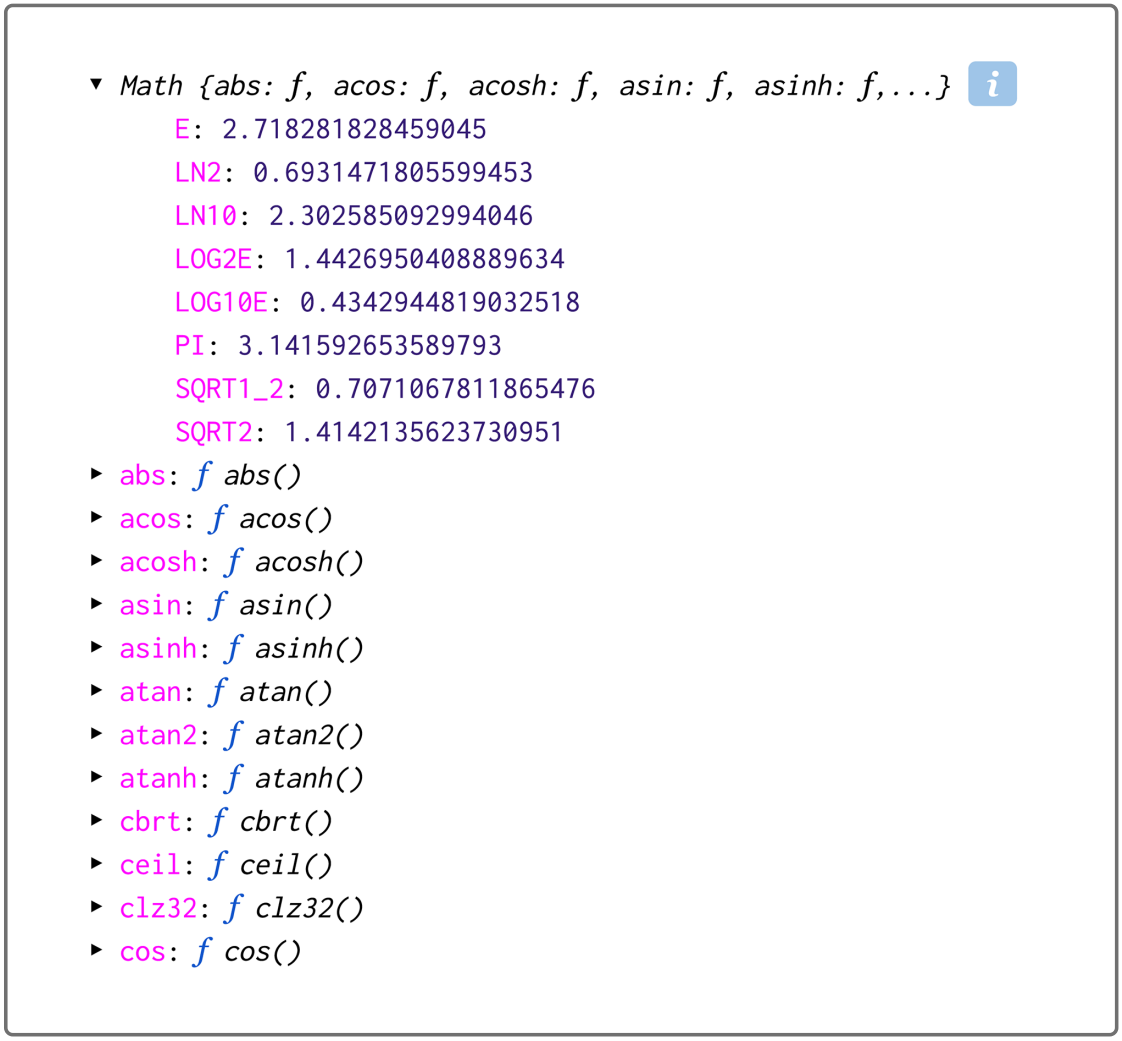
3.4.4 Use the Math Object

If we want to generate random numbers in JavaScript, we'll need the help of a built-in object called `Math`. Like `prompt()` and `alert()`, this is a property of the `window` object, but we're not required to write `window.Math`.

Try console logging the `Math` object and inspect what's printed in the DevTools:



We can see that `Math` has many properties and functions attached to it. When a function belongs to an object, though, we refer to it as a **method**.

