

# Expressions in Alice

Introduction to Multimedia  
Programming  
Fall 2006

# What are expressions?

- > Programming language expressions evaluate a logical relationship between two things returning true or false.
  - For example, are these expressions true or false?
    - >  $(1 > 3)$
    - >  $(2 \leq 3)$
    - >  $(7 \neq 9)$

# Alice Expressions

- > In Alice (as well as other programming languages) we go beyond relationships between numbers .
- > In Alice we are usually interested in logical relationships between different objects in our animation world.
- > For example, if we have a moon world, with a rock and a spider robot, we can evaluate expressions about the distance between the spider robot and the rock.
  - (spiderRobot distance to rock > 2 meters)

# Expressions determine relationships between objects

- > We use expressions to logically evaluate the relationship between objects.
- > So is `(spiderRobot distance to rock > 2 meters)` ?
- > What about in the second world?

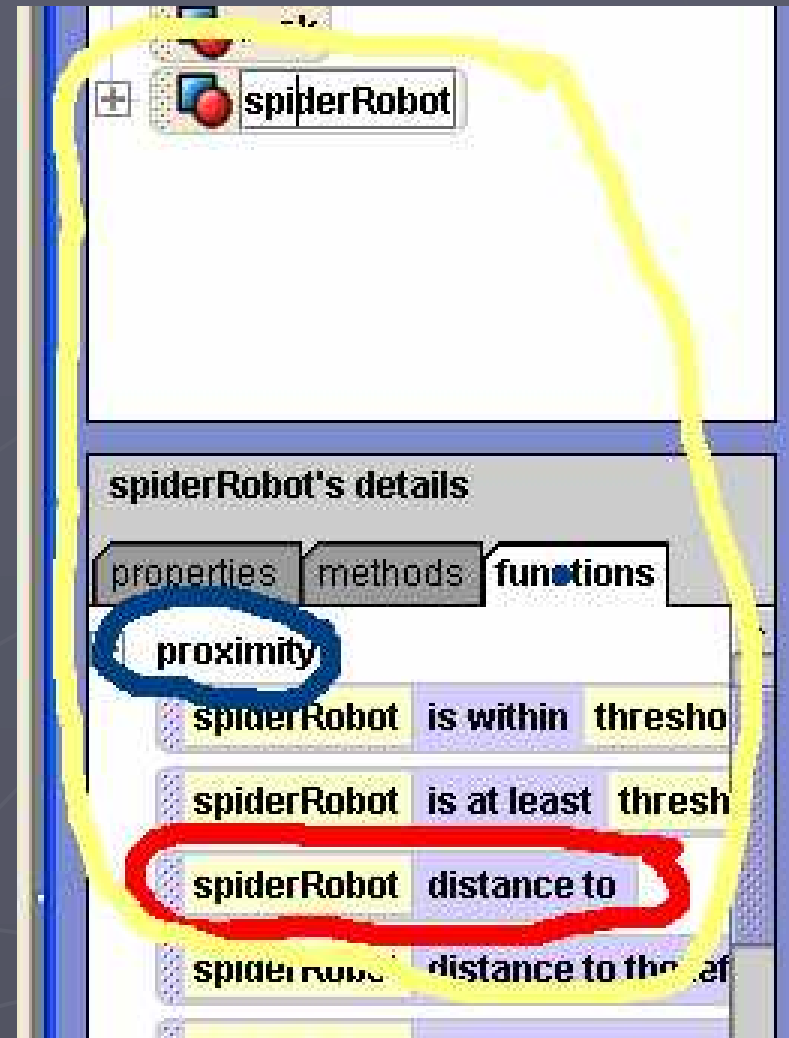


# Expressions and Functions

- > We use expressions usually because we want to take action depending on the relationship between different objects in the world.
- > In Alice we usually get information about relationships between objects using functions specific to those objects

# spiderRobot distance to

- > For example, distance to is a function that tells us the distance from the spiderRobot to another object.
- > spiderRobot functions are broken down by proximity, size, spatial relation, point of view and other



# Expressions and if/else

- > Expressions by themselves don't get us far.
- > We use expressions in other Alice statements, like the if/else statement.

