

CS101

Problem Solving and Object-Oriented Programming

L4: Variables and Functions II; Flow of Control



Variables and Functions

- Variable: named container for storing/retrieving data
 - Memory location for storing data
- Three types of variables
 - Method (local): defined for use within a method
 - Parameter: variable passed to a method
 - Object (property): used to store an object property
- Information needed to define a variable
 - Name: refers to a location in memory
 - Type: the kind of data stored; 4 data types (Number, Boolean, Object, Other: e.g. **Strings**, **Sounds**, **Colors**, etc)
 - Initial value: starting value of specified type; e.g., 1



Method Variables

- Defined using **create new variable** button
- Method variables are local (valid only in the method)
 - e.g. Not accessible anyplace else
- Common uses of method variables
 - Computing and storing values for later retrieval
 - Storing values entered by a user
 - e.g. Sunworhiper2.a2w, MonkeyTalks.a2w



Parameter Variables

- Parameter: variable passed to a method
 - Called an *argument*
 - We've already done this!
 - e.g. when we used the 'say' method for the aliceLiddell object, we passed the string we wanted her to say
 - Can be used to create generic methods for common tasks that require different values at different times
 - e.g. We don't have a *sayHello* method, and a *sayGoodbye* method, and a *sayX* method; we have a *say* method that takes a string as an argument (parameter)
 - If you anticipate needing different values for the action associated with a method, *use parameters!*
 - e.g. `MonkeyTalksParams.a2w`





Property Variables

- Also called instance variables or object variables
 - e.g. distance to another object, point of view (position and orientation), vehicle, etc
- Property variables are defined within an object
 - Use the properties pane of the object's details area
- Values in an object variable are not shared
- How to create and use a property variable
 - Open a new world and add an object
 - Click on object properties
 - Click **create new variable**
 - e.g. SunWorshiper.a2w



Alice Tip: Using the Vehicle Property

● **Vehicle**

- The thing on which an object “rides”
- The default **vehicle** for an object is the **world**

● **Vehicle** property is used to synchronize movements

● Using **vehicle** property in program with girl and horse

- Scene 4 initially shows the **nativeGirl** on a **horse**
- Add **set()** method for **nativeGirl**'s vehicle property
- Set the value of the vehicle to the entire horse
- Include a **move()** method below the **set()** method
- Send **playScene4()** to **my_first_method()** and test



Alice Tip: Using the Vehicle Property (continued)

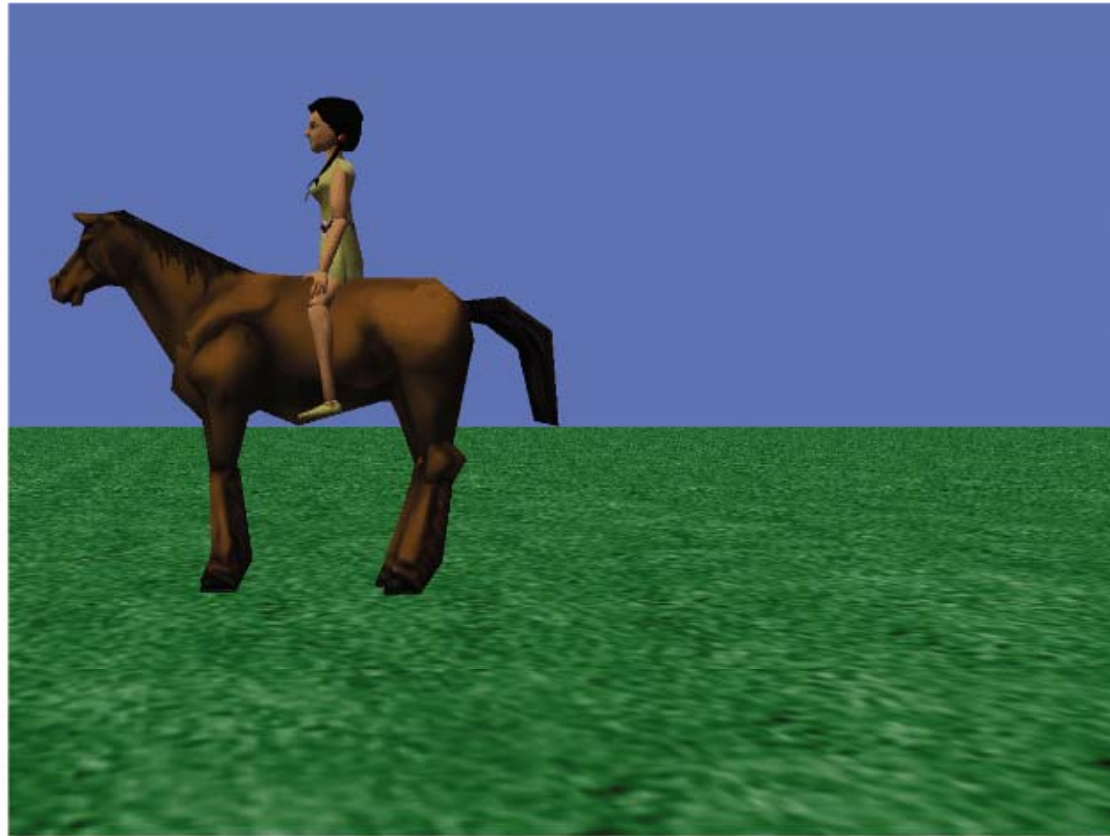
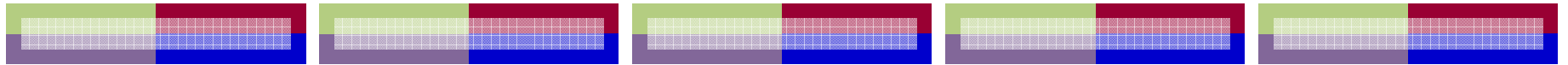


