1 Administrative Topics

• Any questions about project 3?

2 Writing Mini-Lesson

Stephanie passed out a set of sample summaries from previous Project 3 wiki pages. We exam them and develop a list of best practices.

What makes a good summary? You should be able to answer “yes” to all of these questions:

• Does it describe the CS purpose of the project?
• Does it describe the specific project application?
• Does it describe the solution or how it was developed?
• Does it describe the results or outputs?
• Is it concise?
• Are all of the terms well-defined?
• Does it read logically and in the proper order?
3 Writing Our Own Functions that return values

We have written functions to draw scenes with the turtle or that print information, but we haven’t needed to print or store any results from those functions. When we call the mathematical functions, there is always a value returned to us. So, how do we write functions (like math functions) that return results? The answer is that we use a “return” statement.

As a review, here is what we know about writing functions: Functions have input, instructions, and output

- We begin by writing the “header” comments. Name the function, briefly describe what it does, then give detailed information about what it expects as input (including the types), then list what it returns.
- Define the function. using the keyword “def”, the name of the function, and the parameters it takes as input
- Perform the task of the function (e.g. the “do the math”)
- Insert a return statement to return the value to the caller.

Below is the code for two functions that convert temperature values – one from Celsius to Fahrenheit, the other that does the reverse. I have also uploaded the code file to the web page.
# Convert the temperature from Fahrenheit to Celsius

```python
def f2c(ftemp):
    ctemp = (ftemp - 32) * 5.0 / 9
    return ctemp
```

# Convert the temperature from Celsius to Fahrenheit

```python
def c2f(ctemp):
    ftemp = ctemp * 9.0 / 5 + 32
    return ftemp
```

# main code

# Does a temp of 30 degrees Celsius mean it is hot out?
c = 30
f = c2f(c)
weatherF(f)

## 3.1 Stepping through the code

Python reads in the function definitions, so at line 25, the main table is

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
</tbody>
</table>

Then line 27 is executed and c is added to the main symbol table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c</td>
<td>30 &lt;int&gt;</td>
</tr>
</tbody>
</table>
As Python begins to execute line 28, it first evaluates the code for the input to \texttt{c2f}.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Name} & \textbf{Value} \\
\hline
\texttt{weatherF} & \texttt{<function>} \\
\texttt{f2c} & \texttt{<function>} \\
\texttt{c2f} & \texttt{<function>} \\
\texttt{ctemp} & 30 \texttt{<int>} \\
\hline
\end{tabular}
\end{center}

\texttt{ftemp = c2f( ctemp 30 )}

Then \texttt{c2f} begins to execute. It has one parameter \texttt{ctemp}, which is placed in the table with the value that was given as input.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Name} & \textbf{Value} \\
\hline
\texttt{weatherF} & \texttt{<function>} \\
\texttt{f2c} & \texttt{<function>} \\
\texttt{c2f} & \texttt{<function>} \\
\texttt{ctemp} & 30 \texttt{<int>} \\
\hline
\end{tabular}
\end{center}

\texttt{ftemp = c2f( ctemp 30 )}

Then Python evaluates the assignment statement in the function. It begins by evaluating the mathematical expression on the left-hand side.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Name} & \textbf{Value} \\
\hline
\texttt{weatherF} & \texttt{<function>} \\
\texttt{f2c} & \texttt{<function>} \\
\texttt{c2f} & \texttt{<function>} \\
\texttt{ctemp} & 30 \texttt{<int>} \\
\hline
\end{tabular}
\end{center}

\texttt{ftemp = c2f( ctemp 30 )}

\texttt{ftemp = ctemp \times 9.0/5 + 32 86.0}

Then the table is updated and Python moves to the line 23 to execute the return statement. It begins by evaluating the code after the command \texttt{return}:
To indicate that the value is sent back to the caller, I cross out the function call in the main code and replace it with the value that is being returned:

```
main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
</tbody>
</table>

ftemp = c2f( ctemp 30 )
```

```
c2f

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
<tr>
<td>ftemp</td>
<td>86.0 &lt;flt&gt;</td>
</tr>
</tbody>
</table>

return ftemp86.0
```

Then the function’s symbol table disappears

```
main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
</tbody>
</table>

ftemp = c2f(30)86.0
```

And Python can complete the assignment statement from line 28.
And Python moves to line 29, and evaluates the input to the weatherF function:

```
main
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
<tr>
<td>ftemp</td>
<td>86.0 &lt;flt&gt;</td>
</tr>
</tbody>
</table>
```

weatherF(ftemp 86.0)

And weatherF begins execution with its parameter’s value in its symbol table.

```
main
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
<tr>
<td>ftemp</td>
<td>86.0 &lt;flt&gt;</td>
</tr>
</tbody>
</table>
```

Python moves to the code in line 8 and evaluates the conditional expression
Because the condition is True, Python takes the “if-branch” and next executes the code in line 9. This code prints the word ‘hot’ to the Terminal. There is no code after the if-elif-else statement. So Python is now done executing the function. There is no return statement. In the past, we have just crossed out the call to a function without a return statement. But today, I am going to replace it with the value that Python automatically returns. That value is None.

And then Python is done. The main symbol table disappears. And there is nothingness.

4 Default parameters and keyword arguments

When you are defining a function, you identify parameters, and it is possible to provide a default value for that parameter. When you call that function, you can choose to provide a value for that parameter if you want to.
def sayHi( name = 'You' ):
    print "Hey," , name

sayHi()  
sayHi( "Stephanie" )  
sayHi( "You" )

will print out
Hey, You
Hey, Stephanie
Hey, You

When you call a function, the value you provide as input to the function is called an argument. In the above case Stephanie is an argument. There are two ways to provide arguments. One is using positional notation. This is what we have been using so far. The first argument is the value for the first parameter. The second argument is the value for the second parameter. And so on and so forth. But there is another way to do it, using keyword notation. In this case, you use the name of the parameter and the value of the argument together in something that looks like an assignment statement.

sayHi( name = "George" )

For functions with many parameters, you can provide the first set of arguments by position and the last set by keyword, but once you start using keywords, you can’t stop.

In the next example, we examine code that shows the power of both default parameter values and keyword arguments. Consider the following function which has a lot of parameters, some of which have default values:

def circle(x, y, scale, fill=True, fcolor = 'black', pcolor = 'black', psize = 1):
    # lots of code here.

Then the following calls are legal:

circle( 100, 0, 1.0, psize = 3 )
circle( x=100, y=0, scale=1.0, pcolor='red' )
circle( 100, 0, 1.0 )
circle( scale = 1.0, x = 1, y = 2 )

And the following calls are ILLEGAL:
1 circle( 100, 0, 1, pcolor='red', 1 )
Line 1 is illegal because it uses a positional argument (1) after a keyword argument.

Line 2 is illegal because it fails to provide a value for scale. Since scale does not have a default value, the caller must supply an argument for it. Line 3 is illegal for the same reason.