**Conditionals**

**Intro to Conditionals**

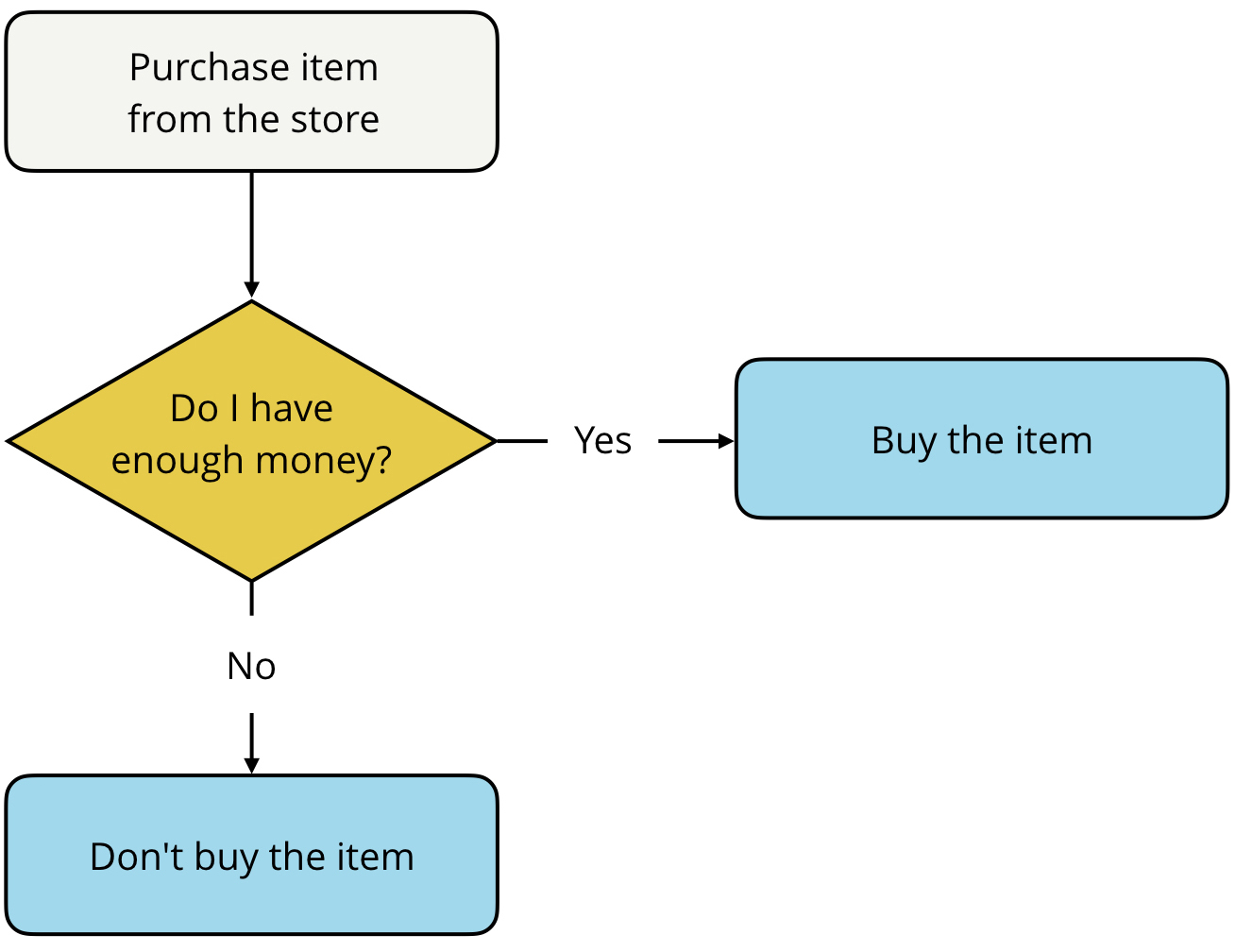
When we’re writing code, the main goal is to solve problems. The steps our code takes to solve a problem is known as an algorithm.

To solve problems we can make use of flowcharts, a diagram for how to solve a problem.