- > For example:

```
if (spiderRobot distance to rock > 2)
```

```
    spiderRobot move to rock
```

```
else
```

```
    spiderRobot say "I'm close to the rock"
```

- > This means that if the spiderRobot is more than 2 meters from the rock, the robot will move to the rock, otherwise the robot will say that it is close to the rock.

# If/else statement evaluation

- > In the example if the expression (spiderRobot distance to > 2 meters) evaluates to true:
  - the spiderRobot moves to the rock
  - but if the expression evaluates to false then the robot says its close to the rock.
- > This is called conditional evaluation. We can take different actions depending on how expressions evaluate.

# Else can be empty (null)

- > We don't need to always supply actions for the else branch.
- > In this exercise we will concentrate on the block that is run when the expression in the if/else statement evaluates to true.

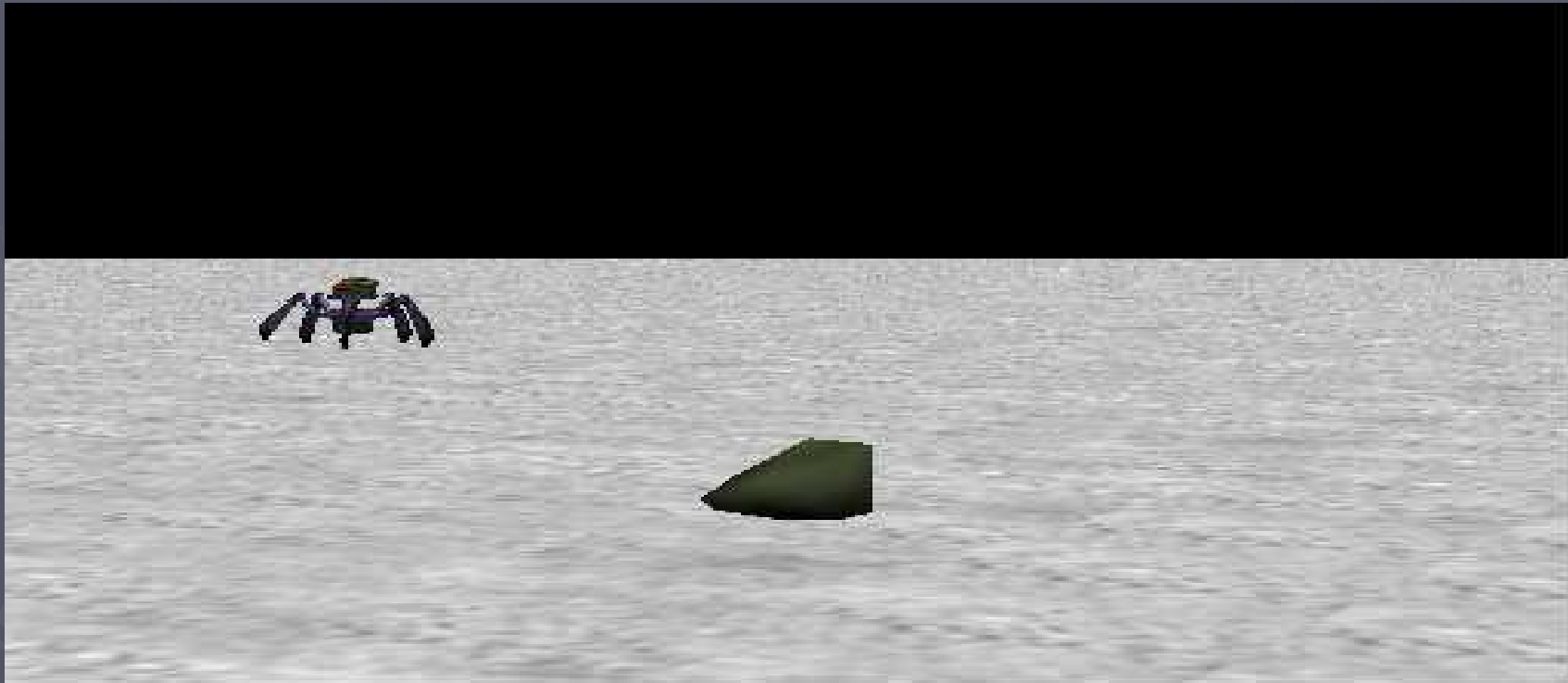
The screenshot shows a programming environment window titled "world.my first method". Below the title bar, there are two buttons: "create new parameter" and "create new variable". The main workspace contains an "If" block with the condition "spiderRobot distance to rock > 2". The "If" block has a green flag icon on its left side. Inside the "If" block, there is a single block: "spiderRobot move to rock more...". Below the "If" block is an "Else" block containing a "Do Nothing" block. The "If" block and its contents are highlighted with a light green background, while the "Else" block and its contents are highlighted with a light orange background.

