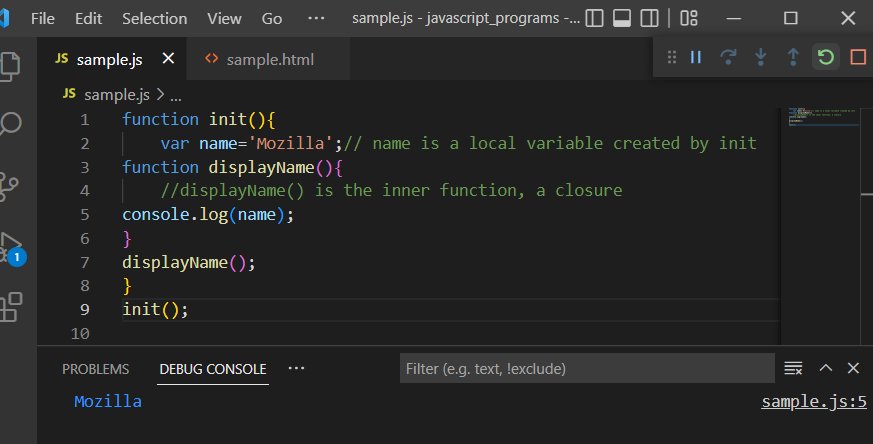
**Chapter 10- Closure**

* In many programming languages when a function is called, a memory is allocated for the variables and the functions inside the stack. And this memory will be wiped off when the function call is exited.
* But in Javascript the variables and the functions defined inside that function still remain in memory even after the function is called.

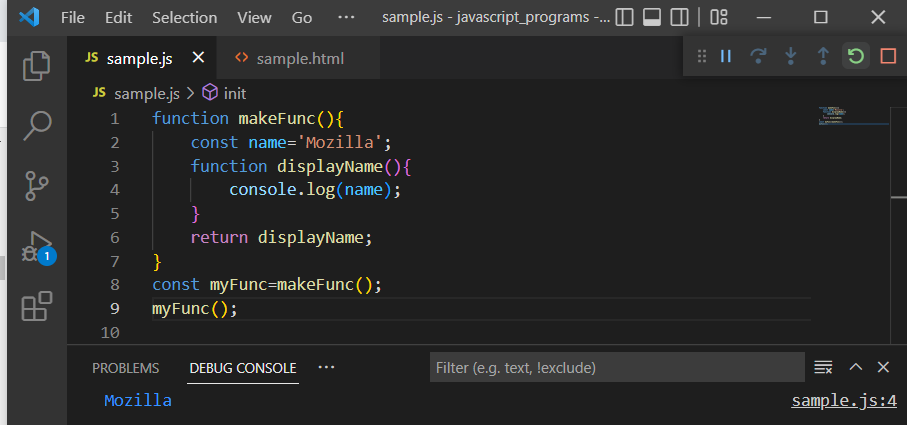
**Lexical scoping:**



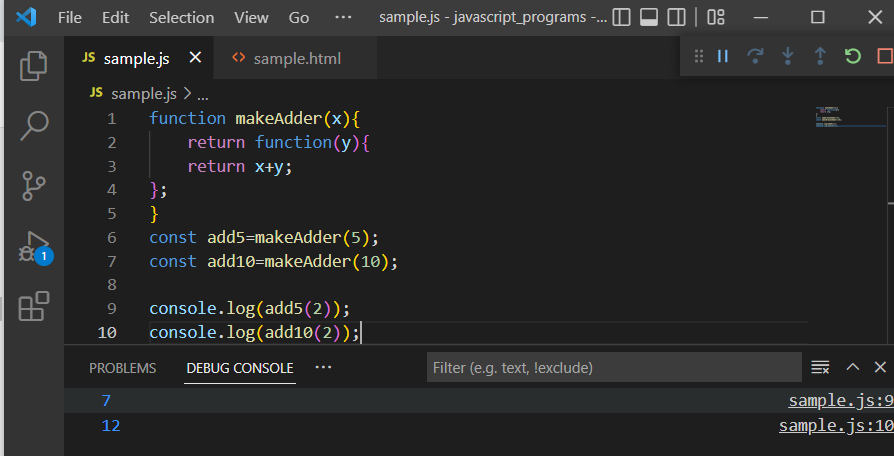
In the above example, init() creates a local variable name and a function displayName() when it is called. The displayName() function is an inner function that is defined inside the function init() and is available within the body of the init() function. The displayName() function has no local variable and has access to its parent function init(). This is called lexical scoping.

**Closure:**

A closure is a combination of a function and the lexical environment within which that function was declared.



In other programming languages when the function finishes execution the variable defined inside the function is no longer accessible, but in case of javascript it is not so. myFunc is a reference to the instance of the function displayName that is created when makeFunc is run. The instance of displayName maintains a reference to its lexical environment, within which the variable name exists. For this reason, when myFunc is invoked, the variable name remains available for use and Mozilla is passed to console.log.