FIGURE 3-47 The girl rides the horse across the screen



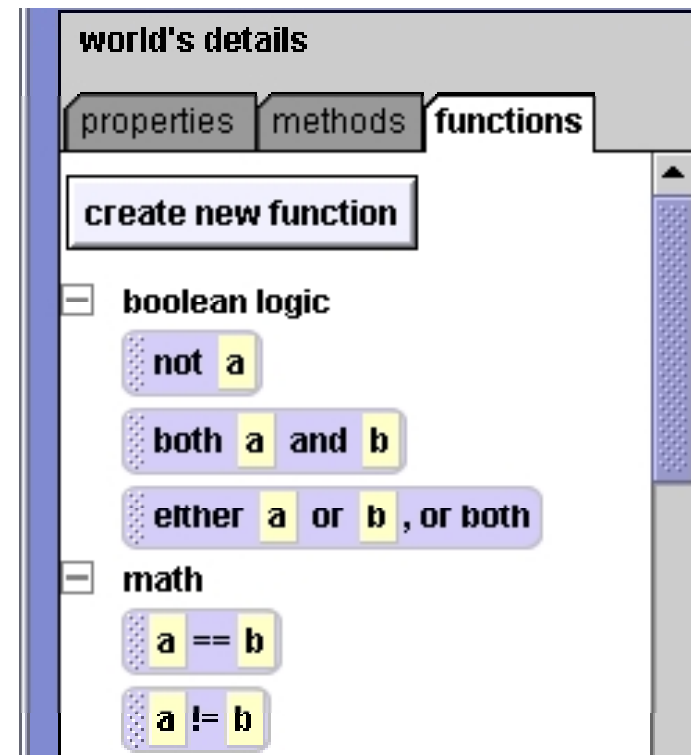
Functions

- Messages used to retrieve information about an object's properties
 - e.g. distance to, etc.
- May be defined by the user
- Comparing functions to methods
 - Functions return a value and appear in parameter list
 - Methods do not return a value and appear as statements



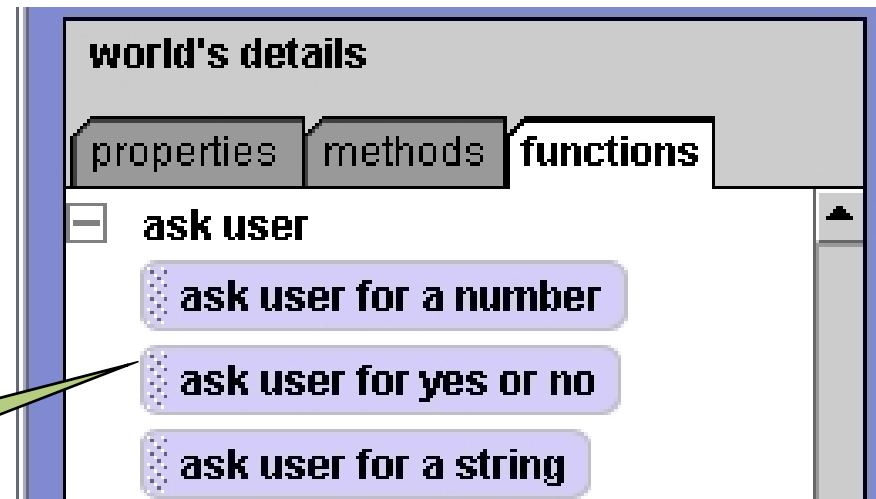
Functions

- Each object has a set of primitive functions
- List of functions is categorized
 - Boolean logic
 - Math
 - Random
 - String
 - Ask user
 - Mouse
 - Time
 - Advanced math
 - Other