**Flowcharts**

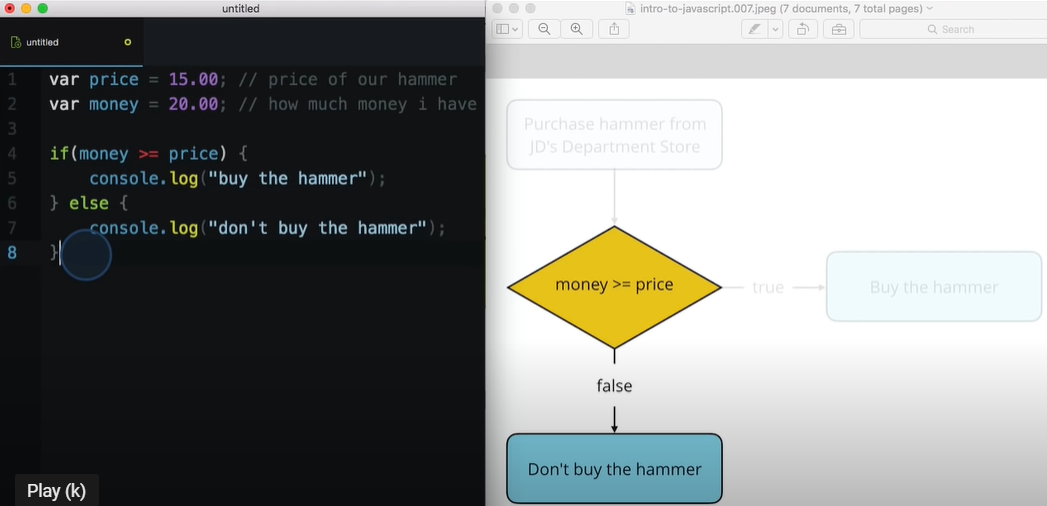
A flowchart is a visual diagram that outlines the solution to a problem through a series of logical statements.

The order in which statements are evaluated and executed is called **control flow**.



**Flowcharts to Code**

We can create the control flow described above:



**If … Else Statements**

If...else statements allow us to execute certain pieces of code based on a condition, or set of conditions, being met.

This is extremely helpful because it let us choose which piece of code we want to execute based on the result of an expression:

var a = 1;

var b = 2;

if (a > b) {

console.log("a is greater than b");

} else {

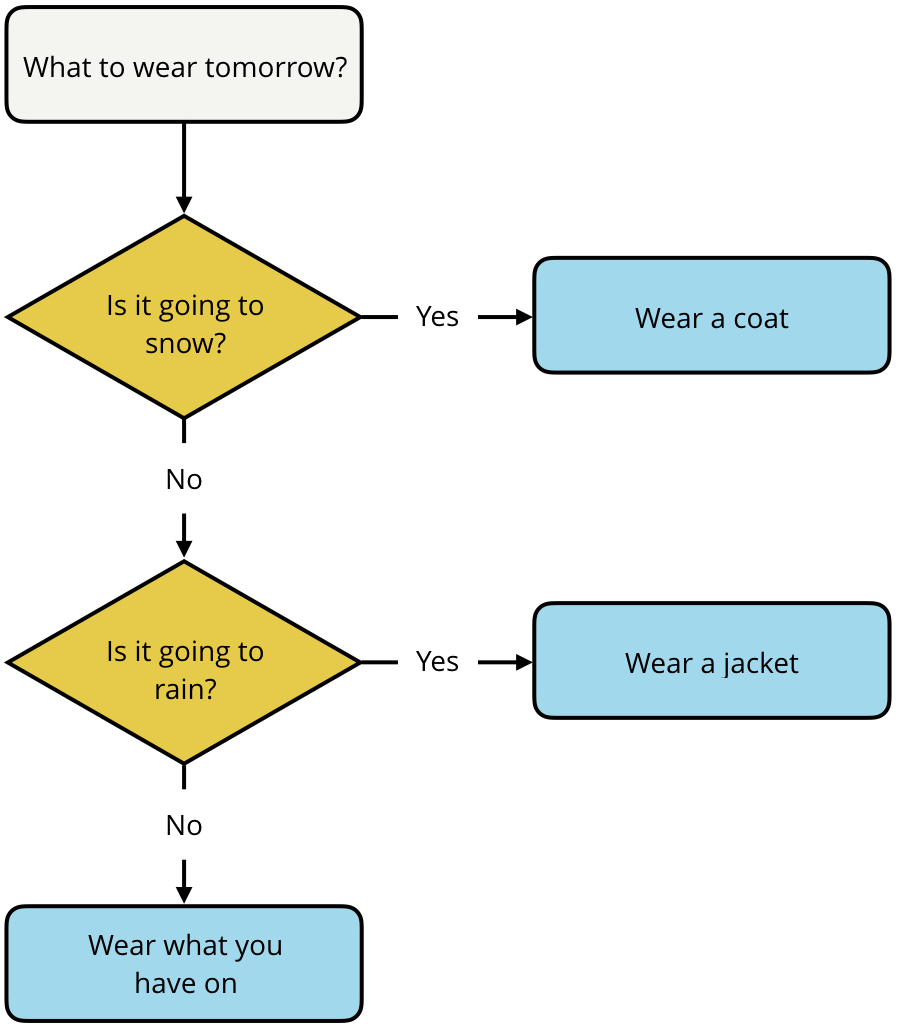
console.log("a is less than or equal to b");

}

The value inside the if statement is always converted to true or false. Depending on the value, the code inside the if statement runs the code inside the else statement run, but not both. We use curly braces {...} to separate conditions and indicate which code should be run.

**Else If Statements**

In some situations, two conditionals aren’t enough.



var weather = "sunny";

if (weather === "snow") {

console.log("Bring a coat.");

} else if (weather === "rain") {

console.log("Bring a rain jacket.");

} else {

console.log("Wear what you have on.");

}

By adding the extra else if statement, we’re adding an extra conditional statement.

