U5L07 Notes:

If statements – conditional statements – conditionals – THEY ARE ALL THE SAME

- If-statements exist so that your program can adapt, respond, or make choices about whether or not to execute certain portions of code based on some condition that you check while the program is executing.
- Because it involves checking conditions, these statements are sometimes called conditional statements, or sometimes just conditionals.
- A conditional statement (if-statement) requires a conditional expression, something that is either true or false and it's what an if-statement uses to decide whether or not to execute a certain portion of code.
- A generic term used by the AP CSP Framework for this is Selection. As in: your program can select whether or not to run certain blocks of code.

The whole point is to be able to handle cases where you can't know ahead of time whether or when certain conditions will exist in your program. So you have to write code to say something like: "Okay, at this point in the program, if* such and such *is true, then do* this*, otherwise do* that*."

U5L08 Notes:

- Boolean - A single value of either TRUE or FALSE
- Boolean Expression - in programming, an expression that evaluates to True or False.
- Conditionals - Statements that only run under certain conditions.
- If-Statement - The common programming structure that implements "conditional statements".
- Selection - A generic term for a type of programming statement (usually an if-statement) that uses a Boolean condition to determine, or select, whether or not to run a certain block of statements.

Reference: Examples

Below are a bunch of examples of how you might see comparisons in code. Review them if you like or continue on and come back if you need reference.

Compares two values - numbers, strings, or other booleans - and returns true if they are equal, otherwise false.

- `"Hello" == "hello"` returns false -- because the strings are slightly different.
- `"3" == 3` returns true -- `==` tries to be forgiving. If it can "coerce" a string into a number it will do so to compare. [1]
- `(2+1) == 3` returns true -- because the arithmetic expression evaluates to 3.
- `x == 7` returns true -- when the variable x has the value 7.

[1] While it is a useful feature that == will coerce a string into a number, it is considered TRICKY since string "3" is not the same as the integer 3 underneath the hood. There are circumstances when you would consider these not equal. There is a "strict" equality operator - the "triple equal" `===` which makes sure that the both the type of data and value are equal. So `"3" === 3` is false.

Compares two values - numbers, strings, or other booleans - and returns `true` if they are not equal, otherwise `false`.

- `"Hello" != "hello"` returns true -- because the strings are slightly different.

- `"3" != 3` returns false -- because the string 3 can be coerced into a number before comparing with 3. (see notes above about the forgiving ==).
- `(2+1) != 3` returns false -- because the arithmetic expression evaluates to 3.
- `x != 7` returns true -- when the variable x is any value other than 7.

is greater than

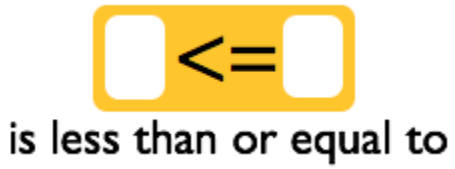
Compares two values to see if the number on the left is greater than the number on the right.

- `4 > 3` returns true
- `3 > 7` returns false
- `age > 17` returns true -- when the value of the variable "age" is strictly greater than 17, otherwise false.

is less than

Compares two values to see if the number on the left is less than the number on the right.

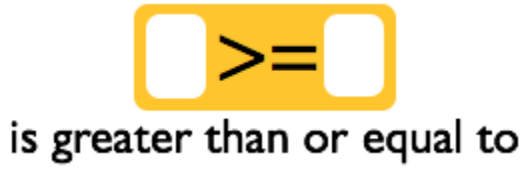
- `4 < 3` returns false
- `3 < 7` returns true
- `age < 17` returns true -- when the value of the variable "age" is strictly less than 17, otherwise false.



is less than or equal to

Compares two values to see if the number on the left is less than or equal to the number on the right.

- `3 <= 4` returns true
- `4 <= 3` returns false
- `age <= 18` returns true -- when the value of the variable "age" is 18 or less.



