# Getting Started With REPL

## REPL

If we type **$node** it will start with **REPL** mode.

REPL – Read Eval Print Loop

Node will just print the result for each line we type with output, for statement it will print undefined.

**CTRL + L** clear the REPL session

For **multiple line** command we node can automatically identify when we start with **{.**

We explicitly can open multi line command by typing **.editor** and enter. To exit we can use **CTRL + D.**

To find other command type **.help**

### Tab and underscore

**TAB** can be used for auto complete and **double tab** will list all possible starting with the provided character.

**Underscore** behaves same as **$?** In shell scripting which stores the last executed value.

>Math.random()

0.99895632

>\_

0.99895632

We can use underscore variable at any place where we can use javascript expression.

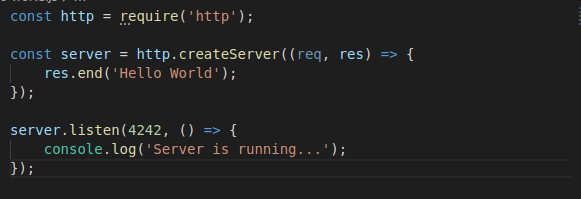
>const random = \_

### Executing script

From the location where the **.js** file is present we can execute the script as below.

$node hello-world.js

Or we can provide absolute path.



createServer() just create the server but doesn’t activate, for activating we need to use **listen()** function.

This script is continuous running and keeps the node REPL busy.

### Working with Timers

Timer functions are popular global functions and node js also has api around it.

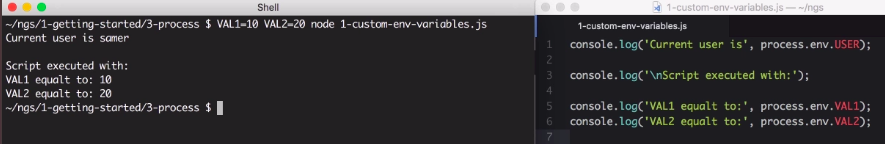
* setTimeout()
* setInterval()

These used to delay or repeat the execution. These method returns a **TimeOut** object which can be used for clearing the time out and intervals.

To clear the timeout or interval we have methods as clearTimeout(timeOut) and clearInterval(timeOut)

### The process object

We can use node command to set custom environment variable.



Here VAL 1 and VAL 2 are the environment variables will be set to **process** object.

**process.env** will hold all the environment variable of the system ex. **USER**

So we can export any variable (here VAL1 and VAL2) directly run the script it will set the variable to process.env.

$ export VAL1 = 150

$ export VAL2 = 200

$ node 1-custom-env-variables.js

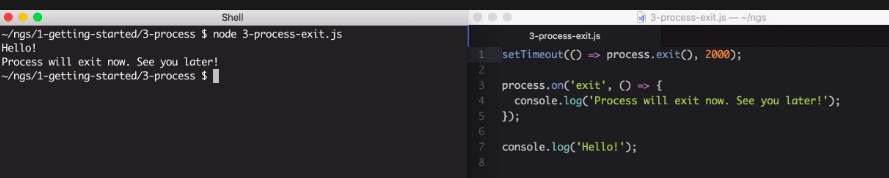
There is another way to pass information to nodes execution context and that’s through

**process.argv: [….]**

the “process” has **stdin** and **stdout** streams to read from input stream and write to output stream.

Ex.

const chunk = process.stdin.read();



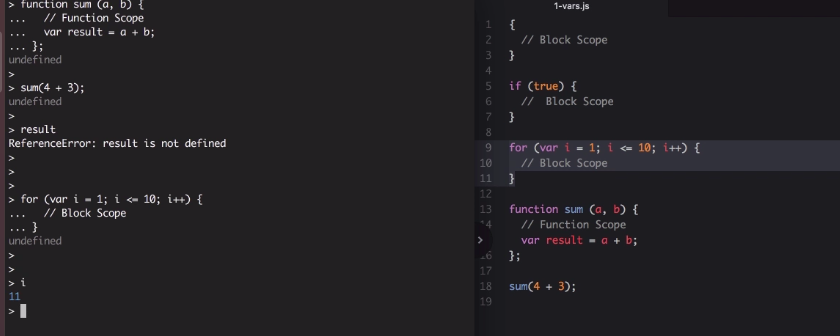
## Modern java script features

### Variable and Block scope

In javascript block scopes are different from function scope. Block scope defines the variable value live within **IF FOR WHILE** etc kind of areas.

