**JS MILESTONE 5**

● Functions

● overloading in JS

● arguments/parameters

● JS liniting

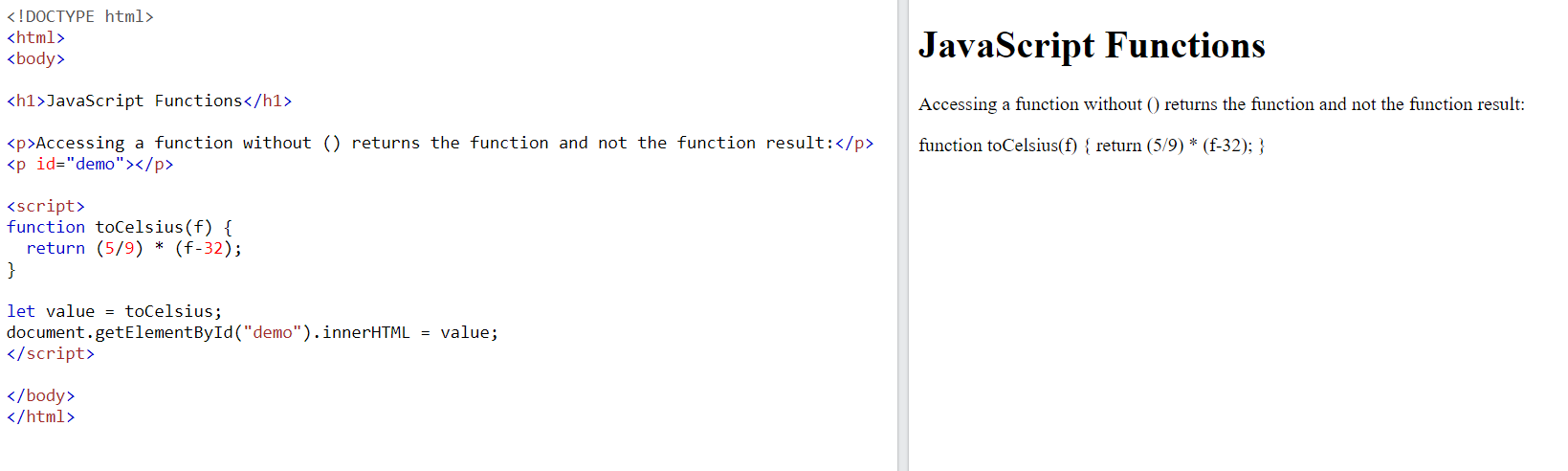
● Curring Funtions

1. Functions:

* JS function is a block of code designed to perform particular task.
* It is executed when something calls it.
* SYNTAX:

function name(parameter1, parameter2, parameter3) {  
  // code to be executed  
}

* When JavaScript reaches a return statement, the function will stop executing.
* Functions often compute a return value. The return value is "returned" back to the "caller".
* Functions are used to reuse the code.
* Same code with different arguments can be used to produce different results.
* Accessing a function without () returns the function and not the function result:



* toCelsius refers to the function object, and toCelsius() refers to the function result.
* Functions can be used directly without making to store the result in a variable value.
* Variables declared within a JavaScript function, become LOCAL to the function.
* Local variables can only be accessed from within the function.
* Local variables are created when a function starts, and deleted when the function is completed.
* When an object/ array is passed as a parameter, any change in object/ array in that function, is reflected in the original object/ array too.
* EX: function myFunc(theObject) {

theObject.make = "Toyota";}

const mycar = {

make: "Honda",

model: "Accord",

year: 1998};

console.log(mycar.make); // "Honda"

myFunc(mycar);

console.log(mycar.make); // "Toyota"

