



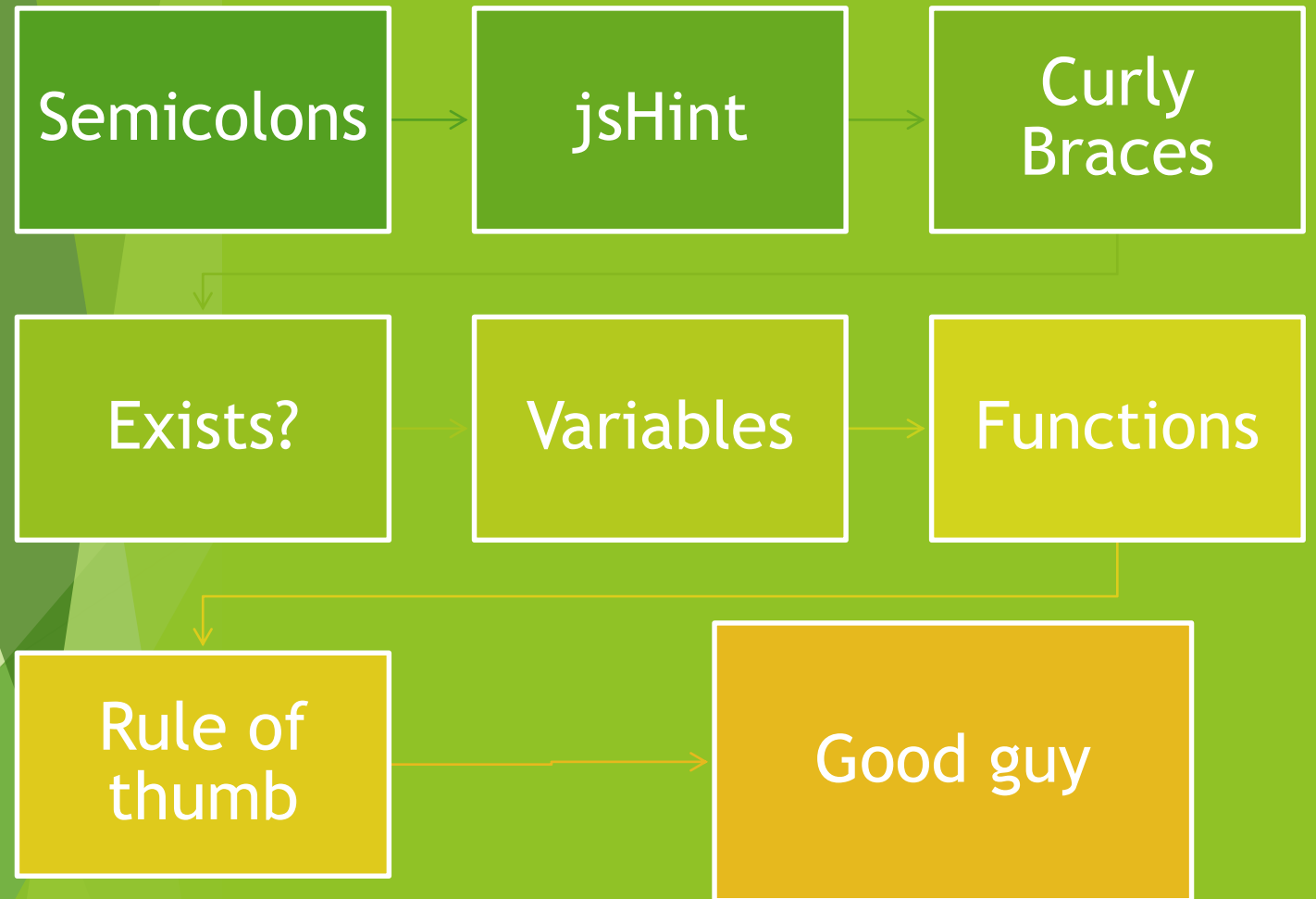
ACHIEVEMENT UNLOCKED
Reach 2 semester





Syntax

Syntax





Semicolons

“Semicolons
are optional”

– *the people*

“Certain ECMAScript statements (variable statement, expression statement, do-while statement, continue statement, break statement, return statement, and throw statement) ***must be terminated*** with semicolons. ”

– *the facts*

“For **convenience**, however, such semicolons may be **omitted** from the source text in certain situations.”

