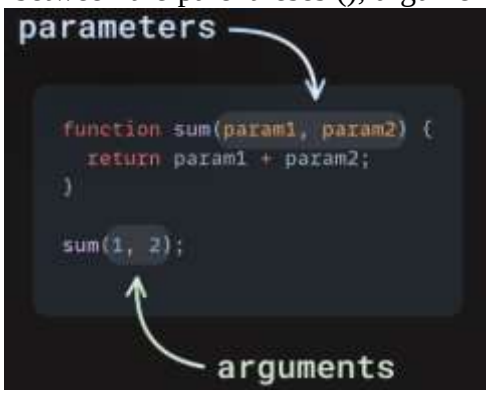
FUNCTION BASICS

Functions are reusable pieces of code which perform specific tasks. parameters are the items listed between the parentheses (), arguments are the actual values we decide to pass to the function.



Note: Functions that are part of objects are called **methods**.

A. Parameters

- Without parameters, example: `Math.random();`
- Needs parameters, example: `myText.replace("string", "sausage");`
- Optional parameters, ex: `myArray.join(" ");` & `myArray.join();`
- 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!
```

Alternative(in old JS code):

```
function showMessage(from, text) {
  if (text === undefined) {
    text = 'no text given';
  }
```

B. Anonymous Functions and Arrow Functions

1. Anonymous functions is a function that has no name. AF usually become “function as arguments” in other function(i.e at addEventListener). Example:

```
function logKey(event) {
  console.log(`You pressed "${event.key}".`);
}

textBox.addEventListener("keydown", logKey);
```

Instead,you can pass an anonymous function into addEventListener():

```
textBox.addEventListener("keydown", function (event) {
  console.log(`You pressed "${event.key}".`);
});
```

2. Arrow functions (https://www.w3schools.com/js/js_arrow_function.asp)
   Arrow functions allow us to write shorter function syntax.

```
hello = function() {
  return "Hello World!";
}
```

⇨

```
hello = () => {
  return "Hello World!";
}
```

   If the function has only one statement, and the statement returns a value, you can remove the brackets and the 'return' keyword:

```
hello = () => "Hello World!";
```

   Other example:

```
hello = (val) => "Hello " + val;
```

   if you have only one parameter, you can skip the parentheses as well:

```
hello = val => "Hello " + val;
```

C. Scope

1. Global Variable
   Variable declared outside the function (or loop) are called global variable, it can be accessed in entire part of document.
2. Local Variable
   Variable declared inside te function (or loop) are called local variable, it can only be accessed inside the function (or loop) where they are declared.

NOTE: If a same-named variable is declared inside & outside the function, the function will igonre the outside variable. if there's a command that execute outside the function, the inside variable will be ignored.

```
let userName = 'John';

function showMessage() {
  let userName = "Bob"; // declare a local variable

  let message = 'Hello, ' + userName; // Bob
  alert(message);
}

// the function will create and use its own userName
showMessage();

alert( userName ); // John, unchanged, the function did not access the
```

Reccomendation: To make the code clean and easy to understand, it's recommended to use mainly local variables and parameters in the function, not outer variables.

D. Naming a Function

Functions are actions. So their name is usually a verb.

Function starting with...

- "get…" – return a value,
- "calc…" – calculate something,
- "create…" – create something,
- "check…" – check something and return a boolean, etc.
- "show…" – show something.

Examples of such names:

```
showMessage(..)     // shows a message
getAge(..)          // returns the age (gets it somehow)
calcSum(..)         // calculates a sum and returns the result
createForm(..)      // creates a form (and usually returns it)
checkPermission(..) // checks a permission, returns true/false
```

Note:

One function – one action

A function should do exactly what is suggested by its name, no more.

Two independent actions usually deserve two functions, even if they are usually called together (in that case we can make a 3rd function that calls those two).

A few examples of breaking this rule:

- `getAge` – would be bad if it shows an `alert` with the age (should only get).
- `createForm` – would be bad if it modifies the document, adding a form to it (should only create it and return).
- `checkPermission` – would be bad if it displays the `access granted/denied` message (should only perform the check and return the result).

A separate function is not only easier to test and debug – its very existence is a great comment! This is the example where the second variant better the first variant:
The first variant uses a label:

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {

    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert( i ); // a prime
  }
}
```

The second variant uses an additional function isPrime(n) to test for primality: (easier to understand, isn't it?)

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);  // a prime
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

E. Callback Functions (or just callbacks)

A callback is a function passed as an argument to another function.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

Two anonymous functions as arguments in the ask() function are callback functions.

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

myDisplayer() is a callback function.

- When to Use a Callback?
  Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

F. Function Expression

Function Expression adalah cara mendefinisikan fungsi di mana fungsi tersebut ditugaskan ke sebuah variabel.

```
function sayHi() {
  alert( "Hello" );
}
```

<= Function Declaration

Function Expression =>

```
let sayHi = function() {
  alert( "Hello" );
};
```

No matter how the function is created, a function is a value. Both examples above store a function in the sayHi variable.

Note: Function declaration dapat dipanggil sebelum dideklarasikan, tetapi Function expression tidak.

