

# Core Modules & globals

Node built in core modules and global

# Our Goals

---

- ▶ **More than remembering the core modules API by heart it is more important to understand the patterns used in node modules**
- ▶ **Understanding the patterns will help us easily learn a new API**
- ▶ **Understanding the pattern will guide us to building our own modules**

# What are the core modules

---

- ▶ **Modules that are installed when you installed node**
- ▶ **Not to many of them**
- ▶ **to use them we don't need to do npm install we simply require them. for example:**
  - **`const fs = require('fs');`**
- ▶ **In this lesson we will go over the main core modules where our goal is to understand what each module is in charge of, and understand the patterns used**

# EventEmitter

---

- ▶ **Javascript is an event driven language, this means that our script finish running and we wait for events to happen**
- ▶ **Events in JS play a major rule in the patterns we will use when creating a node application**
- ▶ **We use EventEmitter to create our custom event**
- ▶ **the event is emitted and we can attach a listener to event we create**
- ▶ **when emitting the event we can send data to the listeners**
- ▶ **Let's examine some of the patterns we can use with the EventEmitter**

# EventEmitter - EX - Hello World

---

- ▶ **Let's start with a small ex that will help us learn the basics of the EventEmitter**
- ▶ **create an instance of the EventEmitter**
- ▶ **create an event called 'hello'**
- ▶ **attach a listener to that event**
- ▶ **after 1 second emit that event and send an hello world message to the listeners**
- ▶ **The listeners should print the message.**

# EventEmitter - Attach a listener

---

- ▶ **Attaching a listener is done with on**
- ▶ **you can also listen with once which will run the listener function once and then unsubscribe the listener**
- ▶ **If the EventEmitter is infinite and keeps emitting pulses you will have to remember to delete the listener with removeListener or off otherwise you might have a memory leak**

# EventEmitter - Inheritance - EX

---

- ▶ **It is a common pattern to inherit from the EventEmitter**
- ▶ **You will do it in cases where you are building a class that can emit event**
- ▶ **For example let's create a class called MyChat**
- ▶ **object of this class can send an event called 'chat' with a string message**
- ▶ **The class should keep the messages in an array.**
- ▶ **Attach a listener and print all the message in the listener**
- ▶ **Try to do it by overriding the emit function**

# EventEmitter - Error handling - EX

---

- ▶ **EventEmitter instance has a special event called 'error'**
- ▶ **When error happens in the EventEmitter instance we emit the error event with the Error instance**
- ▶ **If there is a listener on this event he will be called, if not it will raise an exception and the script will exit with error code.**
- ▶ **Create a method in the previous MyChat we created before that will emit the error**
- ▶ **Try to see what happens if you listen to the error event or not listen**



# global

---

- ▶ **The global namespace object**
- ▶ **global is an object containing the methods and properties that are global and available in every module**
  - **global.setTimeout === setTimeout**
  - **when accessing global method we are searching in the global scope**
- ▶ **EX:**
  - **Let's try and attach a global variable to the global scope**

# process

---

- ▶ **When running a js script with node we are actually running the script with a node process**
- ▶ **the process global provide information and control over the current node process**
- ▶ **the process extends EventEmitter and emits events regarding errors or warnings that are uncaught and bubble all the way up**
- ▶ **We can spawn another process from within a running script and we can communicate with messages between the process**
- ▶ **process.env will contain the environment variables**

# process - ex

---

- ▶ **Write a script that will print an environment variable called foo**
- ▶ **Try to run that script with environment variable set foo=bar**

# File System - EX

---

- ▶ **There is a builtin module called fs that contains methods to deal with the file system**
- ▶ **EX:**
  - **Create a text file containing hello world**
  - **Create a JS script that reads this file and prints the file content to the console**

# File System - Error first callback

---

- ▶ **The async pattern used in the file system module is a common pattern in async code with node, and understanding the pattern means easily understand how to use the majority of the async api's**
- ▶ **It is common for async methods to get a callback as the last argument**
- ▶ **This callback will be with Error instance (or a class that extends the Error) as first argument (this will be equal to null if there was no error)**
- ▶ **The rest of the arguments are the result of the async method**

# File System - Error first callback - EX

---

- ▶ **Try and read a file that does not exist**
- ▶ **Notice that the first argument of the error is filled with the error**
- ▶ **How can we pass the error to the outside? can we wrap it with a try and catch and simply throw the error?**

# File System - Student EX

---

- ▶ **Use the fs module to do the following**
  - **Create a file with an hello world message**
  - **Read from that file**
  - **Update that file**
  - **Read the update**
  - **Delete the file**

# Errors

---

- ▶ **Dealing with errors is an important part of writing code that is often neglected**
- ▶ **Let's go over the best practices when dealing with errors**
- ▶ **The first building block for dealing with errors is the base Error class**



# Errors - base Error

---

- ▶ **generic JavaScript Error class**
- ▶ **All throw errors will either be the Error class or a class that inherits from the Error class**
- ▶ **The base class contains the stack trace**
- ▶ **the class also contain a message property**
- ▶ **the class contains a code property**
- ▶ **Contains name property that can help you differentiate between different error types**
- ▶ **The Error constructor gets a string message describing the error**

# Errors - builtin Errors

---

- ▶ **RangeError**
- ▶ **ReferenceError**
- ▶ **SyntaxError**
- ▶ **TypeError**
- ▶ **EvalError**
- ▶ **URIError**

# Errors - custom Error - EX

---

- ▶ You can't always classify your Error to one of the builtins mentioned before
- ▶ You can customise your Errors by extending the Error base class
- ▶ Let's examine the AssertionError as an example of extending the base Error class
- ▶ You can provide your new Error class with the module you are building
- ▶ You can add additional information to your error in your custom error

# Errors - Error and Exceptions

---

- ▶ **An Error refers to an instance of the Error class or a class that inherits from the Error class**
- ▶ **An exception is when you throw an error (in JS you don't have to necessarily throw an Error object but we will do it anyway as a convention)**
- ▶ **When not catching a thrown exception the script will exit with an error code**
- ▶ **Let's cover the different ways to throw an exception and when they are common**

# Errors - throw exception

---

- ▶ You can use the throw keyword
  - used on sync code and on async functions
  - In this case you will have to catch it with try..catch clause
- ▶ You can use Promise reject
  - used when dealing with async code with promises
  - Unlike other exception throwing this will emit a warning and will not exit the script (in the future this will change to match other exception throwing)
- ▶ Error first callback
  - when dealing with async code that gets a callback
- ▶ EventEmitter
  - You can return an EventEmitter instance and use the error event
  - used when the result is more complex

# Errors - throw exception - EX

---

- ▶ Continuing the ex with reading a file that does not exist with the fs module let's enhance that exercise and try to deal with the error using the following ways
  - Promise
  - Error first callback
  - EventEmitter

# Nerdeez tip

---

- ▶ **The convention in nerdeez is to deal with the async code with promises**
- ▶ **this does not replace the EventEmitter which sometimes is necessary but it does replace the need for error first callback**
- ▶ **It creates a company convention so it is easier to understand other programmers api**
- ▶ **We can use async await and other promise tools**

# fs - promise

---

- ▶ **EX: Try and read a file this time try to wrap it with a promise**
- ▶ **There is an API for fs for reading a file and returning a promise**
  - **require('fs').promises**
  - **This API is still experimental**
  - **At some point probably most of node API will support promises**
- ▶ **use this api to read the file.**
- ▶ **There is a built in module that provides us with a method that will turn every async API with the pattern of error first callback to a promise**
  - **require('util').promisify**
- ▶ **using promisify turn the fs readFile error first to promise**



# path - EX

---

- ▶ **What happens if we try and read a file and we run the process from a different folder**
  - **Where does fs look for the file then?**
- ▶ **Using what variable can we use to specify the absolute location of the file?**
- ▶ **Can you think of a problem if we concatenate the path ourselves?**
- ▶ **Path is a module containing utility functions for manipulating directory path and file path.**
- ▶ **for example the method resolve can concatenate path together and returns us absolute path**
- ▶ **EX: Let's use resolve to fix our problem and supply the full path for readFile**

# Timers

---

- ▶ **node contains as globals methods to activate async code at a certain time in the future**
- ▶ **setTimeout**
- ▶ **clearTimeout**
- ▶ **setInterval**
- ▶ **clearInterval**
- ▶ **For interval it is important to clear otherwise you will have a leak.**

# Summary

---

- ▶ **Understand the patterns used and you will easily dive into every node API**
- ▶ **EventEmitter is an important pattern used to create our custom event based, and extending that class is important when we are creating API with events**
- ▶ **Error first callback is an important pattern that once you know it most of node api's are using it**
- ▶ **Know how to deal with errors even on async code, recommend to turn the async error first pattern to promises.**