– *the facts*

RULES

1.

2.

3.



“When, as the program is parsed from **left to right**, a **token** (called the offending token) is encountered that is not allowed by any production of the **grammar**, then a semicolon is automatically **inserted** before the offending token”

– *the facts*

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern, layered effect on the right side of the slide.

```
var t = 1
```

```
var r = 4
```

```
if(true){console.log(t)}
```

“When, as the program is parsed from left to right, the **end of the input stream of tokens is encountered**, then a semicolon is automatically **inserted at the end of the input stream**”

– *the facts*


```
console.log(r)
```



```
var x =1;  
var y =5;  
var d = x + y  
[1,2,3].foreach(e => console.log(e))
```



```
var x =1;  
var y =5;  
var d = x + y  
(function(){  
console.log('call');  
})();
```

```
var x = [1,2,3]  
var t = x  
[1].toString();  
console.log(t);
```

Continue, Break, return ...

“When, a token is produced that is allowed by some production of the grammar, but the production is a **restricted production** and the token would be the first token of a restricted production and the restricted token is separated from the previous token by at least one **LineTerminator**, then a semicolon is automatically inserted before the restricted token”

– *the facts*


```
function semicolonTest()  
{  
    return  
    {  
        test: 1  
    }  
}  
  
console.log(semicolonTest());
```

```
function example()  
{  
  var get = function()  
  {  
    console.log('get');  
  }  
  
  return  
  {  
    get: get  
  }  
}
```

```
function example(){  
  var get = function(){  
    console.log('get');  
  };  
  
  return{  
    get: get  
  };  
}
```



jsHint

The background is a collage of images featuring illuminated arrows made of lights. On the left, there are several rows of arrows pointing right, composed of small, bright orange and yellow lights. The arrows are set against a dark background, possibly a night scene or a tunnel. On the right, there is a large, semi-transparent green geometric overlay consisting of several overlapping triangles and polygons. The word "Exists?" is written in a light green, sans-serif font, positioned in the center-right area of the image, partially overlapping the green overlay and the background images.

Exists?

```
var x;  
if(x){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```

```
var x=1;  
if(x){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```

```
var x=0;  
if(x){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```

```
if(x){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```

```
var x;  
if(typeof x !== 'undefined'){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```



```
if(typeof x !== 'undefined'){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```

```
var x;  
if(typeof x !== undefined){  
    console.log('X exists');  
}  
else{  
    console.log('X does not exists');  
}
```



Variables

“A **var** statement declares variables that are **scoped** to the running execution context's `VariableEnvironment`. Var variables are **created** when their containing Lexical Environment is instantiated and are initialized to **undefined** when created.”

– *the facts*

```
console.log(r);
```

```
console.log(r);  
var r;
```

```
console.log(r);  
var r=10;
```

```
var myVar = 10;
```

```
function myfun(){  
    myVar = 11;  
}
```

```
console.log(myVar);
```



```
var myVar = 10;
```

```
function myfun(){  
    myVar = 11;  
}
```

```
myfun();
```

```
console.log(myVar);
```

```
var myVar = 10;
```

