Lesson 03.01

functions

parameters and arguments

return values

function-scope vs global scope

default parameter values

A function is a block of code that runs only when it is invoked (called). The function can be called in the code or by an event on the web page, such as a button click. A function typically (but certainly not always) has a name to call it by.

Mozilla mdn web docs defines functions in this way:

> Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

A function *should take some input and return an output*, so it is useful to think of a function as an input-output machine. Think of real world analogies:

- coffee grinder as real world function (input-output machine):
  - input: you put coffee beans in
  - procedure: blades turn inside the machine
  - output: out comes ground coffee

All that said, it is possible to have a function without inputs (parameters). This is where we will begin:

how to define (declare) a function

Follow these steps:

1. Begin by writing the keyword: `function`
2. Give the function a name. Variable naming rules apply. Since a function *does* something, it is customary for the name to begin with a verb: `function sayHello`
3. Next put a pair of parentheses: `function sayHello()`

The parentheses are for passing data -- *inputs* -- to the function, but some functions don't need inputs, and so the parentheses are left empty.

4. Next add a pair of curly braces: `function sayHello() {}`

5. Inside the curly braces, put the code that will run when the function is called:

```
    function sayHello() {
        console.log("Hello!");
    }
```

6. Run this example and check the Console. Nothing happens. Where's "Hello"?

Remember, a function must be *called*, but all we have done is *define* it.

7. We do not have a button to click to call the function, so's call it directly in the code:

```
    sayHello();
```

8. Run the code in the Console again; this time you get the "Hello!".

variable scope

global variables are available everywhere in a script: - **var** declared outside of a function is global. - **let** declared outside of a code block is global. - **const** (constants) follow the same scoping rule as let.

function scoped (local) variables - **var**, **let** declared inside a function is local to that function. - **let** declared inside a code block is available only to that code block.

// 9. Declare a local variables inside the function:

```
    function welcomePlayer() {
        var username = "Pikachubaca";
        let highScore = '6234';
        console.log(`Welcome back, ${username}! Your high score is
${highScore}`);
    }

    welcomePlayer();
```

10. Try to access the function variables in the global scope, outside the function:

```
    // console.log('username', username); // Error: username is not
defined
    // console.log('highScore', highScore); // Error: username is not
defined
```

parameters-arguments

Parameters (params for short) are inputs of a function that go in the parentheses. Inside the function, parameters are local variables. When a function is called, the params are assigned values as arguments passed into the function call parentheses.

11. Write a function with two parameters:

```
    function greetPlayer(username, highScore) {
        console.log(`Welcome, ${username}! Your high score is
${highScore}!`);
    }
```

12. Call the function twice with different arguments each time:

```
    greetPlayer("Brian88", 12345);
    // Welcome, Brian88! Your high score is 12345!

    greetPlayer("Bob", 78967);
    // Welcome, Bob! Your high score is 78967!
```

If a function expects an argument, but none is provided, the missing value will be 'undefined'.

13. Call the function again, but this time omit the second argument:

```
    greetPlayer('Sally123');
    // Welcome, Sally123 undefined
```

default parameter values

A parameter can be assigned a value when the function is declared. That way, if no argument is supplied for it, it uses the default.

14. Write a new function, greetPlayerScore(). It is the same as greetPlayer, but with a default high score of 1000:

```
    function greetPlayerScore(username, highScore=1000) {
        console.log(`Welcome, ${username}! Your high score is
${highScore}`);
    }
```

15. Call the function twice, but the second time, only pass in the username. The high score will default to 1000:

```
greetPlayerScore("X12", 13240);
// Welcome, X12! Your high score is 13240!
greetPlayerScore("yMe");
// Welcome, yMe! Your high score is undefined!
```

Global variables are available everywhere, including to all functions:

16. Declare some global vars and use them in a function;

```
let num1 = 10;
let num2 = 14;
let sum = 0;

function addNums() {
    sum = num1 + num2;
    console.log("sum", sum);
}

addNums(); // sum 24
```

A more self-contained approach would be to num1 and num2 to the funcition as parameters (unless num1 and num2 are needed elsewhere in the script).

refactoring Refactoring code means changing it for the better.