Asking the User for Input

- World has three primitive functions that ask for user input
- Each function is displayed in a dialog box



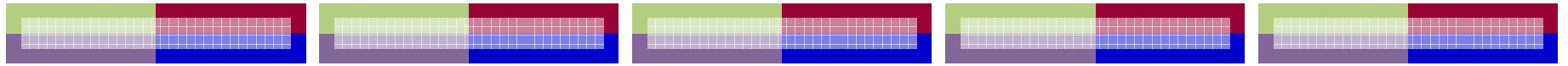
Is it Yes or is it No?





Summary

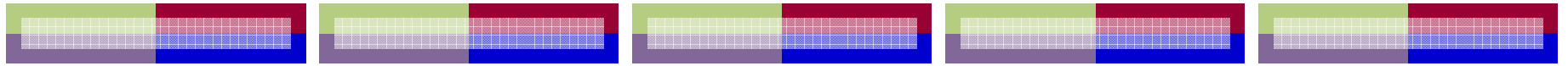
- Variable: named container for storing and retrieving values
- Types of variables: method variables, parameters, object variables
- Data types: **Number**, **Boolean**, **Object**, **String**
-
- Parameter: container for an argument
- Argument: data passed to a method or function



Summary (continued)

- Object **vehicle**: thing on which an object “rides”
- A method does not return a value to its sender
- A method is sent from a place where a statement can appear
- A function does return a value to its sender
- A function is sent from a place where a value can appear



