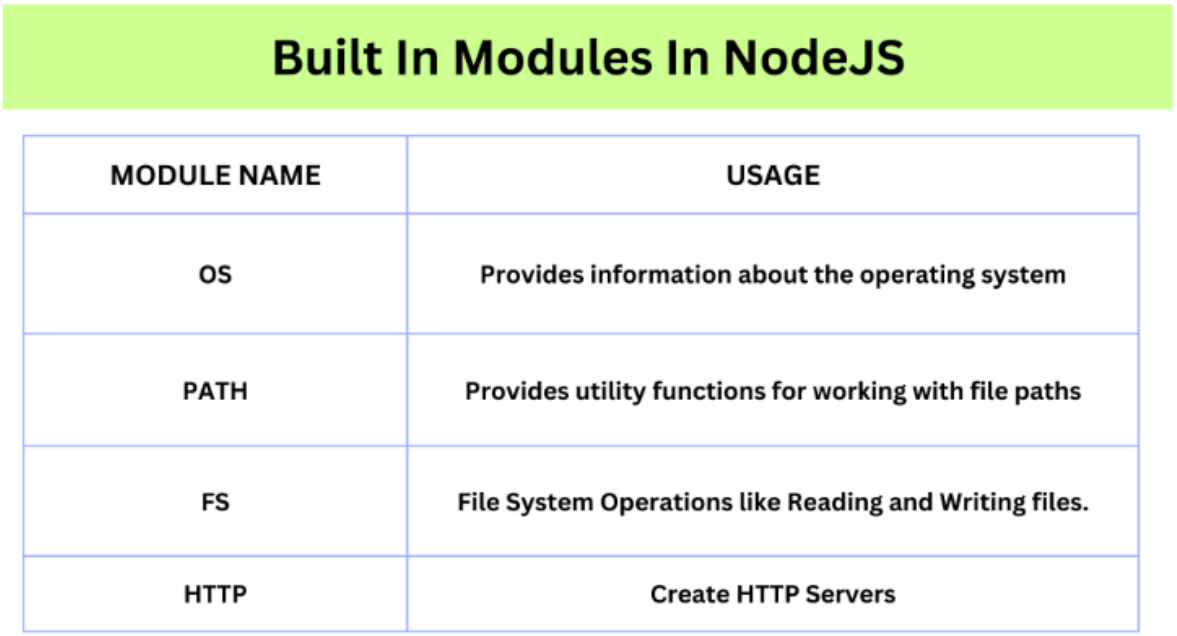
Here is an image of popular built-in modules in NodeJS and what can you do using them:



Built In Modules In NodeJS

MODULE NAME	USAGE
OS	Provides information about the operating system
PATH	Provides utility functions for working with file paths
FS	File System Operations like Reading and Writing files.
HTTP	Create HTTP Servers

Let's go over each of these in more detail so you can learn more about what they do.

The OS Module

The OS Module (as its name implies) provides you methods/functions with which you can get information about your Operating System.

To use this module, the first step is to import it like this:

```
const os = require('os');
```

This is how you can use the OS Module to get information about the Operating System:👇

```
const os = require('os')

// os.uptime()
const systemUptime = os.uptime();

// os.userInfo()
const userInfo = os.userInfo();

// We will store some other information about my WindowsOS in this object:
const otherInfo = {
    name: os.type(),
    release: os.release(),
    totalMem: os.totalmem(),
    freeMem: os.freemem(),
}

// Let's Check The Results:
console.log(systemUptime);
console.log(userInfo);
console.log(otherInfo);
```

This is the output of the above code:

Note that the output shows information about the Windows Operating System running on my system. The output could be different from yours.

```
521105
{
    uid: -1,
    gid: -1,
    username: 'krish',
    homedir: 'C:\\Users\\krish',
    shell: null
}
{
    name: 'Windows_NT',
    release: '10.0.22621',
    totalMem: 8215212032,
    freeMem: 1082208256
}
```

