

Programming: Putting Together the Pieces

Built-in Functions and Expressions

Alice



Putting together the pieces

- A major part of learning how to program is figuring out how to "put together the pieces" that compose a program.

🧩 **Analogy:**

putting together the pieces of a puzzle

- The purpose of this session is to
 - 🧩 define the fundamental pieces of a program
 - 🧩 demonstrate how to put the pieces together



Four Fundamental Pieces

- Instruction
- Control Structure
- Function
- Expression



Instruction

- An **instruction** is a statement that executes (is carried out by the computer at runtime).
- In Object Oriented Programming, an instruction is defined as a **method**.
- In Chapter 2, we used instructions to make objects perform a certain action.

 Examples:

snowman turn to face snowwoman
spiderRobot move up 0.5 meters



Control Structure

🌐 A **control structure** is a statement that controls which instructions are executed and in what order.

🎥 In previous worlds, we used:

💡 Do in order

💡 Do together



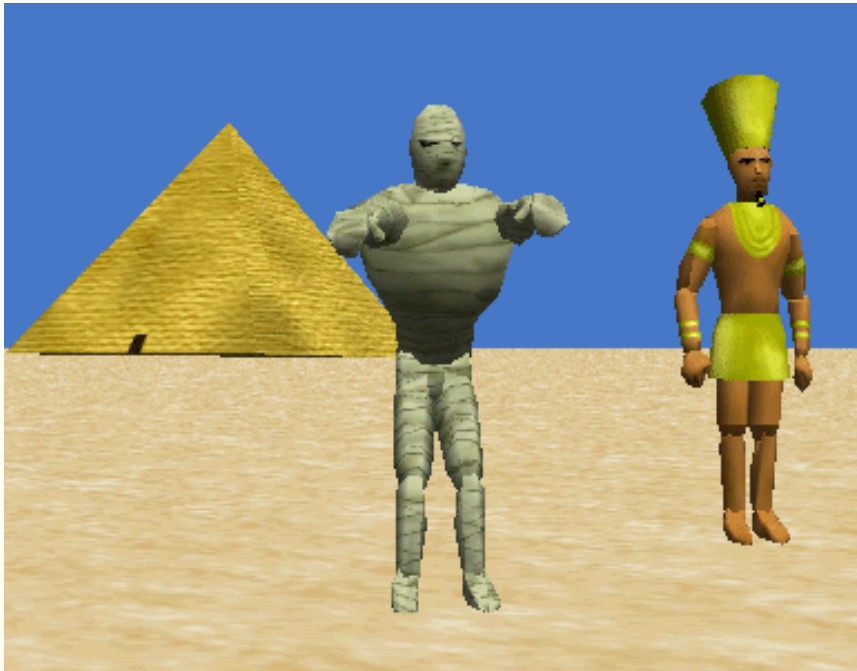
Functions

- 🌐 A **function** asks a question (to check a condition) or computes a value.
- 🌐 In Alice, a function is used to determine
 - 👤 the properties of objects
 - 💡 **Is the snowwoman's face red?**
 - 👤 the relationship of one object to another
 - 💡 **What is the distance between the mummy and pyramid?**
 - 👤 a current condition
 - 💡 **What key (on the keyboard) was pressed?**
- 🌐 Let's look at an example...



Problem Example

- 🌐 A Hollywood movie set. The camera angle influences our perception of the scene. Is the mummy taller than the pharaoh? How far (meters) is the mummy from the pharaoh?



Built-in Functions

Categories

- proximity
- size
- spatial relation
- point of view
- other

The screenshot shows a software interface with a hierarchy of objects on the left and a detailed view of the 'mummy' object on the right. The hierarchy includes 'World', 'Camera', 'Light', 'ground', 'pyramid', 'mummy', and 'pharaoh'. The 'mummy' object is circled in red. The detailed view shows the 'mummy's details' panel with tabs for 'properties', 'methods', and 'functions'. The 'functions' tab is selected and circled in red. Under the 'functions' tab, the 'proximity' category is expanded and circled in red. A red arrow points from the 'proximity' category to a text box at the bottom of the slide.