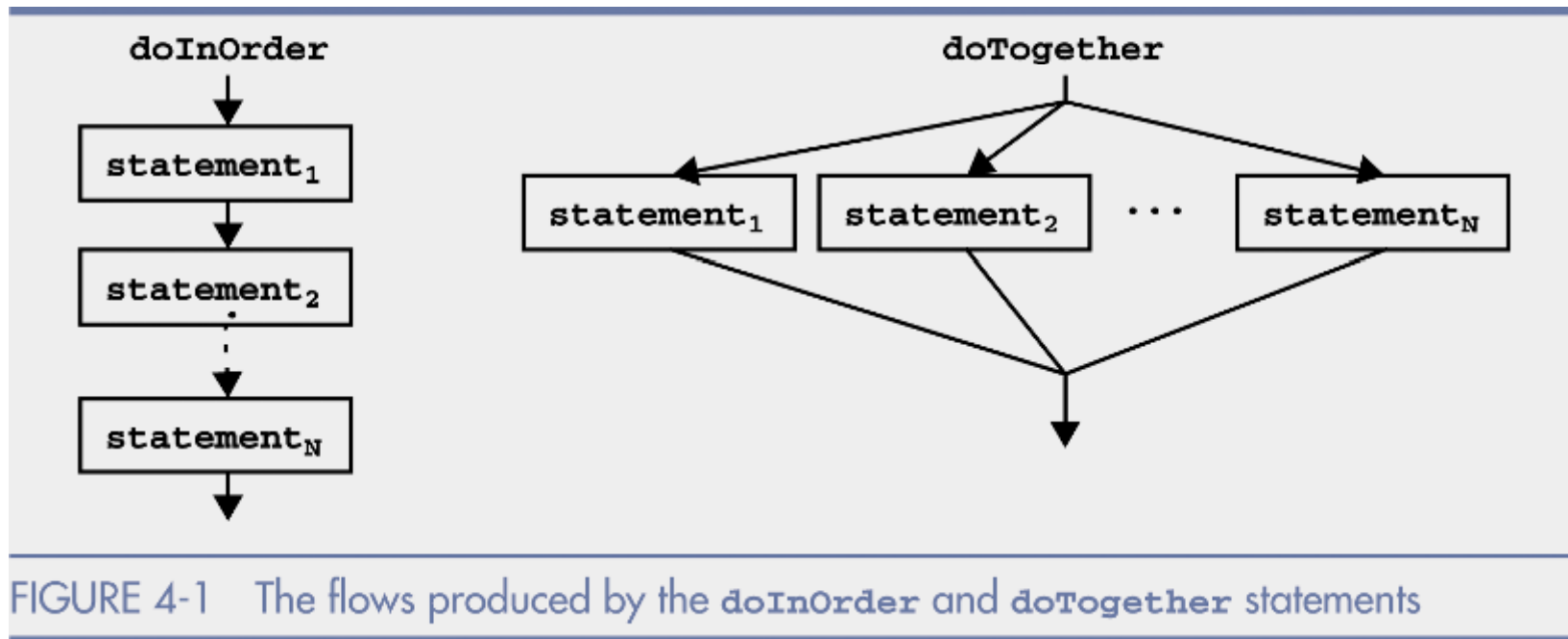


Flow Control

- Flow: sequence of steps for performing a user story
- Flow control statement: structure for managing flow
 - **doInOrder**: produces a sequential execution
 - **doTogether**: produces a parallel execution
- Control statements introduced in the current chapter
 - **if**: directs program flow along one of two paths
 - **for**: directs flow into a fixed number of loops
 - **while**: directs flow into an arbitrary number of loops



Flow Control (continued)



Boolean Variables

- A basic Alice type used to define **Boolean** variables
- Boolean Variables hold only two values

- True
- False
- Initial Value is usually set to be ***TRUE***



- *Condition* (Boolean expression)
 - Produces a value of **true** or **false**
 - Basis for decision-making in programs




Boolean Functions

- Boolean Functions *return* one value from the choice of
 - True
 - False
- Also primitive functions that are Boolean
 - User prompted to answer *Yes* (true) or *No* (false)

The screenshot shows a Scratch code editor window with a blue header bar containing a globe icon and the text "world.animation". Below the header, the code area has a yellow background. The first line of code is "world.animation No parameters" with a "create new parameter" button to its right. The second line is a Boolean variable declaration: a T/F icon, a dropdown menu showing "wantCookie", an equals sign, and a dropdown menu showing "true", with a "create new variable" button to its right. The third line is a function block: a dropdown menu showing "wantCookie", followed by "set value to", a "ask user for yes or no" block with a "question = Do you want a cookie?" dropdown, and two "more..." dropdown menus.



Boolean Functions

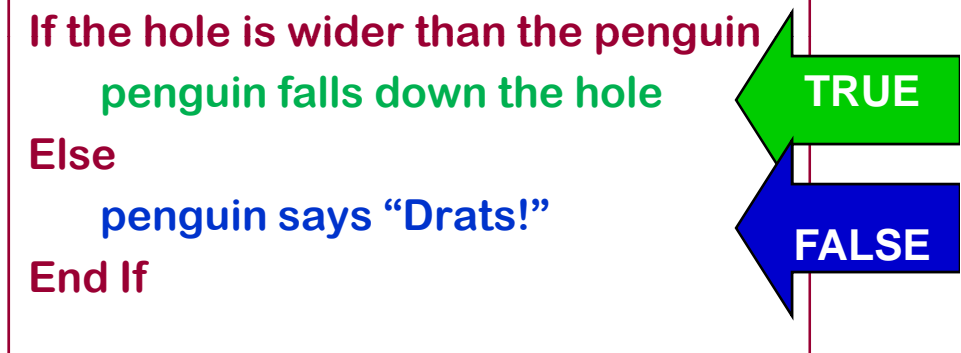
- Return a value of **true** or **false**
 - Can act as a condition in an **if** or **while** statement
 - Many refer to an object's bounding box
 - Example: ***obj.isBehind(obj2)***
 - **true**, if *obj*'s position is beyond *obj2*'s rear edge
 - **false**, otherwise
- 

The **If/Else** Decision Structure

- **If/Else** decision executes one set of instructions if a Boolean condition is *true* and a different set if the condition is *false*
- All previous instructions have been consecutive (*sequence structure*), that is:
 - One
 - After
 - The
 - Other