**More Complex Problems**

Complex problems require multiple conditions to be true.

**Logical Operators**

With logical operators we can combine two or more logical expressions into one large logical expression. If both expressions are true, then the entire logical expression evaluates to true. If either one of the expressions is false, then the whole logical expression is false.

Logical operators can be used in conjunction with booleans values (true and false) to create complex logical expressions.

By combining two boolean values together with a logical operator, we create a logical expression that returns a boolean value:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example** | **How it works** |
| && | Logical AND | value1 && value2 | Returns true if **both** value1 **and** value2 evaluate to true. |
| || | Logical OR | value1 || value2 | Returns true if **either** value1 **or** value2 (**or even both!**) evaluates to true. |
| ! | Logical NOT | !value1 | Returns the **opposite** of value1. If value1 is true, then !value1 is false. |

**Logical AND and OR**

Truth tables are used to represent the result of all the possible combinations of inputs for a logical expression.

**&& (AND)**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A && B** |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

**|| (OR)**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A || B** |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

**Short-circuiting**

If the first argument satisfies a condition of an expression, the latter doesn’t need to be considered. For example, if the first argument returns FALSE while using AND operator, the second argument doesn’t need to be considered, once the result for the expression is already known.

**Advanced Conditionals**

Advanced conditions include truthy and falsy values, the ternary operator and switch statement.

**Truthy and Falsy**

Every value in JavaScript has an inherent boolean value. What that value is evaluated in the context of a boolean expression, the value will be transformed into that inherent boolean value.

**Falsy Values**

A value is falsy if it converts to false when evaluated in a boolean context. For example, an empty string “” is falsy because, “” evaluates to false:

**if** ("") {

console.log("the value is truthy");

} **else** {

console.log("the value is falsy");  
}

***Returns:*** *"the value is falsy"*

Falsy Values:

1. the Boolean value false
2. the null type
3. the undefined type
4. the number 0
5. the empty string ""
6. the odd value NaN (stands for "not a number", check out the [NaN MDN article](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN))

Example of Truthy Values:  
true  
42  
"Pizza"  
"0"  
"Null"  
"Undefined"  
{}  
[]

**Essentially, if it's not in the list of falsy values, then it's truthy!**

**Ternary Operator**

The ternary operator provides us with a shortcut alternative for writing lengthy if … else statements.

conditional ? (if condition is true) : (if condition is false)

We can replace this:

var isGoing = true;

var color;

if (isGoing) {

color = "green";

} else {

color = "red";

}

console.log(color);

With this:

var isGoing = false;

var color = isGoing? "green" : "blue";

console.log(color);

**Cool Example:**

var eatsPlants = false;

var eatsAnimals = false;

/\*

\* Test your code agaist the followig possible input/output combinations of (`eatsPlants`, `eatsAnimals`, expected output):

\* - (true, true, omnivore)

\* - (false, true, carnivore)

\* - (true, false, herbivore)

\* - (false, false, undefined)

\*/

var category = eatsPlants && eatsAnimals? "omnivore":

!eatsPlants && eatsAnimals? "carnivore":

eatsPlants && !eatsAnimals? "herbivore":"undefined";

console.log(category);

**Switch Statement**

If we’re repeating else if statements in our code, where each condition is based on the same value, then it might be time to use a switch statement.

A switch statement is another way to chain multiple else if statements that are based on the same value without using conditional statements.

This code:

var option = 3;

if (option === 1) {

console.log("You selected option 1.");

} else if (option === 2) {

console.log("You selected option 2.");

} else if (option === 3) {

console.log("You selected option 3.");

} else if (option === 4) {

console.log("You selected option 4.");

} else if (option === 5) {

console.log("You selected option 5.");

} else if (option === 6) {

console.log("You selected option 6.");

}

Can be replaced by this:

switch (option){

case 1:

console.log("You selected option 1.");

break;

case 2:

console.log("You selected option 2.");

break;

case 3:

console.log("You selected option 3.");

break;

case 4:

console.log("You selected option 4.");

break;

case 5:

console.log("You selected option 5.");

break;

case 6:

console.log("You selected option 6.");

}

The break statement can be used to terminate a switch statement and transfer control to the code following the terminated statement.

If we don’t add the break statements we can get a falling-through behaviour.

**Falling-Through**

In some cases, we might want to leverage the “falling-through” behaviour of switch statements to our advantage.

For example, when our code follows a hierarchical-type structure.

var tier = "nsfw deck";

var output = "You’ll receive "

switch (tier) {

case "deck of legends":

output += "a custom card, ";

case "collector's deck":

output += "a signed version of the Exploding Kittens deck, ";

case "nsfw deck":

output += "one copy of the NSFW (Not Safe for Work) Exploding Kittens card game and ";

default:

output += "one copy of the Exploding Kittens card game.";

}

console.log(output);

In this example, each successive tier builds on the next by adding more to the output. Without any break statements in the code, it continues to fall-through until reaching the end of the switch statement.

**Default**

When adding the default case to the switch statement, it will be executed when none of the values match the case (value of the switch expression).