

First Class Functions ft. Anonymous Functions

Friday, 18 August 2023 8:00 PM

→ Function Statement;

```
// Function Statement
function a() {
  console.log("a called");
}
```

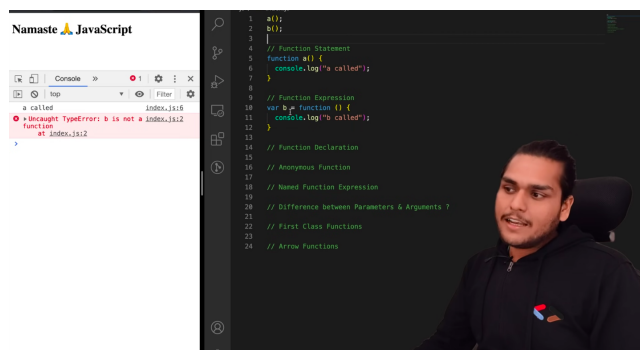
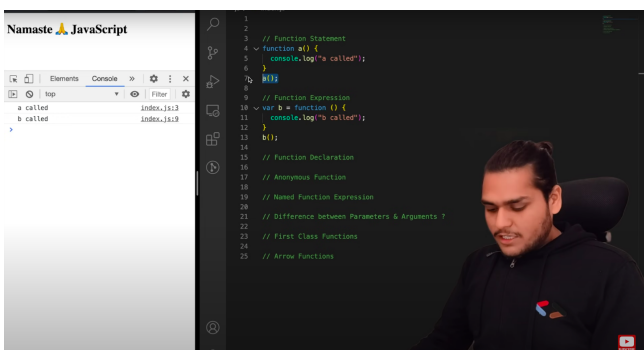
This way of creating a fⁿ is called a **fⁿ statement**.

→ Function Expression;

```
// Function Expression
var b = function () {
  console.log("b called");
}
```

When we assign a fⁿ to a variable then this way of creating a fⁿ is called **fⁿ expression**.

→ Difference b/w fⁿ statement and fⁿ expression :

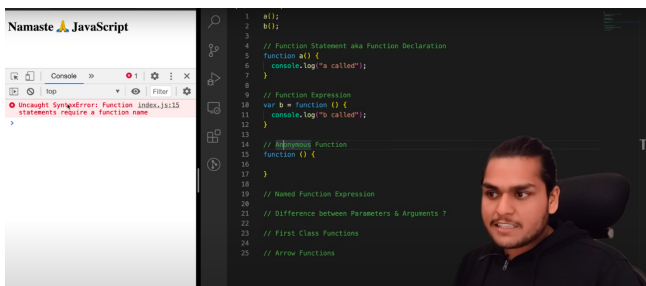


→ The major difference b/w both is **hoisting**. During memory creation phase JS allocates 'undefined' to 'b' and stores 'a' as it is in memory. So if we try to do **console.log(b);** then we will get 'undefined' but if we try to invoke b() then it will give us **TypeError: b is not a function**.

→ Function Declaration: Function statements are also known as function declarations.

```
// Function Statement aka Function Declaration
function a() {
  console.log("a called");
}
```

→ Anonymous function: Anonymous fⁿ is basically a fⁿ statement with no name. But according to ECMAScript a fⁿ statement must have a name.



→ So this results to a **syntax error**.

So by looking at above one can say that what will be the use of anonymous fⁿs?

→ They are used in a place where fⁿs are used as values. So, they basically can be used to assign values to variables and ultimately become fⁿ expressions.

eg:
=

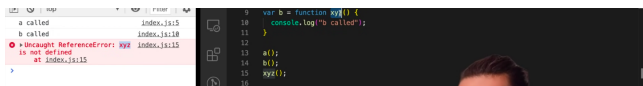
```
var b = function () {  
  console.log("b called");  
}
```

→ Named function expression? They are same as fⁿ expressions but we do not use anonymous fⁿ and instead we use a normal fⁿ.

eg:

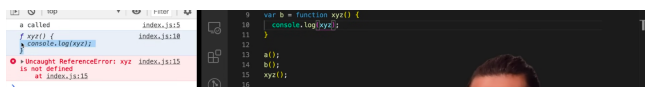
```
var b = function xyz() {  
  console.log("b called");  
}
```

Note:



We can't call xyz() and it gives us **ReferenceError** because xyz is in local scope and is not present in global scope.

But if we try to access xyz inside xyz() then it is fine.



→ Difference b/w parameters and arguments:

Parameters are local variables of a fⁿ they can't be accessed outside the function and are the identifiers which receive values.

Arguments are the variables that are passed to a fⁿ during fⁿ call or fⁿ invocation.

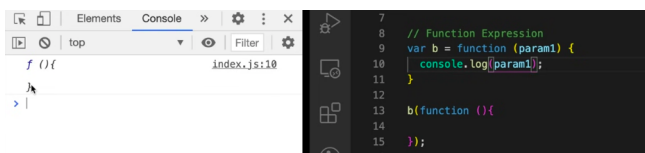
```
var b = function (param1, param2) {  
  console.log("b called");  
}  
  
a();  
b(1, 2);
```

→ First class functions: The ability of fⁿs to be used as values is known as first class fⁿs.

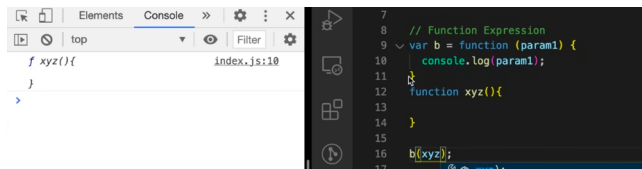
We can pass fⁿs as arguments or even return fⁿ from another fⁿ.

We can assign fⁿs to a variable. These abilities of fⁿs make them first class fⁿs.

eg:
=

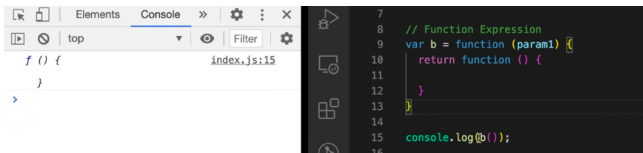


(passing anonymous fⁿ as argument)



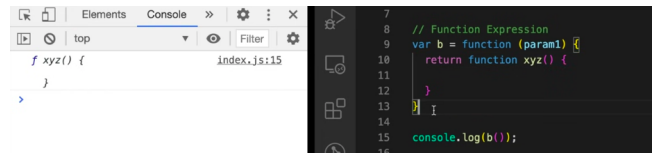
(passing named fⁿ as argument)

eg:



```
7 // Function Expression
8 var b = function (param1) {
9   return function () {
10     // ...
11   }
12 }
13 console.log(b());
```

(anonymous fn returned from another fn)



```
7 // Function Expression
8 var b = function (param1) {
9   return function xyz() {
10     // ...
11   }
12 }
13 console.log(b());
```

(named fn returned from another fn)

Note: The ability of fns to be used as values makes them first class citizens.

Note: If we use `let` & `const` instead of `var` in a fn expression, then they are simply treated like normal variables which follow all rules of hoisting & are in temporal deadzone initially.

→ Arrow functions: This was introduced in ES6.