1 Administrative Topics

- Study for the quiz. It will be on assignment and symbol tables.

2 Overview

The basic idea today is to talk about types and tables. data types – int and str and float functions – fcn modules – tables with functions (difference between import X and form X import *)

So, can I demo turtle stuff and still talk about all of this? This is about table-contents when using predefined functions.

On Friday, talk more about parameters and assignment and math

3 Numeric and String Types

Each value in a symbol table has a type associated with it. So, from now on, we will add type information to the value field.

We know some types already

<table>
<thead>
<tr>
<th>1</th>
<th>1.0</th>
<th>'hi'</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>float</td>
<td>str</td>
</tr>
</tbody>
</table>
Let’s review a bit. Start with a main symbol table. Perform the operation

\[ A = 1 \]

We need to fill in the first row of the table. The name is A. The Value is 1. The type is int. See Fig. 1

```
Name | Value
---   |------
A     | 1 <int>
```

Figure 1: main symbol table

Python has a neat feature which tells you the type of any value.

```
print type(1)
```

What do you think will be printed when I type:

```
print type(A)
```

Because A has the value 1, it has the type <int>. A is just the label for the place in memory where this value is stored.

What happens when I type

```
B = 1.5
```

[We add a new row to the table. We learn that 1.5 is a float. The new row is B, 1.5 <float>]

So the table is now like in Fig. 2.

What about after
A = "hi"

Is a new row created? No.
The new table appears as in Fig. 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 &lt;int&gt;</td>
</tr>
<tr>
<td>B</td>
<td>1.5 &lt;float&gt;</td>
</tr>
</tbody>
</table>

Figure 3: main symbol table

3.1 Modules and Functions

Variables are not the only things stored in the Python memory model. Modules and functions are also stored in symbol tables.

A module is a named collection of functions. For example, there is a module named turtle which provides the turtle graphics commands you will be using in lab today.

Suppose we are running Python in the interpreter model. To have access to the module, we must import it. e.g. import turtle. This command adds
a row to the main symbol table whose name is turtle and whose value is an arrow pointing to a module.

Sidenote: We draw “simple” or “basic” types as entries directly in the value field for an entry. More complicated types, like modules and functions live outside the table, and the entry is actually an arrow to the value.

The module type contains a symbol table, which contains entries for each of the functions defined in that module. See Figure ?? for a depiction.

Figure 4: Main symbol table after the turtle module has been imported. The module has its own symbol table, containing entries for each of the turtle functions/commands.

To access the turtle functions, we must use what is called “dot” notation. I.e. to execute the forward method, we write `turtle.forward()`. I like the think of the dots as providing the links in a chain from table to table.

On the other hand, if we did what you did in lab last week, i.e.
from turtle import *

then all of the turtle functions are placed directly into the main symbol table. This makes the table cluttered. This isn’t a good idea, because it restricts what names you can use for variables. There is nothing to prevent you from executing a line like:

```
left = 'left'
```

which will replace the turtle function with a string. Ick!

### 3.2 Style

Our coding style and the internal organization of the memory should be well-organized. It should be modular. The dot notation helps us with this. Write code in a module.

The code inside function definition will have its own table while it is being executed, but it can also “see” the module’s table. But the code in a function can’t manipulate the module’s symbol table. For the most part this is good news. Because it means we can have the import statement in the module, and not require it in every function. It also forces us to use good coding style. All the code in a function is self-contained. To pass information to a function, it is best to use parameters.

We write the “main” code at the bottom and indicate this by putting in a comment. All module, level code (other than the imports) belongs AFTER this comment.

I do a code demo. The code is posted on the web.

### 4 How to Study for the Quiz

Read over the notes. It will be a question about symbol tables. For example I may ask you to draw the main symbol table after three assignment statements and an import statement have been executed.