Let's break down the above code and output:

- `os.uptime()` tells the system uptime in seconds. This function returns the number of seconds the system has been running since it was last rebooted. If you check the first line of the output: `521105` is the number of seconds, my system has been running since it was last rebooted. Of course, it will be different for you.
- `os.userInfo()` gives the information about the current user. This function returns an object with information about the current user including the user ID, group ID, username, home directory, and default shell. Below is the breakdown of the output in my case:

```
{
    uid: -1,
    gid: -1,
    username: 'krish',
    homedir: 'C:\\Users\\krish',
    shell: null
}
```

The `uid` and `gid` is set to `-1` in Windows, because Windows does not have the concept of user IDs like Unix-based systems. The `username` of my OS is `krish` and the home directory is `'C:\\Users\\krish'`. The `shell` is set to `null` because the concept of a default shell does not exist on Windows. Windows has a default command interpreter program called Command Prompt (cmd.exe), which runs commands and manages the system.

The other methods related to OS Module like `os.type()`, `os.release()` and so on, which you saw in the above code has been used within the `otherInfo` object. Here is a breakdown of what these methods do:

- `os.type()` - Tells the name of the Operating System
- `os.release()` - Tells the release version of the Operating System
- `os.totalMem()` - Tells the total amount of memory available in bytes
- `os.freeMem()` - Tells the total amount of free memory available in bytes

This is the information which the above methods display about my OS:

```
{
    name: 'WindowsNT', // Name of my OS
    release: '10.0.22621', // Release Version of my OS
    totalMem: 8215212032, // Total Memory Available in bytes (~ 8 GB)
     freeMem: 1082208256 // Free Memory Available in bytes (~ 1 GB)
}
```

The PATH Module

The PATH module comes in handy while working with file and directory paths. It provides you with various methods with which you can:

- Join path segments together
- Tell if a path is absolute or not
- Get the last portion/segment of a path
- Get the file extension from a path, and much more!

You an see the PATH Module in action in the code below.

Code:

```
// Import 'path' module using the 'require()' method:
const path = require('path')

// Assigning a path to the myPath variable
const myPath = '/mnt/c/Desktop/NodeJSTut/app.js'

const pathInfo = {
    fileName: path.basename(myPath),
    folderName: path.dirname(myPath),
    fileExtension: path.extname(myPath),
    absoluteOrNot: path.isAbsolute(myPath),
    detailInfo: path.parse(myPath),
}

// Let's See The Results:
console.log(pathInfo);
```

Output:

```
{
  fileName: 'app.js',
  folderName: '/mnt/c/Desktop/NodeJSTut',
  fileExtension: '.js',
  absoluteOrNot: true,
  detailInfo: {
    root: '/',
    dir: '/mnt/c/Desktop/NodeJSTut',
    base: 'app.js',
    ext: '.js',
    name: 'app'
  }
}
```

Let's have a detailed breakdown of the above code and its output:

The first and foremost step to work with `path` module is to import it in the `app.js` file using the `require()` method.

Next, we are assigning a path of some file to a variable called `myPath`. This can be a path to any random file. For the purpose of understanding the `path` module, I chose this: `/mnt/c/Desktop/NodeJSTut/app.js`.

Using the `myPath` variable, we will understand the `path` module in detail. Let's check out the functions which this module has to offer and what can we do with it:

- `path.basename(myPath)`: The `basename()` function accepts a path and returns the last part of that path. In our case, the last part of `myPath` is: `app.js`.
- `path.dirname(myPath)`: The `dirname()` function selects the last part of the path provided to it and returns the path to it's parent's directory. In our case, since the last part of `myPath` is `app.js`. The `dirname()` function returns the path to the parent directory of `app.js` (the folder inside which `app.js` file lies), i.e, `/mnt/c/Desktop/NodeJSTut`. It can be also thought as: the `dirname()` function simply excludes the last part of the path provided to it and returns the leftover path.
- `path.extname(myPath)`: This function checks for any extension on the last part of the provided path and it returns the file extension (if it exists), otherwise it returns

an empty string: `''`. In our case, since the last part is `app.js` and a file extension exists, we get `'.js'` as the output.

