What is a function?

• A function is a subset of code that can accept inputs (called parameters) and return outputs
• Functions help simplify code
• Examples of built-in functions are getHeight and getDistanceTo
• In this tutorial, we will write our own functions
Parts

• Part 1: create function that compares two objects and returns the shorter
• Part 2: create a seagull function that changes the color of the seagull when it is clicked on
• Part 3: create a snowman function that turns the snowman a random color when it is clicked on
• Part 4: create a function that returns the color of the snowman, as a string

PART 1

• Create an Alice world and add objects
• Create a scene function that can be used to compare the height of objects
• Create parameters
Start an Alice world with grass and add lioness (quadruped), bunny (biped), seagull (flyer), and snowman (biped).

Click on “edit code” and add a Scene function.

Click on “unset” under return type and select “Gallery Class.”
Select “SJointedModel” from the list.
Notice that all four objects have checks next to their names, meaning they are included in this category.

Click OK, and name the function “objectThatIsShorter”
What does this function do?

- Functions calculate and return an answer
- Because we chose “SJointedModel”, our function must return an “SJointedModel”, which is a category all four of our objects fall into
- Our function will compare the snowman and the seagull and return the shorter one

Drag in an if statement and choose “true”
Click on the dropdown menu by “true” and choose Relational (DecimalNumber) \( \rightarrow ??? \!< ??? \!
\rightarrow 1 \rightarrow 1 \)

Select this.seagull in the left hand drop-down menu, and click on the functions tab

- If you hover your mouse over functions, Alice will put boxes around places you can drag them to
- Drag this.seagull getHeight onto the first 1.0
Repeat for the snowman’s `getHeight` function. Your code should look like the code below:

We want to return the shorter object, so if the if statement is true, we should return `seagull`.
Otherwise, we want to return `snowman`. 
Now we need to call our method

- Click on myFirstMethod. Right now it should only have a do in order block
- Click on the seagull and drag in “say”
- Have the seagull say “I am shorter!”

But we want the shorter object to say that, not the seagull

Select “objectThatIsShorter” from this→Functions and drag it onto the seagull
• Run your code. The shorter object should say “I am shorter!”.
• Try resizing the snowman and the seagull so the other is shorter and rerun.
• But what if we want to test an object other than the seagull or the snowman? Parameters

Go back to the objectThatIsShorter function and click “Add Parameter”
Under value type, select Gallery Class, then click on SJointedModel

Fill in the name as object1 and check the box (this tells us that we called this function somewhere and we will have to go and fix it, since now we want the main function to give this one a parameter.
Repeat the process for another parameter, named object2

Now to actually use the parameters:

- Drag object1 over “this.seagull” and object2 over “this.snowMan”
Now go back to myFirstMethod

• Your code should look like this:

```
- this
- objectThatIsShorter
- object1: unset, object2: unset
```

• Notice the two red “unset” places? Those are where we need to select two objects to compare

• When we checked that box when creating parameters, we were promising to come back and fill these two places in with real objects

Select two objects and run your code! You should try it multiple times to make sure it works with all the objects in your world
PART 2

• Create a seagull function
• Create a seagull variable to keep track of color
• Change the seagull color in response to a click
  – If the seagull is white, we want it to be blue
  – If it is blue, change to red
  – If red, change to green
  – If green, change back to white

First, we want to create a new seagull function

• Go to the dropdown menu to the right of the scene image
• Select seagull→Add seagull function
We want this function to change the seagull’s color, so set return type to Other Types ➔ Color

Name the function changeColor and click OK
If the seagull is white, we want to turn it blue

• How do we know what color the seagull is?
• In Alice 3, you CANNOT compare colors
• Instead, we will create a seagull variable (called Properties) that will hold the name of the color that the seagull currently is
• Word variables are called Strings

• Go to the dropdown menu
• Select Seagull ➔ Add Seagull Property
• Set the value type to “TextString”
• Name the variable “seagullColor”
• Click initializer and fill in “White”
  – This will be the word the property is set as when the world begins

The property popup should look like this
Now go to the changeColor function

- If you have trouble finding it, select this.seagull from the dropdown menu on the far left
- Click on the functions tab
- At the top you should see this.seagull changeColor
- Click the edit button to the left of it
- Drag in an if statement and select “true”

Click on true and select TextString Comparison ➔ ??? contentEquals ??? ➔ this.seagullColor ➔ Custom TextString

This will let us say “if the word in this.seagullColor (which is “White” right now) is equal to a custom string”
A box will pop up for you to fill in the custom string

- We want to check if the seagull is white, so fill in White
- Make sure to keep the capitalization the same as in the seagullColor property!!

