Web Application Development for Industrial Engineers

การพัฒนาเวปแอปพลิเคชันสำหรับวิศวกรอุตสาหการ

Function

Functions allow you to:

- Store a piece of code that does a single task inside a defined *block*.
- Call that code whenever you need it using a single short command.

You have already seen functions.

The `replace()` string function

- Takes a source string and a target string
- Replaces the source string with the target string
- Returns the newly formed string

```
const myText = 'I am a string';
const newString = myText.replace('string', 'sausage');
console.log(newString);
```

Invoking functions

- Including the name of the function in the code somewhere, followed by parentheses.
- This form of creating a function is also known as function **declaration**.
  - It is always **hoisted**, so you can call function above function definition and it will work fine.

```
function myFunction() {
  alert('hello');
}

myFunction();
// calls the function once
```

Function parameters

- Values that need to be included inside the function parentheses.
- Make functions more useful.

```
function myFunction(text) {
  alert(text);
}

myFunction('hello');
```

Default parameters

- If you're writing a function and want to support optional parameters, you can specify default values by adding `=` after the name of the parameter, followed by the default value:

```
function hello(name = 'Chris') {
  console.log(`Hello ${name}!`);
}

hello('Ari'); // Hello Ari!
hello(); // Hello Chris!
```

Return values

- Function can return values

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

const result = add(1, 2); // 3
```

Anonymous functions

```
function() {
  alert('hello');
}
```

- Has no name.
- Used when a function expects to receive another function as a parameter.

Anonymous function example (1)

```
function operate(a, b, ops) {
  return ops(a, b);
}

// Call the function with anonymous function as an argument.
const result = operate(1, 2, function (a, b) {
  return a + b;
});

console.log(result); // 3
```

Anonymous function example (2)

https://codepen.io/nnnpooh/pen/yLzXjjN

Arrow functions

- If you pass an anonymous function like this, there's an alternative form you can use, called an *arrow function*.
- Instead of `function(event)`, you write `(event) =>`

Arrow functions example (1)

```
function operate(a, b, ops) {
  return ops(a, b);
}

const result = operate(1, 2, (a, b) => {
  return a + b;
});

console.log(result); //3
```

Concise form of arrow functions

If the function only has one line in the curly brackets, you omit the curly brackets and `return`:

```
function operate(a, b, ops) {
  return ops(a, b);
}

const result = operate(1, 2, (a, b) => a + b);

console.log(result);
```

Be careful

```
// Correct
const result = operate(1, 2, (a, b) => a + b);

// Incorrect
const resultWrong = operate(1, 2, (a, b) => {
  a + b;
});

console.log(result); // 3
console.log(resultWrong); //Undefined
```

Arrow functions example (2)

https://codepen.io/nnnpooh/pen/yLzXjjN

Scope

- When you create a function, the variables and other things defined inside the function are inside their *own separate scope*.
- This means that they are locked away in their own separate compartments, unreachable from code outside the functions.
- The top level outside all your functions is called the **global scope**.
    - Values defined in the global scope are accessible from everywhere in the code.

Scope example

- Local variable is not accessible to the outside.

```
function myFunction() {
  let carName = 'Volvo';
}

console.log(carName); // Error
```

Scope example

- Global variable in accessible everywhere.

```
let carName = 'Volvo';

function myFunction() {
  console.log(carName);
}

myFunction(); // 'Volvo'
```

Scope example

- You can declarethe same variable name in a separate scope.

```
let carName = 'Volvo';

function myFunction() {
  let carName = 'Honda';
  console.log(carName); // Honda
}

console.log(carName); // Volvo
```

Conditions

- `if` statement

```
let choice = 'A';

if (choice === 'A') {
  console.log('You chose A.');
}
```

- One-line

```
if (choice === 'A') console.log('You chose A.');
```

`if` - `else`

```
let choice = 'A'; // 'A', 'B', 'C'

if (choice === 'A') {
  console.log('You chose A.');
} else {
  console.log('You did not choose A');
}
```

`if` - `else if` - `else`

```
let choice = 'A'; // 'A', 'B', 'C'

if (choice === 'A') {
  console.log('You chose A.');
} else if (choice === 'B') {
  console.log('You chose B.');
} else if (choice === 'C') {
  console.log('You chose C.');
} else {
  console.log('You did not choose A, B, or C.');
}
```

Truthy values

- A truthy value is a value that is considered `true` when encountered in a Boolean context.

```
if (true)
if ({})
if ([])
if (42)
if ("0")
if ("false")
if (new Date())
if (-42)
if (12n)
if (3.14)
if (-3.14)
if (Infinity)
if (-Infinity)
```

Falsy value

- A falsy value is considered `false` in a Boolean context.

```
if (false)
if (null)
if (undefined)
if (0)
if (-0)
if (0n)
if (NaN)
if ("")
```

Check for `null` or `undefined`

```
function absolute(number) {
  if (!number) return -1; // Error code
  return Math.abs(number);
}

console.log(absolute(-2)); // 2
console.log(absolute(null)); // -1
console.log(absolute(undefined)); // -1
```

`switch` - `case`

- Can be used instead of multiple `else if`

```
let choice = 'A';
let score;
switch (choice) {
  case 'A':
    score = 10;
    break;
  case 'B':
    score = 5;
    break;
  case 'C':
    score = 1;
    break;
  default:
    score = 0;
    break;
}
```

`break`

- *What happens if I forgot a break?*
- If you forget a break then the script will run from the case where the criterion is met and will run the cases after that regardless if a criterion was met.

You can use `object`.

```
let choice = 'A';
let mapping = {
  A: 10,
  B: 5,
  C: 1,
};

result = mapping[choice];
console.log(result);
```

- However, this does not handle the *default* case.

Conditional (ternary) operator

- Executing expressions

```
let loading = true;
loading ? console.log('Loading...') : console.log('Done!');
```

- Return values

```
var age = 26;
var beverage = age >= 21 ? 'Beer' : 'Juice';
console.log(beverage); // "Beer"
```

Back to the *choice* example

- This handles all cases - cool!.

```
let choice = 'A';
let mapping = {
  A: 10,
  B: 5,
  C: 1,
};

result = mapping[choice] ? mapping[choice] : 0;
console.log(result);
```

Form

- https://ie-software-dev.netlify.app/Codes/T09_js/t01_form/