JS

JAVASCRIPT ARROW FUNCTIONS

```
( ) => {

};
```

codewithgauri

INTRODUCTION :

An arrow function expression is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

There are differences between arrow functions and traditional functions, as well as some limitations:

- Arrow functions don't have their own bindings to this, arguments or super, and should not be used as methods.

- Arrow functions don't have access to the new.target keyword.

- Arrow functions aren't suitable for call, apply and bind methods, which generally rely on establishing a scope.

- Arrow functions cannot be used as constructors.

- Arrow functions cannot use yield, within its body.

Comparing traditional functions to arrow functions

Let's decompose a "traditional anonymous function" down to the simplest "arrow function" step-by-step:

```
// Traditional Anonymous Function
(function (a) {
  return a + 100;
});

// Arrow Function Break Down

// 1. Remove the word "function" and place arrow between the argument and opening body bracket
(a) => {
  return a + 100;
};

// 2. Remove the body braces and word "return" — the return is implied.
(a) => a + 100;

// 3. Remove the argument parentheses
a => a + 100;
```

NO ARGUMENTS

If you have no arguments, you'll need to re-introduce parentheses around the arguments:

```
// Traditional Anonymous Function (no arguments)
const a = 4;
const b = 2;
(function () {
  return a + b + 100;
});

// Arrow Function (no arguments)
const a = 4;
const b = 2;
() => a + b + 100;
```

MULTIPLE ARGUMENTS

If you have multiple arguments, you'll need to re-introduce parentheses around the arguments:

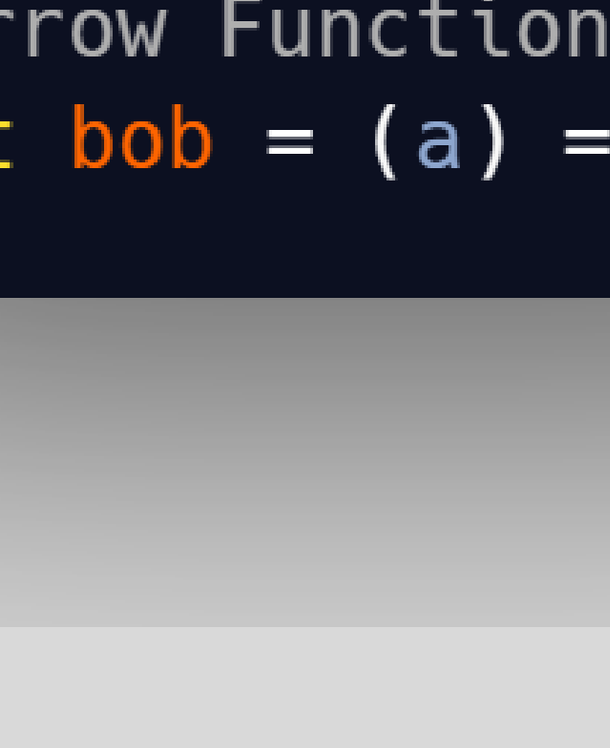
```
// Traditional Anonymous Function
(function (a, b) {
  return a + b + 100;
});

// Arrow Function
(a, b) => a + b + 100;
```

NAMED FUNCTION

```
// Traditional Function
function bob(a) {
  return a + 100;
}

// Arrow Function
const bob = (a) => a + 100;
```



NO BINDINGS OF ARGUMENTS

Arrow functions do not have their own arguments object. Thus, in this example, arguments is a reference to the arguments of the enclosing scope:

```
const arguments = [1, 2, 3];
const arr = () => arguments[0];

arr(); // 1

function foo(n) {
  const f = () => arguments[0] + n; // foo's implicit arguments binding. arguments[0] is n
  return f();
}

foo(3); // 3 + 3 = 6
```

And for amazing stuff you can follow me

Gaurav Pandey

LinkedIn :Gaurav Pandey

Twitter : @gauravcode

References:

Arrow function expressions - JavaScript | MDN (mozilla.org)