==========Functions==========

Function Declaration:
---
To create a function we can use a function declaration.

```
function showMessage() {
  alert( ’Hello everyone!’ );
}
```

*The function keyword goes first, then goes the name of the function,
then a list of parameters between the parentheses.
*finally the code of the function, also named “the function body”, between curly braces.

eg:
---

```
function name(parameter1, parameter2, ... parameterN) {
 // body
}
```

eg:
---

```
function showMessage() {
  alert( ’Hello everyone!’ );
}

showMessage();
showMessage();
```

**The call showMessage() executes the code of the function. Here we will see the message two times.

**one of the main purposes of functions: to avoid code duplication.

Local variables:
---

A variable declared inside a function is only visible inside that function.

For example:
---

```
function showMessage() {
  let message = "Hello, I’m JavaScript!"; // local variable

  alert( message );
}
```

```
showMessage(); // Hello, I’m JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

Outer variables:
---
A function can access an outer variable as well, for example:

```
let userName = ’sathya’;

function showMessage() {
  let message = ’Hello, ’ + userName;
  alert(message);
}

showMessage(); // Hello, sathya
```

ex:
---

```
let userName = ’soma’;

function showMessage() {
  userName = "sathya"; // (1) changed the outer variable

  let message = ’Hello, ’ + userName;
  alert(message);
}

alert( userName ); // sathya before the function call

showMessage();

alert( userName ); // soma, the value was modified by the function
```

*The outer variable is only used if there’s no local one.

**If a same-named variable is declared inside the function then it shadows the outer one.

For instance, in the code below the function uses the local userName.
The outer one is ignored:

```
let userName = ’sathya’;

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

  let message = ’Hello, ’ + userName; // soma
  alert(message);
}

// the function will create and use its own userName
```

```
showMessage();

alert( userName ); // sathya, unchanged, the function did not access the outer variable
```

**We can pass arbitrary data to functions using parameters.

ex:
---

```
function showMessage(from, text) { // parameters: from, text
  alert(from + ': ' + text);
}

showMessage('sathya', 'Hello!'); // sathya: Hello! (*)
showMessage('sathya', "What's up?"); // sathya: What's up? (**)
```

Default values:
==========

ex:
----

```
function showMessage(from, text = "no text given") {
  alert( from + ": " + text );
}

showMessage("sathya"); // sathya: no text given
```

Returning a value:
---
*A function can return a value back into the calling code as the result.

*The simplest example would be a function that sums two values:

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

let result = sum(1, 2);
alert( result ); // 3
```

ex:
---

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
    return confirm('Do you have permission from your parents?');
  }
}
```

```
let age = prompt(’How old are you?’, 18);

if ( checkAge(age) ) {
  alert( ’Access granted’ );
} else {
  alert( ’Access denied’ );
}
```

Function expressions:
---

The syntax that we used before is called a Function Declaration:
---

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

There is another syntax for creating a function that is called a Function Expression:
---

It allows us to create a new function in the middle of any expression.

For example:
---

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

=>Here we can see a variable sayHi getting a value, the new function, created as function() { alert("Hello" ); }.

ex:
---

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

// conditionally declare a function
if (age < 18) {

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

} else {

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

}
```

Arrow functions:

---

There’s another very simple and concise syntax for creating functions,
that’s often better than Function Expressions.

ex:
--

```
let func = (arg1, arg2, ..., argN) => expression;
```

ex:
--

```
let sum = (a, b) => a + b;

/* This arrow function is a shorter form of:

let sum = function(a, b) {
  return a + b;
};
*/

alert( sum(1, 2) ); // 3
```

*As you can see, (a, b) => a + b means a function that accepts two arguments named a and b.
Upon the execution, it evaluates the expression a + b and returns the result.

*If we have only one argument, then parentheses around parameters can be omitted, making that even s
horter.

ex:
--

```
let double = n => n * 2;
// roughly the same as: let double = function(n) { return n * 2 }

alert( double(3) ); // 6
```

*If there are no arguments, parentheses are empty, but they must be present:

ex:
---

```
let sayHi = () => alert("Hello!");

sayHi();
```

ex:
----

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

```
let welcome = (age < 18) ?
  () => alert(’Hello!’) :
  () => alert("Greetings!");

welcome();
```

Multiline arrow functions:
---

**The arrow functions that we’ve seen so far were very simple.
*They took arguments from the left of =>,
evaluated and returned the right-side expression with them.

ex:
--

```
let sum = (a, b) => {  // the curly brace opens a multiline function
  let result = a + b;
  return result; // if we use curly braces, then we need an explicit "return"
};

alert( sum(1, 2) ); // 3
```