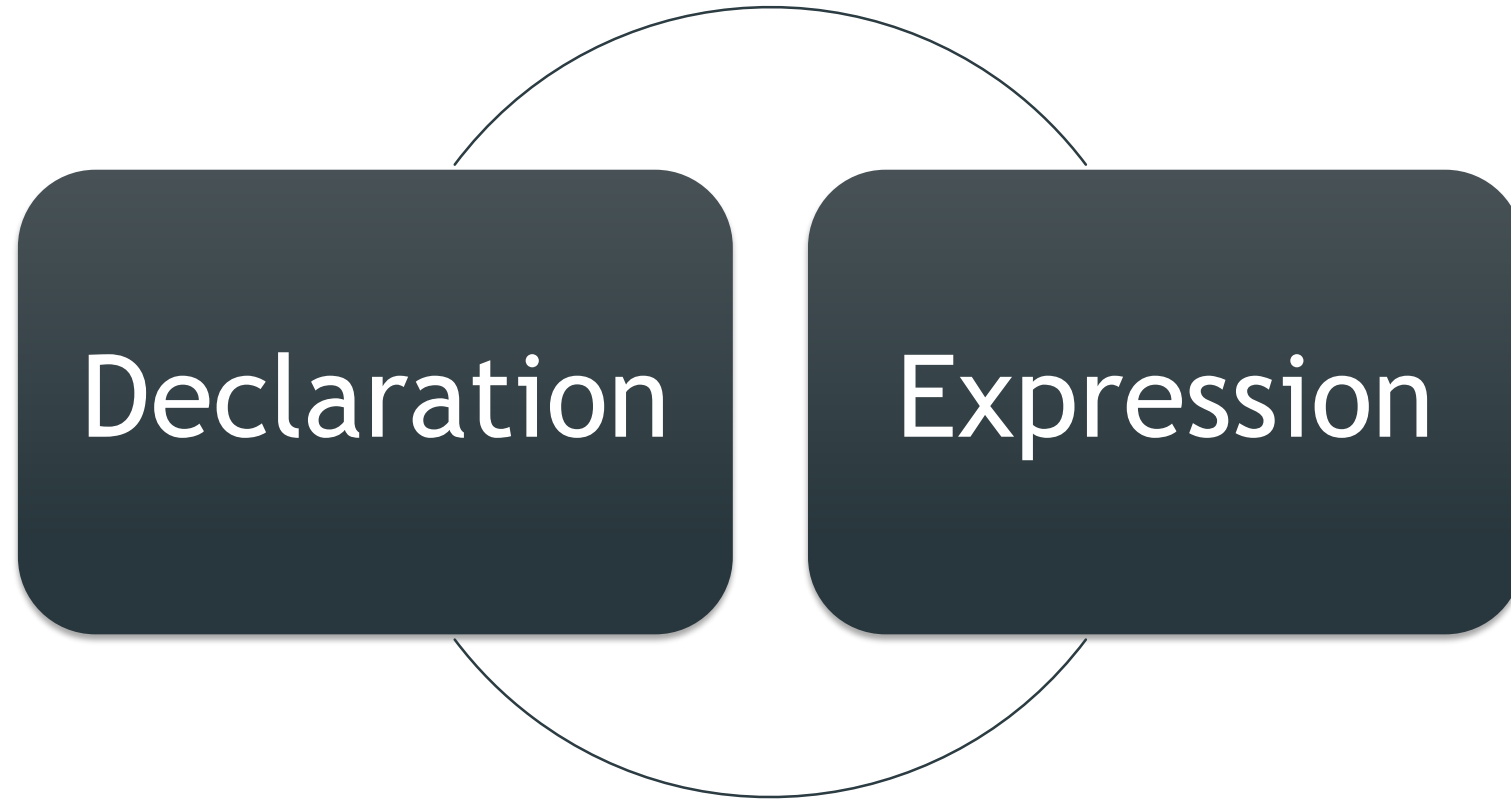
```
function myfun(){  
    myVar = 11;  
    var myVar;  
}
```

```
myfun();
```

```
console.log(myVar);
```



Functions



Functions

```
function declarationFunc(){  
    console.log('declarationFunc');  
}
```

```
declarationFunc();
```

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern, layered effect on the right side of the slide.

```
declarationFunc();
```

```
function declarationFunc(){  
    console.log('declarationFunc');  
}
```



```
var expresionFunc = function(){  
    console.log('expresionFunc');  
};
```

```
expresionFunc();
```