World

- Camera
- Light
- ground
- pyramid
- mummy**
- pharaoh

mummy's details

properties methods **functions**

- proximity**
 - mummy is within threshold of object
 - mummy is at least threshold away from object
 - mummy distance to
 - mummy distance to the left of
 - mummy distance to the right of

This example illustrates some built-in proximity functions.



Values

- 🌐 When a function is used to ask a question or perform a computation, an answer is returned.
- 🌐 The answer is called a **value**.
- 🌐 The **type** of value depends on the kind of function.
 - 👤 In our example, we want to ask the question:
 - 💡 What is the distance of the mummy to the pharaoh?
 - 👤 We expect to get a number value. It could be a whole number or a fractional value, such as
 - 💡 3 meters or 1.2 meters



Demo

- Ch03Lec1mummyDistanceFunction

- Concepts illustrated in this example program:

- ✦ The built-in *distance to* function determines the distance from the center of one object to the center of another object.

- ✦ A function is not a "stand-alone" instruction; it is **nested** within another instruction.



Types of Values

🌐 In our example, we used a function that has a number value. Other types of values include:

🌐 **Boolean**

🐙 *true, false*

🌐 **String**

🐙 "Oh, Yeah!"

🌐 **Object**

🐙 snowman, helicopter

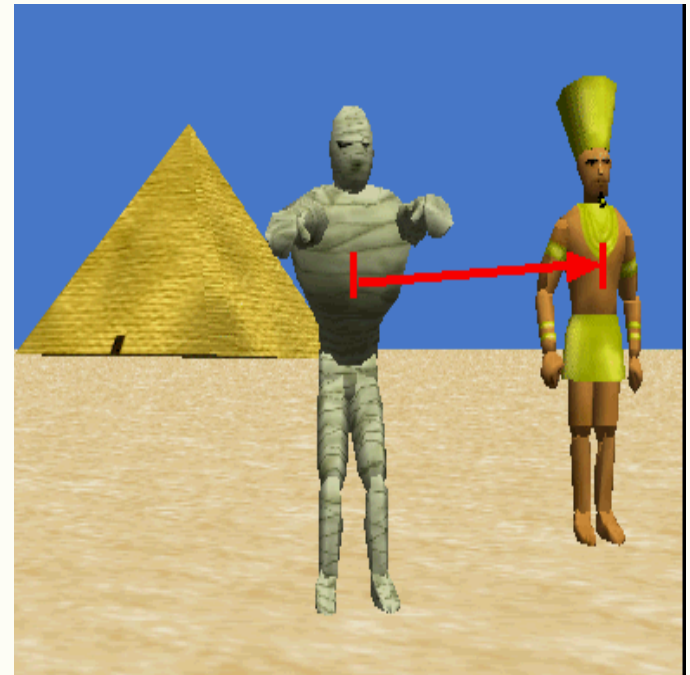
🌐 **Position in the world**

🐙 (0, 0, 0) – the center of an Alice world



Problem: Collision

- When the program is run, the mummy collides with the pharaoh.
- The problem is the distance between two objects is measured center-to-center.
- One way to avoid a collision is to subtract a small number (1 or 2) from the distance.



Expressions

- 🌐 An expression is a math or logic operation on numbers or other types of values
- 🌐 Alice provides math operators for common math expressions:
 - 💡 addition +
 - 💡 subtraction -
 - 💡 multiplication *
 - 💡 division /



Demo

● Ch03_mummyExpression

● Concept illustrated in this example:

- 📺 Math expressions are created within an instruction.
- 📺 Available math operators are +, -, *, /



Demo

- Ch03_mummyExpressionV2
- Subtracting 2 meters from the distance is an arbitrary amount.
- To be more precise, we could subtract the width of the pharaoh.
- The resulting expression subtracts the value of one function from the value of another function.



