1. (12 pts) Consider the following snapshot of an Alice world in which there are three objects in front which are (from left to right) hare, penguin, and tortoise. There is a fourth object, a table, that is at least five meters behind all the animals. The hare and penguin are 3 meters apart. The penguin and tortoise are 3 meters apart. Imagine the three of them are standing on an invisible line. The procedures, properties and events are all shown on two pages.
A) List the events in this world.

B) List the event handlers in this world.

C) Explain what happens in the world when the Play button is pressed.

D) After the world has played for 30 seconds, the user clicks on one object and moves the object close to the table. Explain what would happen in each case after the object is moved to the table.
   A. Suppose the object moved is the tortoise. What happens?

   B. Suppose instead the object moved is the lemur. What happens?

   C. Suppose instead the object moved is the penguin? What happens?
2. (4 pts) Consider the following array of friends shown below as a picture and as a property, and the following code.

Explain what happens when this code runs.
3. (11 pts) Consider the following Alice word, which includes a 3D text called score, and three objects: soccerball, volleyball, and pumpkinHead. The world is a game with the objects moving around. Whenever the user clicks on one of the objects they get one point. A procedure updateScore is also shown along with two events. MyFirstMethod is not shown but is empty with no code. Also not shown is how the objects move around randomly.

A. Rewrite the mouseClicked Event so that when the user clicks on the soccerball, they get one point; when they click on the volleyball they get 2 points, and when they click on the pumpkinHead they get 5 points.
B. Suppose the change is made in Part A, and this new event is added. Explain what happens now when the game runs.
4. Assume there is an array called `penguins` that has 10 penguins in it. Write the Scene procedure `makeAllButNumInvisible` that has one number parameter called `number`. This procedure makes all but “number” penguins invisible. For example in the first picture below, there is an array of penguins that has ten penguins in it. After calling `makeAllButNumInvisible` with number set to 3, then only three random penguins are still visible as shown in the second picture. If you ran the program again, say with number set to 5, then five random penguins would be visible. **YOUR PROGRAM SHOULD STILL WORK IF THE NUMBER OF PENGUINS IN THE ARRAY changes.** In writing this you will first write a Scene function called `numberVisible` that will return how many penguins in the array are visible. You will also write a Scene procedure called `makeInvisible` that makes one randomly chosen visible penguin, invisible.

A. First write the Scene function `numberVisible`. It returns the number of penguins in the array `penguins` that are visible. Your program should still work if the size of the array changes.
B. Write the Scene procedure `makeInvisible` that randomly picks one visible penguin in the array and makes it invisible.

```plaintext
declare procedure makeInvisible
```

C. Write the Scene procedure `makeAllButNumInvisible` that has one parameter `number`. This procedure makes all but “number” penguins invisible in the penguins array. You must call `numberInvisible` and `makeInvisible` in this procedure.

```plaintext
declare procedure MakeAllButNumInvisible with parameter: WholeNumber number
```
5. (10 pts) Consider the following Alice world, that has one peacock and an array of penguins, falcons and chickens. The peacock is NOT in the array. There is a different number of penguins compared to the number of falcons compared to the number of chickens in the array. You are suppose to figure out which type of the three types of birds appears most in the array, and then replace one of those birds in the array with the peacock. Note your program should work regardless of which birds are in the array.

Here is myFirstMethod. It figures out how many birds of each type are in the array. Then it has all the birds in the array jump up and down. Then for the bird type that has the most birds in the array, it replaces one of them with the peacock, that is, it puts the peacock in the array in place of that bird. Note it is calling the function numberOfTypeBird and the procedure changeBirdInArray.

You will write some of the functions/procedures needed to run this program.
A. Write the Scene function named `numberOfTypeBird` that is given a parameter of type `Flyer` with the name `birdType`. You should return the number of Flyers of type `birdType`. For example, if `birdType` is `chicken`, then using the array on the previous page you should return 4, since there are four chickens in the array. How do you tell what type a bird is? Every object has a “toString” method that will give you the name of the object as a string of characters. The chickens are named `chicken`, `chicken2`, `chicken3` and `chicken4`, so you just need to make sure the `toString` of the name of the object starts with the word `chicken`. For example if you were going to compare the string name of `chicken2` with the string name of `chicken` you would ask:

```javascript
if (this.chicken2.toString().startsWith(this.chicken.toString()))
```

Complete the function below.

```javascript
function numberOfTypeBird
```

B. Write the Scene function named indexLocationOfBird, that has a parameter named oneBird that is of type Flyer. This function should return the index location of where oneBird is located in the Scene array named birds. You may need to use a counter and count the index locations as you step through the array until you find the bird. Then return the index location of that bird in the array.

```
declare WholeNumber function indexLocationOfBird with parameter: Flyer oneBird
```
C. Write the Scene procedure named changeBirdInArray that has two parameters, both of type Flyer named birdOne and birdTwo. This procedure finds the location of birdOne in the array and replaces that bird with birdTwo at that location. This procedure should call the function IndexLocationOfBird to calculate the index location of birdOne in the array.

declare procedure changeBirdInArray with parameters: Flyer ‹birdOne›, Flyer ‹birdTwo›