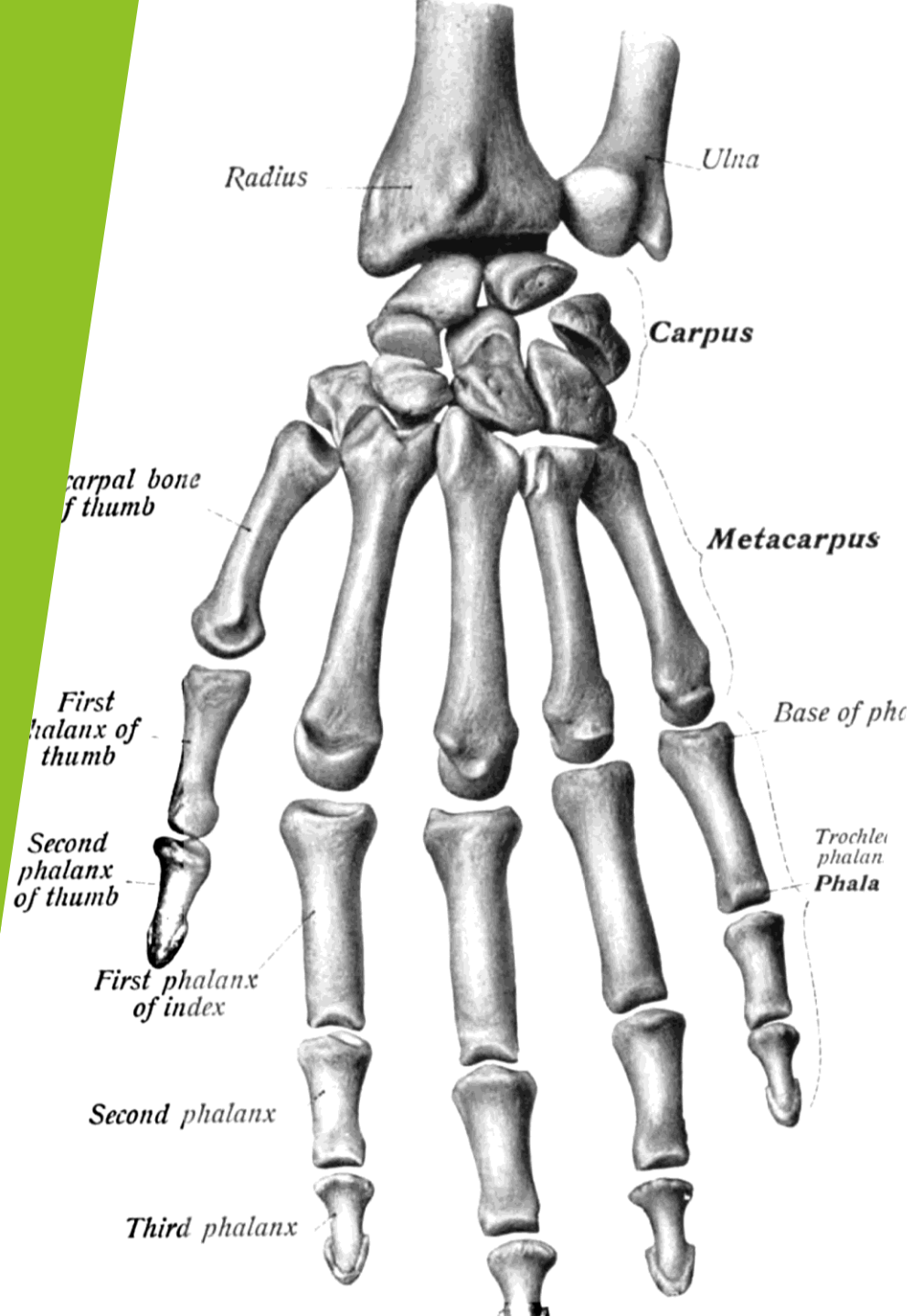
The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern, layered effect on the right side of the slide.

```
expresionFunc();
```

```
var expresionFunc = function(){  
    console.log('expresionFunc');  
};
```