# Exercise 1: spiderRobot moves to rock

- > 1. Create a moon world.
- > 2. Add a rock (or an island – if you add the island then shrink it) to the world.
- > 3. Add a spiderRobot to the world. Place the spiderRobot a distance from the rock.

# Exercise 1 so far:

- > So far our world looks something like this (your version can vary of course):



# Exercise 1: Add the if/else

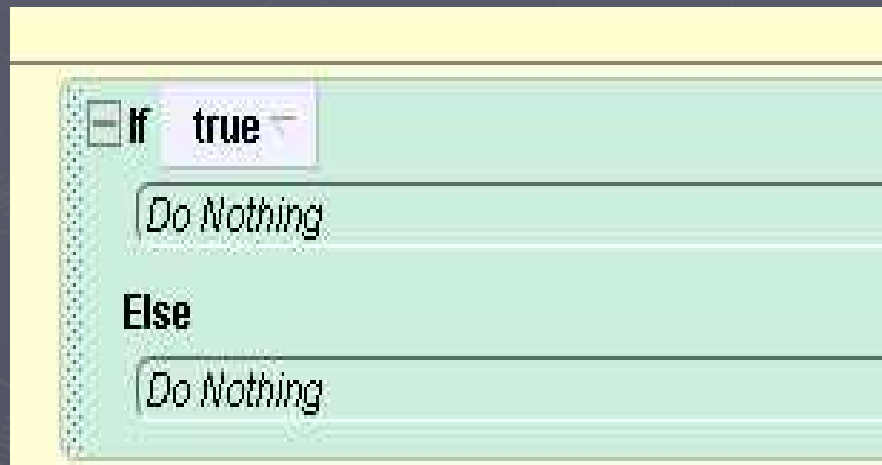
- > Drag the if/else statement
- > And drop it in the Do Nothing slot in the edit area



# Exercise 1: Set the if/else expression

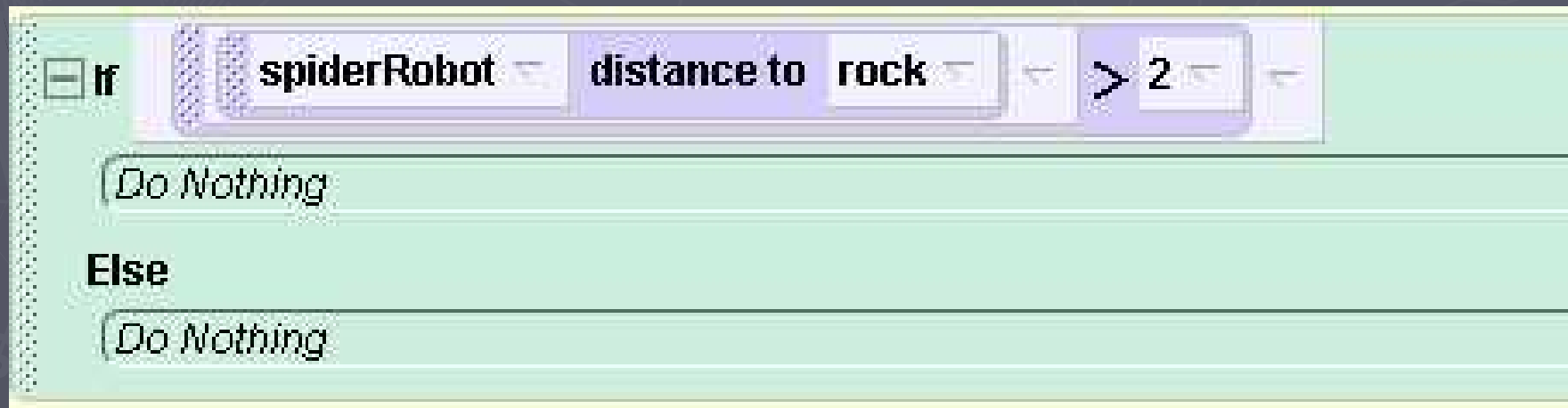
- > When you drop the if/else statement into the Do Nothing slot Alice will want you to set a default expression and will give you a choice between true and false.
- > Select true.
- > We will change this in the next step because we want to make the expression a relationship between two objects in the world.

