1 Administrative Topics

• Change in my office hours the week: Office hours for the Week of Feb 15-19: M 1-5, T 1-3:30, F 12-2

• No need for partners in proj 2.

• Personal folders are for back-ups. Academics folders are for hand-in.

• Unary operators have precedence lower than ** on the left, and higher than exp on the right

• The reason rounding didn’t work was that the computer can’t store 0.2, it can store only 0.200000001.

2 More Math Functions

There are several built-in math functions:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>absolute value of x</td>
</tr>
<tr>
<td>int(x)</td>
<td>$x$ converted to an integer</td>
</tr>
<tr>
<td>float(x)</td>
<td>$x$ converted to a float</td>
</tr>
<tr>
<td>pow(x,y)</td>
<td>$x$ to the power $y$</td>
</tr>
<tr>
<td>round(x)</td>
<td>$x$ rounded to 0 digits after the decimal point</td>
</tr>
<tr>
<td>round(x,n)</td>
<td>$x$ rounded to n digits after the decimal point</td>
</tr>
</tbody>
</table>
3 Executing Assignment Statements – a Review

The two steps to executing an assignment statement are

1. Evaluate the right hand side

2. Look for the left hand side in the symbol table. If it is not there, we add a row (and put the name of the symbol in the name column). Put the value from the right hand side into symbol table in the appropriate row.

4 Writing Our Own Functions that return values

We have written functions to draw scenes with the turtle, 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.
What about mathematical functions that are not in the math module? What if I want to cube a number?

I will create a file lec6_fcns.py. In it, I place

```python
# Function cube - cubes a number
# input:
# number : a number (<float> or <int>)
# output: <int> or <flt> (the number cubed)
def cube(number):
    result = number * number * number
    return result
```

Now we have our function. Let’s try it out and investigate how memory is changing.

## 5 Executing Functions

I start Python in interactive mode. To use the functions in lec6_fcns.py, I need to import it.

I type `import lec6_fcns`

The main symbol table is updated to reflect this. Note that I usually draw the arrow outside the table, and then include the table within the module with entries for each function. For the purposes of these notes, I am going to use a slightly simpler notation, which places the text “<module>” inside the table. I am doing this because the typesetting language I am using to write these notes is much much more amenable to this, simpler notation. Thus, the symbol table (after the import) now looks like this:

```
main
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lec6_fcns</td>
<td>→ &lt;module&gt;</td>
</tr>
</tbody>
</table>
```

Let’s step through the following example.

At the prompt, type `a = lec6_fcns.cube(2)`
When I press return, Python will execute this line of code. I like to write
the line of code currently being executed directly underneath the appropriate
symbol table, i.e.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lec6_fcns</td>
<td>→ &lt;module&gt;</td>
</tr>
<tr>
<td>a = lec6_fcns.cube(2)</td>
<td></td>
</tr>
</tbody>
</table>

Now, let’s evaluate the statement. Do you remember how assignment works?
First, the right hand side is evaluated. In our case the function cube is called.
It is given as input the value 2. The input is “passed in” to the function. A
new symbol table is created called cube. A variable named number is created
and it is assigned the value given in the argument. The new line is added to
the cube symbol table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>2&lt;int&gt;</td>
</tr>
</tbody>
</table>

Next, the line result = number*number*number is executed. Since this line
is part of the cube function, we write it underneath the symbol table for the
cube function:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>number<em>number</em>number</td>
</tr>
<tr>
<td>number</td>
<td>2&lt;int&gt;</td>
</tr>
<tr>
<td>a = lec6_fcns.cube(2)</td>
<td></td>
</tr>
</tbody>
</table>

This is an assignment statement. How do we execute assignment statements?
Evaluate the r.h.s.
Look for the symbol in the current table. If it isn’t there, create a row for it.
Put the value from the r.h.s. into the value column.

So, we add a row for `result` with the value 8. Then, the assignment statement has finished execution:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>2&lt;int&gt;</td>
</tr>
<tr>
<td>result</td>
<td>8&lt;int&gt;</td>
</tr>
</tbody>
</table>

Now the function gets ready to return. It returns the value in `result`. First, we add the statement `return result` underneath the cube table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>2&lt;int&gt;</td>
</tr>
<tr>
<td>result</td>
<td>8&lt;int&gt;</td>
</tr>
</tbody>
</table>

```
return result
```

Now, to be extra clear that it is the value being returned, let’s cross out the variable name and replace it with the value:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>2&lt;int&gt;</td>
</tr>
<tr>
<td>result</td>
<td>8&lt;int&gt;</td>
</tr>
</tbody>
</table>

```
return 8 <int>
```

Now, we need to indicate that the value is being thrown back to the caller (i.e. the code in main). Let’s represent that by going back to the code under the main symbol table and replacing it with the return value:
Now, `cube` has finished executing. We do not need its memory any more. So, we erase the cube symbol table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lec6_fcns</td>
<td>→ &lt;module&gt;</td>
</tr>
<tr>
<td>a = lec6_fcns.cube(2)</td>
<td>8 &lt;int&gt;</td>
</tr>
</tbody>
</table>

Now, we continue evaluating our original assignment statement. We have finished evaluating the right hand side of the assignment statement `a = lec6_fcns.cube(2)`. We finish executing the assignment statement by updating the main symbol table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lec6_fcns</td>
<td>→ &lt;module&gt;</td>
</tr>
<tr>
<td>a</td>
<td>8 &lt;int&gt;</td>
</tr>
</tbody>
</table>

Phew! We are done.

Let’s reiterate what happens when a function, like `cube`, is executed:

1. A symbol table is created for the function
2. One row is added to the symbol table for each parameter. The values are taken from the caller.
3. The code in the function is executed.
4. Any return value is sent back to the caller.
5. The function exits and the symbol table is deleted.
6. “Control” is returned to the caller.

Let’s do another example.

How cold out is it today? Do you “think in celsius” or “think in fahrenheit”? I think in fahrenheit but would like to develop an intuition for celsius. The conversion isn’t all that easy in our heads, so I have written two functions – one that converts from celsius to fahrenheit and another that converts it back.

Here is the one converting from celsius to fahrenheit:

```python
# celsius2fahrenheit : converts a temperature in 
# celsius to fahrenheit
# input:
# c : the temperature in degrees celsius
#     (<int> or <float>)
# output:
# the temperatue in degrees fahrenheit (<float>)
def celsius2fahrenheit(c):
    f = c*9.0/5.0 + 32.0
    return f
```

and here is the other:

```python
# fahrenheit2celsius : converts a temperature in 
# fahrenheit to celsius
# input:
# c : the temperature in degrees fahrenheit
#     (<int> or <float>)
# output:
# the temperatue in degrees celsius (<float>)
def fahrenheit2celsius(f):
    c = (f-32.0)*5.0/9.0
    return c
```