

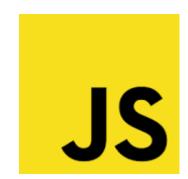
# 웹 시스템 설계

Web System Design

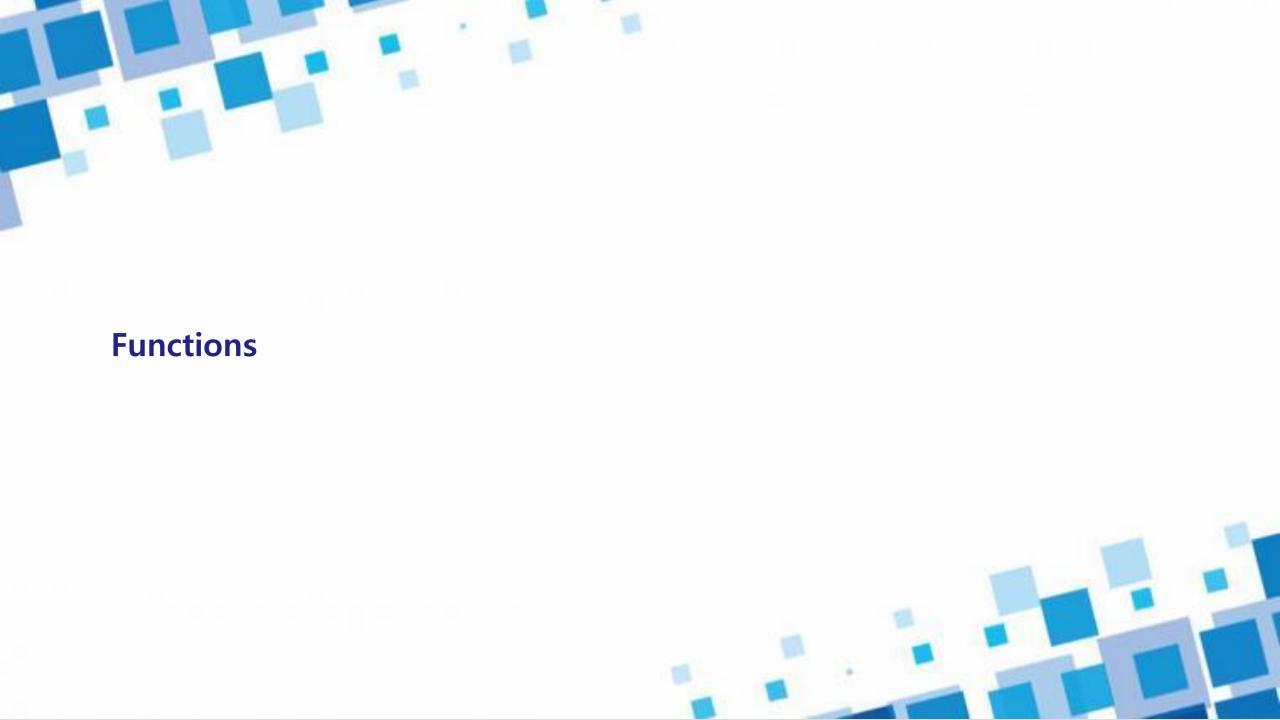
10. JavaScript: Functions

#### References

- Learning JavaScript by Ethan Brown
- Mozilla Developer Network: JavaScript tutorials (https://developer.mozilla.org/ko/docs/Learn/JavaScript)







### **Functions**



- ❖ A function is a self-contained collection of statements that run as a single unit
- ❖ A function declaration
  - Define a body of function, which is the collection of statements





### **Function Declaration**

```
// Compute the distance between Cartesian points (x1,y1) and (x2,y2).
function distance(x1, y1, x2, y2) {
       let dx = x2 - x1;
       let dy = y2 - y1;
       return Math.sqrt(dx*dx + dy*dy);
// A recursive function (one that calls itself) that computes factorials
// Recall that x! is the product of x and all positive integers less than it.
function factorial(x) {
       if (x \le 1) return 1;
       return x * factorial(x-1);
```