* **FUNCTION EXPRESSION:** function is not necessarily needed to have name. (anonymous function)
* **EX:** const square = function (number) {

return number \* number;};

console.log(square(4)); // 16

* A function referring to itself is called as recursion.
* Below code doesn’t produce an error. Every time the function is hoisted when the scope is being called.
* console.log(square(5)); // 25

function square(n) {

return n \* n;}

above code is equivalent to

* function square(n) {

return n \* n;}

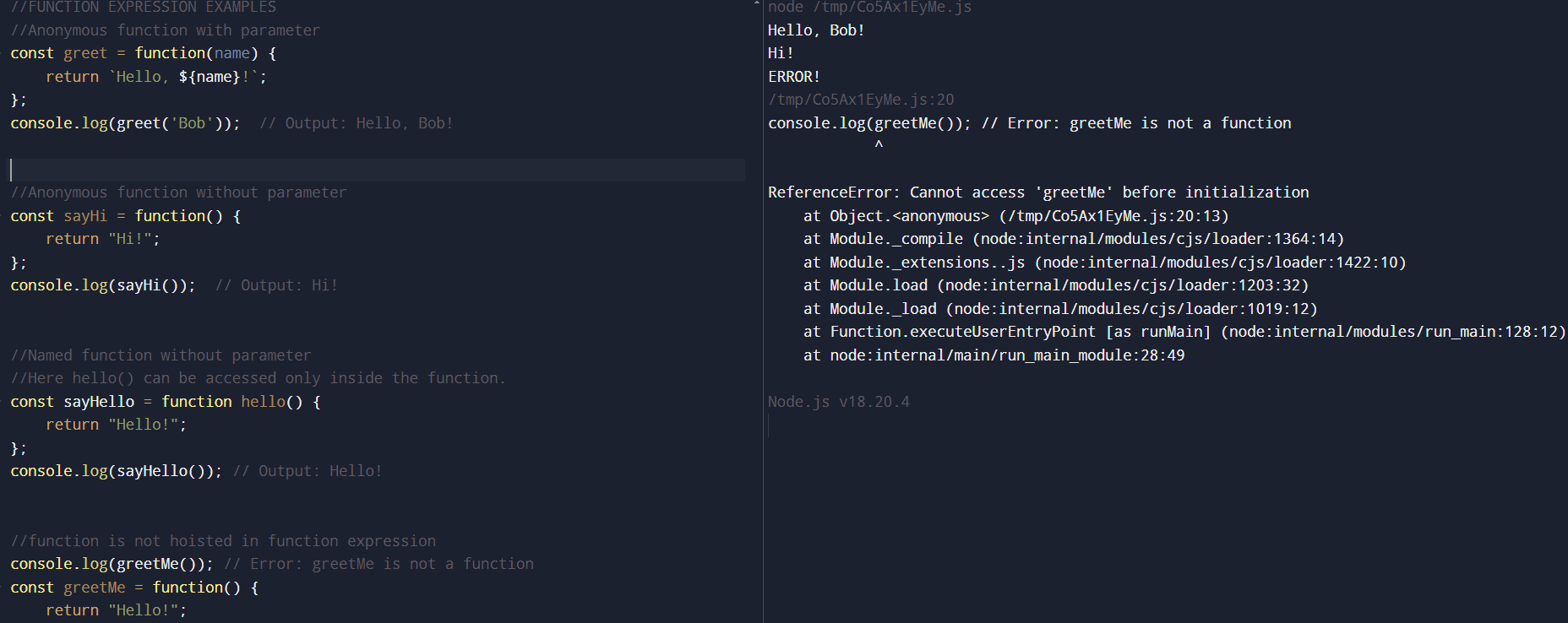
console.log(square(5)); // 25

* **FUNCTION DECLARATIONS:**
  + A function declaration defines a named function using the function keyword.
  + It is hoisted, meaning it can be called before it is defined in the code.
  + **EX:** console.log(sayHi()); // Output: Hi!

function sayHi() {

return "Hi!";}

* + Objects can be passes as an argument
  + It is always named function.
  + It has Longer syntax. It also hoisted
* **FUNCTION EXPRESSIONS:**
* It can either be anonymous or named function.
* It is not hoisted, meaning you can’t use it before it is defined.



