Press Esc to exit full screen

SLIDE

# DEBUGGING & COMMON TYPES OF ERRORS

PROGRAMMING HERO

## What is debugging?

- Debugging is the process of **finding** and **fixing** errors or bugs in the source code
- computer programmers study the code to determine **why any errors occurred.**
- checks the code **step by step,** and **analyze** and **fix** the issue

## How does the debugging process work?

**Error identification**

- Developers locate the exact line of codes or code module causing the bug.

**Error analysis**

- Coders analyze the error by recording all program state changes and data values.

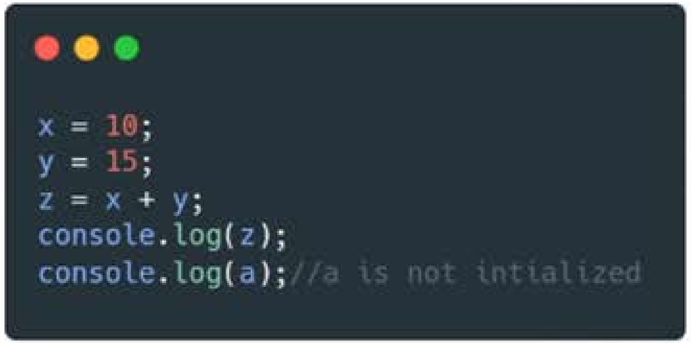
**Fix and validation**

- Developers fix the bug and run tests to ensure the software continues to work as expected.

## Some debugging techniques

- **Using console.log() method**

The **console.log()** method displays the result in the console of the browser. If there is any mistake in the code, it generates the error message.

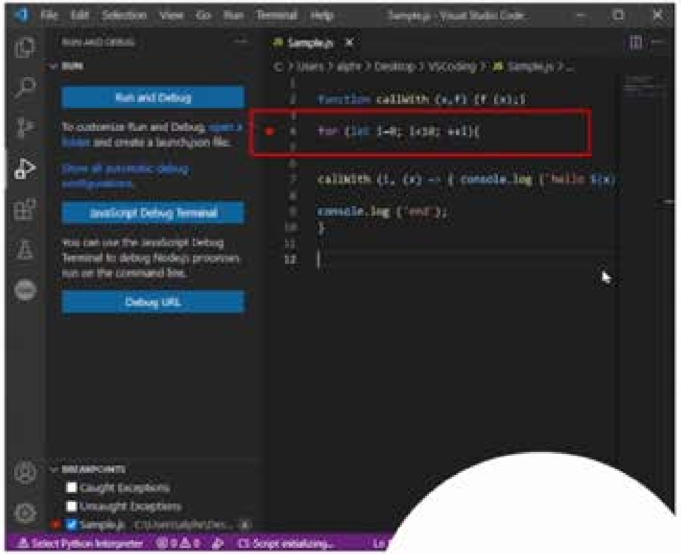


```
x = 10;
y = 15;
z = x + y;
console.log(z);
console.log(a);//a is not intialized
```

## Some debugging techniques

- **Setting the breakpoints**

  Marking some lines of code at different points of the program can create breakpoints. When the program reaches the marked point during execution, it pauses. This helps the developer to check for any available error and sort it out.



To customize Run and Debug, open a folder and create a launch.json file.

Show all automatic debug configurations.

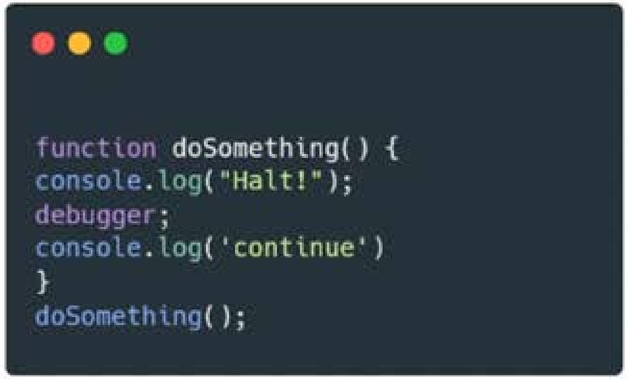
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.

```
function callWith (x,f) {f (x);}

for (let i=0; i<10; ++i){

callWith (i, (x) => { console.log (`hello ${x}

console.log ('end');
}
```