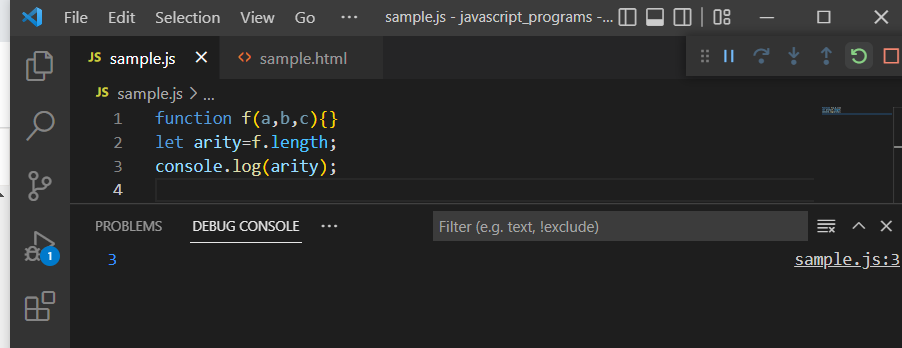


In the above example, the function makeAdder(x), is the function factory. It creates functions that can add a specific value to their argument.

Add5 and add10 are both closures. They share the same function body definition but store different lexical environments. In add5’s lexical environment, xis 5, while in the lexical environment for add10, x is 10.

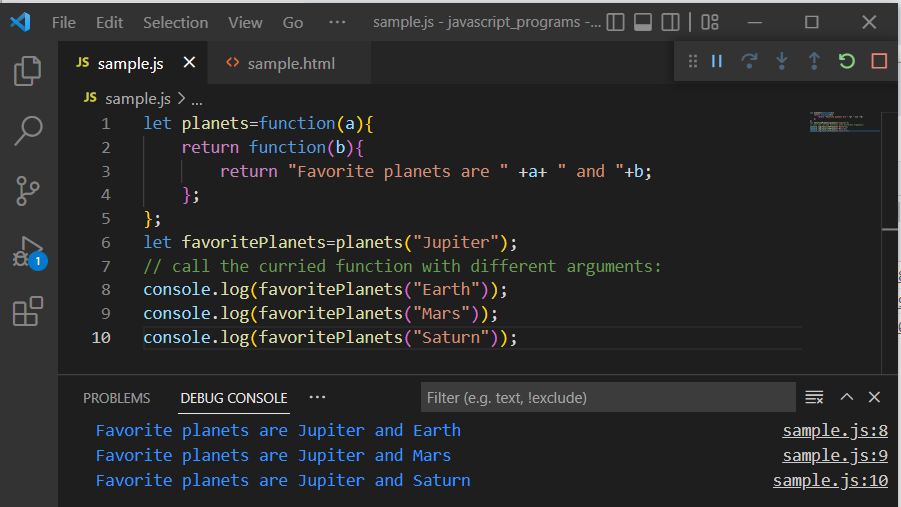
**Arity:**

Arity is the number of arguments a function takes. It can be accessed by function.length property.



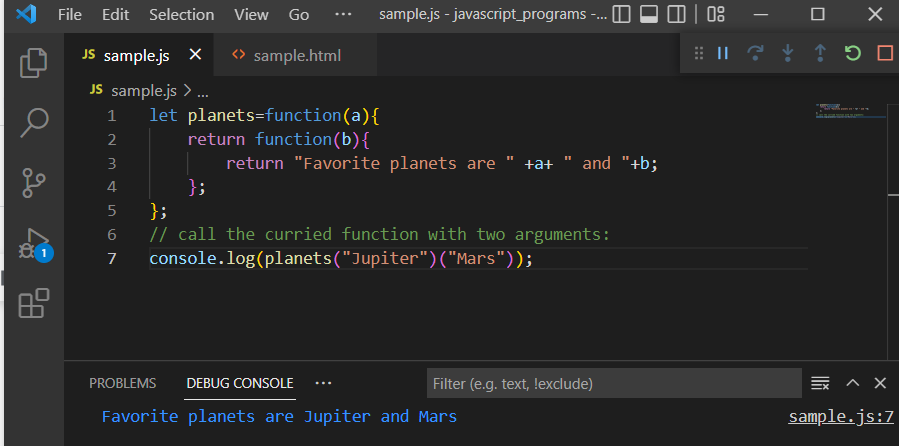
**Currying:**

A curried function can be constructed by chaining closures by defining and immediately returning all inner functions at the same time.



Function planets returns another anonymous function. So, when it is assigned to favoritePlanets with one argument “Jupiter”, it can be called again with a secondary argument.

The above program can also be written like this:



The older currying syntax can be rewritten with a far more elegant arrow function format:

