Functions

tsevdos.me / @tsevdos

Rules

Feel free to interrupt me for:

- questions
- relevant comments

Agenda

- function declarations
- function expressions
- arrow function (expressions)
- function invocation (function calling)
- default parameters
- function scope

Functions

- code reuse
- they either return a value,
- produce a side effect,
- or both

Function declarations

- aka. function statement
- name of the function
- a list of parameters (enclosed in parentheses and separated by commas ",")
- function body (enclosed in curly braces {})

```
function sayHi(name, age) {
  return `Hi, my name is ${name} and I'm ${age} years old.`;
}
```

Function invocation

- aka. function calling
- name of the function
- pass the parameters

```
sayHi("John", 43);
sayHi("Mary", 23);
```

Default parameters

```
function greet(greeting = "Hello", name = "stranger") {
  return `${greeting}, ${name}`;
}

greet();
greet("Hi", "John");
```

Function expressions

- can be anonymous and named
- the function is stored in a variable

```
// anonymous
const square = function (number) {
  return number * number;
};

console.log(square(4));
```

Function expressions

```
// named
const square = function square(number) {
  return number * number;
};

console.log(square(4));
```

Arrow function

- aka. arrow function expression
- always anonymous
- don’t have bindings to arguments, `this`, or `super`, and cannot be used as methods
- cannot be used as constructors (calling them with `new` throws a `TypeError`)
- implicit return

```
const add = (number1, number2) => number1 + number2;

console.log(add(2, 3));
```

Function scope

Variables defined inside a function cannot be accessed from outside the function because they are defined only in the function's scope. However, a function can access all variables and functions defined in the scope where it is declared. (MDN)

Workshop

- demo

Happy coding!