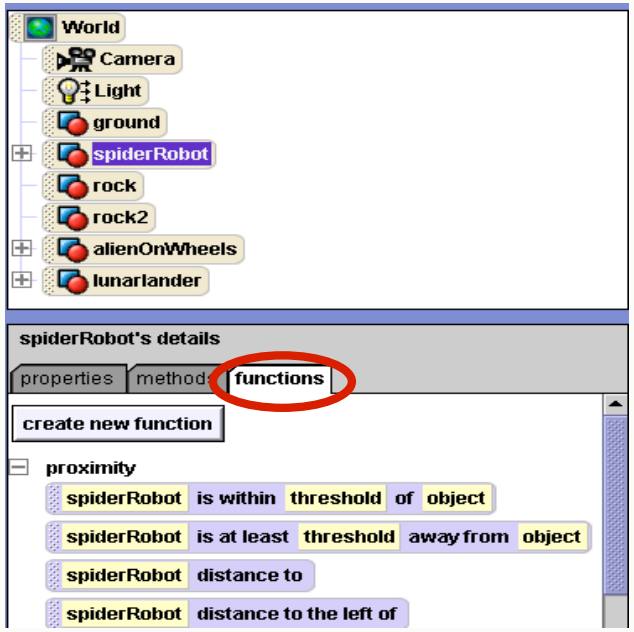
Functions

Alice

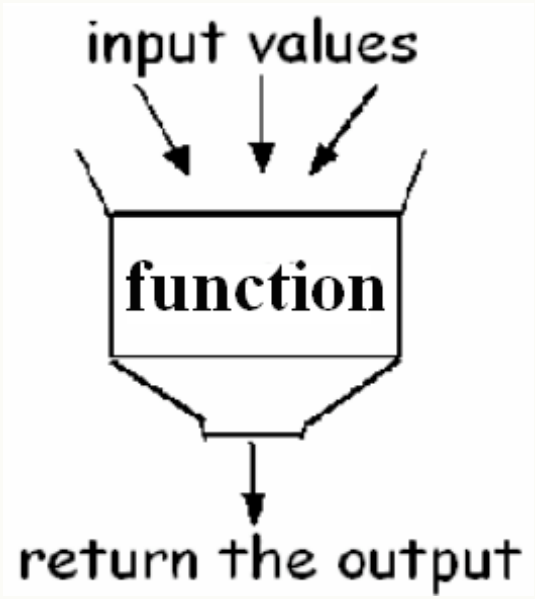
Built-in Functions

- We have been using built-in functions.



How a function works

- A **function** receives value(s), performs some computation on the value(s), and returns (sends back) a value.

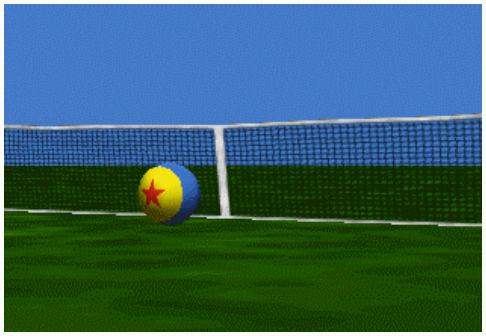


Types of Functions

- The type of a function depends on the type of value it returns.
  - a number
  - a specific object
  - a color
  - a Boolean (true or false)
  - other property values…

A new example

- A common task in sports game programs is to bounce a ball. To illustrate how this is done, let's bounce the ball over the net. (The ball has been positioned 1 meter from the net.)



- The ball should move up and forward and then down and forward in a pattern that looks something like this:

Note: This looks easy – but do not be deceived!

