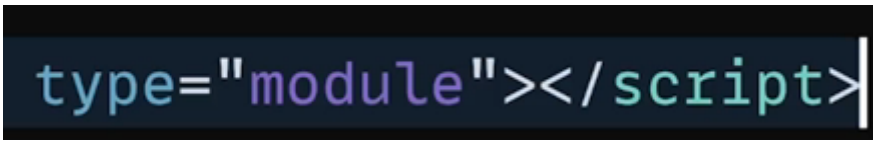
1. Importowanie

Ważne jest przy importowaniu, ze musi być



a)

```
export let apiKey = "adnasdoasflak1";
```

```
import { apiKey } from "./util.js";
```

(można napisać as)

b)

```
export default "adnasdoasflak1";
```

```
import apiKey from "./util.js";
```

(dowolną nazwę wówczas można dać)

c)

Można importować wszystkie let jako obiekt

```
import * as util from "./util.js";
```

2. Zmienne i kontenery (Variables and Values)

a)

```
let userMessage = "Hello World!";
```

Let na początku

b)

Const jak nie zmieniasz

c) === - zwraca wartość Boolean (do porównywania)

3. Funkcje

a)

Można nadać domyślną wartość, mogą zwracać wartości

b) Arrow functions (lambdy)

c)

d) **More on the Arrow Function Syntax**

1) Omitting parameter list parentheses

Instead of

```
1. (userName) => { ... }
```

you could write

```
1. userName => { ... }
```

2) Omitting function body curly braces

Instead of

```
number => { return number * 3;}
```

you could write

```
number => number * 3;
1. number => if (number === 2) { return 5 }; // invalid because if statements
                                               can't be returned
```

3) Special case: Just returning an object

To *"tell"* JavaScript that an object should be created (and returned) instead, the code would need to be adjusted like this:

```
1. number => ({ age: number }); // wrapping the object in extra parentheses
```

By wrapping the object and its curly braces with an **extra pair of parentheses**, JavaScript understands that the curly braces are not there to define a function body but instead to create an object. Hence that object then gets returned.

4. Obiekty

a)

```
const user = {
  name: "Max",
  age: 34,
  greet() {
    console.log('Hello!');
  }
};
```

b)

```
class User {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log('Hi!');
  }
}

const user1 = new User("Manuel", 35);
```

5. Tablice

a)

```
const hobbies = ["Sports", "Cooking", "Reading"];
console.log(hobbies[0]);
```

Tablice mogą zawierać co chcesz (tzn. inne tablice, itd...)

b)

```
const index = hobbies.findIndex((item) => {
  return item === "Sports";
});
```

```
const index = hobbies.findIndex((item) => item === "Sports");
```

c)

```
hobbies.map((item) => item + "!");
```

Zwraca nową tablicę, nie modyfikuje starej

d) Mapowanie na obiekty

```
const editedHobbies = hobbies.map((item) => ({text: item}));
```

6) Destrukturyzacja

a) dla tablic

```
const [firstName, lastName] = ["Max", "Schwarzmüller"];
```

Przypisujemy nazwy zmiennych

b) dla obiektu

```
const { name: userName, age } = {
  name: "Max",
  age: 34
};
```

Można też aliasami

c) destrukturyzacja funkcji przyjmującej obiekt

Przed:

```
function storeOrder(order) {
  localStorage.setItem('id', order.id);
  localStorage.setItem('currency', order.currency);
}
```

Po:

```
function storeOrder({id, currency}) { // destructuring
  localStorage.setItem('id', id);
  localStorage.setItem('currency', currency);
}
```

Funkcja wciąż przyjmuje jeden parametr (obiekt)

7) Rozprzestrzenianie (spread) [...]

a) dodawanie tablic do siebie

```
const newHobbies = ["Reading"];

const mergedHobbies = [...hobbies, ...newHobbies]
```

b) dodawanie obiektów

```
const extendedUser = {
  isAdmin: true,
  ...user
}
```

Mamy teraz też pola z user

8) Wyrażenia warunkowe, kontroli

a) w if porównujemy przy pomocy ===

b)

```
for (const hobby of hobbies) {
  console.log(hobby);
}
```

9) Dostęp do DOM – Document Object Model

```
const list = document.querySelector("ul");
list.remove();
```

10) Funkcje jako wartości

a)

```
function handleTimeout() {
  console.log("Timed out!");
}

const handleTimeout2 = () => {
  console.log("Timed out ... again!");
};

setTimeout(handleTimeout, 2000);
```

11) Definiowanie funkcji wewnątrz innych

```
function init() {
  function greet() {
    console.log("Hi!");
  }

  greet();
}

init();
```

12) Referencja i wartości prymitywne

a) obiekty i tablice są przechowywane przez referencję. Dlatego jak dasz const to możesz dować do niej