Rule of Thumb

- Variables
- Functions
- Code



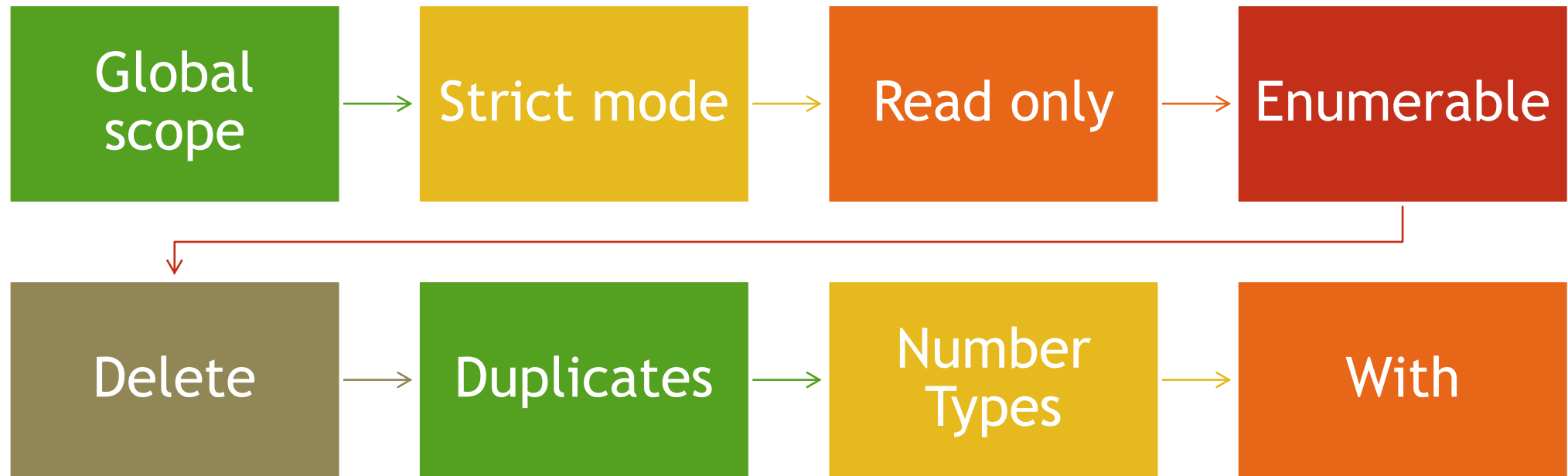


Good guy



Behaviors

Behaviors





Global scope

```
function show(param){  
    var innerParam = param;  
    console.log(param);  
}  
  
show('test');
```

```
var val1 = 'show' ;

function show(param){
    var innerParam = param;
    console.log(param);
    console.log(val1);
}

show('test');
```

```
function show(param){  
    var innerParam = param;  
    console.log(param);  
}
```

```
console.log(innerParam);
```

```
show('test');
```

