Functions

IIKG1002/IDG1011 – Front-end web development

Johanna Johansen
johanna.johansen@ntnu.no

- Functions allow us to group together a series of statements that are required to perform a task
- The steps that the function needs to perform in order to do its task are packaged up in a **block of code**
- **Remember**
  - a block of code consists of one or more statements contained within curly braces
  - semicolon after statements; NO semicolon after code blocks

- To define a JavaScript function, we can use the *function* keyword, which can be used as a **declaration** or an **expression**
- ES6 (ECMAScript 2015) allows to define functions without the *function* keyword, through a more compact syntax → called **arrow functions**

Function declarations

- **Named functions**
- Creating a function
  - using the *function* keyword
  - we give it a **name (identifier**) followed by parentheses
  - we include the statements to perform a task in a code block, inside curly braces

Function declarations

- example of a **function declaration**:

```
let message = 'Welcome to the Front-end web development
course!';

function updateMessage() {

    let el = document.querySelector('#welcomeMessage');

    el.textContent = message;

}
```

Function declarations

- Name/value pair principle used in programming languages
  - the function name `updateMessage`
  - the value is the code block consisting of statements
- The name of the function becomes a variable whose value is the function itself
  - the name of the function is used as a variable

Calling a function

- or invoking a function
- when needing to perform the same task repeatedly
  - without having to rewrite the same statements again → we **call the function** by its name
  - a way to reuse the same code
- the statements that a function contains get **executed** ONLY when we call the function
- we can call the function as many times as we want within the same JavaScript file

Calling a function

- by using the function name, followed by parentheses

```
let message = 'Welcome to the Front-end development course!';
function updateMessage() {
     let el = document.getElementById('welcomeMessage');
     el.textContent = message;
}
updateMessage();
```

Parameters

- Some functions require some information in order to perform their tasks
- The required information is given to the function in the form of **parameters**
  - given in the parentheses after the function name
  - e.g., a function to calculate the area of a box will need to know its width and height

Parameters

```
function getArea(width, height) {
  return width * height;
}
```

- function that calculates and returns the area of a rectangle
- the parameters behave like local variables within the body of the function
- the code can perform the task without knowing the exact details in advance; we do not know yet what is the value of *width* and *height*

Arguments

- Calling function that need information
  - the values are specified in the parentheses that follow the name of the function
  - these values are called **arguments**

```
getArea(3, 5);
```

  - *3* and *5* are the values used for *width* and *height* when calculating the area of a certain box

Arguments

```
function getArea(width, height) {
    return width * height;
}
// Some code
getArea(3, 5);
// Some other code
```

```
function getArea(width, height) {
    return width * height;
}
// Some code
let wallWidth = 3;
let wallHeight = 5;
getArea(wallWidth, wallHeight);
// Some other code
```

- **Parameters** are used when declaring a function
  - act like variables inside the function
- **Arguments** are used when calling the function
  - specify the values that are going to be used to perform the calculations

Exercise

- Do the exercise *Function 1* from

  https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Test_your_skills:_Functions

  - **Note**: you can edit the code directly on the exercise's page
  - for the part requiring

    “... prints a random name from the provided array (names) to the provided paragraph (para)”,

    use this example as inspiration for what you are supposed to write

    ```
    const para = document.createElement('p');
    para.textContent = 'We hope you enjoyed the ride.';
    ```

  - Try also an implementation where the chooseName() function takes the name as parameter.

Return

```
function getArea(width, height) {
    return width * height;
}
// Some code
let wallWidth = 3;
let wallHeight = 5;
let wallLivingroom = getArea(wallWidth,
wallHeight);
// wallLivingroom variable holds the value
15 which was calculated by the getArea()
function
```

- some functions return information to the code that call them
  - returning the result of a calculation
  - getArea() function returns the area of a rectangle to the code that called it
    - computes a value
  - The interpreter leaves the function when return is used
    - any subsequent statements in this function will not be processed
  - if return does not have an associated expression it returns **undefined** to the caller

Return

```
function getArea(width, height) {
    return width * height;
}

let wallWidth = 3;
let wallHeight = 5;
let wallLivingroom =
getArea(wallWidth, wallHeight);
```

```
function getArea(width, height) {
    let area = width * height;
    return area;
}
…
```