Design Storyboard

- A design for a world-level method:

World.ballOverNet:

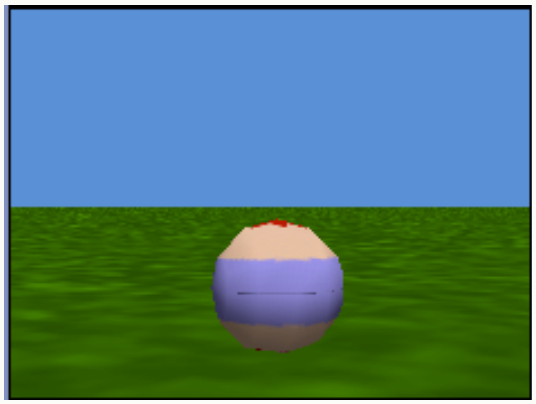
Do in order
  toyball turn to face the net
  Do together
    toyball move up
    toyball move forward
  Do together
    toyball move down
    toyball move forward

Demo

- ToyballOverNet-V1
- ToyballOverNet-V2
- Concepts illustrated in this example:
  - Movement of an object is sometimes relative to another object. In this example,
    - a built-in *height* function is used to determine the height of the net. Then the ball moves up enough to clear the net.
    - the toyball's up and down movements are relative to the ground. An *orient to* method is used because it is not easy to tell "which way is up."

Rolling the ball

- Another sports game action involves rolling a ball along the ground.
- We want a realistic motion rather than a slide.
- The ball must simultaneously move and roll.



realisticRoll

Do together
  move ball forward 1 meter
  turn ball forward 1 revolution

Demo

- Ch06Lec1ToyballRoll-V1
  - Our design assumed that the ball's motion is relative to the ground. But the ball turns relative to it's own sense of direction.
- Ch06Lec1ToyballRoll-V2
  - *AsSeenBy ground* can be used to make the ball turn relative to the ground.

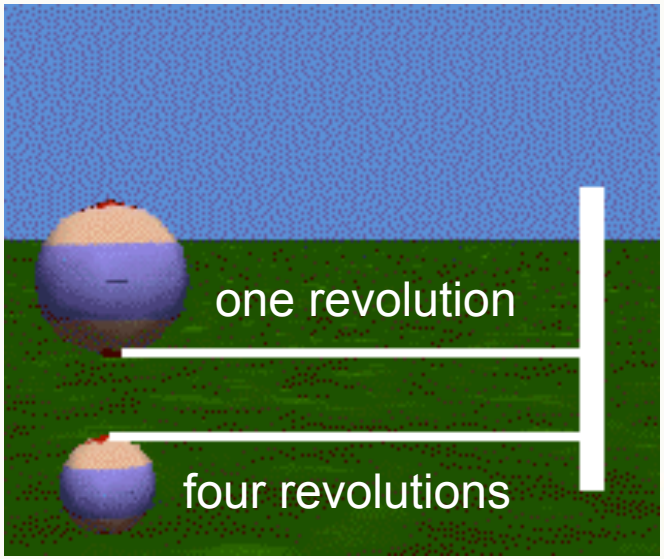
Revising the approach

- The ball is made to roll 1 revolution. Suppose we want the ball to roll a certain distance (say 3 or 4 or even 10 meters) along the ground.
- How many times should the ball turn 1 complete revolution?

Number of revolutions

- The number of revolutions depends on the size of the ball
  - The number of revolutions is

    distance/(P * diameter)



- But there is no built-in function to return the number of revolutions
- We will write our own!