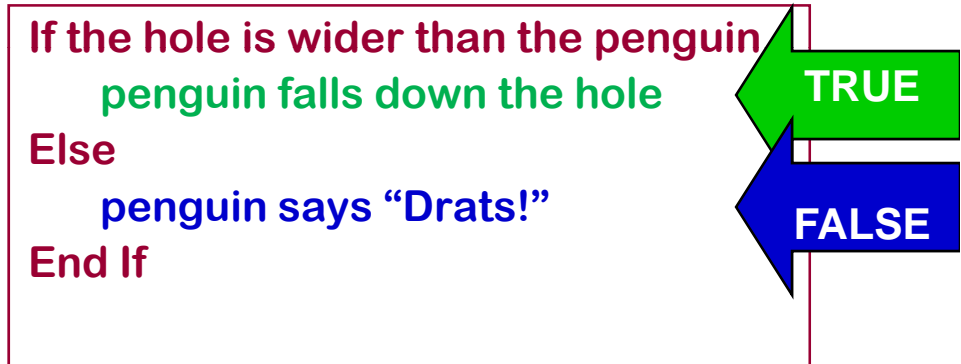
If/Else

- If/Else is a *decision structure*
- Instruction tests a *condition* (which gives a Boolean value) and then based on a true value does task “A” or task “B” if the value was false



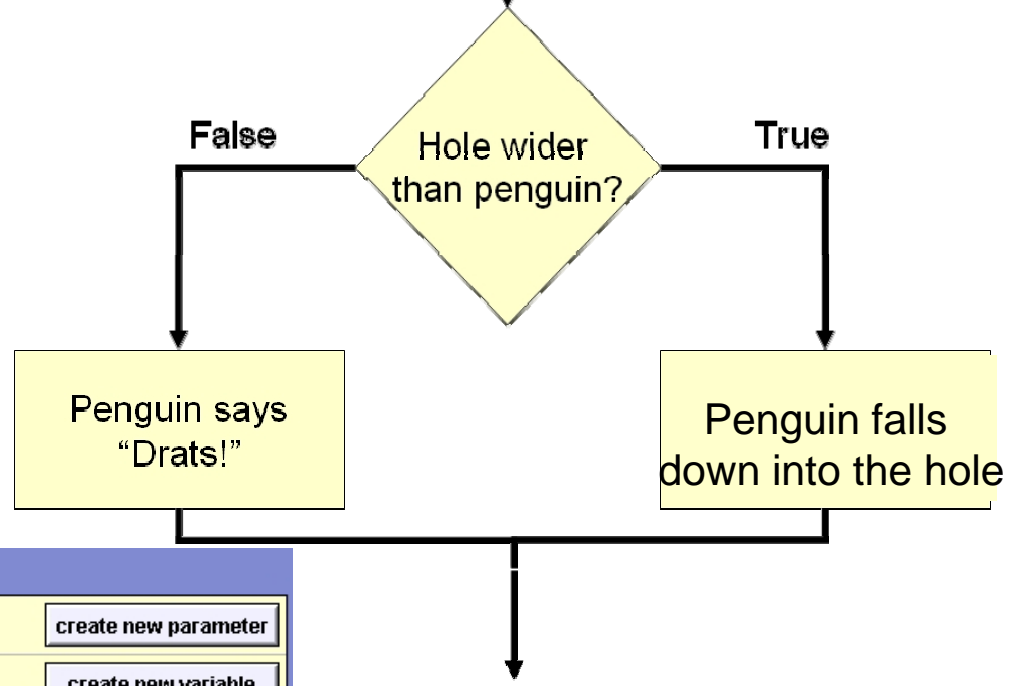
If/Else

- BOTH actions will NOT be performed...only ONE can be!
- The penguin falling or speaking are instructions that are *conditionally executed*
 - They do NOT always execute
 - Executed only under certain conditions



Decision Structure

Flowchart



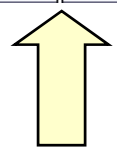
world.my first method

world.my first method *No parameters* create new parameter

No variables create new variable

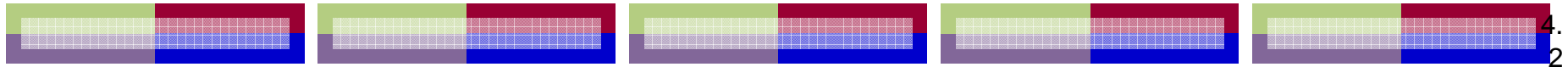
Do Nothing

Do in order Do together **If/Else** Loop While For all in order For all together Wait print //



If/Else Tile





If/Else

Empty If/Else instruction

- Note to empty “slots” in instruction.
 - IF part
 - ELSE part
- The TRUE placeholder is replaced with the Boolean variable or a Boolean function
- Instructions tiles added to both the IF part and the ELSE part
- More than one instruction can be added to both the IF and ELSE part!