Console log some of these properties and methods to see what they do:

```
// prints 3.141592653589793
console.log(Math.PI);

 // rounds to the nearest whole number (4)
console.log(Math.round(4.4));

 // prints the square root (5)
console.log(Math.sqrt(25));
```

DEEP DIVE

DEEP DIVE

DEEP DIVE

There are plenty more math-related methods available. To learn more, see the **MDN web docs on Math (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math#Methods)** .

The method that we're most interested in is `Math.random()`, but this can be tricky to use and understand. Let's warm up with `Math` by first using the `Math.max()` method. Given a series of numbers, this method will return the largest one.

Here are a few examples:

```
// prints 100
console.log(Math.max(10, 20, 100));

// prints 0
console.log(Math.max(0, -50));
```

How does that help us with the game? Think about areas where one of our number values could dip into negative territory. Player health, enemy health, and money are all deducted from at various points, and there's a chance these values could turn negative. Does it break the game if they do? No, not really, but it also doesn't look very professional. If we use something like `Math.max(0, variableName)`, we can ensure that deducted values always stop at zero.

Make the following changes in the `fight()` function:

- Replace `enemyHealth = enemyHealth - playerAttack;` with `enemyHealth = Math.max(0, enemyHealth - playerAttack);`
- Replace `playerHealth = playerHealth - enemyAttack;` with `playerHealth = Math.max(0, playerHealth - enemyAttack);`
- Replace `playerMoney = playerMoney - 10;` with `playerMoney = Math.max(0, playerMoney - 10);`

Save and test the game, verifying that your `alert()` or `console.log()` methods never display a negative health or money value.

PAUSE

Alternatively how could we have prevented negative values with `if` statements instead of `Math.max()`?

```
playerMoney = playerMoney - 10;

if (playerMoney < 0) {
  playerMoney = 0;
}
```

Hide Answer

Now that we have a taste of the `Math` object, let's dive into its random capabilities. Console log `Math.random()` a few times and you'll get some interesting numbers like 0.7353300720527607 or 0.25000120638240264.

The `Math.random()` method returns a random decimal number between 0 and 1 (but not including 1, meaning you would never get exactly 1). For this decimal number to be useful, we have to pair it with other math operations.

In the `startGame()` function, replace the line `enemyHealth = 50;` with the following:

```
enemyHealth = Math.floor(Math.random() * 60);
```

By multiplying `Math.random()` by 60, we've now specified a random range from 0 to 59.xx (remember, `Math.random()` will never be 1, so we would never get an even 60). We don't want decimal numbers cluttering up our game, though, so we can use `Math.floor()` to round down to the nearest whole number. This means that at the start of each round, `enemyHealth` would be a random whole number from 0 to 59.

Hmm. This still isn't perfect. Even though the odds are low, we don't want to risk an enemy starting with zero health. Ideally, enemy health should be between 40 and 60, which we can still achieve with a little extra math!

Update the line in `startGame()` to look like this:

```
enemyHealth = Math.floor(Math.random() * 21) + 40;
```

Okay, now this random logic is getting confusing! To understand what's happening, we should start from the inside out. Let's break it down:

1. `Math.random() * 21` will give us a random decimal number between 0 and 20.xx.

2. `Math.floor()` will round this number down, so now the range is a whole number between 0 and 20.

3. We'll always add 40 to the generated number. If the random number is 0, we at least have 40. If the random number is 20, we have our upper limit: 60.

Play the game again and notice how each enemy starts with a different health value! There are still other places where we could use a random number, though, making this a good use case for a function.

Add this function alongside the other functions in `game.js`:

```
// function to generate a random numeric value
var randomNumber = function() {
  var value = Math.floor(Math.random() * 21) + 40;

  return value;
};
```

Wait, there's a keyword in there that we haven't talked about yet: `return`. What does that do? Think back to how we used the `window.prompt()` method. When called, this method would give us a string that we could then store in a variable. As we write our own methods and functions, they can optionally give something back, too, using a `return` statement.

To see this in action, replace the random logic in `startGame()` with a call to the new `randomNumber()` function:

```
enemyHealth = randomNumber();
```

Because `randomNumber()` returns a value, that returned value can be stored in the `enemyHealth` variable.