Parameters

- We want to return the value computed as

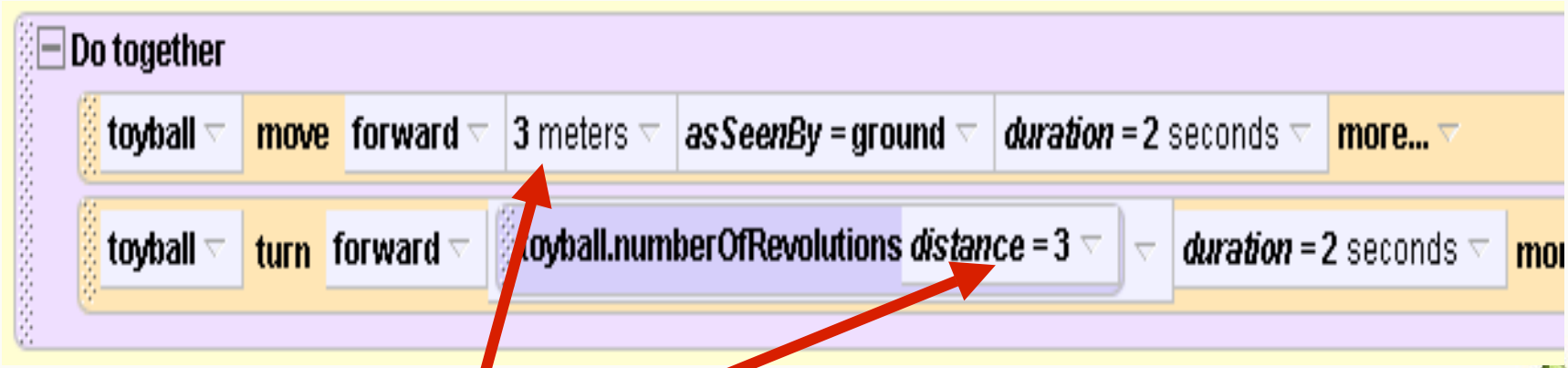
  distance/(Π * diameter)

- Obviously, what is needed is
  - the ball's diameter
    - the ball object has a built-in *width* function
  - the distance the ball is to travel
    - can be sent as a parameter to the function

Demo

- Ch06Lec1ToyballRoll-V3
- Concepts illustrated in this example
  - A function must have a *return* statement to send back a computed value.
  - In this example, a **math expression** is created to compute a number value.
  - The order of evaluation is indicated by the use of parentheses, as in traditional math expressions.

Calling the function



test values

We should run the animation with several test values to be sure it works as expected.

What happens if you use a negative value?

Levels of Functions

- As with methods, functions can be either class-level or world-level. (The function just presented was class-level.)
- The guidelines for class-level methods also apply to class-level functions:
  - No references to other objects.
  - No references to world-level functions you have written (**built-in** world-level functions are fine to use).

Execution Control with *If/Else* and Boolean Functions

Example: Single Condition

Alice

Thinking About More Advanced Worlds

- No doubt you have started to think about building animations like simulations and video games…
- To build more advanced worlds, you will need to write code that involves **decisions**

