Arrow Function and Classes in JavaScript

Arrow Function

1. In arrow function, this refers to the global object where as in the normal function, it refers to the block object. So, whenever you are using this keyword use normal function if not then use arrow function. Also arrow function processes quick.

Demo

```
let user = {
    name: "GFG",
    gfg1:() => {
        console.log("Arrow -> " + this.name);
    },
    gfg2(){
        console.log("Normal ->" + this.name);
    }
 };
user.gfg1();
user.gfg2();
```

2. Do not use arrow function when you're using a class. Since `this` refers to the global object.

Demo

```
1. Do not use this

class Counter {
  constructor() {
    this.count = 0;
  }
  next = () => {
    return ++this.count;
  }
  current = () => {
    return this.count;
  }
}

2. Use this

class Counter {
  constructor() {
    this.count = 0;
  }
  next() {
    return ++this.count;
  }
  current() {
    return this.count;
  }
}
```

Classes

Classes are also a functions in JavaScript.

Declaring a class in a JavaScript

```
Function Type
const x = function(){}
const y = class(){}

Initialize a constructor function

function FunctionOne(name, age){
  this.name = name;
  this.age = age;
}

class ClassOne{
  constructor(name,age){
  this.name = name;
  this.age = age;
  }
}
```

Methods in classes

```
// Function
FunctionOne.prototype.getName = function(){
  return `I am ${this.name} and my age is ${this.age}`;
}
// This is also valid. But the problem here is that everytime you create a
// FunctionOne object it calls the getName. So it is not good. It's better
// to use with the prototype.
```

```
this.getName = function(){
  return `I am ${this.name} and my age is ${this.age}`;
}

// Class
getName(){
  return `I am ${this.name} and my age is ${this.age}`;
}
```

Demo

```
// Full Demo using Function
function FunctionOne(name, age){
  this.name = name;
  this.age = age;

}
// Function
FunctionOne.prototype.getName = function(){
  return `I am ${this.name} and my age is ${this.age}`;
}
// This is also valid. But the problem here is that everytime you create a
// FunctionOne object it calls the getName. So it is not good. It's better
// to use with the prototype.
this.getName = function(){
  return `I am ${this.name} and my age is ${this.age}`;
}

const getFunction = new FunctionOne('Pramish',24)
console.log(getFunction.getName())
```

```
// Full Demo using Class

class ClassOne{
  constructor(name,age){
  this.name = name;
  this.age = age;
  }

getPeople(){
return `I am ${this.name} and my age is ${this.age}`;
}
}

const hello = new ClassOne('Pramish',24)
console.log(hello.getPeople())
```