Compares two values to see if the number on the left is greater than or equal to the number on the right.

- `3 >= 4` returns false
- `4 >= 3` returns true
- `age >= 18` returns true -- when the value of the variable "age" is 18 or greater.

Understanding Program Flow

Programs are said to have a "flow of execution". You start by executing some line of code and then the next and so on.

A flow chart is a common visual that's used to represent the various paths of execution that your program might take. Many people use them to help plan.

1. This flow chart depicts a program executing one line after another until it gets to a point where it needs to make a decision.
2. In order to determine which path to take you state some condition. It should be a Boolean expression - something that evaluates to true or false. Here we have a simple comparsion of two values: the person's age and the number 18.
3. The program does one thing if the condition is true, and something else if the condition is false.
4. The program can continue a single thread of execution after the condition as well.

How If-statements work

`if` statements are the lines of code you need to change the flow while you're program is running. You can write code that makes a decision that determines which lines of code should be run next.
At the right is a diagram showing the elements of a basic `if` statement in JavaScript.

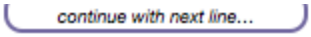
There are two basic parts to an if-statement.

1. A condition to be evaluated (A Boolean expression that evaluates to true or false)
2. Code that should run if the expression was true - enclosed in curly braces

How If-Else Statements work

With an if-else statement you are giving an either-or command:
either the lines of code inside the if will execute or the lines inside the else will execute. Those are the options.
You saw in the video how to add an else clause to an if-statement -- hit the little `+` symbol on the tail of the if statement.
Inside the curly braces for the else clause you put lines of code that you want to run if the Boolean condition from the if statement is false.
Some important notes about the else clause:

- The `else` must come immediately after the closing curly brace of an if statement
- The `else` also has its own set of opening and closing curly braces to encapsulate lines of code



Considering our flow chart from before, until now we haven't had a way to make the program do something different if the condition was false. With an `if-else`statement we do.
We can now write a program that "branches" at a particular point, running one of two possible sections of code.

How Dropdown Menus Work

A dropdown menu provides a fixed list of choices for a user to choose from. If you're expecting only a fixed set of responses from a user then it's a good alternative to a plain text input box. For example, if you want the user to select a day of the week.
Dropdown menu v. text input
Choosing a day of the week is an example of when it's a good idea to use a dropdown menubecause there are only 7 days of the week, but there are countless ways a user might type it if you used a text input box. Rather than trying to figure out what the user typed, you can provide your own list. It makes it much easier to check.

How to get what the user chose

To get the value the user selected from the dropdown menu is very similar to getting the text out of a text input.

Design Mode

In the events tab you can click the shortcut to insert an event handler into the code.

1. The event type "change" gets triggered when the user has made a new selection. The "change" event fires when the user releases the mouse button on a new selection. (You don't want "click" because that would trigger the moment the user clicked the arrow to open up the menu in the first place.)

2. You get the value that was selected the same way you get data out of a text input boxwith `getText(id)`. If you call `getText(id)` on a dropdown menu it returns the currently selected value in the menu.

Lesson 09: if-else-if and Conditional Logic

- Boolean - A single value of either TRUE or FALSE
- Boolean Expression - in programming, an expression that evaluates to True or False.
- Conditionals - Statements that only run under certain conditions.
- If-Statement - The common programming structure that implements "conditional statements".
- Selection - A generic term for a type of programming statement (usually an if-statement) that uses a Boolean condition to determine, or select, whether or not to run a certain block of statements

New Code

- `if ( ){ // if code } else { // else code }`
- `&&`
- `||`
- `!`

How if-else-if Works

Not all conditions you want to check have only two possible outcomes. However a computer can only check one true/false condition at a time.

- You add an `else-if` clause to an ifstatement when you have another condition you want to check.
- When you hit the `+` on the tail of an else clause it will add an else if into the statement.
- You can add as many `else-if`s as you want.
- Each condition in an if-else-if is checked in order from top to bottom and the final else clause is executed if all the previous conditions evaluated to false.