Examples of Decisions

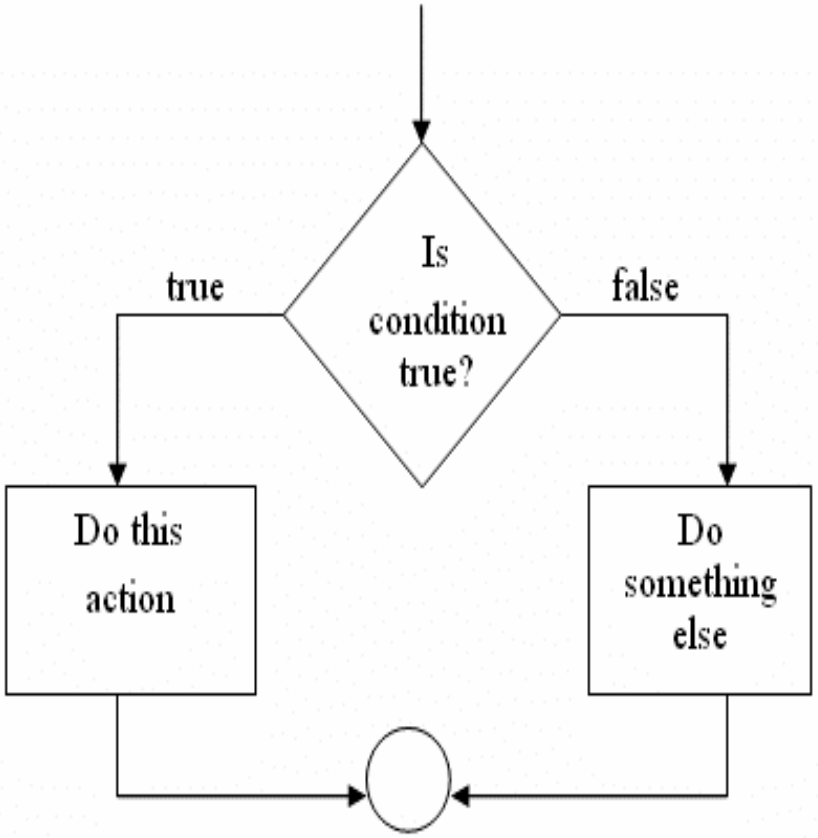
Decisions

- In a car-race simulation, the driver steers the car around curves and past mile-markers.
  - If the car stays on the road, the score increases.
  - If the car goes off the road into the stands, the car crashes.
  - If the driver gets the car over the finish-line, the time is posted and the driver wins!

Logical Expressions

- A decision is made based on current conditions
- A condition is checked in a logical expression that evaluates to *true* or *false* (Boolean) value
  - car on road ⇨ *true*
  - car over finish line ⇨ *false*

If/Else



- In Alice, a logical expression is used as the condition in an ***If/Else*** control structure.
- Decisions (using If/Else) are used in
  - functions
  - methods

Example: Boolean Function

- Suppose you are building a simulation system used to train air traffic controllers.
- One of the tasks of an traffic controller is to be alert for possible collisions in the flight space.

Storyboard

- One factor in determining whether two aircraft are in danger of collision is the vertical distance (difference in altitudes) between them.
- We can write a function that checks the vertical distance against a minimum difference in altitudes.
- The function returns *true* if they are too close, otherwise *false*.

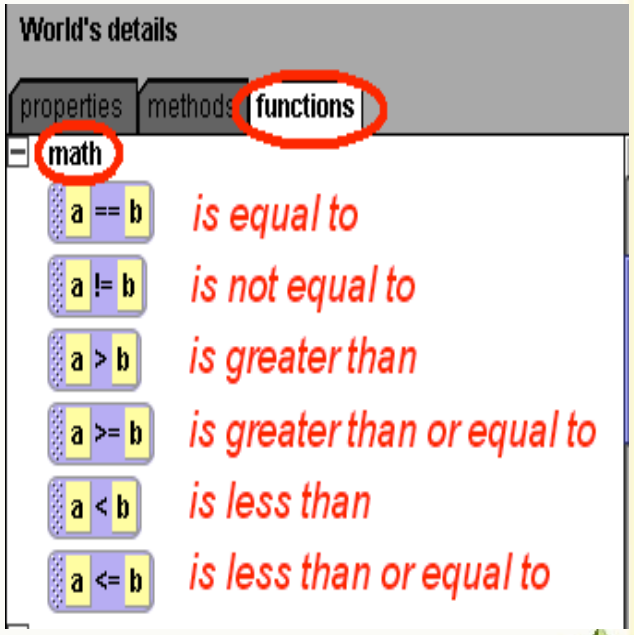
isTooClose

Parameters: *aircraftOne, aircraftTwo, minimumDistance*

If the vertical distance between *aircraftOne* and *aircraftTwo* is less than *minimumDistance*
    return *true*
Else
    return *false*

Demo

- Ch06Lec2aFlightCollision-V1
- Concepts illustrated in this example
  - A world-level *relational operator* is used to create a Boolean expression as the condition.
  - The *absolute value* function is used to make sure the computed difference in altitude is not a negative number.