Return

- returning **multiple values** with **arrays,** e.g., calculating both the area and the volume of a box

```
function getSize(width, height, depth) {
  let area = width * height;
  let volume = width * height * depth;
  let sizes = [area, volume];
  return sizes;
}
```

```
let areaOne = getSize(3, 2, 3)[0]; // getting the value of the array from the
index 0, which is area in this case
let volumeOne = getSize(3, 2, 3)[1]; // getting the value of the array from the
index 1, which is the volume in this case
```

```
function updateMessage() {
  let el =
  document.getElementById('welcomeMe
  ssage');
  el.textContent = message;
}
```

```
function getArea(width, height) {
  return width * height;
}
```

- `updateMessage()` function
  - adds some content to a HTML page
  - no return value is necessary
  - if a function does not contain a return statement, it simply executes each statement in the function body until it reaches the end
  - returns *undefined* to the caller

Function expressions

- they appear within the context of a larger expression or statement

```
//Note that we assign it to a variable
const area = function(width, height) { return width * height; };
let size = area(3,4);
```

  - **Good practice:** use `const` with function expressions so you do not accidentally overwrite your functions by assigning new values
- they are used for code that only needs to run once within a task, rather than repeatedly being called by other parts of the script
  - e.g., the value computed by a function expression can be given as an argument to another function

Function expressions

- the name is optional *;
- when a function does not have a name, they are called **anonymous functions**
- a function expression does not declare a variable, as the function declaration
  - we need to assign the defined function object to a constant or variable if you are going to need to refer to it multiple times
  - in order to invoke a function we must be able to refer to it; this is not possible until the function is assigned to a variable

* For the examples in this course, we will omit the name; it is allowed for functions, like factorial functions, that need to refer to themselves (not covered in this course)

Function expressions

- are sometimes defined and immediately invoked
- they are executed immediately the interpreter comes across them

```
let area = (function(width, height) { return
width * height; }(3,4));
```

`(3,4)` → used to call the function immediately

- The *area* variable holds the value returned from the function, rather then storing the function itself to be called later

Arrow functions

- a compact syntax for anonymous functions
- use an **`=>`** “arrow” to separate the function parameters from the function body
- the *function* keyword is not used
- are expressions instead of statements
  - they do not need a function name either

```
const area = (width, height) => { return width * height; };
```

Arrow functions

- even more compact
  - if the body of a function is a single return statement, you can omit the *return* keyword, the semicolon, an the curly braces

    ```
    const area = (width, height) => width * height
    ```

  - if the arrow function has only one parameter, we can omit the parentheses around the parameter list

    ```
    const square = x => x * x;
    ```

  - if no parameter, the parentheses are kept

    ```
    const square = () => 2 * 2;
    ```

Variable scope

- The **scope** of a variable is the location in your code in which the variable was defined
  - the location where you declare a variable will affect where it can be used within your code
- inside a block of code → **local variables**
- outside any code blocks → **global variables**

Local variables

- Variables and constants can only be used inside the *block of code* in which they are defined
- Blocks can be: functions, classes, bodies of if/else statements, while loops ...
- If a variable is defined within a set of curly braces, those curly braces delimit the region of code in which the variable or constant can be used
- The interpreter creates local variables when the code of block is run, and removes them as soon as the respective code of block finished its task

Global variables

- We have global variables when a variable or constant declaration appears at the top level (depth-wise), outside of any code blocks
- In traditional JavaScript the scope of a global variable is the HTML document in which it is defined
  - can be used anywhere within the script and other script files used by the respective HTML page
  - this can cause naming conflicts (variables defined with the same name)
- are stored in memory for as long as the web page is loaded into the web browser
  - therefore, they take up more memory than the local variables

```
function getArea(width, height) {
   // area is a local variable to the getArea function
   let area = width * height;
   return area;
}

// wallSize is a global variable
let wallSize = getArea(3, 2);
document.write(wallSize);
```

Exercise

- Read *Function scope and conflicts*

  https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#function_scope_and_conflicts
- Look at the code
- click on the link “Running live on GitHub”

  https://mdn.github.io/learning-area/javascript/building-blocks/functions/conflict.html
- open the Developer tools (F12) for the above page and look at the error
  - click on the Console to see the error message