Variable inside the **function scope** doesn’t leak out from that area, so we can’t access the variables declared inside function outside.

But the variables from **IF FOR** we can use outside of it.



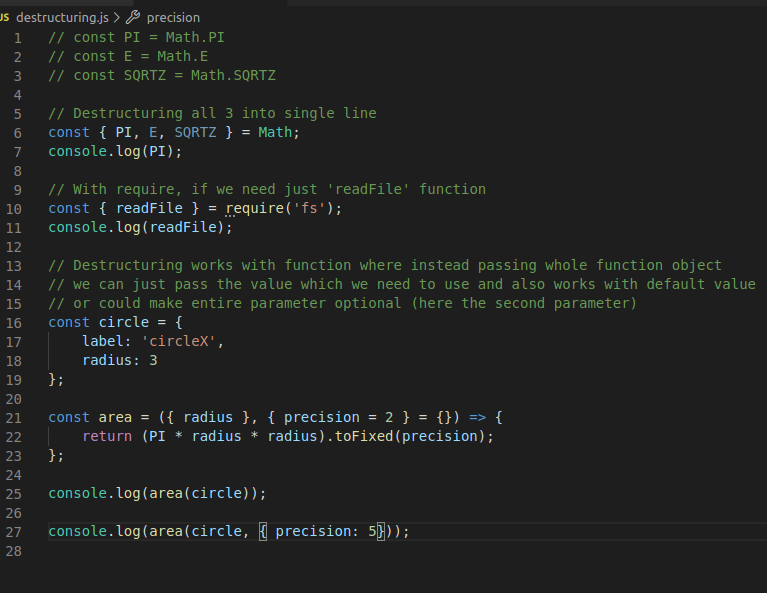
In the above example if we try to access **result** variable, it will print **undefined** which is not accessible.

But we can access the index variable of the **FOR** outside of the loop and it will print its value, which is weird.

So it is recommended to declare variable with **let** keyword or **const**, which doesn’t have this access scope problem.

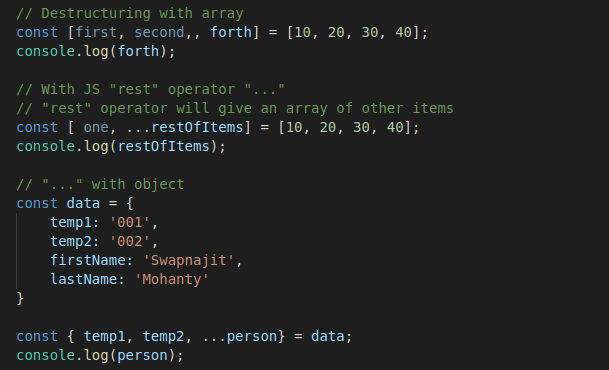
### De-structuring

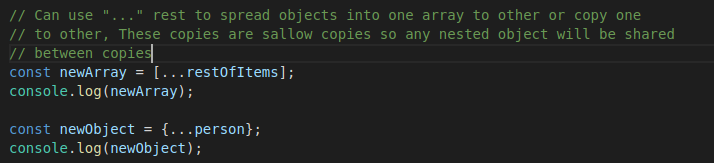
De-structuring works for both Array and Objects.



**Using with Array:**

Modern java-script has **rest operator** denoted with 3 dots . . . to bring rest of the items even copy array or objects. These copies are **sallow** copies, so any nested objects will be shared.