Demo

- Ch03_mummyExpressionV3

- To be even more precise, we could subtract half of the sum of the pharaoh's and mummy's width. That is,

(width of the pharaoh + width of mummy)/2

- Now, the expression gives a very close estimate of the distance to move.



Programming: Simple Control Structures

Alice



Control Statements

- We have been using *Do in order* and *Do together* to control the way instructions are executed in your Alice program.
- Control statements can also be used for
 - 🏰 conditional execution
 - 🏰 repetition



Conditional Execution

- **Conditional execution** is where some condition is checked and a decision is made about whether a block of the program will be executed.
- Conditional execution is extremely useful in
 - 🤖 games
 - 🤖 simulations
 - 🤖 real-time controls, e.g. robot systems



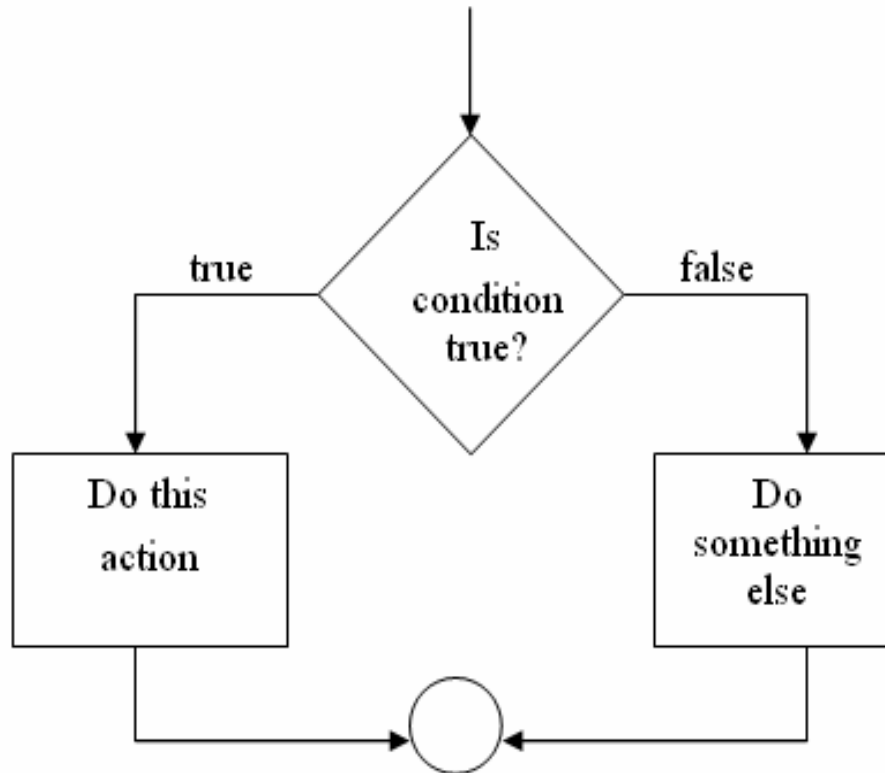
Example

- As a simple example of conditional execution, let's revisit the Egyptian scene in the Hollywood movie set.
 - ▶ The camera angle can mislead our perception of the scene
 - ▶ We want to check a condition (is the mummy taller than the pharaoh?) and then perform an action based on the whether the condition is true.



If/Else

- In Alice, an **If/Else** control statement is used to check a condition and make a decision.

