1. RangeError

This is thrown when a number is outside an allowable range of values.

For example,

```
const l = console.log

const arr = [90,88]
arr.length=90**99
```

We have an array, arr with two elements. Next, we try to grow the array to contain `90**99 == 2.9512665430652753e+193` elements.

This number is way past the size arrays can be grown to. Running it will throw a RangeError:

```
$ node errors
errors.js:4
arr.length=90**99
 ^

RangeError: Invalid array length
```

because the number we want to increase the arr array to is out of the range specified by JS.

3. SyntaxError

This is the most common error we encounter. This error occurs when we type code that the JS engine can understand.

This error is caught by the JS engine during parsing. There are different stages in the JS engine our code is put through before we see those results on the terminal.

- tokenization
- parsing
- interpreting

tokenization breaks the source of the code into individual units. At this stage, numbers, keywords, literals, operators are sorted out and individually marked.

Next, the token stream generated will be passed to the parsing stage, which is handled by a parser. This is where an AST is generated from the token stream. AST is an abstract representation of the structure of our code.

During these two stages, tokenization and parsing, if the syntax/source of our codes doesn't conform to the syntax rules of JS makes the stages fail and throw SyntaxError. For example,

```
const l = console.log

let cat h = "cat"
```

What is with the lone "h"? The "h" being there breaks the code.

```
$ node errors
errors.js:3
let cat h = "cat"
 ^

SyntaxError: Unexpected identifier
```

See, Node.js points out the problem. It says that the "h" was unexpected. IT being there breaks the declaration of the cat variable.

So we can say syntax error occurs during parsing/compile time.

4. TypeError

TypeError is used to indicate an unsuccessful operation when none of the other NativeError objects are an appropriate indication of the failure cause.

TypeError occurs when an operation is performed on a wrong data type. Maybe a boolean is expected but a sting is found.

For example,

if we try to convert a number to uppercase like this:

```
const num = 123
num.toUpperCase()
```

This will throw a TypeError

```
$ node errors
errors.js:4
num.toUpperCase()
 ^


TypeError: num.toUpperCase is not a function
```

because the toUpperCase function expects a string data type. The toUpperCase function is intentionally generic; it does not require that its this value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

Only strings are converted to uppercase or lowercase if we call the toUpperCase function on Objects, Boolean, Symbol, null, undefined data types we will get the TypeError because it is the wrong data type it operates.

5. URIError

This indicates that one of the global URI handling functions was used in a way that is incompatible with its definition.

URI (Uniform Resource Indicator) in JS has the functions: decodeURI, decodeURIComponent, etc.

If we call any of them with the wrong parameter we will get a URIError

```
decodeURI("%")
^

URIError: URI malformed
```

decodeURI, gets the unencoded version of a URI. “%” is not the right URI, so a URIError was thrown.

URIError is thrown when there’s a problem encoding or decoding the URI.

6. EvalError

This is used to identify errors when using the global `eval()` function.

According to EcmaSpec 2018 edition:

This exception is not currently used within this specification. This object remains for compatibility with previous editions of this specification.

7. InternalError

This error occurs internally in the JS engine, especially when it has too much data to handle and the stack grows way over its critical limit.

This occurs when the JS engine is overwhelmed by too many recursions, too many switch cases, etc

```
switch(num) {
 case 1:
 ...
 break
 case 2:
 ...
 break
 case 3:
 ...
 break
 case 4:
 ...
 break
 case 5:
 ...
 break
 case 6:
 ...
 break
 case 7:
 ...
 break
 ... up to 1000 cases
 }
```

Too much recursion, a simple example is this:

```
function foo() {
 foo()
}
foo()
```

2. ReferenceError

This error is thrown when a reference made to a variable/item is broken. That is the variable/item doesn't exist.

For example,

```
const l=console.log

const cat = "cat"
cat
dog
```

We have a variable cat initialized to "cat". Next, we referred to the cat variable and dog variable. cat variable exists but dog variable doesn't.

cat will return "cat", while dog will throw a reference error because the name dog can't be found on the environment record.

```
$ node errors
errors.js:3
dog
^

ReferenceError: dog is not defined
```

Whenever we create or define a variable, the variable name will be written to an environment record. This environment record is like key-value storage,

```
+--------------+
| Key | Value |
---------------
| cat | "cat" |
+--------------+
```

that stores variables defined in our program, whenever we reference a variable. The environment record is searched with the name of the variable as key when it is found on the record the value is extracted and returned. calling a function that hasn't been defined.

Now, when we create or define a variable without assignment. The variable is written to the environment record, with the key as the variable name but the value will hold undefined.

```
var cat


env record
+-----------------+
| Key | Value |
-------------------
| cat | undefined |
+-----------------+
```

When the variable is later assigned a value, the variable is searched in the env record and when found the initial undefined valued is overwritten with the assigned value.

```
var cat
cat = "cat"


env record
+-------------+
| Key | Value |
---------------
| cat | "cat" |
+-------------+
```

So, a RefernceError is thrown by the JS engine when a variable name can't be found in the env record.

```
+-------------+
| Key | Value |
---------------
| cat | "cat" |
+-------------+


cat // "cat", yes, :) it's there
dog // :( what's this? can't find it
```

Note: an undefined variable won't throw ReferenceError because it exists in the env record just that its value hasn't been set.