## Some debugging techniques

- **Using the debugger keyword**

  You can also use the JavaScript debugger keyword. This places a halt in execution where you place the debugging keyword. Here is an example:



```
function doSomething() {
console.log("Halt!");
debugger;
console.log('continue')
}
doSomething();
```

## Some debugging techniques

### Backtracking

- Backtracking uses different solutions until you find the right solution to the error.
- It is effective for debugging.
- You apply possible solutions to solve the problem in the JavaScript program.
- If it doesn't work you retreat and select another move.

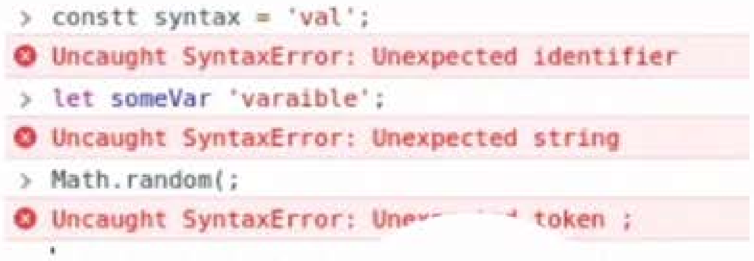
# COMMON TYPES OF ERRORS

PROGRAMMING HERO

## Syntax Error

**Syntax error** is triggered when you write code that is not syntactically correct.

- Missing inverted commas
- Missing closing parentheses
- Improper alignment of curly braces or other characters

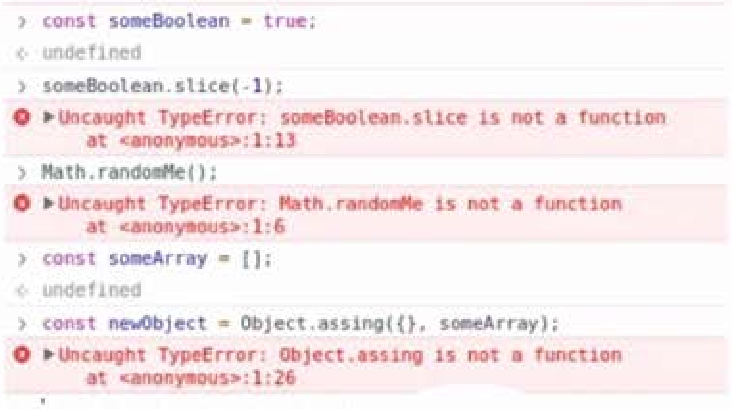


```
> constt syntax = 'val';
✖ Uncaught SyntaxError: Unexpected identifier
> let someVar 'varaible';
✖ Uncaught SyntaxError: Unexpected string
> Math.random(;
✖ Uncaught SyntaxError: Unex[illegible] token ;
```

## Type Error

**Type error** is created when some value doesn't turn out to be of a particular expected type.

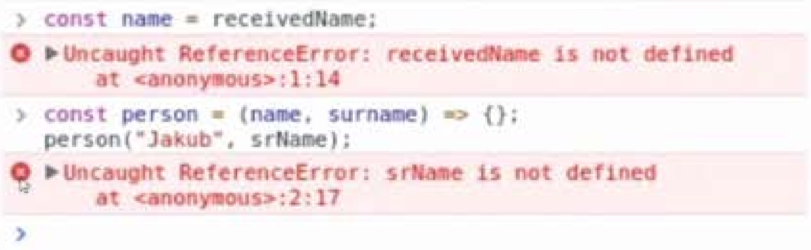
- Calling objects that are not methods.
- Attempting to access properties of null or undefined objects
- Treating a string as a number or vice versa



```
> const someBoolean = true;
< undefined
> someBoolean.slice(-1);
✖ ▶Uncaught TypeError: someBoolean.slice is not a function
      at <anonymous>:1:13
> Math.randomMe();
✖ ▶Uncaught TypeError: Math.randomMe is not a function
      at <anonymous>:1:6
> const someArray = [];
< undefined
> const newObject = Object.assing({}, someArray);
✖ ▶Uncaught TypeError: Object.assing is not a function
      at <anonymous>:1:26
```