# Exercise 1: the if/else so far



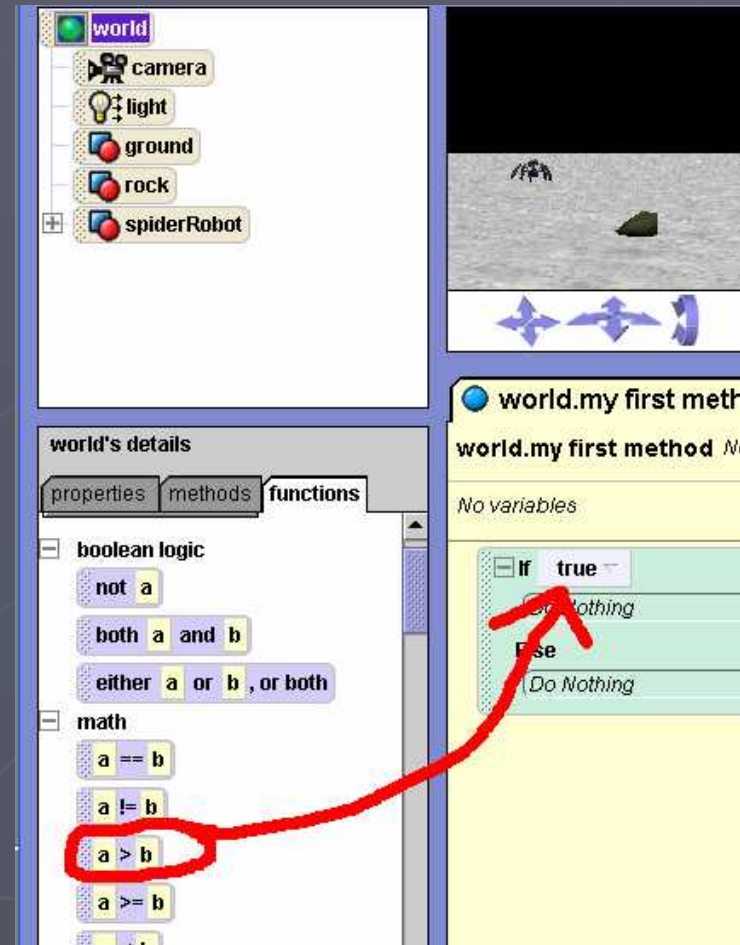
# Exercise 1: Change the expression

- > We want to change the expression to a  $>$  (greater than) relationship – because we want to run the true branch of the if/else when the spiderRobot is more than 2 meters away from the rock.



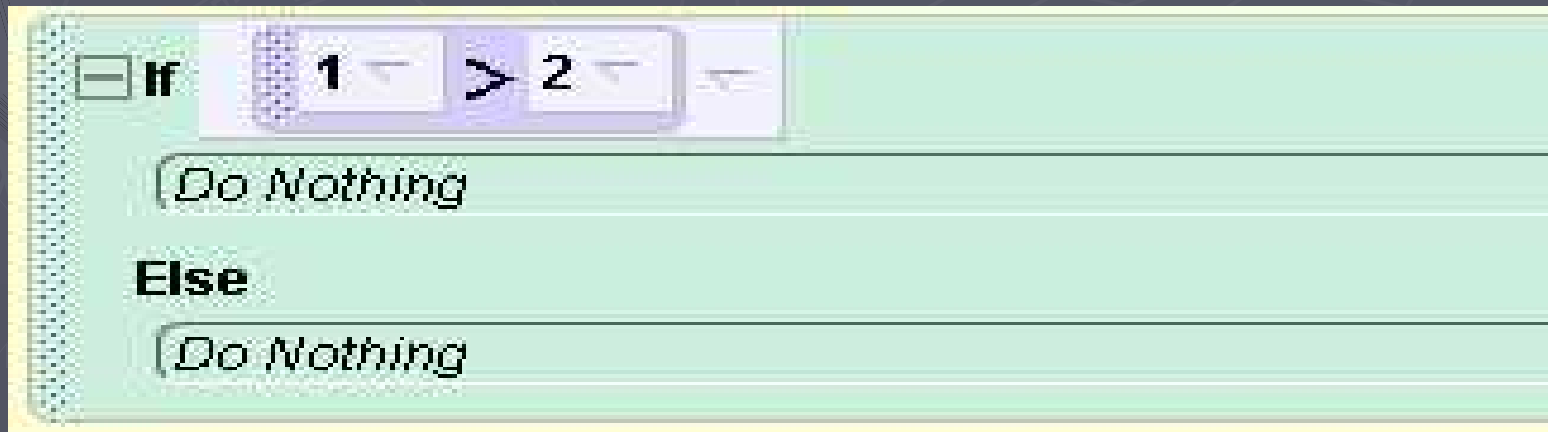
# Exercise 1: change expression to $a > b$

