CONDITIONALS AND BUILT-IN METHODS

CONDITIONALS

So, what do **conditionals** involve (at least for today)?

- **if/else statements**
- **Boolean logic** to combine and manipulate conditional tests
- **for, forEach, while, do while**

for, forEach, while, do while are also **loops**, which we'll expand on in a later week

Conditional statements allow us to decide which bit of code to **execute** and which to skip based on the results of whatever **condition** we stated.

A **condition** is sort of like a test

JavaScript makes use of two conditional statements: **if/else** and **switch**

if/else statements are dependent on **boolean logic**

Remember what **boolean logic** is?

The block of code within the body of the **if statement**, curly brackets: { }, executes only if the **boolean logic** evaluates to true

```
if ( boolean logic ) {
 // run this code if 'boolean logic',
 // as a parameter, evaluates to true
}
```

An actual **if statement**

```
if ( 1 > 0 ) {
 console.log( "The number 1 is greater than 0" );
}
```

Take a second, try that in repl.it

That's very useful, but also kind of limiting no?

Why?

What if the **boolean logic** evaluates to **false**?

Where does the code go? What's the next step?

Else statement!:

Here's an example:

```
if ( boolean logic ) {
 // run this code if 'boolean logic',
 // as a parameter, evaluates to true
} else {
 // evaluate this code if
 // 'boolean logic' evaluates to false
}
```

Here's another example:

```
let number = 7;

if ( number > 5 ) {
 console.log("The variable number is greater than 5");
} else {
 console.log("The variable number is less than 5");
}
```

Take another second, try that in repl.it as well, and change the value of **number** to **console.log** both strings

But that's only two options, and in the real world, we'll want more.

So we can expand to **else if statements**

else if statements can test more than one criteria

Note, **JavaScript** will stop checking **conditionals** once it hits one that evaluates to **true**

An example:

```
let name = "puppies";

if ( name === "kittens" ) {
 name += "!";
 console.log(name);
} else if ( name === "puppies" ) {
 name += "!!";
 console.log(name);
} else {
 name = "!" + name;
 console.log(name);
}
```

Take a second, do that one in repl.it and play around, see if you can get it to work

A word of caution:

Do NOT assign values within a **conditional statement**

```
if ( x = "puppies" ) {
 console.log("False!");
}
```

Try that one in repl.it too, see what it says

Something you may encounter when Googling are **ternary operators**

In short, it's a concise **if/else statement**

Ternary:

```
( expression ) ? /* true value */ : /* false value */ ;

( 1 > 0 ) ? console.log( 'true' ) : console.log( 'false' );
```

So, looks different, probably faster to type, works the same

I'm sure some would argue **ternary operators** are best practice, but I'm not too worried about it

We're essentially doing this:

```
if ( expression ) {
 /* true value */ ;
} else {
 /* false value */ ;
}
```

So an example:

```
let age = 30;
let minAgeToVote = 18;
let allowedToVote = ( age > minAgeToVote ) ? "yes" : "no";
console.log( allowedToVote );
```

Take a crack at that one in repl.it

Alright, let's talk about **comparison operators**.

We can make comparisons using **equality comparison operators** and **relational operators**

Comparison:

```
==, !=, ===, !==
```

Relational:

```
>, <, >=, <=
```

Equality operator:

The double equals, ==

Note, JavaScript will perform something called **type conversion** in the background if the **operands** are different **types** to check if they're equal

```
"dog" == "dog";
"dog" == "cat";
"1" == "1";
1 == "2";
```

You shouldn't rely on **type conversion** though.

Try some of those in repl.it

Just to reiterate, **numbers** and **strings** that contain **numbers** of the same value will be considered equal.

Identity operator: also refered to as the **strict equality** operator. It compares *both* type and values

```
1 === "1";
```

Try that in repl.it, what's it return?

So, no **type conversion**. We should ALWAYS use this because it's the most precise.

We can also compare **objects**. So, any **object** (like an **array**) is only equal to itself.

Objects are compared by **reference**, not **value**

```
[] === [];
// => false

let a = [];
a === [];
// => false

a === a;
// => true
```

Again, to be explicit, **primitives** are compared by **value** whereas **objects** are compared via where they're stored in **memory**

Moving on to **inequality operators**, we have **!=**, and **!==** where the latter is the strict equality operator

The **inequality operator** returns true if opperands are not equal. And just like before, if the two operands are not of the same type JavaScript will try and perform **type conversion**

```
1 != "1"
// => false

1 !== "1"
// => true

1 !== 2
// => true
```

There are also **logical operators**, of which we've been using without defining.

&& - means "and"

|| - means "or"

The **&&** operator requires both values to be **true** to **return true**, otherwise it will **return false**

```
true && true
// => true

true && false
// => false

false && false
// => false
```

Okay, that's a lot of information. Let's try and practically apply this via checking a password.

Try this in repl.it:

```
let network = "SVA-Guest";
let pw = "Paintbrush";

if ( (network === "SVA-Guest") && (pw === "paintbrush") ) {
 console.log( "Wifi Access Granted" );
} else {
 console.log( "Wifi Access Denied" );
}
```

What will the console show?