The order of the conditions matters!

Without a final else clause it's possible that the whole structure can be skipped

Else-if is Convenient but Not Required (for AP)

NOTE: The AP Pseudocode DOES NOT Have else-if

Boolean operators && and || and !

The logical operators -- also known as the Boolean Operators -- AND (`&&`), OR (`||`) and NOT (`!`) allow you to compare the results of more than one Boolean operation at a time.

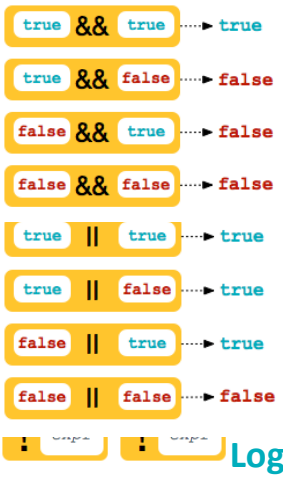
NOTE: the OR is made with two vertical "pipe" characters. The "pipe" is on the keyboard with same button as `\` -- it's right next to the key with `}]`on it, just above the `Return/enter` key.

Example:
The && operator (called "AND") lets you check whether two conditions are both true at the same time. Consider the statement below:

Compound Boolean Expression

This says: "True or false: is it the case that BOTH `age >= 13` AND `age < 21`?"
If both of expression 1 and expression 2 return true then the larger compound boolean expression returns `true`. You can replace expression 1 and expression 2 in that statement with anything that evaluates to true/false.

Breakdown: AND, OR and NOT



Logical AND

expr2 are true then the whole expression returns `true`.

se then the whole expression returns `false`.

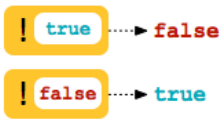
Logical OR

expr2 are true then the whole expression returns `true`.

alse then the whole expression returns `false`.

Logical NOT

This operator takes a single true/false value and inverts it. For example, maybe you want to say something like, "if it's NOT the case that age is less



rite this: `!(age < 21)`. Even though that's logically the same as `(age >= 21)`

ficient to state something in the negative with NOT.

Lesson 10: Building an App (Color Sleuth)

Vocab

- Boolean Expression - in programming, an expression that evaluates to True or False.
- Conditionals - Statements that only run under certain conditions.
- If-Statement - The common programming structure that implements "conditional statements".
- Selection - A generic term for a type of programming statement (usually an if-statement) that uses a Boolean condition to determine, or select, whether or not to run a certain block of statements.

New Code

- `setProperty(id, property, value)`
- `rgb(r, g, b, a)`

How setProperty Works

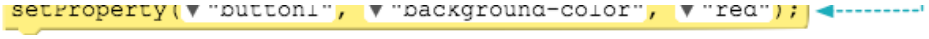
`setProperty` is the code that lets you set properties of UI elements like color, width, font size, etc. Any property you can set in Design Mode can also be set in code using `setProperty`

1. In Code Mode you can find `setProperty` in the UI Controls toolbox.
2. Choose the id of the UI element you want to change a property of
3. The pulldown menu shows a list of properties that you can set for that element. It matches the list of properties that you see for an element in Design Mode
4. The last parameter is the value you want to set for that property. We're interested in background-color which has a variety of values you can give it.

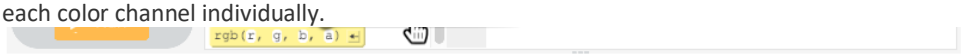
How to set the "background-color"

There are a number of ways you can specify colors in App Lab.

1. background-color is a string (in quotes) that represents the color. Typically the string is a named color like "red" or a hex value.



2. you can also use the `rgb` function to set the color. The `rgb` function is a convenient way to compose colors by setting each color channel individually.