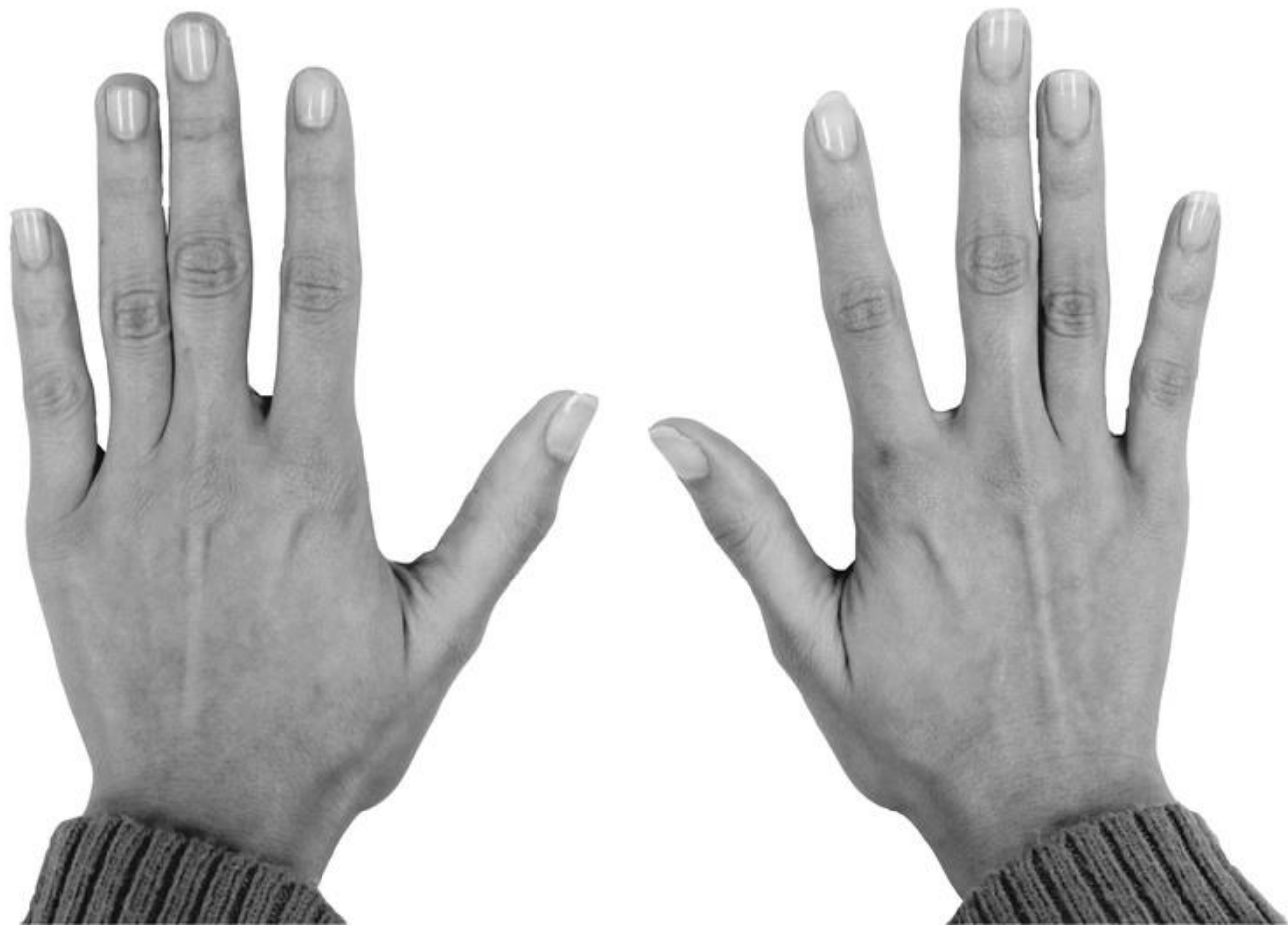
```
function show(param){  
    var innerParam = param;  
    console.log(param);  
}
```

```
show('test');
```

```
console.log(innerParam);
```

```
function show(param){  
    innerParam = param;  
    console.log(param);  
}  
  
console.log(innerParam);  
  
show('test');
```

```
function show(param){  
    innerParam = param;  
    console.log(param);  
}  
  
show('test');  
console.log(innerParam);
```



PLZ

Strict mode

JavaScript please just stop helping !!!

STOP

```
'use strict';
```

```
function show(param){  
    innerParam = param;  
    console.log(param);  
}
```

```
show('test');
```

```
console.log(innerParam);
```

```
function show(param){  
    'use strict';  
    innerParam = param;  
    console.log(param);  
}  
  
show('test');  
  
console.log(innerParam);
```

```
function show(param){  
    'use strict';  
    var innerParam = param;  
    console.log(param);  
}
```

```
show('test');
```

```
notCreatedVariable = 5;
```

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern, layered effect on the right side of the slide.

```
'use strict';
```

```
notCreatedVariable = 5;
```

The background features abstract, overlapping green geometric shapes in various shades, primarily on the right side of the slide.

```
'use strict';
```

```
var obj = {};
```

```
obj.a = 'sdfs';
```

```
console.log(obj);
```




