Just when you had become an expert at string slicing, you discovered another sliceable data type: lists. However, lists are different from strings in that they are mutable. Not only can we slice a list, but we can also change its contents. The purpose of the lab is to introduce you to these new features, and demonstrate just how powerful the list type can be.

Getting Credit for the Lab. There is exactly one file to download for this lab: windows.py. You will need this lab to import it, but you do not need to do anything else with it. Everything in this lab will be done via the Python interactive prompt, just as you did with Lab 1. You will write all of your answers on this piece of paper.

When you are done, show this handout to your instructor, who will record that you did it. As with previous labs, if you do not finish during your section, you have until the beginning of lab next week to finish it. You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

1. Windows

As we said in class, a type is a collection of values together with its operations. Normally when we think of values, we think of things like numbers, or True and False. But these values could take on any form. A value could be a window on your screen. That is the motivation for the type Window. In this part of the lab, you will create and manipulate Window objects.

All of the types that we have seen so far have natural ways to represent their values. We represent ints by whole numbers; we represent strings as characters inside double quotes. But there is no natural way to represent values of type Window. For this type, we have to use a special function – called a constructor – to create a new object of that type.

The name of a constructor function is generally the same name as the type. The type Window is provided by the module window (which you should have downloaded). Import this module and enter the assignment statement

```python
>>> w = window.Window()
```

What happens? What is stored in the variable w?
1.1. **Window Attributes.** As we discussed in class, attributes are named variables that are stored inside an object. You can use attributes in expressions, or even assignment statements.

One of the interesting thing about GUI objects is that assigning new values to an attribute can have visible effects. In the table below we have a list of expressions and assignment statements. **Enter these into the Python shell in exactly the order presented.** If it is an expression, give (or guess) the value that Python returns. If it is an assignment statement explain (or guess) the result of the assignment.

<table>
<thead>
<tr>
<th>Statement or Expression</th>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Reason for Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.x = 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.y = 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.width = 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.height</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.title</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w.title = 'window'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.2. **Window Methods.** Window objects also have methods. Unlike string methods, which are functions, these methods are procedures that do something to the object. Execute calls for the three methods shown in the table below. Explain what happens when you call them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result When Called</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.beep()</td>
<td></td>
</tr>
<tr>
<td>w.iconify()</td>
<td></td>
</tr>
<tr>
<td>w.deiconify()</td>
<td></td>
</tr>
</tbody>
</table>
1.3. **Positioning a Window.** You have already seen the Window size and position is controlled by attributes. For the Window whose name is in \( w \), look at the attributes for the \( x \)-coordinate and \( y \)-coordinate. Write their values here:


Next, create a second Window object, storing its name in another variable. This should pop up a new window. What are the \( x \)-coordinate and \( y \)-coordinate for this window?


Is there something unusual about how screen coordinates work? What do you notice about the difference in coordinates between the two windows?


1.4. **Resizing a Window.** Create a new Window, storing its name in variable \( w \). Try resizing the Window with your mouse to make it bigger. Look at the attribute function `resizable` of \( w \) to see whether the Window is resizeable. What is the value of this attribute?

Now execute the assignment

\[ w\text{.resizable} = \text{False} \]

Try resizing the Window whose name is in \( w \) with your mouse. Is it resizeable now?

Assign the attribute `resizable` to True. Once you have done that, call the procedure

\[ w\text{.setMaxSize}(50,100) \]

What happened to your Window?

What happens when you try to resize this Window?
2. **List Expressions and Commands**

This part of the lab will take place in the Python interactive prompt, much like the first two labs. You do not need to create a module. First, execute the following assignment statement:

```python
lablist = ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

Like a string, this is a list of individual characters. Unlike a string, however, the contents of this list can be changed.

Enter the following statements **in the order they are presented**. Many of the commands below are always type in expressions, Python will immediately display the value; the commands below are all followed by a print statement showing the new contents of the list. Each case, describe what you see and **explain the result**.

<table>
<thead>
<tr>
<th>Commands</th>
<th>Result/Explanation</th>
</tr>
</thead>
</table>
| `lablist.remove('o')`  
`print lablist` |  |
| `lablist.remove('x')` |  |
| `pos = lablist.index('o')`  
`print pos` |  |
| `pos = lablist.index('B')` |  |
| `lablist[0] = 'J'`  
`print lablist` |  |
| `lablist.insert(5, 'o')`  
`print lablist` |  |
| `s = lablist[:]`  
`print s` |  |
| `s[0] = 'C'`  
`print s`  
`print lablist` |  |
| `a = '-'.join(s)`  
`print a` |  |