- `path.isAbsolute(myPath)`: This tells whether the provided path is absolute or not. On Unix-based systems (such as macOS and Linux), an absolute path always starts with the forward slash (`/`). On Windows systems, an absolute path can start with a drive letter (such as `C:`) followed by a colon (`:`), or with two backslashes (`\\`). Since the value stored in `myPath` variable starts with `/`, therefore `isAbsolute()` returns `true`.

However, if you just change the `myPath` variable to this: `Desktop/NodeJSTut/app.js` (converting it to a relative path), `isAbsolute()` returns `false`.

- `path.parse(myPath)`: This function accepts a path and returns an object which contains a detailed breakdown of the path provided to it. Here is what it returns when we provide the `myPath` variable to it:
- `root`: The root of the path (in this case, `/`).
- `dir`: The directory of the file (in this case, `/mnt/c/Desktop/NodeJSTut`).
- `base`: The base file name (in this case, `app.js`).

- `ext`: The file extension (in this case, `.js`).
- `name`: The base name of the file, without the extension (in this case, `app`).

Before continuing with the other functions of the `path` module, we need to understand something called **path separator and the path structure**.

You must have seen that the path to a same file looks different in different Operating Systems. For example, consider the path to a file named `example.txt` located in a folder called `Documents` on the desktop of a Windows user:

```
C:\Users\username\Desktop\Documents\example.txt
```

On the other hand, the file path to the same file for a user on a macOS system would look like this:

```
/Users/username/Desktop/Documents/example.txt
```

2 differences are to be noted here:

1. **Difference in path separators:** In Windows, file paths use the backslash (`\`) as the separator between

directories, while in macOS/Linux (which is a Unix-based system), file paths use the forward slash (`/`) as the separator.

2. **Difference in root directory of the users files:** On Windows, the root directory for a user's files is commonly found at `C:\Users\username`, whereas on macOS and Linux, it is located at `/Users/username/`. While this holds true for most Windows, macOS, and Linux systems, there may be some variations in the exact location of the user's home directory based on the system's configuration.

With this in mind, let's move ahead and understand some other functions provided by the `path` module:

- `path.sep`: `sep` is a variable which contains the system specific path separator. For Windows machine: `console.log(path.sep)` prints `\` in the console while in case of macOS or Linux, `path.sep` returns a forward slash (`/`).
- `path.join(<paths>)`: The `path.join()` function accepts path(s) as strings. It then joins those paths using the system specific path separator and returns the joined path. For example, consider this code:

```
console.log(path.join('grandParentFolder', 'parentFolder', 'child.txt'))
```

The above code prints different results for different Operating Systems.

In Windows, it will give this output: `grandParentFolder\parentFolder\child.txt` while in macOS/Linux, it will give this output: `grandParentFolder/parentFolder/child.txt`. Note that the difference is only in the path separators - backward slash and forward slash.

- `path.resolve(<paths>)`: This function works in a similar way as compared to `path.join()`. The `path.resolve()` function just joins the different paths provided to it using the system specific path separator and then appends the final output to the absolute path of the present working directory.

Suppose you are a Windows user and the absolute path to your present working directory is this: `C:\Desktop\NodeJSTut`, If you run this code:

```
console.log(path.resolve('grandParentFolder', 'parentFolder', 'child.txt'));
```

You will see the following output in the console:

```
C:\Desktop\NodeJSTut\grandParentFolder\parentFolder\child.txt
```

The same is applicable to a macOS or a Linux user. It's just the difference in the absolute path of the present working directory and the path separator.