Read only

```
var obj = {};  
  
Object.defineProperty(obj, 'ro',{  
  enumerable: true,  
  configurable: true,  
  writable: false,  
  value: 'Original Value'  
});  
  
console.log(obj.ro);
```

```
var obj = {};
```

```
Object.defineProperty(obj, 'ro', {  
  enumerable: true,  
  configurable: true,  
  writable: false,  
  value: 'Original Value'  
});
```

```
obj.ro = 'Altered Value';
```

```
console.log(obj.ro);
```

```
'use strict';
```

```
var obj = {};
```

```
Object.defineProperty(obj, 'ro', {  
  enumerable: false,  
  configurable: false,  
  writable: false,  
  value: 'Original Value'  
});
```

```
obj.ro = 'Altered Value';
```

```
console.log(obj.ro);
```



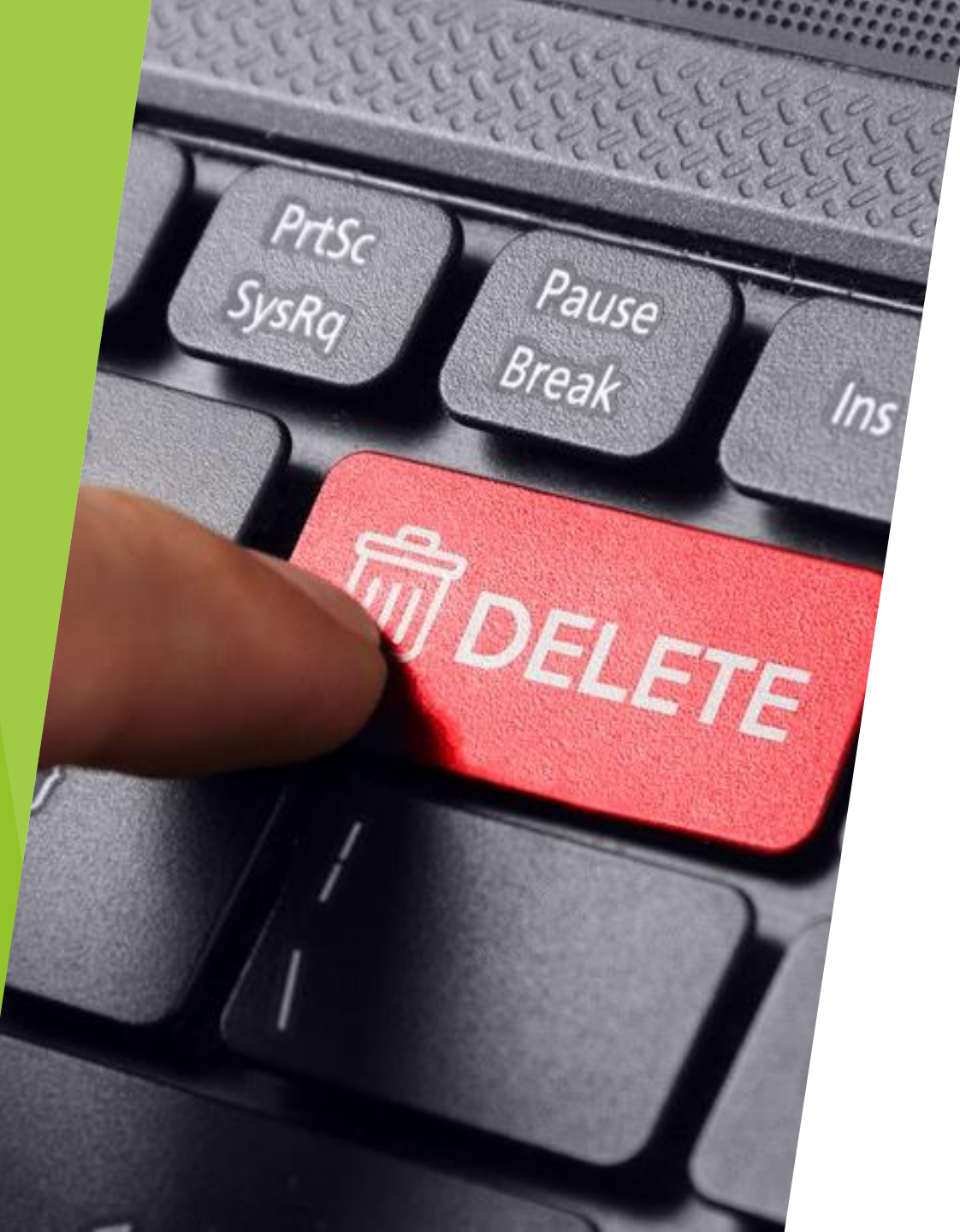
Enumerable

```
var obj = {  
    c : 'C Value'  
};  
  
obj.a = 'A Value';  
  
Object.defineProperty(obj, 'b',{  
    enumerable: true,  
    configurable: true,  
    writable: true,  
    value: 'B Value'  
});  
  
console.log(obj);  
for (var key in obj) {  
    console.log(key);  
}
```

```
var obj = {  
    c : 'C Value'  
};  
  
obj.a = 'A Value';  
  
Object.defineProperty(obj, 'b',{  
    enumerable: false,  
    configurable: true,  
    writable: true,  
    value: 'B Value'  
});  
  
console.log(obj);  
for (var key in obj) {  
    console.log(key);  
}  
  
console.log(obj.b);
```



```
var obj = {  
  c : 'C Value'  
};  
  
obj.a = 'A Value';  
  
    Object.defineProperty(obj, 'a',{  
      enumerable: false  
    });  
  
Object.defineProperty(obj, 'b',{  
  enumerable: false,  
  configurable: true,  
  writable: true,  
  value: 'B Value'  
});  
  
console.log(obj);  
for (var key in obj) {  
  console.log(key);  
}  
  
console.log(obj.a);  
console.log(obj.b);
```