- > In the object tree, select World and then the function tab
- > Then move the  $a > b$  math relation to the expression



# Exercise 1: so far

- > When we move the  $a > b$  tile, we have to provide default values for both  $a$  and  $b$ . Select any numeric values you want. We will change them later although its best if the  $b$  value is 2 (because we will set it to 2 later if its not).



# Exercise 1: the left hand slot

- > Now both slots in the expression have numbers in them.
- > We are going to put the spiderRobot distance to function in the left hand slot.
- > That way we will get the actual distance from the spiderRobot to the rock.
- > Why will this work? Because the distance to function returns a number.

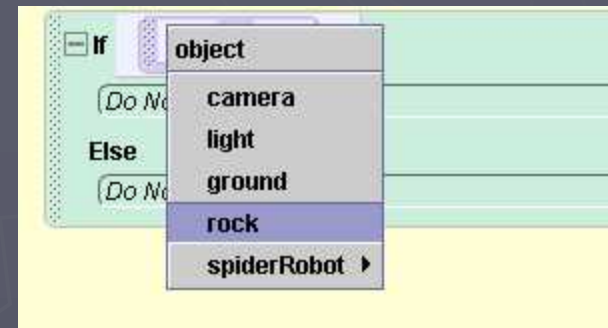
# Exercise 1: distance to substitution

- > Click on the spiderRobot object in the object tree. The click on the function tab. Look through the functions for spiderRobot distance to.

The screenshot displays a software interface with two main panels. The left panel, titled "spiderRobot's details", has three tabs: "properties", "methods", and "functions". The "functions" tab is active, showing a list of functions. The function "spiderRobot distance to" is highlighted with a red circle. The right panel shows a function editor for "world.my first method". It includes a "No variables" section and a conditional block with "If" and "Else" sections. The "If" section contains a comparison "1 > 2" and a "Do Nothing" action. A red arrow points from the "spiderRobot distance to" function in the left panel to the "If" section in the right panel.

# Exercise 1: move the distance to function to the left hand slot in the expression.

- > Move the distance to function to the left hand slot in the expression.
- > You will have to supply an object that the function takes so that you can get the distance from the spiderRobot to some object.
- > Select the rock. Now the function in the left hand slot returns the distance from the spiderRobot to the rock.
- > And the expression will return true if the distance from the spiderRobot to the rock is greater than 2 meters.



# Exercise 1: add code

- > Now we need to add code to the block that will execute if the expression is true.
- > Add spiderRobot move method. Move forward 5 meters.
- > Then run the animation.