17. Refactor addNums() as addNumbers() and give it parameters. The values of num1 and num2 will be passed in as the arguments when the function is called:

```
function addNumbers(num1, num2) {
    let sum = num1 + num2;
    console.log(sum, sum);
}

addNumbers(12, 18); // sum 30
addNumbers(123, 184); // sum 307
```

return value

So far, our output has consisted only of console.log. This is useful for tesing and debugging, but once the output is logged, that data is gone. The answer is for the output to be a "return value" which we can save in the script.

return value

To return a value from a function, put the keyword 'return' in front of a value.

Save the return value by setting the function call equal to a variable.

A return not only returns a value, it exits the function.

18. Define a new function, that is similar to the previous one, but with one big exception: this one has a return value:

```
let numeroUno = 7;
let numeroDos = 8;

function multiplyNumbers(n1, n2) {
    return n1 * n2;
}
```

19. Call the function, setting the function call itself equal to a variable to save the return value:

```
let prod1 = addNums3(12, 18);
console.log('prod1', prod1);

let prod2 = addNums3(3412, 5618);
console.log('prod2', prod2); // 9030
```

20. Refactor the greet function to return a value. Use a new name: getGreeting.

- "get" in function names means it returns a value
- replace the `console.log` with `return`
- set the function calls equal to variables
- log the variables to output the return values

```
function getGreeting(username, highScore) {
    return `Welcome, ${username}! Your high score is ${highScore}`;
}

let welcome1 = getGreeting("Viper", 73489);
console.log('welcome1:', welcome1);
// Welcome, Viper! Your high score is 73489!

let welcome2 = getGreeting("Ktrav26", 12345);
console.log('welcome2:', welcome2);
// Welcome, Ktrav26! Your high score is 12345!
```

21. Declare a function **calcRestaurantBill()**.

- Give it three parameters: **subTotal**, **tipPct** and **taxRate**, the latter two with default values: **tipPct=15** and **taxRate=4**. This assigns a default 15% tip and default tax rate of 4% if no value(s) are provided for these.

```
function calcRestaurantBill(subTotal, tipPct=15, taxRate=4) {
```

```
}
```

The function calculates the tax and tip and adds those amounts to the subtotal

- calculate tip:
  - **let tip = subTotal * tipPct / 100;**
- calculate tax:
  - **let tax = subTotal * taxRate / 100;**
- add tax and tip to subtotal to get total:
  - **let total = subTotal + tax + tip;**

22. Write the calculations. The tax and tip percents are divided by 100, because 15% is not 15, but rather 0.15:

function calcRestaurantBill(subTotal, tipPct=15, taxRate=4) { let tax = subTotal * taxRate / 100; let tip = subTotal * tipPct / 100; let total = subTotal + tax + tip; }

23. Concatenate the bill and return it:

```
function calcRestaurantBill(subTotal, tipPct=15, taxRate=4) {
    let tax = subTotal * taxRate / 100;
    let tip = subTotal * tipPct / 100;
    let total = subTotal + tax + tip;
    let bill = `Sub-total:  $ ${subTotal}
    Tax Rate:   ${taxRate} %
    Tip Pct:    ${tipPct} %
    Tip:        ${tip}%
    Tax:        $ ${tax}
    TOTAL:      $ ${total}
    Thank you!`;
    return bill;
}
```

24. Add toFixed(2) to round the cents to 2 decimal places:

```
Tip:        ${tip.toFixed(2)} %
Tax:        $ ${tax.toFixed(2)}
TOTAL:      $ ${total.toFixed(2)}
```

25. Call the function, and supply all three parameters. Make the tip 20 percent and the tax 6 percent. This overrides the default tipPct and taxRate.

```
// $80 subtotal, 20% tip, 6% tax
let bill1 = calcRestaurantBill(80, 20, 6);
console.log(bill1);
```

26. Call the function again, but this time omit the third argument (tax). The tax rate will therefore default to 4%:

```
// $90 subtotal, 18% tip, 4% tax
let bill2 = calcRestaurantBill(90, 18);
console.log(bill2);
```

27. Call the function, passing in only one parameter, understood to be subTotal. tipPct and taxRate will use the defaults:

```
// $65 subtotal,  15% default tip,  4% default tax
let bill3 = calcRestaurantBill(65);
console.log(bill3);
```

End Lesson 03.01

Next: Lab 03.01