Delete

```
var obj = {  
  a: 'A',  
  b: 'B'  
};
```

```
console.log(obj);
```

```
var obj = {  
  a: 'A',  
  b: 'B'  
};
```

```
delete obj.b;
```

```
console.log(obj);
```

```
var x = 6;
```

```
delete x;
```

```
console.log(x);
```

```
var obj = {  
  a: 'A',  
  b: 'B'  
};
```

```
delete obj;
```

```
console.log(obj);
```

```
'use strict';
```

```
var x = 6;
```

```
delete x;
```

```
console.log(x);
```



```
'use strict';
```

```
var obj = {  
  a: 'A',  
  b: 'B'  
};
```

```
delete obj;
```

```
console.log(obj);
```



Duplicates

```
function test(x,y,x){  
  console.log(x);  
}
```

```
test('a','b','c');
```

```
'use strict';
```

```
function test(x,y,x){  
    console.log(x);  
}
```

```
test('a','b','c');
```

The background features a dense, overlapping pattern of 3D numbers in shades of blue and white. A large, semi-transparent white triangle is positioned in the center, containing the text. The entire composition is framed by vibrant green geometric shapes, including triangles and polygons, which create a modern, abstract aesthetic.

Number Types

```
var x = 120,  
y = 012;
```

```
console.log(x+y);
```

```
var x = 120,  
y = 0x12;
```

```
console.log(x+y);
```

```
'use strict';
```

```
var x = 120,  
y = 012;
```

```
console.log(x+y);
```



```
'use strict';
```

```
var x = 120,  
y = 0x12;
```

```
console.log(x+y);
```

```
'use strict';
```

```
var x = 120,  
y = parseInt(12,8);
```

```
console.log(x+y);
```



with

```
var obj = {  
  a:{  
    b: 'B'  
  }  
};
```

```
console.log(obj.a.b);
```

```
var obj = {  
  a:{  
    b: 'B'  
  }  
};
```

```
with(obj.a){  
  console.log(b);  
}
```

```
var obj = {  
  a:{  
    b: 'B'  
  }  
};
```

```
var b = 'C';
```

```
with(obj.a){  
  console.log(b);  
}
```

```
'use strict';
```

```
var obj = {  
  a: {  
    b: 'B'  
  }  
};
```

```
var b = 'C';
```

```
with(obj.a){  
  console.log(b);  
}
```

```
'use strict';
```

```
var obj = {  
  a: {  
    b: 'B'  
  }  
};
```

```
var b = 'C';
```

```
(function(tempVar){  
  console.log(tempVar);  
})(obj.a.b));
```