Function expression(anonim) dapat digunakan sebagai:

1) Argumen untuk fungsi lain, seperti dalam metode array ‘map’, ‘filter’, atau ‘forEach’.

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(function(num) {
    return num * 2;
});
console.log(doubled); // Output: [2, 4, 6, 8, 10]
```

2) Nilai yg direturn dari fungsi lain

```
function createGreeting(name) {
    return function() {
        console.log("Hello, " + name);
    };
}
const greetJohn = createGreeting("John");
greetJohn(); // Output: Hello, John
```

3) Metode dalam objek

```
const person = {
    name: "Alice",
    greet: function() {
        console.log("Hello, " + this.name);
    }
};
person.greet(); // Output: Hello, Alice
```

4) Elemen dalam array

```
const funcs = [
    function() { console.log("First function"); },
    function() { console.log("Second function"); },
    function() { console.log("Third function"); }
];
funcs[0](); // Output: First function
funcs[1](); // Output: Second function
funcs[2](); // Output: Third function
```

5) Dalam IIFE(Immediately Invoked Function Expression)

```
(function() {
    console.log("This is an IIFE");
})(); // Output: This is an IIFE
```

- How we can make Function Expression visible outside of ‘if’ ?

```
let age = prompt("What is your age?", 18);

let welcome;

if (age < 18) {

  welcome = function() {
    alert("Hello!");
  };

} else {

  welcome = function() {
    alert("Greetings!");
  };

}

welcome(); // ok now
```

We could simplify it:

```
let age = prompt("What is your age?", 18);

let welcome = (age < 18) ?
  function() { alert("Hello!"); } :
  function() { alert("Greetings!"); };

welcome(); // ok now
```

- When to choose Function Declaration versus Function Expression?
  When we need a conditional declaration (we've just seen an example above), then Function Expression should be used.
  Full(see the last chat): https://chatgpt.com/share/4249f5da-8180-45e7-99b6-b496fba96ad2

G. JavaScript Call Stack (https://www.javascripttutorial.net/javascript-call-stack/)

the JavaScript engine uses a 'call stack' to manage execution contexts:

- Global execution context : denoted by main() or global() function on the call stack
- Function execution contexts : other than main() or global()

The call stack works based on the last-in-first-out (LIFO) principle.

When you execute a script, the JavaScript engine creates a global execution context and pushes it on top of the call stack.

Whenever a function is called, the JavaScript engine creates a function execution context for the function, pushes it on top of the call stack, and starts executing the function.

If a function calls another function, the JavaScript engine creates a new function execution context for the function being called and pushes it on top of the call stack.

When the current function completes, the JavaScript engine pops it off the call stack and resumes the execution where it left off.

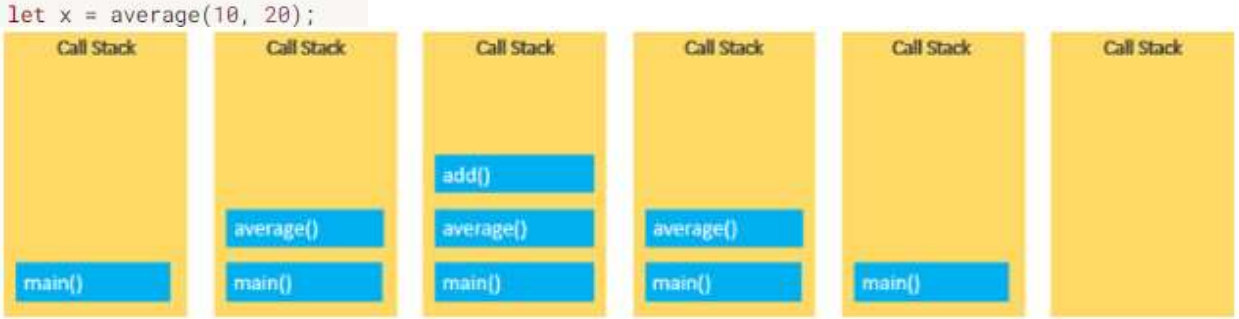
The script will stop when the call stack is empty.

JS Call Stack Example:

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

function average(a, b) {
    return add(a, b) / 2;
}

let x = average(10, 20);
```



- Stack Overflow

  If the number of execution contexts exceeds the size of the stack, a stack overflow error will occur. example, a recursive function that has no exit condition:

```
function fn() {
    fn();
}
fn(); // stack overflow
```

- Asynchronous JavaScript

  Asynchronous means the JavaScript engine can execute other tasks while waiting for another task to be completed. Hal ini berarti beberapa operasi dapat dilakukan tanpa harus menunggu operasi sebelumnya selesai. Teknik ini sangat berguna dalam meningkatkan kinerja aplikasi web, terutama untuk operasi yang memakan waktu lama seperti pengambilan data dari server, pengolahan file, atau animasi yang kompleks.

H. Note

- Rewrite the function using '?' or '||'

Using a question mark operator `'?'`:

```
function checkAge(age) {
  return (age > 18) ? true : confirm('Did parents allow you?')
}
```

Using OR `||` (the shortest variant):

```
function checkAge(age) {
  return (age > 18) || confirm('Did parents allow you?')
}
```