example 1: a function that converts miles to kilometres
  
  ```python
  def kilometers(miles):
      """ Returns the kilometers that correspond to the given miles """
      return miles * 1.609
  ```

• To use the function we use its name and provide values for its parameters (called *arguments*).

• Example of function calls:
  
  – kilometers(100) will return 160.9
  – distance_in_miles = 10
    distance_in_kilometers = kilometers(distance_in_miles)
    then distance_in_kilometers \(\rightarrow\) 16.09

• Example 2: A function that calculates the income tax given a income and a tax rate:
  
  ```python
  def income_tax( income, rate ) :
      """ Returns the income tax for the given income and tax rate """
      return (income - 9000.00) * rate
  ```

• Example call: income_tax( 50000, 0.30 )
Evaluation Order

• Because functions produce values, they can be used in expressions
  >>> \texttt{round}(-12.2) + \texttt{max}(2,5)
  -7

• Python evaluates arguments from left to right, from the inner most function to the outmost functions
  >>> \texttt{round} ( \texttt{min} ( 3.1, 4.0 ))
  evaluates in the following order:
  3.1
  4.0
  min (3.1, 4.0)
  round(3.1)
Local Variables

• A function may need to declare its own variables to simplify the calculations. These variables
  – are only accessible from the statements inside the function's body
  – are called local variables

• Example: An income_tax function:
  
def income_tax( income, rate ) :
    """Returns the income tax for the given income and tax rate"""
    exemption = 9000.0
    net_income = income - exemption
    return net_income * rate

• Note: parameters are only accessible inside the function body as well; they are also local variables
  – in the above example: income, rate, exemption and net_income are only known inside the income_tax function
  – we say that the scope of these variables is the code from their declaration to the end of the function
Each function call is executed in its own name space or frame: an environment with its local variables and parameters.

When a function returns its environment is destroyed and the value is returned to the environment that called the function.

Example:

```python
def income_tax(income, rate):
    """
    
    exemption = 9000.00
    net_income = income - exemption
    return net_income * rate
    """

# Main Program
my_income = 10000.00
my_rate = 0.20
my_tax = income_tax(my_income, my_rate)
```

Main's frame:
income_tax → income_tax(income, rate)
my_income → 10000.0
my_rate → 0.20
my_tax → _________
The return statement

• Determines flow of control
  – When return is encountered in a function, the function call is done and control is returned to the caller
  – Any further statements in the function are ignored

• A function call is an expression which always has a value
  – That value can be specified in the return statement which completes the function call
  – If no value is specified in the return statement, the value of the function call is None
  – If no return statement is encountered before the body of the function ends, the value of the function call is None
Change Log

• Added some additional examples of built in functions (min and max)
• Added slide on evaluation order
• In Function Call slide: added the initial placing of income_tax function name on the main program frame when the code first runs