If the seagull is white, we want to change it to blue

- When we create a property, Alice 3 automatically makes a procedure and a function to go along with it: setPropertyName and getPropertyName
- So under the Procedures tab you will see setSeagullColor and under the Functions tab you will see getSeagullColor
If the seagull is white, we want to change it to blue

- Drag in `setSeagullColor` from the procedures tab and select Custom TextString
- Fill in “Blue”
- We still want to return a Color type, so drag in the return statement from the bottom bar, and select BLUE
- (picture on next slide)

Your code should look like this
We are one fourth of the way there

- If the seagull is white, we want it to be blue
- If it is blue, change to red
- If red, change to green
- If green, change back to white

Drag another if block under the else and repeat as before

- This time, select TextString Comparison \(???\) contentEquals \(???\) this.seagullColor \(\rightarrow\) Custom TextString and fill in “Blue”
- Drag in setSeagullColor and fill in “Red”
- Return RED
Drag another if statement under the else

• This time, select TextString Comparison → ?? contentEquals ??? → this.seagullColor
  → Custom TextString and fill in “Red”
• Drag in setSeagullColor and fill in “Green”
• Return GREEN
• Under the else
  – Drag in setSeagullColor and fill in “White”
  – Return WHITE

![Diagram of nested if statements with conditions and actions for seagull color determination.]
We only want this to happen if the seagull is clicked on

- Go to the `initializeEventListeners` function
- Click on Add Event Listener

Select Mouse ➔
`addMouseClickOnObjectListener`
This listener is true whenever you click on any object, but we want it to be true for only the seagull

• Click on add detail ➔ setOfVisuals ➔ Custom Array

Add this.seagull to the Visual array
Drag in seagull’s setPaint method and select BLACK

• This will change the seagull’s color to black
• But we want to change it to the color returned by changeColor
• Drag the changeColor function over BLACK

Run your world! When you click on the seagull, it should cycle through colors
PART 3

• Create a snowman function
• Create a snowman variable to keep track of color
• Change the snowman color in response to the value of a random variable
  – If the number is 1, we want it to be red
  – If it is 2, change to blue
  – If 3, change to green
  – If 4, change to white

Start by creating a new snowman function

• Click on the dropdown menu
• Snowman ➔ Add Snowman function
Like before, set the return type to Other types → Color

Name the function changeColorRandom
We want to click a local random number variable

- Select variable from the bottom bar
- Drag into the function

- For value type, selectWholeNumber
- For name, fill in colorNumber
- For initializer, pick 1
But we don’t want the value to be one, we want the value to be a random number

• Under 1, select Random \rightarrow \text{nextRandomIntegerFromAUpToAndIncludingB} \rightarrow 1 \rightarrow \text{Custom WholeNumber}

• We want the numbers to be between 1 and 4, so when the custom number calculator pops up, pick 4

• See next slide for images
Your code should look like this

```plaintext
WholeNumber = colorNumber ← nextRandomIntegerFromAUpToAndIncludingB: 1, 4
```

Next drag in an if block and select true

Under true, select Relational (WholeNumber) → ??? == ??? → colorNumber → 1

This basically says “if colorNumber = 1”
If colorNumber is 1, we want to return red

Repeat for the following

- If colorNumber==2
  return Blue
- If colorNumber==3
  return Green
- Else
  return White

- Give it a try before you look at the code on the next slide!
We want the snowman to change color when we click it

- Go to initializeEventListeners
- Pick mouse → objectListener
  - Add detail → setofVisuals → customArray
Add this.snowMan to the Visual array

Drag in setPaint and select BLACK, then drag in changeColorRandom over it
Run your code!

- Does it work?
- How would we know?
- What if we have the snowman say the random number, so we can check

Go back to changeColorRandom

- Drag in say and select “hello”
- Then under “hello”, select ???+???→Custom TextString → WholeNumber → colorNumber
- Because colorNumber is a Number type and not a String type, it does not come up as an option to say
- But if we say a word and then tack on colorNumber, Alice will to the conversion
- I left the Custom TextString blank, but you could also fill in something along the lines of “The random number is ”. Just make sure you end with an extra space!
Run your code and match up the number the snowman tells you with the color he turns.

What if we want to find out what color the snowman turns?
Part 4

• Write a function to return the current color of the snowman

Add a new snowman function
Make the return type TextString and the name getNameOfColor

Now we need to create the variable. Go to Snowman ➔ add snowman property

- Value type = TextString
- Name = snowmanColor
- Initializer = White
For the snowmanColor variable to have the right variable, we have to change it with the color

- Go to changeColorRandom and, before the return statement, drag in setSnowmanColor and type “Red”
- Do the same for the other three colors
Go back to `getNameOfColor`

• All we need to do is return the property we just made
• Drag in a return and select `snowmanColor`

Now to have a character say it

• Since we already have the snowman saying the random number, we will have the bunny say the color
• Note that even though the function is the snowman’s, other objects can use it
• Go to the `initializeEventListeners` function
We want the bunny to say the color every time the snowman turns color

• Drag a say statement directly under the snowman setPaint line
• Pick “hello”
• Then under hello select ???+???

Custom TextString ➔ “hello”
• For the custom text, type “The snowman is ”
• Make sure to leave a space after “is”!
• Then select snowman from the left dropdown menu and drag getNameOfColor over “hello”

Run your program!

• Click on the snowman and watch the bunny tell you his color!