Storyboard

- To avoid a collision, the aircraft that is above the other should move up and the lower aircraft should move down.

avoidCollision

Parameters: *aircraftOne, aircraftTwo*

If aircraftOne *is above* aircraftTwo
    Do together
        aircraftOne move up
        aircraftTwo move down
Else
    Do together
        aircraftOne move down
        aircraftTwo move up

Demo

- Ch06Lec2aFlightCollision-V2
- Concepts illustrated in this example
  - Decisions were made to
    - control whether a method is called
    - determine which set of instructions are immediately executed

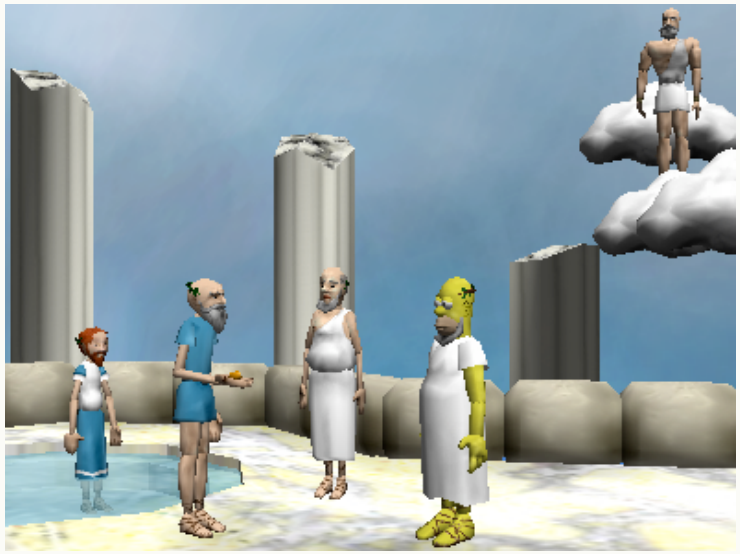
Execution Control with *If/Else* and Boolean Questions

Example: Multiple Conditions

Alice

The Zeus world revisited

- Recall the Zeus world (Chapter 5 Section 2)



- Testing this world, we found two significant problems
  - Anything the user clicked was zapped by Zeus' bolt – not just philosophers!
  - Philosophers could be zapped with lightning more than once!

Problem

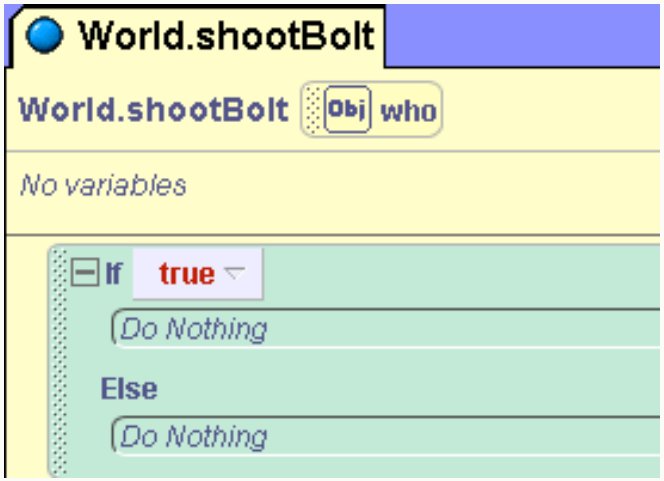
- What we need in the Zeus world is conditional execution
  - Check conditions:
    - Is the selected object a philosopher?
    - If so, has the philosopher already been zapped by lightning?
  - Conditional execution:
    - lightning bolt will be shot or not

Multiple Conditions

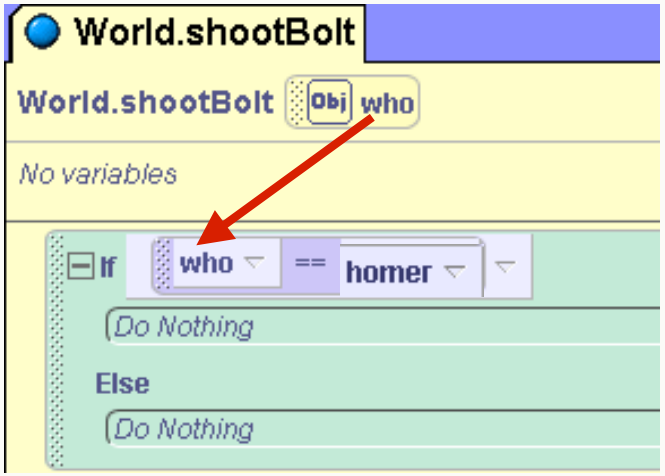
- First, we'll tackle the problem of restricting Zeus to shoot thunderbolts only at philosophers.
- This is different from previous examples in that four possible conditions must be checked – the user could click on any one of the four philosophers.

