# Static methods

The static methods are functions attached directly to the class. They hold logic related to the class, rather than to the instance of the class.

To create a static method using the special keyword `static` followed by a regular method syntax:

```
static myStaticMethod() { ... }
```

When working with static methods, there are 2 simple rules to remember:

1. A static method ***can access*** static fields
2. A static method ***cannot access*** instance fields

For example, let's create a static method that **detects whether a user with a specific name was already taken**.

```
class User {
  static #takenNames = [];

  static isNameTaken(name) {
     return User.#takenNames.includes(name);
  }
  name = 'Unknown';

  constructor(name) {
    this.name = name;
    User.#takenNames.push(name);
  }
}

const user = new User('Jon Snow');

console.log(User.isNameTaken('Jon Snow'));
console.log(User.isNameTaken('Arya Stark'));
```

```
// Output
True
False
```

## Description of above code

- `isNameTaken()` is a static method that uses the **static private field** `User.#takenNames` to check whether that particular name has been taken or not with the help of `includes` method. `includes` method is an in-built method provided by *Javascript* to check *`whether a particular data exist in a variable or not`* and returns value as `true` or `false` accordingly.
- First of all, we have created an **instance** of **User** class by passing the name "Jon Snow", so the static private field of **User** class `#takenNames` has an element "Jon Snow".
- Now We have printed `User.isNameTaken('Jon Snow')`. Since "Jon Snow" is present in the **static private field** `#takenNames` so, it will print `true`.
- As the **static private field** `#takenNames` doesn't contain "Arya Stark" hence, `User.isNameTaken('Arya Stark')` will print `false`.