DEEP DIVE

DEEP DIVE

The `return` statement actually serves two purposes. Yes, it returns a value, but it also ends function execution right then and there. Consider the following example:

```
var doubleIt = function(num) {
 console.log("beginning of function");

 var double = num * 2;

 return double;

 console.log("end of function");
};

var newNumber = doubleIt(5); // is now 10
```

The second console log, `"end of function"`, never happens, because the function has returned, or ended, before it reached that line. It's similar to using a `break` statement in a `for` or `while` loop.

We have a `randomNumber()` function in place now, but it's set up to return a random value between 40 and 60. Other areas of the game will need a random number between a different range. Fortunately, we can reuse the same `randomNumber()` function by adding **parameters**. This will be similar to the `enemyName` parameter that was added to `fight()` earlier:

```
var fight = function(enemyName) {

};
```

In the case of `randomNumber()`, we actually want two parameters: one to represent the lower limit and one to represent the upper limit. We'll adjust our `Math.random()` logic to accommodate both values.

Rewrite the `randomNumber()` function like so:

```
var randomNumber = function(min, max) {
  var value = Math.floor(Math.random() * (max - min + 1) + min);

  return value;
};
```

That looks pretty confusing. Let's try breaking it down again. If we want a random number between 40 and 60, we would call the function as `randomNumber(40, 60)`. That means `min` would be 40 and `max` would be 60. We can mentally swap out those numbers if it helps:

```
var randomNumber = function(40, 60) {
  var value = Math.floor(Math.random() * (60 - 40 + 1)) + 40;

  return value;
};
```

Then start performing math operations, and suddenly we're back in familiar territory:

```
var randomNumber = function(40, 60) {
  var value = Math.floor(Math.random() * (21)) + 40;

  return value;
};
```

Working with random numbers can definitely be tricky. Fortunately, there are many helpful articles online that can be found with a quick Google search for "js random numbers". For now, our `randomNumber()` function seems to do what we want. We won't need to edit it any further and can focus on where to call it.

ON THE JOB

Random numbers are used often in game development, but it might be harder to see their application in non-gaming apps. However, random numbers can still be useful for situations like helping users generate a password, assigning students to a study group, picking a sweepstakes winner, or shuffling an image gallery.

If you haven't already, make one last change in `startGame()` to set `enemyHealth` correctly:

```
enemyHealth = randomNumber(40, 60);
```

In the `fight()` function, we'll need to update the places where health is deducted based on attack damage. These are the same lines of code where we added `Math.max()`.

Replace the `enemyHealth` line with these two lines:

```
// generate random damage value based on player's attack power
var damage = randomNumber(playerAttack - 3, playerAttack);

enemyHealth = Math.max(0, enemyHealth - damage);
```

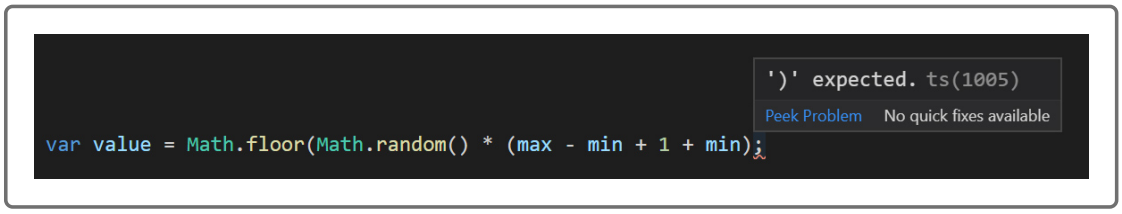
Do the same for `playerHealth` later in the `fight()` function:

```
var damage = randomNumber(enemyAttack - 3, enemyAttack);

playerHealth = Math.max(0, playerHealth - damage);
```

Save and test the game in the browser, making sure the enemy health and damage values are different each time. If anything broke along the way, remember to check the console for errors. Sometimes it's easy to misplace a parentheses. For instance, `Math.floor(Math.random() * (max - min + 1) + min);` (correct) vs. `Math.floor(Math.random() * (max - min + 1 + min);` (incorrect).

VS Code will likely warn you of these syntax errors as well, though the character(s) it underlines in red can be misleading:



In this case, the semicolon isn't the problem. VS Code is simply highlighting that something was supposed to come before the semicolon. If you point your cursor at the underlined character, you'll see the message `')' expected`, meaning a parentheses was forgotten somewhere.

Before we move on, we've learned a lot about functions throughout this project and may be a little confused about all the new jargon we've come across. Let's take a moment to refresh ourselves with this video walkthrough to help visualize what all of these words mean and what purposes they serve: