Inside your extension the last two functions must be this.init and then this.cosmos. Don't write them anywhere else inside the document.

WRONG

```
function i_test()
{

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    this.somefunction = function()
    {

    };

    this.someotherfunction = function()
    {

    };

}
```

CORRECT

```
function i_test()
{

    this.somefunction = function()
    {

    };

    this.someotherfunction = function()
    {

    };

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

}
```

Every application must have three core variables (cosmos_exists, is_init, cosmos) which you define them at the end of your file with this order. Also there are some checks at the this.init and this.cosmos functions which are the same at every application and inside these checks you write your code.

WRONG

```
function i_test()
{

    this.init = function()
    {

        // Code of your application

        return true;

    };

    this.cosmos = function()
    {

        if (cosmos_exists === true)
            return false;

        if (cosmos_object === undefined)
            return false;

        cosmos_exists = true;

        // Code of your application

        return true;

    };

    var is_init = false,
        cosmos_exists = false,
        cosmos = null;

}
```

CORRECT

```
function i_test()
{

    this.init = function()
    {

        if (is_init === true)
            return false;

        // Code of your application

        is_init = true;

        return true;

    };

    this.cosmos = function()
    {

        if (cosmos_exists === true)
            return false;

        if (cosmos_object === undefined)
            return false;

        // Code of your application

        cosmos_exists = true;

        return true;

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null;

}
```

In every application you define some global variables which contain services or containers from cosmos (vulcan , colony etc etc). These ones that you get directly from cosmos hub, you must define them inside this.cosmos function. The other ones that you get from containers like matrix, you define them inside this.init function. And all these inside the appropriate checks that we mentioned before.

WRONG

```
this.init = function()
{

    if (is_init === true)
        return false;

    // Code of your application
    vulcan = cosmos.hub.access('vulcan');
    pythia = cosmos.hub.access('pythia');
    colony = cosmos.hub.access('colony');
    swarm = cosmos.hub.access('swarm');
    matrix = cosmos.hub.access('matrix');
    dev_box = cosmos.hub.access('dev_box');
    i_test = dev_box.get('bee');
    infinity = dev_box.get('infinity');
    infinity.init(cosmos, id + '_data');
    fx = dev_box.get('fx');
    fx.init(cosmos);

    var __nature = matrix.get('nature');

    __nature.themes(['i_test']);
    __nature.apply();

    is_init = true;

    return true;

};

this.cosmos = function()
{

    if (cosmos_exists === true)
        return false;

    if (cosmos_object === undefined)
        return false;

    // Code of your application

    cosmos_exists = true;

    return true;

};
```

CORRECT

```
this.init = function()
{

    if (is_init === true)
        return false;

    // Code of your application
    i_test = dev_box.get('bee');
    infinity = dev_box.get('infinity');
    infinity.init(cosmos, id + '_data');
    fx = dev_box.get('fx');
    fx.init(cosmos);

    var __nature = matrix.get('nature');

    __nature.themes(['i_test']);
    __nature.apply();

    is_init = true;

    return true;

};

this.cosmos = function()
{

    if (cosmos_exists === true)
        return false;

    if (cosmos_object === undefined)
        return false;

    // Code of your application
    vulcan = cosmos.hub.access('vulcan');
    pythia = cosmos.hub.access('pythia');
    colony = cosmos.hub.access('colony');
    swarm = cosmos.hub.access('swarm');
    matrix = cosmos.hub.access('matrix');
    dev_box = cosmos.hub.access('dev_box');

    cosmos_exists = true;

    return true;

};
```

At the end of your file where you define the global variables, you must write them in some kind of order. First you write the three core variables (cosmos_exists, is_init, cosmos) then the system's variables (vulcan, colony etc) and then your application's variables. We mentioned the order of the core variables, but you must also write the system's variables in order too. You have to put all the utilities / service first and then the others. A general example you should use is the following one.

WRONG

```
var cosmos_exists = false,
    is_init = false,
    cosmos = null,
    infinity = null,
    i_test_bee = null,
    id = 'i_test',
    vulcan = null,
    pythia = null,
    dev_box = null,
    app_box = null,
    hive = null,
    colony = null,
    swarm = null,
    forest = null,
    matrix = null,
    fx = null;
```

CORRECT

```
var cosmos_exists = false,
    is_init = false,
    cosmos = null,
    vulcan = null,
    fx = null,
    pythia = null,
    infinity = null,
    colony = null,
    swarm = null,
    hive = null,
    matrix = null,
    dev_box = null,
    app_box = null,
    i_test = null,
    id = 'i_test';
```

When you want to define a variable in your application, take a look at the right code block here and decide where you need to put it.

You must understand the scope of our technique. I am going to make an example using PHP in order to understand it better. In PHP, inside a class we have public functions and private functions (also protected but w/e, not our case here) . When you create an object of this class, you can only access the public functions. Private functions can be accessed only by the class itself. So basically private functions exist for “inside” reasons of the class. With the scope we have developed here, we can do the same thing in Javascript. So from now on, you need to consider what is the purpose of the function that you creating and if you need to make it private or public. A quick example is that must not create the Ajax function public but private, because whoever is going to create an instance of this class/ application shouldn't be able to make any Ajax call he wants.