Demo

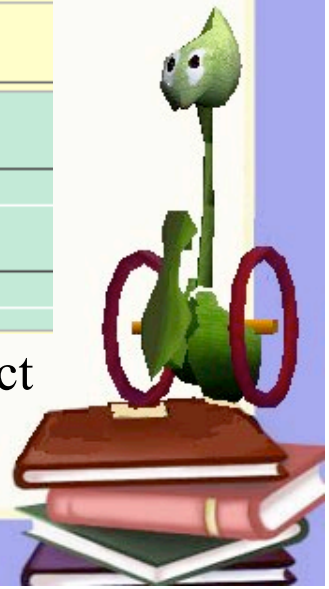
- **Ch06Lec2bZeus-V1**
- **Concept illustrated:**
  - Begin with just one object – we chose homer, but any one philosopher would do.



Start with a blank *If* statement…

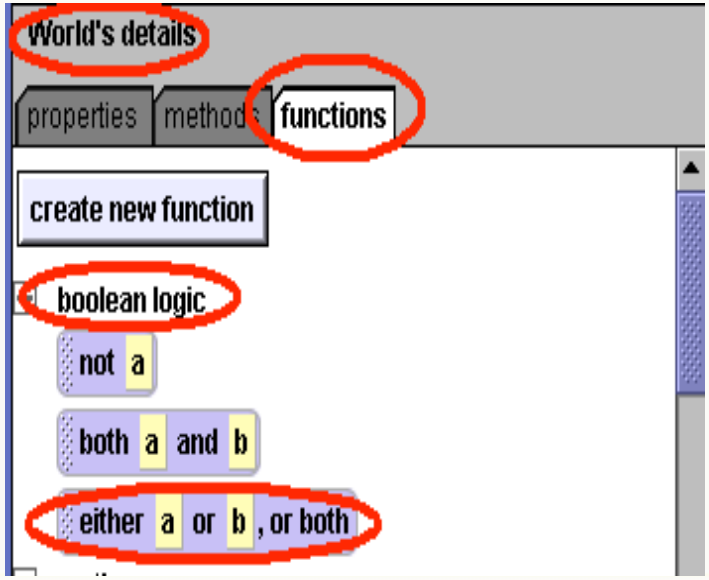


drag in the *who* tile, then select == and homer



Demo

- Ch06Lec2bZeus-V2
- Concept illustrated
  - Boolean logic operators are used to build an expression composed of multiple conditions



Abstraction

- Multiple conditions, as in this example,
  - become complex
  - are difficult to read
  - are difficult to debug
- A better solution is to write our own Boolean question that checks the conditions.
- This is another example of abstraction – allowing you to think on a higher plane.

Demo

- Ch06Lec2bZeus-V3
- Concepts illustrated in this example
  - Multiple conditions can be checked using nested *If* statements to check each possible condition.
    - If one of the conditions is found to be *true*, the function immediately returns *true* and the function is all over!
    - If none of the conditions are *true*, the function returns *false*.

Completing the Zeus world

- Now, we are ready to prevent lightning striking the same philosopher more than once.
- How do we know if a philosopher has already been struck by lightning?
  - When a lightning bolt strikes, the philosopher “is zapped” and the his *color* property is set to black.
  - Checking color is a convenient way to check for a previous lightning strike.

Demo

- Ch06Lec2bZeus-V4
- Concept illustrated in this example
  - We only need to check for a possible duplicate-strike if we already know that a philosopher is clicked
  - the way to handle this is to nest a second *If* statement inside the first.