ICS3U: Multidimensional Lists

We have seen how lists can hold a variety of objects, including strings, integers, floating-point values and so on. *Any* Python object can be stored in a list. This includes other list-like types, such as tuples and even other lists. Here is some code that stores a sequence of tuples in a list. Even though there are eight integers in total, the list `L` still contains only four elements. It just happens that each element itself contains two integers. Using `len` on `L` will not count the individual values.

```
>>> L = [(1, 2), (3, 4), (5, 6), (7, 8)]
>>> len(L)
4
```

There are many ways in which we can reference the values stored inside of the list. Using standard indexing, we can obtain an entire tuple.

```
>>> L[0]
(1, 2)
>>> L[-1]
(7, 8)
```

Slicing will return a list consisting of one or more tuples. It is important to know that a slice will always be a list, even if it only contains a single tuple.

```
>>> L[:2]
[(1, 2), (3, 4)]
>>> L[3:]
[(7, 8)]
```

Let's say that we want to obtain the individual values inside of the tuples, rather than the tuples themselves. One way to do this is to obtain a tuple (via indexing the list, looping through elements, or some other process) and then indexing the tuple.

```
>>> T = L[3]
>>> T
(7, 8)
>>> a = T[0]
>>> b = T[1]
>>> a
7
>>> b
8
```

Another way is to assign both values in the tuple directly to the variables a and b at once.

```
>>> T = L[3]
>>> a, b = T[0], T[1]
>>> a
7
>>> b
8
```

We can also combine the indexes of the tuple with the index of the list. References like this have the form `[main list index][inner list index]`.

```
>>> a = L[3][0]
>>> b = L[3][1]
>>> a
7
>>> b
8
```

Which method we choose is mostly a matter of style, not of function.

To demonstrate the use of a two-dimensional list, we will make a small "drawing" program that will allow us to create a 10 × 10 image, similar to the one below.

```
....XX....
...XXXX...
..XX..XX..
.XX....XX.
XXXXXXXXXX
X........X
X..XXXX..X
X..X..X..X
X..X..X..X
XXXXXXXXXX
```

Initially, a "blank" image will consist only of '.' characters. The user will select a position between (0, 0) and (9, 9), and the character at that position will change from a '.' to an 'X'. If there is already an 'X' at that position, it will revert back to a '.' instead.

First, we need to set up a two-dimensional grid. We can do this by creating an empty list, and then adding 10 sublists to it, each representing a row in the picture. Here is a function that does this.

```
def blank_picture():
   pic = []
   for row in range(10):
      pic.append(["."]*10)
   return pic
```

Next, we need to be able to display the picture as a 10 × 10 image. If we output the list as-is, it will look something like this.

```
>>> pic
[['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.',
'.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'], ['.', '.',
'.', '.', '.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.', '.', '.',
'.', '.'], ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.',
'.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.',
'.', '.', '.', '.']]
```

Obviously, this is not what we want. To display the image in a more human-friendly format, we can use a simple `for` loop to process each row in the image. Processing involves joining the contents of a row into a single string so that it can be displayed on a single line.

```
def draw_picture(pic):
   for row in pic:
      chars = "".join(row)
      print(chars)
```

This will give us an image something like this.

```
..........
..........
..........
..........
..........
..........
..........
..........
..........
..........
```

Next, we will need to obtain a valid coordinate between (0, 0) and (9, 9) for the user to toggle between a '.' and an 'X'. We would like the user to be able to enter both values at once. We can achieve this using `split`, and then checking whether both values can be represented as positive integers. If they are, we use a list comprehension to convert the values to integers and check whether they fall between 0 and 9.

```
def get_pos(prompt):
   while True:
      pos = input(prompt).split()
      if len(pos) == 2:
         if pos[0].isdigit() and pos[1].isdigit():
            pos = [int(n) for n in pos]
            if pos[0] >= 0 and pos[0] <= 9 and pos[1] >= 0 and pos[0] <= 9:
               return pos[0], pos[1]
      print("Invalid position.")
```

Note that the code will continue to loop unless *all* criteria are met. Finally, our main program consists of a loop that repeats indefinitely, allowing the user to toggle between characters.

```
pic = blank_picture()
while True:
   draw_picture(pic)
   r, c = get_pos("Enter the coordinates to toggle: ")
   if pic[r][c] == ".":
      pic[r][c] = "X"
   else:
      pic[r][c] = "."
```

Each character in the picture is accessed by specifying the values of `r` and `c` in `pic[r][c]`. The first value, `r`, specifies the row of the image while the second, `c`, specifies the column. For example, if `r` was 2 and `c` was 3, then the statement `pic[r][c]` would set the value at position (2, 3).

Here is the full program. Try running it, and create something interesting.

```
def blank_picture():
   pic = []
   for row in range(10):
      pic.append(["."]*10)
   return pic

def draw_picture(pic):
   for row in pic:
      chars = "".join(row)
      print(chars)

def get_pos(prompt):
   while True:
      pos = input(prompt).split()
      if len(pos) == 2:
         if pos[0].isdigit() and pos[1].isdigit():
            pos = [int(n) for n in pos]
            if pos[0] >= 0 and pos[0] <= 9 and pos[1] >= 0 and pos[0] <= 9:
               return pos[0], pos[1]
      print("Invalid position.")

pic = blank_picture()
while True:
   draw_picture(pic)
   r, c = get_pos("Enter the coordinates to toggle: ")
   if pic[r][c] == ".":
      pic[r][c] = "X"
   else:
      pic[r][c] = "."
```