Extension only with public functions

```
function i_test()
{

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

}
```

Extension with public and private functions

```
function i_test()
{

    function utilities()
    {

        this.ajax = function()
        {

        };

    };

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var utils = new utilities();

}
```

If I create a new instance of this class and console.log this instance, you will see that in both cases I have access only to .init and .cosmos functions.

```
> var inst = new i_test();
  undefined
> console.log(inst);
  ▼ i_test {init: function, cosmos: function}                    VM130:2
    ▶ cosmos: function ()
    ▶ init: function ()
    ▶ __proto__: i_test
```

When you create private functions , this means that you have created subclasses. All the subclasses must be we organized and you should define them at the end of EACH class which they belong, right after the declaration of the variables. I am going to demonstrate two examples, one for the main class of our extension and one for a subclass inside a subclass.

WRONG

```
function i_test()
{

    function utilities()
    {

        this.ajax = function()
        {

        };

    }

    var utils = new utilities();

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test';

}
```

CORRECT

```
function i_test()
{

    function utilities()
    {

        this.ajax = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

WRONG

```
function i_test()
{

    function utilities()
    {

        this.ajax = new ajax();

        function ajax()
        {

            this.data = function()
            {

            };

            this.response = function()
            {

            };

        }

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

CORRECT

```
function i_test()
{

    function utilities()
    {

        function ajax()
        {

            this.data = function()
            {

            };

            this.response = function()
            {

            };

        }

        this.ajax = new ajax();

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

In order to use these subclasses, you need to define a inside scope variable in which you will assign the “this” keyword. For the main class name it “self” and for subclasses name it “me”. Define at the beginning of the class. I am going to demonstrate an example for the main class and one for the subclass.

WRONG

```
function i_test()
{

    function utilities()
    {

        this.somefunction = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        self = this,
        utils = new utilities();

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function utilities()
    {

        this.somefunction = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

WRONG

```
function i_test()
{

    var self = this;

    function utilities()
    {


        this.somefunction = function()
        {

        };

        this.someotherfunction = function()
        {

        };

        var me = this;

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function utilities()
    {

        var me = this;

        this.somefunction = function()
        {

        };

        this.someotherfunction = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

Sometimes you want to create public subclasses inside your main class. For example when you access a bee, you can go to .gui which it has many functions inside. When you create these subclasses, you need to create the inside instance of the this class, at the very end of the main file, after you finish declaring the private functions.

WRONG

```
function i_test()
{

    var self = this;

    this.settings = new settings();

    function utilities()
    {

        var me = this;

        this.somefunction = function()
        {

        };

    }

    function settings()
    {

        this.container = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function utilities()
    {

        var me = this;

        this.somefunction = function()
        {

        };

    }

    function settings()
    {

        this.container = function()
        {

        };

    }

    this.init = function()
    {


    };

    this.cosmos = function()
    {


    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test',
        utils = new utilities();

    this.settings = new settings();

}
```

Don’t write so many variables at the end of your extension. Organize your code so everything is categorized and where it belongs. For example, we all use a global variable id, which contains the id of the extension. From now on, you have two options: The first is to create a public subclass in which you will define the id. With this way , whoever is using your application will have access to this id. The second way is to create a private subclass in which you will access / set your id but only inside the scope of your extension.

First Option :

WRONG

```
function i_test()
{

    var self = this;

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test'; // <-- WRONG APPROACH

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function settings()
    {

        var __id = null;

        this.id = function()
        {
            // Put arguments with mode to check if you want
            // to set the id or to return it/
        };

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null;

    this.settings = new settings();

}
```

Second Option :

WRONG

```
function i_test()
{

    var self = this;

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        id = 'i_test'; // <-- WRONG APPROACH

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function config_model()
    {

        this.id = null;

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test = null,
        config = new config_model();

}
```

All the variables that are not global , you need to define them with __.

WRONG

```
function i_test()
{

    var self = this;

    function settings()
    {

        var id = null; // <-- WRONG APPROACH

        this.id = function()
        {
            // Put arguments with mode to check if you want
            // to set the id or to return it/
        };

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null;

    this.settings = new settings();

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function settings()
    {

        var __id = null; // <-- RIGHT APPROACH

        this.id = function()
        {
            // Put arguments with mode to check if you want
            // to set the id or to return it/
        };

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null;

    this.settings = new settings();

}
```

At the top of every public function , or function of a public subclass, you need to have a check to see if the extension was initialized. This is very important, because if it wasn't initialized then the whole extension will eventually broke. The only exception is at this.init and this.cosmos functions in which we mentioned before that they have different checks.

WRONG

```
function i_test()
{

    var self = this;

    this.start = function()
    {

    };

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null;

}
```

CORRECT

```
function i_test()
{

    var self = this;

    this.start = function()
    {

        // MUST in every public function
        if (is_init === false)
            return false;

    };

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null;

}
```

Break down your code as much as possible, so you don't write long functions.

WRONG

```
function i_test()
{

    var self = this;

    function utilities()
    {

        this.mail_accounts_add = function()
        {

        };

        this.mail_accounts_remove = function()
        {

        };

        this.mail_accounts_disable = function()
        {

        };

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null,
        utils = new utilities();

}
```

CORRECT

```
function i_test()
{

    var self = this;

    function utilities()
    {

        function mail()
        {

            function accounts()
            {

                this.add = function()
                {

                };

                this.remove = function()
                {

                };

                this.disable = function()
                {

                };

            }

            this.accounts = new accounts();

        }

        this.mail = new mail();

    }

    this.init = function()
    {

    };

    this.cosmos = function()
    {

    };

    var cosmos_exists = false,
        is_init = false,
        cosmos = null,
        .
        .
        .
        i_test_bee = null,
        utils = new utilities();

}
```