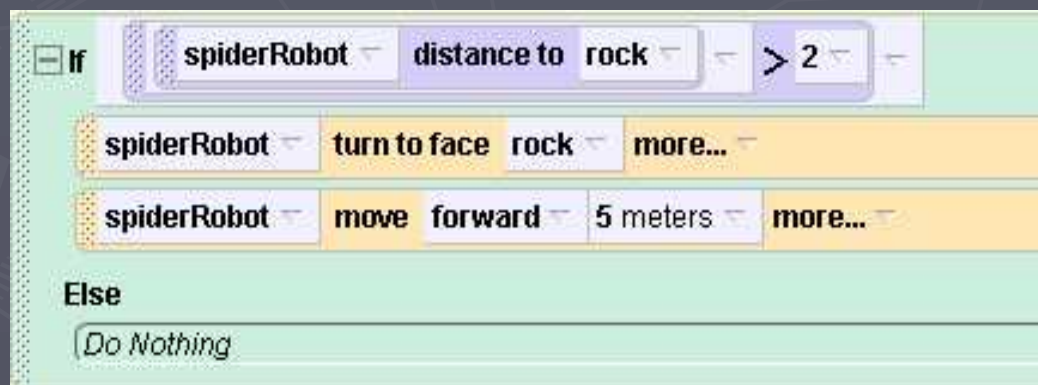


# Exercise 1: evaluate the result

- > In most cases the spiderRobot probably did not move toward the rock.
- > That's because we did not orient the robot to the rock.
- > If we had used the move toward method this would have worked.
- > But bear with me –

# Exercise 1: turn to face

- > Just before the move method we add a statement to orient the robot.
- > Drag the spiderRobot turn to face method and add it to the code. Alice will ask what you want to turn to face – in this case the rock.



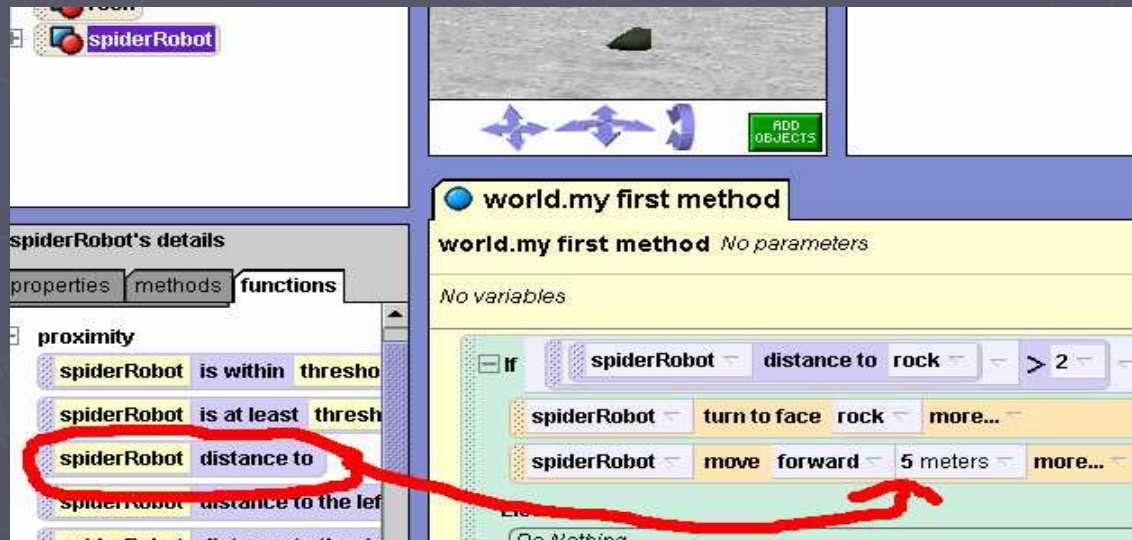
# Exercise 1: test again

- > Now run the animation again.
- > This time the spiderRobot at least moves toward the rock

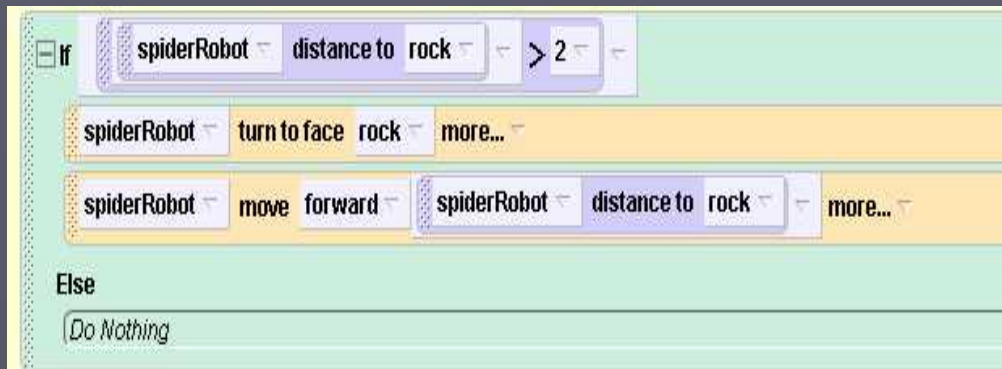


# Exercise 1: add distance to again

- > Now we will substitute the distance to function for the 5 meter guess in the move statement.
- > Select spiderRobot distance to function and substitute it for the 5 meter guess in the move statement.



