Functions

CSE 1310 – Introduction to Computers and programming

Alexandra Stefan
University of Texas at Arlington
Book reading

• Chapter 6,
  – except for:
    • 6.3.4 (it is an example that uses files. can be read as soon as we cover files) and
    • 6.4 (uses the Turtle graphics).

• Chapter 8:
  – 8.1.1 - 8.1.3, 8.5 (covers variable scope, namespaces, mutable objects)
  – Excluded:
    • 8.1.4 - uses list comprehension, NOT covered in this class
  – Not required, but interesting:
    • 8.2 - default values for parameters and parameter keywords - very interesting topic, recommended to students who are comfortable with the material and want to read more.
    • 8.3 - functions as objects (annotations, docstrings, ...)
    • 8.4 - an example, uses default parameters.
Review: Function call

• Syntax:  
  
  function_name(argument1, argument2,..)
  
  – It may take none, one, or more arguments
  – It may or may not return a value (an object)
    • If it returns a value, this value can be saved/mapped to a variable with an assignment statement

• E.g.:  
  
  N = len("today")

• The parenthesis, (), after the name make it a function call (as opposed to a reference to an object).
• It ‘communicates’ with the outside world via the arguments and the return value.

• Notice that function calls are different from method calls (using dot notation).
  • E.g.:  
    
    my_str = "today".upper()
  • Syntax:  
    
    object.method_name(argument1, argument2, ...)


New material:
Write our own functions

• Why do we need functions?

• To make our life easier!
  – Easier to read and write programs
  – Easier to test and debug
  – Easier to share
  – and more …

• We’ll see that soon.
New material: Write our own functions

• When we ‘write’ a function we give the *function definition*.

• The function definition associates a *name* (the function name) with a *piece of code*.

• The code in the function definition is not executed until the function is called.
def function_name(_,_,_,...):
    statement
    ...
    statement
    return some_object
Function definition

def C_to_F(C):
    F = C * 1.8 + 32
    return F
Function definition

```
def C_to_F(C):
    F = C * 1.8 + 32
    return F
```

Function name: `C_to_F`

Parameter: `C`

Function body:

```
    F = C * 1.8 + 32
    return F
```

Return value
Function definition

After this function definition the name \texttt{C\_to\_F} is added to the local namespaces and associated with a function object. From now on, this name will be defined (it can be used).

\texttt{C} is called a \textbf{parameter} (here, in the definition). It indicates that the function will receive some data at the time it is executed (at function call). When the function will be called it will have to be given exactly one argument (because it has one parameter in the definition).

(We can also write function definitions that allow function calls with fewer arguments than parameters.)

The function body is the set of instructions that “do the work” of the function. Indented.

The \textbf{return} statement has 2 purposes:

- ends the function definition. Any code following it (at the same indentation) will not be executed.
- specifies what data (object) should be returned to the caller program after the function finishes executing. Only one object can be returned, but that object can be a collection (e.g. str).
Function call

• Function call: $\text{C\_to\_F}(10)$

• We refer to the program that makes the function call as the ‘caller’.

• We refer to the function being called, $\text{C\_to\_F}()$, as the ‘invoked function’.
The "Main" Code

• In order for a Python file to do something, it must include some code outside function definitions.
  – The code that is outside definitions is the place where Python starts executing the code from a file. It is called the main code of the program.

  – Until we did functions, the entire code was main code (outside function definitions, because we had none).

  – Next we will place a few function calls to `C_to_F()` in the main code.
def C_to_F(C):
    F = C * 1.8 + 32
    return F

# main code (main program)
f1 = C_to_F(10)  # first line to execute
f2 = C_to_F(20)
cel = 30
f3 = C_to_F(c)

Use the debugger to follow the flow of control.
Function call

• The flow of control jumps from the caller (the program that makes the function call) to the invoked function and then back to the caller.

