Quizlet

IIFE

Study online at quizlet.com/_8jmkux

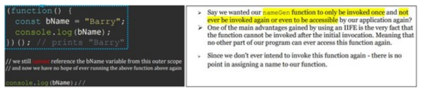
1. **How can IIFEs help to keep the global namespace from being polluted by variables that are only used once?**

   Using IIFEs ensures our global namespace remains unpolluted by one off variable names because those variables are only available in the scope of the IIFE. If an IIFE has been invoked then the variables are no longer available in any other scope - inlcuding the global scope.

2. **It is best practice to write an IIFE as what kind of function?**

   An anonymous function expression. Since an IIFE will not be reinvoked - it is best practice to keep it anonymous.

3. **True or False: A single IIFE can be invoked multiple times throughout an application.**

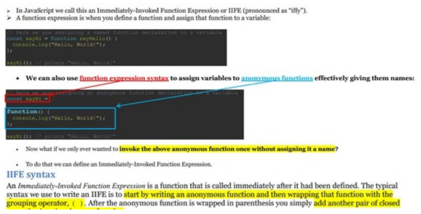
   

```
(function() {
  const bName = "Barry";
  console.log(bName);
})(); // prints "Barry"

// we still cannot reference the bName variable from this outer scope
// and now we have no hope of ever running the above function above again

console.log(bName); //
```

   - Say we wanted our nameGen function to only be invoked once and not ever be invoked again or even to be accessible by our application again?
   - One of the main advantages gained by using an IIFE is the very fact that the function cannot be invoked after the initial invocation. Meaning that no other part of our program can ever access this function again.
   - Since we don't ever intend to invoke this function again - there is no point in assigning a name to our function.

   False! An IIFE is invoked once then never again.

4. **What does IIFE stand for?**

   - In JavaScript we call this an Immediately-Invoked Function Expression or IIFE (pronounced as "iffy").
   - A function expression is when you define a function and assign that function to a variable:

```
// here we are assigning a named function declaration to a variable
const sayHi = function sayHello() {
  console.log("Hello, World!");
};

sayHi(); // prints "Hello, World!"
```

   - We can also use function expression syntax to assign variables to anonymous functions effectively giving them names:

```
// here we are assigning an anonymous function declaration to a variable
const sayHi =

function() {
  console.log("Hello, World!");
};

sayHi(); // prints "Hello, World!"
```

   - Now what if we only ever wanted to **invoke the above anonymous function once without assigning it a name**?
   - To do that we can define an Immediately-Invoked Function Expression.

   IIFE syntax

   An *Immediately-Invoked Function Expression* is a function that is called immediately after it had been defined. The typical syntax we use to write an IIFE is to start by writing an anonymous function and then wrapping that function with the grouping operator, ( ). After the anonymous function is wrapped in parenthesis you simply add another pair of closed

   Immediately-Invoked Function Expression

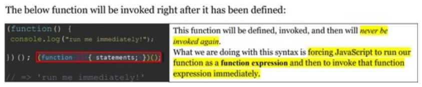
5. **What is is the value of the test variable when the below code is run?**
   1 const test = (function() {
   2 return "hello";
   3 })();
   4
   5 console.log(test); // ???

   The value of the test variable will be "hello". When an IIFE is assigned to a variable the function will be invoked and then the return value of that function will be assigned as the variable's value.

6. **What will happen when the below function is run? 1 (function() {**
   2 const test = "Hello world!";
   3 })();
   4 console.log(test); // ???

   An error will be thrown. The variables defined within an IIFE are not available in an outer scope.

7. **When is an IIFE invoked?**

   The below function will be invoked right after it has been defined:

```
(function() {
 console.log("run me immediately!");

}) (); (function () { statements; })();

// => 'run me immediately!'
```

   This function will be defined, invoked, and then will *never be invoked again*.
   What we are doing with this syntax is forcing JavaScript to run our function as a **function expression** and then to invoke that function expression immediately.