## Reference Error

- forgotten to define a value for the variable before using it
- we might be trying to use an inaccessible variable in our code.
- Making a typo in a variable name.
- Trying to access block-scoped variables outside of their scopes.



```
> const name = receivedName;
⊗ ▶Uncaught ReferenceError: receivedName is not defined
      at <anonymous>:1:14
> const person = (name, surname) => {};
  person("Jakub", srName);
⊗ ▶Uncaught ReferenceError: srName is not defined
      at <anonymous>:2:17
>
```

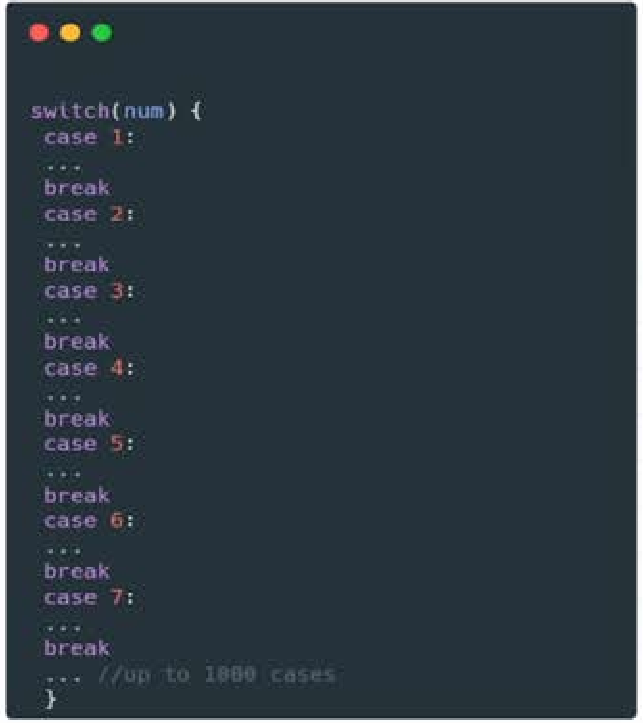
## Internal Error

The Internal Error occurs when an exception occurs in the JavaScript runtime engine. It may or may not indicate an issue with your code.

InternalError occurs in two scenarios only:

- When a patch or an update to the JavaScript runtime carries a bug that throws exceptions
- When your code contains entities that are too large for the JavaScrint engine (e.g. **too many switch cases, too large array initia[illegible] much recursion**)

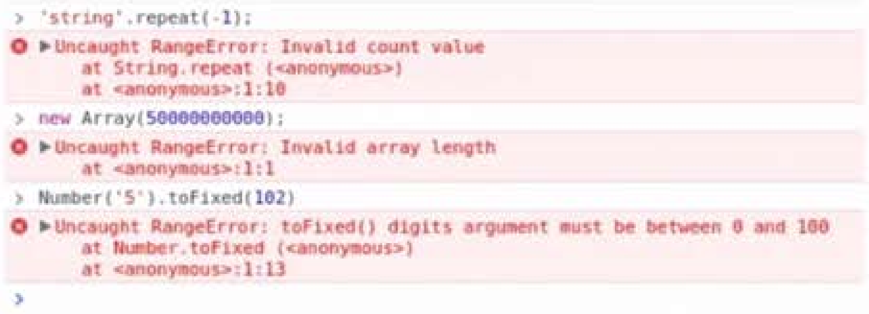
## Internal Error Example



```
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
 }
```

## Range Error

**RangeError** —thrown when a value is not in an allowed range



```
> 'string'.repeat(-1);
⊗ ▶Uncaught RangeError: Invalid count value
      at String.repeat (<anonymous>)
      at <anonymous>:1:10
> new Array(50000000000);
⊗ ▶Uncaught RangeError: Invalid array length
      at <anonymous>:1:1
> Number('5').toFixed(102)
⊗ ▶Uncaught RangeError: toFixed() digits argument must be between 0 and 100
      at Number.toFixed (<anonymous>)
      at <anonymous>:1:13
>
```