• The code in the body of the function is executed only when the function is called. Note that it is not executed when the function is defined.
Function call and expression evaluation in an assignment

• There is a similarity between:

  – \( f_1 = C\_to\_F(10) \) and
  – \( f_1 = 1+4 \)

  – They both:
    • First evaluate the expression on the right hand side
    • Then replace the expression \( (C\_to\_F(10) \text{ and } 1+4) \) with the computed value
    • Execute instruction: \( f_1 = \text{value} \), which maps \( f_1 \) to the computed value.
Understanding Function Evaluation (Function execution)

• Every function evaluation involves:
  – a calling line of code
  – the namespace of the calling line
  – arguments provided by the calling line of code
  – the function that we actually call
  – the namespace of the function
  – the parameters: the names of the arguments that the function uses
  – the body of the function
  – (optionally) a return value of the function
Flow of control with functions

• Flow of control that we have seen:
  – Sequential
    • statements executed in the order that they appear
    • if-else, while, for

• With functions:
  – The flow of control jumps from invocation call (in the calling program, or caller) to the function definition (the callee) and then back to the invocation call statement (in the calling program).
    • When the body of the function starts executing, code execution follows the same rules we have been using.
Calling a Function

• When a function is called:
  – The function namespace becomes active
  – The function parameters are assigned values obtained from the calling line, using the namespace of the calling line.
  – The next line of code that is executed is the first line of the body of the function.
Completing a Function Call That Returns a Value

• If the execution of the body of a function, gets to a return statement:
  – The expression after the return keyword is evaluated.
  – The value of that expression is transferred to the calling line (and it will be later used by the calling line).
  – The function call ends:
    • Any values computed by the function call, that were not returned, are lost forever.
  – The active visible namespace becomes again the namespace of the calling line.
  – The code resumes execution from the calling line (using the value returned by the function).
Defining a *procedure* (a function with no return value)

def print_greeting(name):
    print("hello," + name + ", how are you?\"")

>>> print_greeting("mary")
hello, mary, how are you?

• Function `print_greeting()` does not compute and return a value, it just does something useful.
Local variables

• Variables created in the body of a function are not visible to the main code or to any other function.

Example 1: x is undefined in main

```python
def foo():
    x = 10
    print("foo:", x)

# main code
foo()
print(x)  # Error
```

Example 2: x is undefined in bar()

```python
def foo():
    x = 10
    print("foo:", x)

def bar():
    print("bar:", x)  # Error

# main code
foo()
bar()```
Global variables

- Variables defined in the main program are visible to functions from that file if they were defined before the function executes (before the function call).
- These variables are called *global variables*.

Example 1: y is defined and visible when `foo()` is executed.
```python
def foo():
    print("foo:", y)
# main code
y = 10 #global variable
foo()
```

Example 2: y is undefined when `foo()` is executed.
```python
def foo():
    print("foo:", y)
# main code
foo()
y = 10 #global variable
Global variables

• However you should not use global variables inside a function. The transfer of data between a function and the main code or any other function should be done through the parameters and the return value of the function, not with global variables.

  – To avoid the creation and use of global variables, we will place our higher level code in a function named \textit{main()} and the main code (code that is outside of function definitions) will consist of only a call that function to get the program started. (This way there will be no global variables in our program.)
Code organization

Intermediary version:

```python
def C_to_F(C):
    F = C * 1.8 + 32
    return F

# main program
f1 = C_to_F(10)
f2 = C_to_F(20)
cel = 30  # global
f3 = C_to_F(c)
```

Final version:

```python
def C_to_F(C):
    F = C * 1.8 + 32
    return F

def main():
    f1 = C_to_F(10)
f2 = C_to_F(20)
    cel = 30  # local
    f3 = C_to_F(c)

# the execution starts here
main()
```
What happens when you modify, inside a function, an object that was passed as an argument?