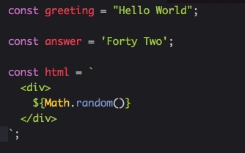




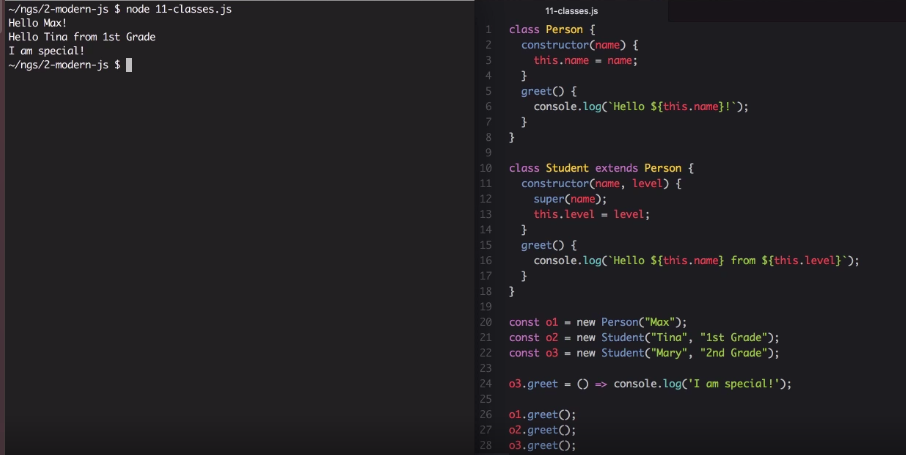
### Template string

Template string is an advanced way of defining string in javascript which supports dynamic value replacement through **interpolation.**

Template string defined by enclosing in **back-tick ~**

****

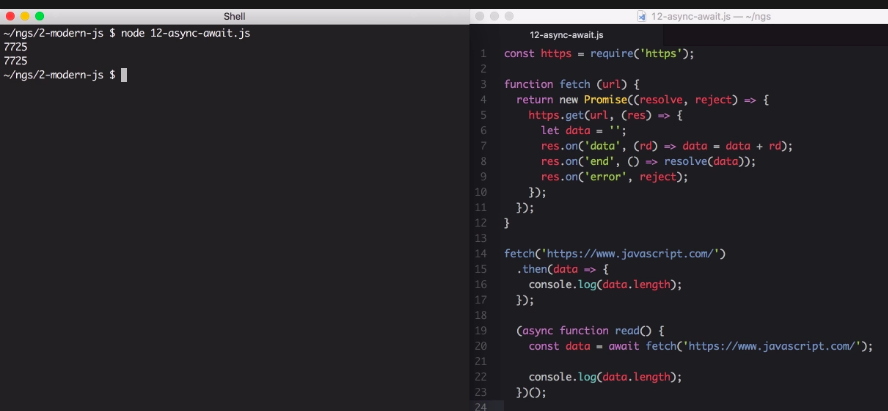
### Classes



### Promises and Async/Await

Node is event driven and most of the function return **Promise** and we have to consume them using **.then()** and **.catch()**

**promise** is an modern alternative to **(async function await …. )**



## NPM Node Package Manager

NPM stands for Node Package Manager. It used to do 3 major things

* Share the code with other developers
* Re-use own code in other projects
* Use code written by others

NPM is all about code sharing and reusability.

### NPM Commands

NPM comes in handy with Node. We may need to update NPM frequently than node it self and below is the command.

$ npm –v

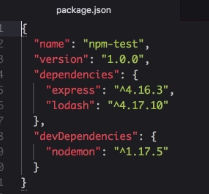
$ npm install –g npm

To download package we can use install, which will be placed under node\_module folder.

$ npm i express

To make a dependency only “development” dependency we can use –D flag.

$ npm i –D nodemon



### packge.json and package-lock.json

To create a package.json file from start we can use **init**

$ npm init

Can use --yes flag to set all default instead interactively answering all the questions.

**packge-lock.json** will get the exact same version even for the sub dependency tree. Let’s assume by the time I added **express** dependency and a time team member pulled my code to use a new version of some other dependent dependency got released our new team member will not get that new version but will get the exact same version which I use because of **package-lock.json**

In package-lock.json we will get the direct and even whole dependency tree for the project.