The || operator only requires *either* of the values to be **true** to **return true**, other it **returns false**

```
true || false
// => true

false || true
// => true

false || false
// => false
```

The || operator with an **if statement**:

```
let day = "Monday";
if ( (day === "Monday") || (day === "Wednesday") ) {
 console.log( "We have class!" );
}
```

The || operator can often be used for **default** values, since only one value needs to be **true**

```
// our saySomething() function takes an
// argument called 'message'
function saySomething(message) {
 let loggedMessage = message || "Hello World!";
 console.log( loggedMessage );
}
// but what happens if you invoke the saySomething()
// function without passing an argument?
saySomething();
```

Try that in repl.it

With all of this in mind, try an eligibility exercise:

Using repl.it, write a program that outputs a message based on a user's age.

The program must **console.log** *only* the most recent item a person can do. For example, if a user's age is 46, the message should **console.log** "You can run for president!"

Stipulations:

- Under 16: 'You can go to school!'
- 16 or older: 'You can drive!'
- 18 or older: 'You can vote!'
- 21 or older: 'You can (legally) drink alcohol!'
- 25 or older: 'You can rent a car!'
- 35 or older: 'You can run for president!'
- 62 or older: 'You can collect social security!'

You can hardcode the age as a variable to test your code.

Don't forget, once **JavaScript** evaluates one of these **expressions** as **true**, it will stop.

As usual, there are other ways of going about this. Specifically, a **switch statement**

A **switch statement** first evaluates the **expression** and then matches the **expression's** value to a **case** clause. If there's a **match**, it **executes** the **statements** for that **clause**.

We also have to use a **break** to stop it from continuing to **evaluate** statements if there's a match. There's also an option for default.

```
switch ( expression ) {
 case valueOne:
  // statements
  break;
  ...
 case valueN:
  // statements
  break;
 default:
  // statements
  break;
}
```

An actual switch statement, try this in repl.it:

```
let num = 1;

switch ( num ) {
 case "1":
        console.log("You entered the string '1'");

 case valueTwo:
        console.log("You entered the number 1");

 default:
        console.log("You did not enter 1");
}
```

What happened?

Try this one, what's the difference?

```
let num = 1;

switch ( num ) {
 case "1":
        console.log("You entered the string '1'");
        break;
 case valueTwo:
        console.log("You entered the number 1");
        break;
 default:
        console.log("You did not enter 1");
}
```

If we're comparing against specific values in an **if/else statement**, we can almost always refactor to cleaner code using a **switch statement**.

Using repl.it, refactor the following code to use a **switch statement**:

```
let grade = 'B';
if ( grade === 'A' ) {
 console.log('Awesome job');
} else if ( grade === 'B' ) {
 console.log('Good job');
} else if ( grade === 'C' ) {
 console.log('Okay job');
} else if ( grade === 'D' ) {
 console.log( 'Not so good job' );
} else if ( grade === 'F' ) {
 console.log('Poor job');
} else {
 console.log('Unexpected grade value entered');
}
```

ANSWER...

```
let grade = 'B';
switch ( grade ) {
        case 'A':
                console.log('Awesome job'); break;
        case 'B':
                console.log('Good job'); break;
        case 'C':
                console.log('Okay job'); break;
        case 'D':
                console.log('Not so good job'); break;
        case 'F':
                console.log('Poor job'); break;
        default:
                console.log('Unexpected grade value entered');
}
```

And what happens if you take the **break;** statement out?

```
// Good job
// Okay job
// Not so good job
// Poor job
// Unexpected grade value entered
```

There's a technique, similar to || in **if/else statements**. For example, what if we only cared about whether or not the student passed?

```
let grade = 'B';
switch ( grade ) {
        case 'A':
        case 'B':
        case 'C':
        case 'D':
                console.log('You passed!');
                break;
        case 'F':
                console.log('You failed!');
                break;
        default:
                console.log('Unexpected grade value entered');
}
```

BUILT-IN METHODS / FUNCTIONS

Like I've mentioned before, functions are re-usable pieces code that we can call later in a program that perform some sort of operation or process.

Methods are essentially the same, with different terms.

Specifically, a method is a procedure or function and associated with an object in object-oriented programming.

We'll differentiate and elaborate on these futher, later on in the semester, as we get more complex.

For now, just take for granted that there are a number of already existing methods/functions within JavaScript that we can make use of.

Check out a list of them here.

Take a bit of time to just read through the names of descriptions of those methods available to us. It will begin to start painting a picture of the basic buildings blocks to accomplish tasks.

For us, in the demo last week, we made use of:

- charAt()
- Math.floor()
- Math.random()
- console.log()

And others...

We'll introduce and use more built-in methods when we introduce the concept of arrays in Week 7

We technically used an array to temporarily store our generated password in the password demo, but we haven't formally defined it.

If you want to jump ahead, check it out on MDN:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Arra

Also, here are some built-in methods to be used in conjunction with arrays:

https://www.w3schools.com/js/js_array_methods.asp

As usual, email me with questions.