### Invoking (calling, executing, running) Functions

- Function invocation
  - let total = distance(0,0,2,1) + distance(2,1,3,5);
- Method invocation
  - Method: function stored in a object
  - If you have a function f and an object o, you can define a method named m of o with the following line:
    - o.m = f;
    - Invoking: o.m();

```
const o = {
  name: 'Wallace',
  bark: function() {return 'Woof'; }
}
```

```
// we can do this in ES6
const o = {
  name: 'Wallace',
  bark() {return 'Woof'; }
}
```

### **Return Values**



- The return keyword will immediately terminate the function and return the specified value, which is what the function call will resolve to.
  - Calling a function is an expression and expressions resolve to a value
- Functions that are going to return a value must use the return statement.
  - Otherwise, the return value will be undefined

### **Example**

```
function prod(a,b)
{ x=a*b;
 return x; }
```

```
product=prod(2,3);
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.



# Dynamic Sjou Tomorrow

# Calling vs. Referencing

- In JavaScript, functions are objects (first class)
  - can be passed around and assigned just like any other object.
- calling a function
  - follow a function identifier with parentheses
  - JavaScript knows that you're calling it: it executes the body of the function, and the expression resolves to the return value.
- \* referencing a function
  - don't provide the parentheses
  - you're simply referring to the function just like any other value

```
getGreeting();  // "Hello, World!"
getGreeting;  // function getGreeting()
```

```
function getGreeting() {
    return "Hello world!";
}
```



# **Function Arguments**



- The primary mechanism to pass information to a function call
  - Also called parameters
  - Arguments are like variables that don't exist until the function is called.
- Formal arguments
  - Arguments in a function declaration (i.e., a and b)

```
function prod(a,b)
{ x=a*b;
 return x; }
```

- Actual arguments
  - When a function is called, formal arguments receive values and become actual arguments (i.e., the values 2 and 3)

```
product=prod(2,3);
```

The arguments exist only in the function



매개변수 해체(Destructing Arguments)와 해체 할당 (Destructed Assignment)는 이후 Advanced Topics 주제로 다루겠습니다.

# this keyword



- ❖ Inside a function body, a special read-only value called this is available
  - The this keyword relates to functions that are properties of objects.
  - When methods are called, the this keyword takes on the value of the specific object it was called on:

```
const o = {
  name: 'Wallace',
  speak() {return `My name is ${this.name}!`; }
```

```
o.speak(); // My name is Wallace!
```





### **Function Expressions and Anonymous Functions**

- ❖ Function expressions are syntactically identical to function declarations except that you can omit the function name.
- ❖ How are we to call it?
  - understanding *function expressions* (something that evaluates to a value) & functions are values like any other in JavaScript.
  - A function expression is simply a way to declare a (possibly unnamed) function. A function expression can be assigned to something (thereby giving it an identifier), or called immediately.
- The example
  - use a function expression and assign the result to a variable

```
let distance = function (x1, y1, x2, y2) {
    let dx = x2 - x1;
    let dy = y2 - y1;
    return Math.sqrt(dx*dx + dy*dy);
};
let factorial = function (x) {
    if (x <= 1) return 1;
    return x * factorial(x-1);
};</pre>
```



대학

function distance(x1, y1, x2, y2)

- When you're defining a named function that you intend to call later → use a function declaration
- If you need to create a function to assign to something or pass into another function → use a function expression.



### **Arrow function (fat arrow notation =>)**

- Arrow functions are always anonymous
- Simplify function in syntax in three ways:
  - 1. You can omit the word function.
  - 2. If the function takes a single argument, you can omit the parentheses.
  - 3. If the function body is a single expression, you can omit curly braces and the return statement.

```
// expression at the right side
let sum = (a, b) => a + b;  // 1, 3
// or multi-line syntax with { ... },
// need return here:
let sum = (a, b) => {  // 1
    // ...
    return a + b;
}
// without arguments
let sayHi = () => alert("Hello");// 1, 3
// with a single argument
let double = n => n * 2;  // 1, 2, 3
```