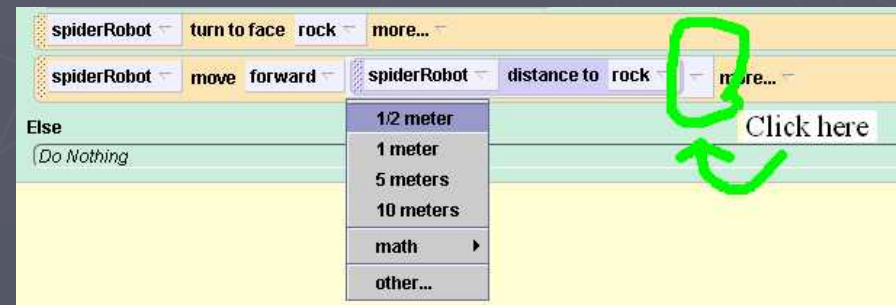
# Exercise 1: Substitute and test



- > After the substitution run the animation. What happens this time?
- > The robot moves over or into the target object.
- > This is because the distance is measured from the center of the robot to the center of the target (in this case, the rock).

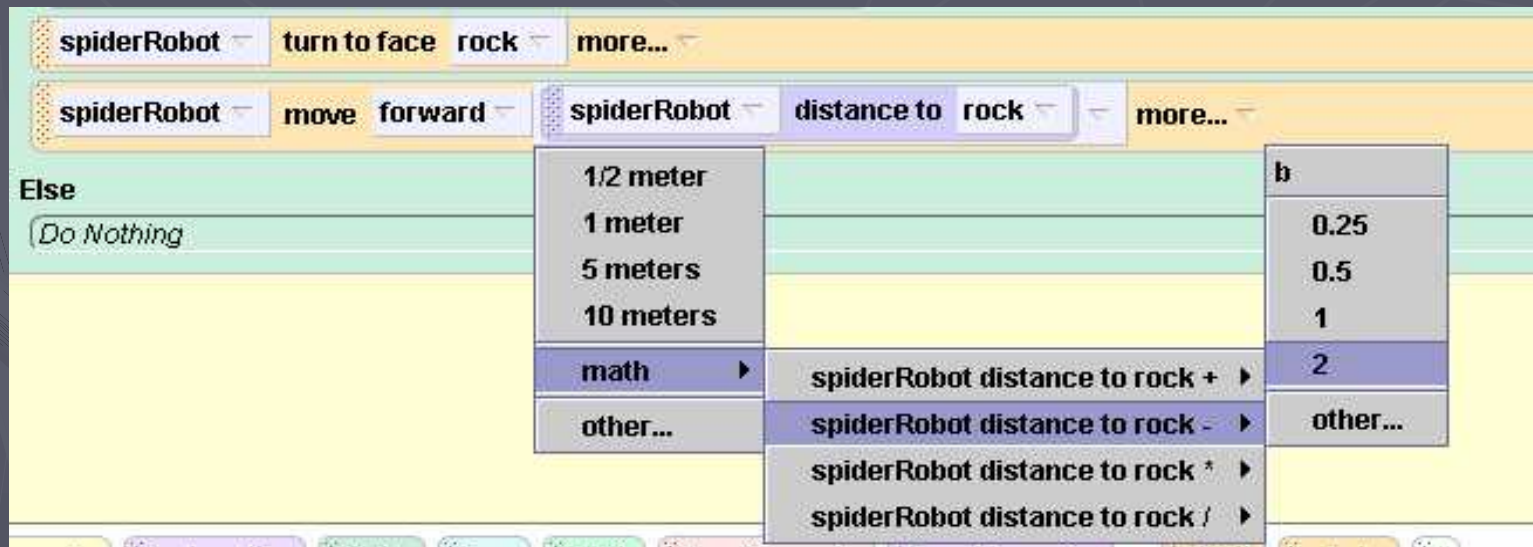
# Exercise 1: back off a little

- > We would like to back the spiderRobot off a little bit. We can do this by subtracting the width of the rock from the distance from the robot to the rock.
- > How do we do this?
- > First we click on the dimple at the end of the expression in the move statement. A menu will popup ...