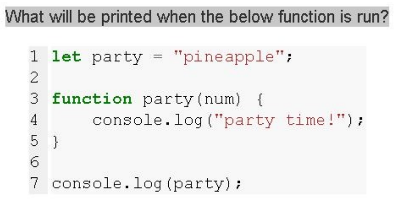
   Immediately after it has been defined.

Quizlet

Hoisting in JavaScript

Study online at quizlet.com/_8m8tvx

1. **What will be printed when the below function is run?**



What will be printed when the below function is run?

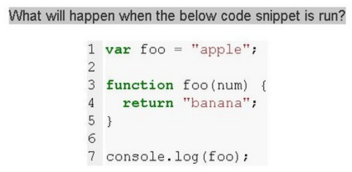
```
let party = "pineapple";

function party(num) {
    console.log("party time!");
}

console.log(party);
```

A ReferenceError will be thrown. As soon as the function declaration is run an error will be thrown because the above let variable has the same "party" name.

2. **What will happen when the below code is run?**
bark();
function bark() {
console.log("Woof!");
}

"Woof" will be printed to the console. The bark function is a function declaration and it will therefore be hoisted and available everywhere in the program.

3. **What will happen when the below code snippet is run?**



What will happen when the below code snippet is run?

```
var foo = "apple";

function foo(num) {
  return "banana";
}

console.log(foo);
```

"apple" will be printed to the terminal. When declaring a variable using var that variable assignment will take precedence over any declared functions with the same name.

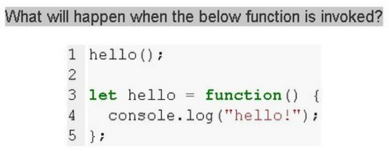
4. **What will happen when the below code snippet is run?**
1 hello();
2 function hello() {
3
console.log("hello!");
4 }

"hello" will be printed to the console. Since this is a function declration the name of the hello function will be hoisted to the top of the scope and so it will be available in memory starting from the first line of the snippet.

5. **What will happen when the below code snippet is run?**
var foo;
function foo(num) {
return "I'm a function";
}
console.log(foo);

The foo function will be printed to the console. Function declaration takes presedence over var variable declaration (but not variable assignment).

6. **What will happen when the below function is invoked?**



What will happen when the below function is invoked?

```
hello();

let hello = function() {
  console.log("hello!");
};
```

An "ReferenceError: hello is not defined" error will be thrown. A let declared Function Expression is not hoisted so the "hello" let variable will be unavailable.

7. **What will happen when the below function is invoked?**
hello();
var hello = function()
{
console.log("hello!");
};

"TypeError: hello is not a function" will be thrown. The declaration of hello is hoisted but not the assignment. So hello will exist is the namespace above where this function is defined but the value is undefined and therefore cannot be invoked.

8. **Which way of defining a new function will hoist the name of the function to the top of it's current scope?**

A function using function declaration.

Falsey-Values-Javascript

Quizlet

Study online at quizlet.com/_8m8udh

1.	**Is 0 truthy or falsey in JavaScript?**	0 is falsey!
2.	**Is an empty object, ({}), truthy or falsey in JavaScript?**	{} is truthy! This is a tricky one - because an empty string ("") is falsey!
3.	**Is an empty string, (""), a truthy or falsey value in JS?**	An empty string is falsey.
4.	**Is an empy array, ([]), truthy or falsey in JavaScript?**	An empty array is truthy!
5.	**Is ("false") truthy or falsey in JavaScript?**	Truthy! A non-empty string like "false", no matter the characters within, will always be truthy.
6.	**Name the seven falsey values in JavaScript.**	false, 0, NaN, null, undefined, "" (empty string), and 0n. 0n is the BigInts primitive data type falsey value.

7.	**What will be printed to the console when the below code is run?**
	
	if (NaN || 0 || "" || null || undefined) {
	console.log("hello");
	} else {
	console.log("goodbye");
	}
		"goodbye" will be printed because all of the values within the if conditional are falsey.

8.	**What will be printed when the below code is run?**
	
	if (0 || "" || "0") {
	console.log("hello");
	} else {
	console.log("goodbye");
	}
		A non empty string ("0") will evaluate to true so the above code will print "hello"