* **ARROW FUNCTION:**
  + It is introduced in ES6.
  + They are always anonymous in nature.
  + Also they don’t have *this* binding.
  + Arrow functions are not hoisted.
  + They inherit this from lexical scope.
  + When the function has a single statement, the curly braces and return can be omitted.

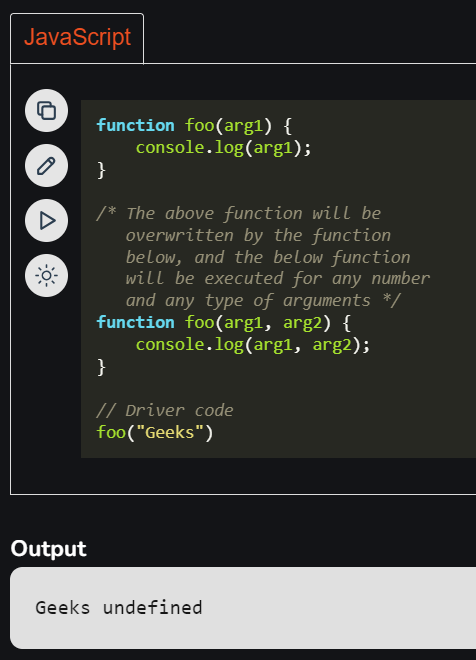


1. **Overloading JS:**

* Overloading – same function with different parameters.
  + EX: int add(arg1, arg2){ };

double add(arg1, arg2) {};

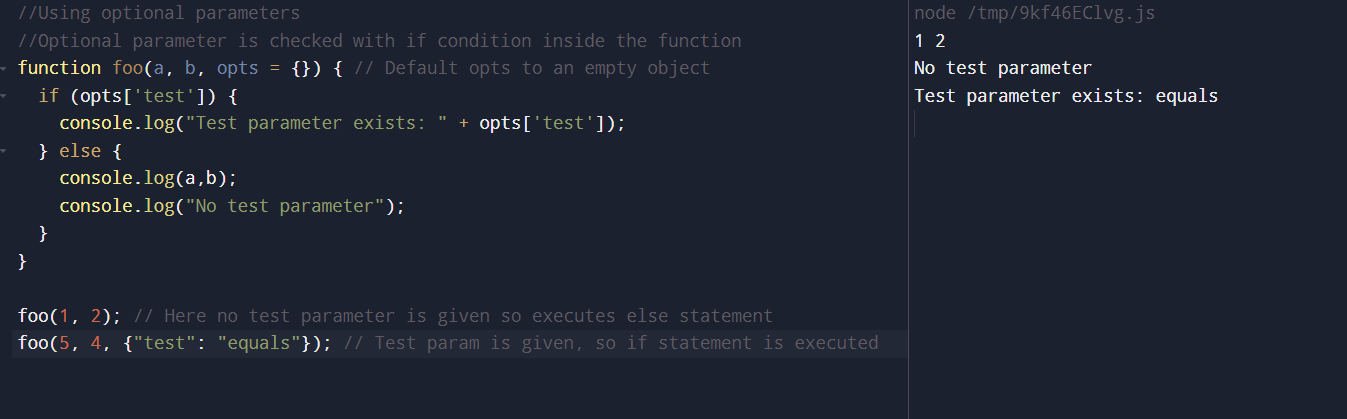
* Overloading is not supported in JS, overriding is supported in JS.
* If you declare 2/ more functions with same name, the last one will override the previously defined one.



* But overloading can be achieved by using optional parameters, arguments object, checking type for different arguments.

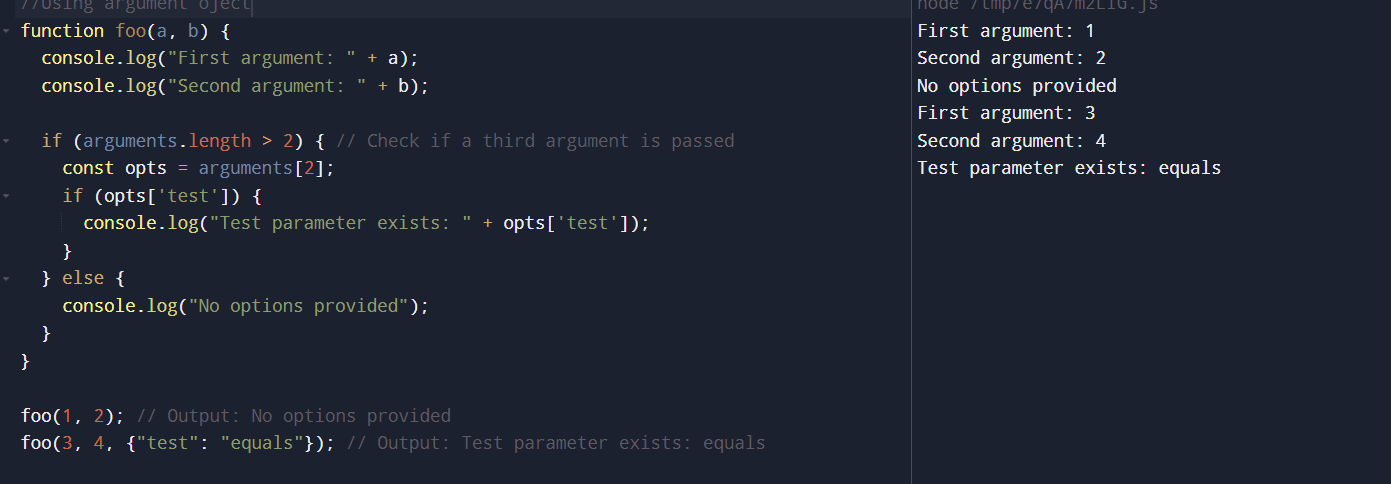
1. Optional Parameter:

The opts parameter defaults to an empty object if no third argument is provided. This way, the function can be called with 2 or 3 arguments.



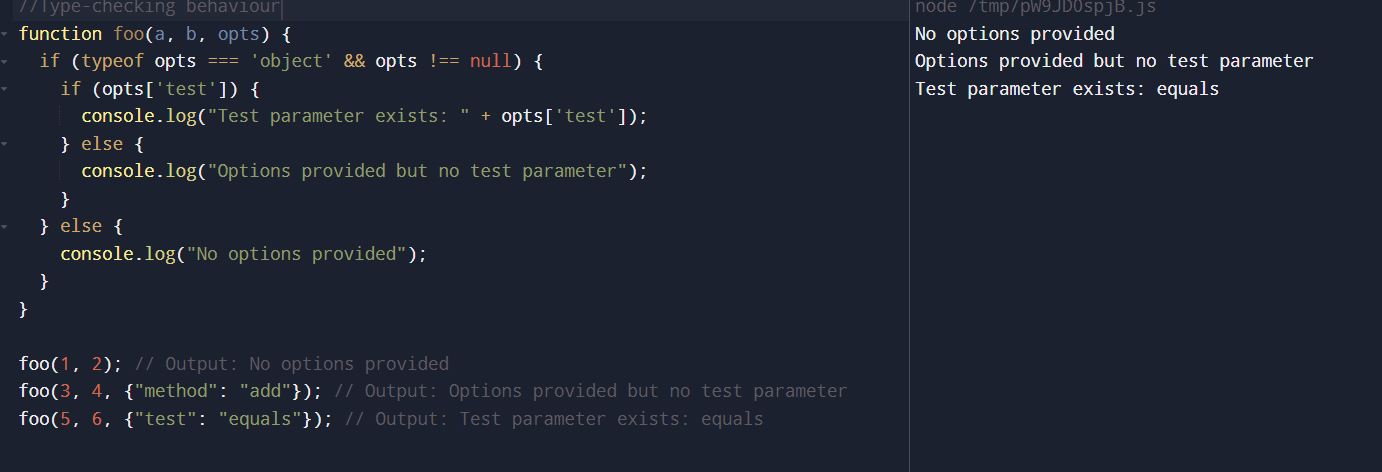
1. Argument object

Using the arguments object allows you to check how many arguments are passed and handle them accordingly.



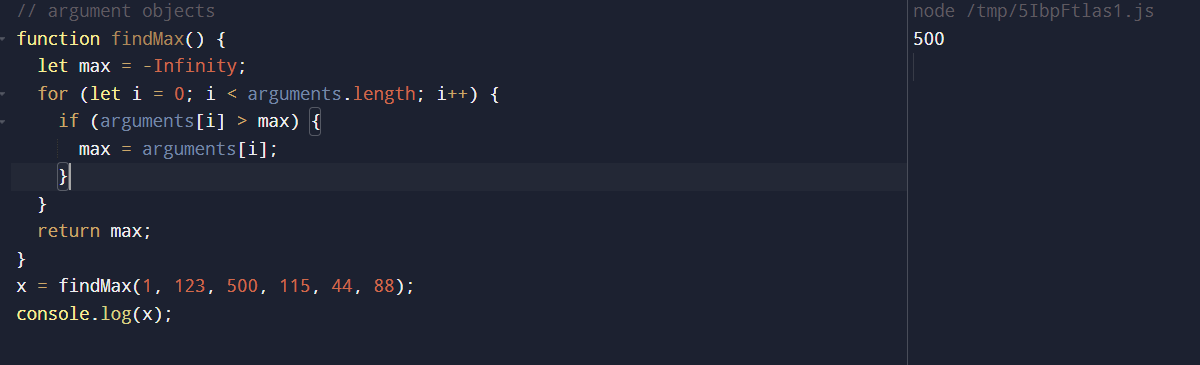
1. Type checking for overloading behaviour:

You can use typeof to check if opts is an object, and then act based on its content. This allows for flexible handling of different argument types.



3.Arguments/ Parameters:

* Function parameters are the names listed in the function definition.
* Function arguments are the real values passed to (and received by) the function.
* SYNTAX: function functionName(parameter1, parameter2, parameter3) {  
    // code to be executed  
  } functionName(arg1,arg2,arg3);
* JavaScript function definitions do not specify data types for parameters.
* JavaScript functions do not perform type checking on the passed arguments.
* JavaScript functions do not check the number of arguments received.
* In ES6, function parameters can have default values.
* EX: function myFunction(x, y = 10) {  
    return x + y;  
  }  
  myFunction(5);
* Rest parameter allows a function to treat an indefinite number of arguments as an array
* function sum(...args) {  
    let sum = 0;  
    for (let arg of args) sum += arg;  
    return sum;  
  }let x = sum(4, 9, 16, 25, 29, 100, 66, 77);
* JavaScript functions have access to a special array-like object called arguments
* If a function is called with too many arguments (more than declared), these arguments can be reached using the arguments object.



1. JS Linting:

* Linter is a tool that programmatically scans your code with the goal of finding issues that can lead to bugs or inconsistencies with code health and style. Some can even help fix them for you!
* Popular JS Linters are ESLint, JSLint, JSHint
* JS linting is a powerful tool for improving code equality, preventing errors, ensuring consistency across the project.
* JS Linting can be used to detect errors like undeclared variables, missing semicolons, incorrect use of functions.
* Some linters can automatically fix minor issues, like formatting ie., converting var to let, adding semicolons etc., which allows developers to focus on more complex problems.

1. Currying functions:

* This technique is named after Haskell Curry, an American mathematician and logician.
* Currying breaks down a function that takes multiple parameters into a chain of functions that each take one parameter.
* It helps us to create higher-order function
* It reduces chances of error in our function by dividing it into multiple smaller functions that can handle one responsibility.
* It makes code more reusable
* EX: // Regular function

function simpleFunction(param1, param2, param3) {

return param1 + param2 + param3;}

// Curried function

function curriedFunction(param1) {

return function(param2) {

return function(param3) {

return param1 + param2 + param3;

};

};

}