Using Lists and Functions to Create Dialogue

Hello, tiger! (ENTER to continue)
Using Lists and Functions to Create Dialogue

Alice's default “say” method allows you to create dialogue, but it has limitations.
For example, what if you wanted to give a user as much time as they needed to read a character's lines, but also wanted another user to be able to skip through them very quickly?

One way to do this is to use a list, a counter, and a while statement.
You'll learn to store a character's lines in a list, then let the user select which line to display based on a count variable.
First, go to “add objects”.
Go to “Animals” and add the Tiger and Boar objects to your world. These will be our conversation partners.
Make sure they're reasonably close to each other, like above.
Now we'll create some dialogue for the boar. Go to the boar's properties and create a new variable. Make it a **string**, check **make a List**, and call it **dialogueList**.

Add the five lines shown in the image by clicking “new item” and then clicking on “default string” next to each item to edit it.
Now for the tricky part: making the boar actually speak. We'll do this by having it say whatever line from its dialogue corresponds to the value of a specific variable. By changing the value of this variable, we can change the line said by the boar.

Start by making a variable in world (NOT world.my first method!) called count. Make it a number and give it a starting value of 0.
Next, create an event to increment \textbf{count} by 1 when the \textbf{enter} key is typed. (Drag-and-drop \textbf{count} from the world properties page into the box that says \textless None\textgreater.)

We will use count to keep track of the boar's lines. Now that we can change its value, it's time to make the boar talk. Create a while loop.
Go to the boar's methods, then drag `say` into the `while` loop. Choose “Hello” to start.

```plaintext
While true
  boar say Hello more...
```

Go to the boar's properties and drag “dialogueList” into the box where it says “hello”. Select “ith item from list” on the menu that appears, then go to expressions and select `count`.

This means the boar will say the item from `dialogueList` that corresponds to `count`. When `count` is 1, the boar will say item 1, etc.

```plaintext
While true
  boar say item world.count from boar.dialogueList more...
```
If you press play now, the boar should speak!

Hello, tiger!

The line it's saying should change when you press enter. However, press enter too many times, and... !!!

Alice has detected a problem with your world: what must not be null.
As we have it set up now, if you increase count until it is a larger number than the number of the last item in the list, then Alice will still search the list for the item with the corresponding number, but will not find it and will crash.

We need to change the while loop so that it will stop executing once count passes the last item in dialogueList.

To do this, change the while loop's condition (the box next to while) so that the loop will only execute when count is less than 6 (the number of lines in dialogueList). Drag count in (from properties, under world) and select "count <" then type in 6.
Congratulations, the world should now be working properly!
Now the boar is speaking properly. However, if you think you're up to some more challenging work, you can make this a proper conversation.
Go to the tiger's properties this time and create a new `dialogueList` variable. Make it a list of strings.

Add 12 items. Write responses to the boar's dialogue, but only in **odd** items. Leave the rest as default string.
Now go to the boar's properties, delete its old `dialogueList`, and make a new one.

Add 12 items, just like before. Put the boar's old dialogue back in, but only in `even` items. Leave the rest as default string.
Since 12 total lines (6 from the boar and 6 from the tiger) will now be said, we're going to increment count up to 12 this time.

However, we will print boar lines only when count is even, and tiger lines only when count is odd.

First, let's make an isEven function in `world` to check if count is even. Set its return type to `boolean`.

Create a parameter for `isEven`. Make it a number called `input`.
A common way of determining if a number is even is to see if it leaves a remainder when divided by 2. The **IEEERemainder** function under **world** lets you do this in Alice.
Drag an `a==b` (a is equal to b) function into the box next to “if” where it says “true”. For the first option, select `input` (under expressions), and for the second option, enter 0.

Next, drag an `IEEERemainder of a/b` function into the box where it says `input` in the if/else statement. Select 2 from the menu.
Your if/else statement should now look like this.

If it does, your function is complete!
Now to tie it all together. Go back to `world.my first method` and delete “count < 6” from the while loop you have. Add a second loop in the same format as the loop for the boar's dialogue in which the tiger says lines from its own dialogue list.

Time to use that new function! Go to `world functions` and drag `isEven` over “true” in both loops. In the loop with the tiger, click the second drop-down arrow by `isEven`, which will open up a menu.
Click the drop-down arrows next to “world.isEven” in each condition to change “input” to count (under expressions).

Finally, drag both loops into a third while loop. Drag count from world properties into the condition of this loop (where it currently says “true”), choose “count <”, and enter 12.
Congratulations!

Hello, tiger! (ENTER to continue)