In the lab today, you will continue to work with lists and loops, and learn how to apply those concepts to matrix multiplication.

Problem 1.

Write a function with the signature `list1_start_with_list2(list1, list2)`, which returns `True` iff `list1` is at least as long as `list2`, and the first `len(list2)` elements of `list1` are the same as `list2`. Note: `len(lis)` is the length of the list `lis`, i.e., the number of elements in `lis`.

First write the function without using slicing (slicing means saying things like `list1[2:5]`), and using a loop.

Problem 2.

Write a function with the signature `match_pattern(list1, list2)` which returns True iff the pattern `list2` appears in `list1`. In other words, we return `True` iff there is an `i` such that $0 \leq \text{i} \leq$ (list1)-len(list2) and

```
list1[i] = list2[0]
list1[i + 1] =list2[1]
.
.
.
list1[i + len(list2)  1] = list2[-1]
```

For example, if `list1` is `[4, 10, 2, 3, 50, 100]` and `list2` is `[2, 3, 50]`, `match_pattern(list1, list2)` returns `True` since the pattern `[2, 3, 50]` appears in `list1`

Problem 3.

Write a function with the signature `duplicates(list0)`, which returns `True` iff `list0` contains at least two adjacent elements with the same value.

Problem 4.

In Python, you can use a *list of lists* to store a matrix, with each inner list representing a row. For example, you can store the matrix

$$\begin{pmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{pmatrix}$$

by storing each row as a list: `M = [[5,  6, 7], [0, -3, 5]]`. For ease of reading, since Python allows for line breaks inside brackets, you can write it as follows:

```
M = [[5,  6, 7],
     [0, -3, 5]]
```

You can use `M[1]` to access the second row of `M` (i.e., `[0, -3, 4]`), and you can use `M[1][2]` to access the third entry in the second row (i.e., `5`).

Part (a)

Write a function with the signature `print_matrix_dim(M)` which accepts a matrix M in the format above, and prints the matrix dimensions. For example, `print_matrix_dim([[1,2],[3,4],[5,6] ])` should print `3x2`.

Part (b)

Recall that the product of a matrix M of dimension (n, m) and a vector v of dimension $(m, 1)$ can be written as

$$Mv = \begin{pmatrix} (\sum_{i=1}^{m} M_{1,i}v_i) \\ \dots \\ \dots \\ (\sum_{i=1}^{m} M_{n,i}v_i) \end{pmatrix}.$$

Write a function with the signature `mult_M_v(M, v)` which returns the product Mv of a matrix M and a vector v. Vectors are stored as lists of `float`s.

To write this function, you will need to create a new vector. Here are two ways to create a new vector (stored as a list) with 10 zeros in it:

```
ten_zeros1 = [0]*10

ten_zeros2 = []
for i in range(10):
  ten_zeros2.append(0)
```

Problem 5.

As you saw in the last problem, list elements may be lists themselves. A list with list elements is called a *nested list.* An example is:

```
pets = [["Shoji", "cat", 18],
        ["Hanako", "dog", 15],
        ["Sir Toby", "cat", 10],
        ["Sachiko", "cat", 7],
        ["Sasha", "dog", 3],
        ["Lopez", "dog", 13]]
```

We can access each element of list pets using its index:

```
>>> pets[3]
["Sachiko", "cat", 7]
```

We can also access elements of the inner lists. For example, since `pets[3]` refers to a list, we can use `pets[3][2]` to access the element at position 2:

```
>>> pets[3][2]
7
```

This is saying that element 2 of element 3 of the list pets (the age of the cat named “Sachiko”) is 7.

Part (a)

Write a function that prints each list from the list `pets` on a separate line.

Part (b)

Write a function that prints the second element (the species) of each inner list in list pets on a separate line.

Part (c)

Write a function that examines the list `pets` and computes the sum of the ages of the animals in the list. Ages are the third element of the inner lists.

Part (d)

Write a function that examines the list `pets` and computes the number of dogs in the list.