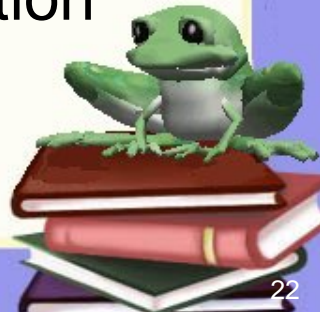


Storyboard

- In our example, we can demonstrate which object is the tallest by having that object turn around.
- A storyboard design for this action is:

```
If mummy is taller than pharaoh  
    mummy turns 1 revolution  
Else  
    pharaoh turns 1 revolution
```

- The condition in an *If* statement is a Boolean function returning a Boolean (*true* or *false*) value.

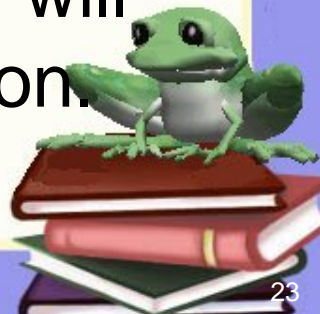


Demo

● Ch03Lec2mummyIfElse

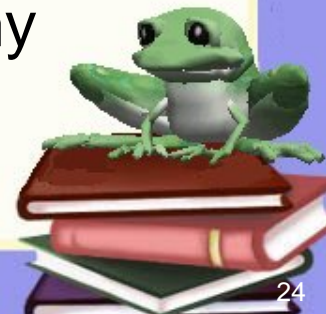
● Concepts illustrated in this example program:

- 🎬 The condition of an If/Else statement is constructed by using a function that returns a Boolean value (true or false).
- 🎬 *Do nothing* in the Else part of the statement means that if the condition is false, Alice will skip this part and go to the next instruction.



A different scenario

- 🌐 In some cases, the built-in functions are not sufficient for a condition that we want to check.
 - 👤 For example, the built-in function *is taller than* compares the heights of two objects. We used this function to compare the heights of the mummy and the pharaoh.
 - 👤 Suppose, however, that the casting call for the mummy requires that the actor is more than 2 meters tall.
 - 👤 How can we compare the height of the mummy to a specific measurement (2 meters)?



Relational Operators

- In situations where you need to write your own comparison, you can use a relational operator.
- Relational operators are provided in the World's built-in functions.

The diagram shows a hierarchy of built-in functions. At the top is a grey box labeled "world's details". Below it are three smaller grey boxes: "properties", "methods", and "functions". The "functions" box is circled in red. Below the "functions" box is a white box labeled "math", also circled in red. To the right of the "math" box is a list of six relational operators, each in a yellow box with a dotted left side, followed by its corresponding text description in red italics.

$a == b$	<i>is equal to</i>
$a != b$	<i>is not equal to</i>
$a > b$	<i>is greater than</i>
$a >= b$	<i>is greater than or equal to</i>
$a < b$	<i>is less than</i>
$a <= b$	<i>is less than or equal to</i>



Demo

🌐 Ch03Lec2mummyRelationalOps

🌐 Concepts illustrated in this example program:

- 📺 Relational operations are defined using the world's built-in functions.
- 📺 Placeholder values are first selected for the "a" and "b" components of the relational expression. Then the placeholder values are each replaced by either a function or a number.



Example

- Let's write code to make the mummy "walk" – a somewhat stilted motion like you would see in an old Hollywood horror film.
- The code will be more complex because we need to
 - alternate left and right leg motions
 - coordinate leg motions with the body moving forward



Storyboard

Do in order

Do together //move body and left leg

mummy move forward 0.25 meters

Do in order

mummy's left leg move backward

mummy's left leg move forward

Do together //move body and right leg

mummy move forward 0.25 meters

Do in order

mummy's right leg move backward

mummy's right leg move forward



Demo

 Ch03Lec2mummySimpleSteps



Need for Repetition

- In this example the mummy takes 1 step forward on each leg.
- Suppose we want the mummy to take 20 steps forward?



Loop

- The **Loop** statement is a simple control structure that provides for repeating an instruction (or block of instructions) a counted number of times.



Demo

 Ch03Lec2mummyLoop



Assignment

- Do Lab 3
- Read Chapter 4

