DEV ACADEMY
TE KURA HANGARAU
O AOTEAROA

Functions

Week 1, Day 2, Part B

Agenda

- Functions as First Class Citizens
- Different ways to write functions:
  - Named functions
  - Anonymous functions
  - Arrow functions
- Using functions as arguments:
  - Array methods (`` `.forEach()` ``, `` `.find()` ``, `` `.filter()` ``, `` `.map()` ``, `` `.reduce()` ``)

First Class Citizens

In JavaScript, functions are considered "first class citizens". This means that functions can be:

- Assigned to variables
- Passed as arguments to other functions
- Returned from functions

Assigned to variables

Demo

```
function helloWorld() {
  return 'Hello World'
}

// Assign to variable
const greeting = helloWorld

console.log(greeting()) // 'Hello World'
```

Passed as arguments to other functions

Demo

```javascript
function helloWorld() {
  return 'Hello World'
}

function sayHello(fn) {
  const result = fn()
  console.log(result)
}

// Pass as argument
sayHello(helloWorld) // 'Hello World'
```

Returned from functions

Demo

```
function helloWorld() {
  return 'Hello World'
}

function getGreeting() {
  // Return from function
  return helloWorld
}

const greeting = getGreeting()

console.log(greeting()) // 'Hello World'
```

Different ways to write functions

Demo

Named:

```js
function add(a, b) {
  return a + b
}
```

- Name is `add`
- Uses the return keyword (explicit return)
- Can be called by name `add(1, 2)`

Anonymous:

```js
function (a, b) {
  return a + b
}
```

```js
const add = function (a, b) {
  return a + b
}
```

- Has no name
- Uses the return keyword (explicit return)
- Has no name to directly call it...
- ...but as a *first class citizen* it...
  - can be assigned to a variable
  - can be passed as an argument to another function
  - can be returned from another function

Different ways to write functions (cont.) Demo

Arrow functions: `() => {}`

Arrow functions are anonymous by default, but since they are first class citizens, they can be assigned to a variable.

Explicit return:

```
const add = (a, b) => {
  return a + b
}
```

- Uses the return keyword (explicit return)
- Can be called by name `add(1, 2)`

Implicit return:

```
const add = (a, b) => a + b
```

- Does not use the return keyword (implicit return)
- Can be called by name `add(1, 2)`

Omits the **return keyword** and **curly braces** `{}`

Array methods

… `.forEach()`, `.find()`, `.filter()`, `.map()`, `.reduce()`

Since functions can be **passed as arguments to other functions**

We often call array methods with a function as an argument.

```
const isOdd = (number) => {
  return number % 2 !== 0
}

[1, 2, 3].find(isOdd)
// or
[1, 2, 3].find((number) => number % 2 !== 0)
```

This example uses the `.find()` method to find the first odd number in an array.

Demo

Array methods

Summary

`.forEach()`:

```
['A', 'B', 'C'].forEach((letter, index) => {
  console.log(`Index ${index}: ${letter}`)
  // Index 0: A, Index 1: B, Index 2: C
})
```

- Executes a function for each element in the array, **returns** `undefined`
- **Signature**: `(element, index, array) => {}`

`.find()`

```
[1, 2, 3, 4].find((number) => number % 2 === 0) // 2
```

- **Returns**: the first element in the array that satisfies the condition
- **Signature**: `(element, index, array) => {}`

`.filter()`

```
[1, 2, 3, 4].find((number) => number % 2 === 0) // [2, 4]
```

- **Returns**: a new array with elements that satisfy the condition
- **Signature**: `(element, index, array) => {}`

`.map()`

```
[1, 2, 3, 4].map((number) => number * 2) // [2, 4, 6, 8]
```

- Transforms each element in the array, **returns** a new array with transformed elements
- **Signature**: `(element, index, array) => {}`

Array methods (cont.) `Summary`

`` `.reduce()` ``:

```
// Usage:
[].reduce((currentValue, element) => newValue, initialValue)


const sum = [1, 2, 3, 4].reduce((sum, number) => {
  return sum + number
}, 0)

sum // 10
```

- Reduce
  - Executes a function for each `` `element` `` in the array,
  - passing the **return value** from the *previous iteration* as the `` `accumulator` `` in the *next iteration*
- **Returns**: the final value of the accumulator
- **Signature**: `` `(accumulator, element, index, array) => {}` ``

Read more about reduce

Other useful Array methods

- ``.some()`` MDN
- ``.every()`` MDN
- ``.includes()`` MDN
- ``.sort()`` MDN
- ``.reverse()`` MDN
- ``.slice()`` MDN