# Exercise 1: select math expression

- > From the popup menu select math
- > then select spiderRobot distance to rock – 2 (or some other number – we will substitute a function for the number next).



# Exercise 1: our story so far:

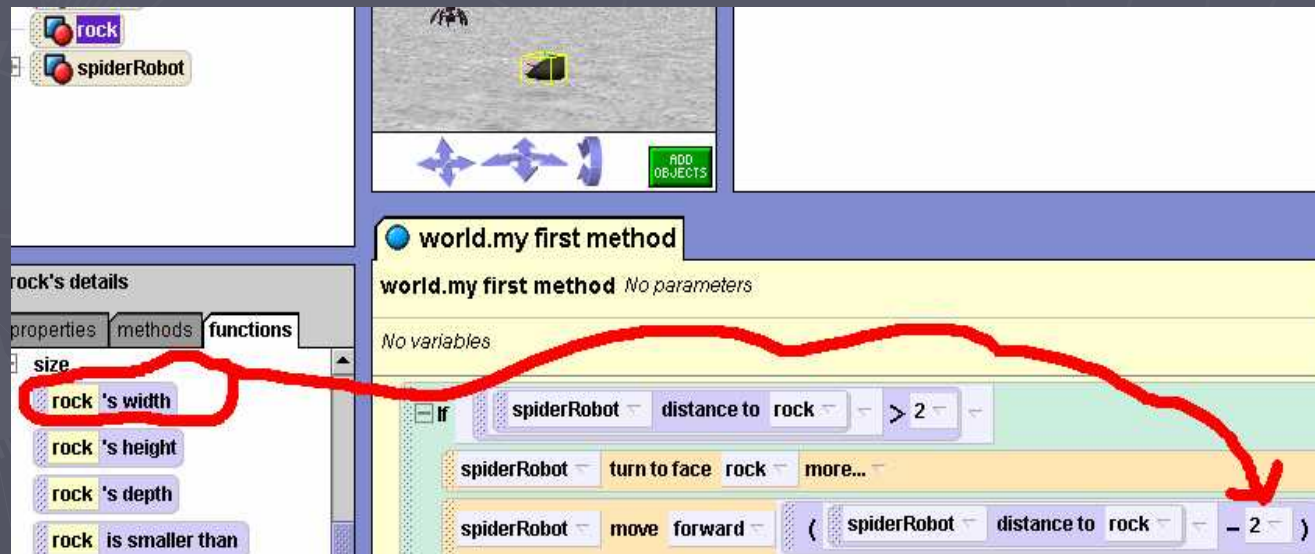
- > This is what the code looks like up to this point:

The image shows a Scratch script for a spider robot. It begins with an 'if' block that checks the condition 'spiderRobot distance to rock > 2'. If this condition is true, the script enters an 'if' block containing two actions: 'spiderRobot turn to face rock' and 'spiderRobot move forward ( spiderRobot distance to rock - 2 )'. If the condition is false, the script enters an 'else' block with the action 'Do Nothing'.

```
if ( spiderRobot distance to rock > 2 )  
  spiderRobot turn to face rock  
  spiderRobot move forward ( spiderRobot distance to rock - 2 )  
else  
  Do Nothing
```

# Exercise 1: subtract width from distance

- > Now we'll substitute the rock's width from the distance.
- > We click on the rock in the object tree, then the functions tab and select the function that gives us the rock's width:



# Exercise 1: substitute and test

- > Substitute the rock's width function and test the animation.

The image shows a Scratch script with the following blocks:

- If** block: `spiderRobot distance to rock > 2`
- Then** block: `spiderRobot turn to face rock more...`
- Then** block: `spiderRobot move forward ( spiderRobot distance to rock - subject = rock 's width )`
- Else** block: `Do Nothing`

# Exercise 1: finale

- > So now the spiderRobot stops a little bit short of the rock.