3. You'll notice that the `rgb` function has 4 parameters for the red, green, blue and alpha values you want. The alpha value is decimal number between 0 and 1 that represents how opaque the color is (opaque is the opposite of transparent) as a percentage. `1` means fully opaque, and for example `0.5` means 50% opaque.

4. You can omit the alpha value entirely - it will default to 1.0 - by clicking the parameter arrow to collapse the last parameter. For the app we're making we probably don't need transparent colors so this will make our code simpler to read.
5. The `rgb` function actually just returns a color string as well. If we store the value that `rgb` returns in a variable then we can simply plug in that variable as the value for `setProperty` for several different UI elements if we want. Below we use `myColor` to set both the button color and the icon color of the screen.

Lesson 11: While Loops

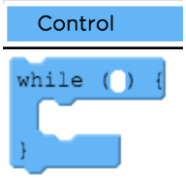
Vocab

- Iterate - To repeat in order to achieve, or get closer to, a desired goal.
- while loop - a programming construct used to repeat a set of commands (loop) as long as (while) a boolean condition is true.

New Code

- `while( ){ // code }`

while Loops



The `while` loop uses a boolean condition to repeatedly run a block of code. It checks the expression, and if it is true it runs the block of code contained within it. This process of checking the condition and running the block of code is repeated as long as the boolean condition remains true. Once the boolean expression becomes false it will stop.

Infinite while Loops

`while` loops run until their condition becomes false, which raises an interesting question. What happens if the condition never becomes false? In these cases the program enters what is called an infinite loop over the commands in the `while` loop, and it never reaches the rest of your program. We normally avoid infinite loops in our programs

Update Condition

In order for a `while` loop to stop at some point, the code inside the loop must change something about the state of the program - usually the value of a variable - so that eventually the boolean expression becomes false. Otherwise you'd have an infinite loop!

Expressing Stopping Conditions: "Until Loops"

It is often more natural to think about looping in terms of when the loop should end rather than when it should continue. For example you might say "keep going down the road until you see the gas station" or "keep calling until you get through to someone." You might think of these as

"until loops" rather than "while loops," since we want the loop to continue until a condition is true rather than while a condition is true.

There is no "until loop" in JavaScript but it is actually quite easy to translate "until loops" into `while` loops so that you can use them in programs. An "until loop" runs until a condition is true, as opposed to a while loop which runs as long as a condition is true. That means an until loop is the logical inverse of a while loop - it runs as long as the condition is false. The table below shows how you can use the NOT (`!`) operator to translate stopping conditions into `while` loop conditions.

Stop once you reach the gas station	Keep going until you reach the gas station	Keep going while you have NOT reached the gas station
Stop calling when you get through to someone	Keep calling until you get through to someone	Keep calling while you have NOT gotten through to someone
Stop when `x == y`	`until(x == y){...`	`while(!(x == y)){...`

Note how we can use the NOT operation to find the logical inverse (or opposite) of the condition from our "until loop" to create a `while`loop. Let's do a little practice of that now.

Introducing the ++ Operator

You can write `count++` to add 1 to `count`. `count++` does the exact same thing as `count = count + 1`! In fact the computer turns `count++` into `count = count + 1` behind the scenes - it really is just a convenient shorthand.

Note: This is more of a programmer style choice so if you want to write your code using `count = count + 1` instead there is nothing wrong with that!

++ Has a Friend! Introducing --

As you may have guessed, just as we can write `count = count + 1` as `count++` we can also write `count = count - 1` as `count--`.

Let's write a program that counts down from 10 down to 1.

Introducing `+=` and `-=`!

We can use the `+=` or `-=` operator to add or subtract any value we want to the current value of a variable.

So, the shorthand versions of `count = count + 3` and `count = count - 3` would be:

- `count += 3`
- `count -= 3`

The -= Operator

The `-=` operator works almost identically to `+=`, but instead it subtracts the value provided from the variable.