```
'use strict';
// Lecture 32: Activating Strict Mode
/* This mode is a special mode that we can activate in JS
which makes it easier for us to write a secure JS code.
All we have to do is to write the above string at the
beginning of the script.

This line has to be the very first line of code in the script,
if there is code before that line, then this line of code will be ignored. Comments are allowed though, but no code.

We can activate strict mode for more local purposes - like within a function or within a code block. Put strict mode at the beginning of your scripts, and write more SECURE CODE.

What is SECURE?
'strict mode' helps prevent developers from make accidental errors. This prevents developers from introducing bugs into their code. Thats because of two reasons:
    - strict mode will prevent you from doing certain things
    - strict mode will create errors for
    us, such that in certain situations (without sttrict), JS will simply fail silently without letting us know that we made a mistake.
*/
let hasDriversLicense = false;
const passTest = true;

if (passTest) hasDriverLicense = true; // wrong variable name!
// strict mode can help you avoid wrong variable name errors!

if (hasDriversLicense) console.log('I can drive :D');
// strict mode also includes a shortlist of variable names which are reserved for features which are going to be added to the language a little later.
const interface = 'Audio'; // ==> Unexpected strict mode reserved word
const private = 534; //==> unexpected strict mode reserved word
// JS is reserving this word for a new feature in the future.
// (In the future, there might be private fields within classes.)
/////////////////////////////////////////////////////////////
// Lecture 33: Functions
// Functions are just like a variable. - A variable can hold a value, but a
```

```
A variable can hold a value, but a function can hold one or more complete lines of code.
// Function Declaration
function logger() {
  console.log('My name is Jonas');
}
// here's the syntax:
function funcName(funcParameters){
    console.log("Function body");
    //Declare local variables
    const x = funcParameters + 5;
    return x; // return statement ( can be none!)
}
// For Example:
// Apples and Oranges are placeholders which will get substituted away with the actual function arguments.
function fruitProcessor(apples, oranges) {
    // juice is a local variable
  const juice = `Juice with ${apples} apples and ${oranges} oranges.`;
  return juice; // return by value?
}

// calling / running / invoking function
logger();
logger();
logger(); // this function does not return a value
// we also don't save its value to any
```

```javascript
variable here.

const appleJuice = fruitProcessor(5, 0); // 5 apples, 0 oranges
console.log(appleJuice);

const appleOrangeJuice = fruitProcessor(2, 4);
console.log(appleOrangeJuice);
// Functions allow us to write more maintainable code. With functions, we can create more REUSABLE chunks of code, instead of having to manually write the same code over and over again.
// DON'T REPEAT YOURSELF (DRY) principle: Functions are perfect for creating DRY code.

//Built in functions
// - Type conversion operators
const num = Number('23');
// - Console.log()
// This function returns undefined! (or void)
console.log(num);
////////////////////////////////////////////////////////////
// Lecture 34: Function Declarations vs. Expressions

// Function declaration
function calcAge1(birthYear) {
  return 2037 - birthYear; // no need to create local variables if your operations are simple.
}
// Function call
const age1 = calcAge1(1991);

// Function Expressions (NEW) (Anonymous Functions)
// Instead of writing a function with the callAge name, we can write function WITHOUT A NAME, and then we STORE this entire expression into a VARIABLE!
// This variable will THEN be a function!
// Remember everything on the RHS would give a value - hence it is an EXPRESSION.
// there are sometimes where we need to write functions like these - so we assign this whole value produced by the function to this variable calcAge2.
// In JS, functions are NOT A TYPE, they are just a value.
// If they are a value, you can store it in a variable.
const calcAge2 = function (birthYear) {
  return 2037 - birthYear;
}
// Calling the function
const age2 = calcAge2(1991);

console.log(age1, age2);
```

```
/* Whats the difference?
In fact, you can actually call functions in the code BEFORE they are defined!
But this is NOT possible with the function expression.
This is because of HOISTING, which will be covered somewhere in the future.

Some people use function expressions more, because they feel like it adds structure to their code? They may also like how things are stored in variables. However, which type of function you declare is of personal preference. For someone well-versed in a C & C++ background, you probably should try the anonymous functions since its something new.
*/
//////////////////////////////////////////////////////////////
```