Day 7 - Functions

A function is a reusable block of code/programming statements meant to perform a certain tasks. They're declared by the *function* keyword, followed by a name, followed by parentheses (), which can take a parameter. A parameter taken by a function is called an argument. A function can also take a default parameter.

Functions make code:

- clean and easy to read
- reusable
- easy to test

A function can be declared or created in a few ways:

- Declaration function
- Expression function
- Anonymous function
- Arrow function

Function Declaration

```
function functionName() {

}
functionName()
```

Function without a parameter and return

Function can be declared without a parameter

```
function addTwoNumbers() {
  let numOne = 10
  let numTwo = 20
  let sum = numOne + numTwo
```

```
  console.log(sum)
}

addTwoNumbers()
```

Function returning value

Functions can also return values. If a function doesn’t return values the value of the function is undefined.

```
function addTwoNumbers() {
  let numOne = 2
  let numTwo = 3
  let total = numOne + numTwo
  return total
}

console.log(addTwoNumers())
```

Function with a parameter

We can pass different data types (number, string, boolean, object, function) as a parameter.

```
function functionName(parm1) {
  // code goes here
}
functionName(parm1) // during calling or invoking one argument needed
```

Function with two parameters

```
function functionName(parm1, parm2) {
  // code goes here
}
```

Function with many parameters

```
// function with multiple parameters
function functionName(parm1, parm2, parm3,...){
  // code goes here
}

// this function takes array as a parameter and sum up the numbers in the array
function sumArrayValues(arr) {
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum = sum + arr[i];
  }
  return sum;
}
const numbers = [1, 2, 3, 4, 5];
```

Function with unlimited number of parameters

The way we create functions with an unlimited number of arguments different between a function declaration and arrow function.

Unlimited number of parameters in regular function

A function declaration provides a function scoped arguments array like object. Anything we passed as argument in the function can be accessed from arguments object inside the functions.

```
// Let us access the arguments object

function sumAllNums() {
  console.log(arguments)
}

sumAllNums(1,2,3,4)

// function declaration

function sumAllNums() {
  let sum = 0
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i]
  }
  return sum
}
```

Unlimited number of parameters in arrow function

Arrow function doesn't have the function scoped arguments object. We use the spread operator followed by any parameter name to implement a function which takes unlimited number of arguments. Anything we pass as an argument in the function can be accessed as array in the arrow function

```
// Let us access the arguments object

const sumAllNums = (...args) => {
  console.log(args)
}

sumAllNums(1,2,3,4)
// [1, 2, 3, 4]

// function declaration

const sumAllNums = (...args) => {
  let sum = 0
  for(const element of args) {
    sum += element
  }
  return sum
}

console.log(sumAllNums(1, 2, 3, 4)) // 10
```

Anonymous Function

Anonymous function or without name

```
const anonymousFun = function() {
  console.log('This is an anonymous function, stored in anonymousFun')
}
```

Expression Function

These are anonymous functions. We create a function without a name and assign it to a variable. To return a value from the function we should call the variable.

```
// Function expression
const square = function(n) {
```

```
  return n * n
}

console.log(square(2)) // -> 4
```

Self Invoking Functions

These are anonymous functions that don’t need to be called to return a value.

```
(function(n) {
  console.log(n * n)
}(2) // 4

let squaredNum = (function(n) {
  return n * n
})(10)

console.log(squaredNum)
```

Arrow Function

This is an alternative to writing a function, however function declaration and arrow function have some minor differences.

Arrow function uses arrow instead of the keyword *function* to declare a function.

```
// Function declaration
function square(n) {
  return n * n
}

// Arrow function
const square = n => {
  return n * n
}

// Arrow function on one line
const square = n => n * n

const printFullName = (firstName, lastName) => {
  return `${firstName} ${lastName}`
}

console.log(printFullName('Chris', 'Guerrero'))
```

```
// Above function has only the return statement, so we can explicitly return it as follows
const printFullName = (firstName, lastName) => `${firstName} ${lastName}`
console.log(printFullName('Asabeneh', 'Yetayeh'))
```

Function with default parameters

Sometimes we pass default values to parameters. When we invoke the function and don’t pass an argument, the default value will be used. Both function declaration and arrow functions can have a default value or values.

```
function functionName(param = value) {
  // code
}

functionName()
functionName(arg)

// Example:
function greetings(name = 'Peter') {
  let message = `${name}, welcome to 30 Days Of JavaScript!`
  return message
}

console.log(greetings())

// Example 2:
function calculateAge(birthYear, currentYear = 2019) {
  let age = currentYear - birthYear
  return age
}

console.log('Age: ', calculateAge(1819))

// Default parameters with arrow functions
const functionName = (param = value) => {
  // code
}

// Example:
const greetings = (name = 'Peter') => {
  let message = name + ', welcome to 30 Days Of JavaScript!'
  return message
}

console.log(greetings())
console.log(greetings('Chris'))
```