If the object is mutable and it is modified (not reassigned), the modification will persist after the function execution ends (it will also be modified in the caller space).
Import a function from a file

• Functions written in one file can be used in another
  - Assume that our C_to_F function is saved in file conversion

• **Version 1** - make sure you know this one
  - First import the name of the file:
    ```python
    import conversion
    ```
  - Usage (with filename): instead of `function_name`, use
    ```python
    conversion.function_name
    ```

• Version 2
  - Import all the definitions in the file:
    ```python
    from conversion import *
    ```
  - Usage:
    ```python
    function_name
    ```

• Version 3
  - Import only the desired function definition:
    ```python
    from conversion import C_to_F
    ```
  - Usage:
    ```python
    function_name
    ```
Import a function from a file

• Functions written in one file can be used in another file, but the files must be in the same directory. Otherwise the import statement fails.
Import files – extra, update

- When asked to import a file, Python looks in the local directory and at other locations specified by the Python path. In order to ask it to look at another directory of your choice, you have to add it to the path:

  ```python
  import sys
  sys.path # see the current path
  sys.path.append('C:\Users\Alex_2\Desktop\_code\my_libraries')
  sys.path # notice that your directory was added to the list
  ```

  - If you modify the sys.path as shown above, it will be reset (to the original value) when you restart the Shell

- You can develop your own library/collection of functions
Docstrings

def function_name(_,_,...,):
    """Brief description
    # empty line
    More details (several lines)
    """
    #notice: on a separate line
    statement
    ...
    statement
    return some_object

Function

– A small program that performs a specific task:

**A function must do one thing**

• If it does more than one thing, it should be redesigned.

– Encapsulation:

  (implementation) details are hidden from us
How to write a function: basic skills

• When you write a function you must pay attention to the following:
  – Follow the syntax for function definition
  – Set up the function communication with the calling line using the parameters and the return value
    • See how many pieces of information (objects) the function needs from the outside and pass those to it using parameters.
    • See if there is any data that the function computes and is needed outside. If there is, send it out using the return statement.
Why functions?

• Better code: easier to read
• Divide-and-conquer:
  – Divide a larger problem in smaller problems that can be solved individually with functions
  – make programs easier to write
• Decoupling
  – Prevents you from modifying code that is irrelevant for this task
• Abstraction
  – A function provides an interface to an implementation of a solution
  – Hides away the implementation details
• Code reuse
  – You write and test it once, reuse it however many times you need
• Code sharing
  – Others can use or even improve your function
  – Others can use your function, while you are still working on it!!!
• Security
  – Easier to test and verify a function
  – Verified functions can then be combined together and verified again
• Simplification and readability
  – Avoids duplication of code.
Summary

- Definition vs invocation/call: know how to do both
- Parameters vs arguments
- Definition
  - `def`, ..., colon, parameters, indentation
  - Return statement
  - The docstring: a string describing what the function does
- Using functions
  - Flow of control of a function call jumps to the function definition and then back to the calling line
  - Namespaces: one for main, one for each function definition
  - Parameter passing
  - Functions are *callable* entities: use ‘()`’ after the name of the function
  - Import functions written in other files
  - Function vs procedure: procedure does not return a value
  - Mutable objects passed as arguments and modified (not reassigned) during the function call will keep the change after the function finishes.
FIGURE 6.2 Function flow of control.
Practice

• **Write functions**
  – Based on problem description:
    • “Write a function that takes [as argument] a string \( S \) and returns the sum of the ASCII code of the characters in it.

    • Write a function that takes 2 strings, a piece size, \( p\_sz \), and merges the strings in chunks of \( p\_sz \).

    • “Write a function that takes [as argument] a list \( L \) and returns the sum of the elements in \( L \). If \( L \) has strings, it should return the concatenation of all the strings.”

    • Write a fct that takes two arguments: a list, \( L \), and a number, \( N \), and removes all the occurrences of \( N \) from the list. Test cases:
      - \( L = [4,7,4,3,4], N = 1 \)
      - \( L = [4,4,4,4,4], N = 4 \)
      - \( L = [4,4,4,4], N = 4 \)
      – A) modify the list \( L \) (Pay attention to how you iterate over the list.)
      – B) return a new list with the modifications .

  – Based on your own choice.

• **Write functions for matrix operations** (involving lists and nested lists).