

Programming fundamentals in JS

Day - 2

GIRLSCRIPT EDUCATION OUTREACH PROGRAM



Variables



So, what are variables?

Variables are used to store

values (name = "John") or

expressions (sum = $x + y$).



**Declare
Variables!
Not War!**




Types of Variables in JS

- undefined
- null
- boolean
- number
- string
- object

All other types of values are objects





```
1 let p = undefined;
2 let q = null;
3 let x = true;
4 let y = 10;
5 let z = "Hello"
6 let obj = {name: "Rick", lastName: "Sanchez", age: 80,
  isDead: false};
```



null vs undefined

```
1 myVar; // ReferenceError: myVar is not defined
2
3 var myVar;
4 myVar; //undefined
5
6 myVar = 'mine!';
7 myVar; // 'mine!'
8
9 myVar = null;
10 myVar; // null
```



Objects

1. Property Access
2. Bracket Notation
3. Dot Notation
4. Dot vs Bracket
5. Nested Objects
6. Object Literals
7. Iteration

0

They are mutable keyed collections.

Arrays, functions, regular expressions, and of course, objects are objects.

Objects can contain other objects.



Assignment 0

```
1 var newObject = new Object();
2 var anotherObject = Object.create(null);
3
4 var empty_object = {};
5
6 var flight = {
7     airline: "Oceanic",
8     number: 815,
9     departure: {
10         IATA: "SYD", time: "2004-09-22 14:55", city:
11         "Sydney"
12     },
13     arrival: {
14         IATA: "LAX",
15         time: "2004-09-23 10:42",
16         city: "Los Angeles"
17     }
18 };
```



Assignment

with dots

```
1 var box = {};  
2  
3 box.material = "cardboard";
```

with brackets

```
1 var box = {};  
2  
3 box["material"] = "cardboard";
```



Access

with

```
1 var box = {};  
2  
3 box.material = "cardboard";  
4  
5 let cb = box.material; //cardboard
```

with

```
1 var box = {};  
2  
3 box["material"] = "cardboard";  
4  
5 let cb = box["material"]; //cardboard
```



3



Question?

What is the
output?

```
1 var box = {};  
2  
3 box["material"] = "cardboard";  
4  
5  
6 var key = "material";  
7  
8 box[key]; //What will it return?
```



Delete

It will remove a property from the object if it has one.

```
1 var obj = {name: "myName"};  
2 obj.name; // "myName"  
3 obj.name = null;  
4 obj.name // null  
5 delete obj.name;  
6 obj.name; // undefined
```



Reference

Objects are passed around by reference. They are never copied.

```
1 var stooge = {nickname: "Moe"}
2 var x = stooge;
3 x.nickname = 'Curly';
4 stooge.nickname;
5 // 'Curly' because x and stooge are references to the same object
```



Do's && Don'ts

```
1 var box = {};  
2  
3  
4 box['material'] = "cardboard";  
5  
6 var key = 'material';  
7  
8  
9 box['key']; //undefined  
10 box.key; //undefined  
11 box[key]; //"cardboard"
```



The Rules

Dots

- strings
- ~~numbers~~
- ~~quotations~~
- ~~weird characters~~
- ~~expressions~~


Brackets

- strings
- numbers
- variables
- weird characters
- expressions




Iteration





```
1 var box = {};  
2  
3 box['material'] = "cardboard";  
4 box[0] = 'meow';  
5 box['^&*'] = "testing 123";  
6  
7  
8 for(var key in box){  
9   console.log(key); //??  
10 }
```





```
1 var box = {};  
2  
3 box['material'] = "cardboard";  
4 box[0] = 'meow';  
5 box['^&*'] = "testing 123";  
6  
7  
8 for(var key in box){  
9   console.log(box[key]); ???  
10 }
```



Arrays

- Arrays vs Objects
- Access & Assignment
- Native Methods & Properties
- Iteration



Arrays vs Objects



Access and Assignment

Array

```
1 var box = [];  
2  
3 box[0] = true;  
4 box[1] = 'meow';  
5 box.push({'hello' : 'goodbye'});  
6  
7  
8 var i = 0;  
9  
10 box[i]; // Returns the data stored at  
    0th index which is -> true  
11 box[1];  
12 box.pop() // Removes the last element  
    in the array
```

Object

```
1 var box = {};  
2  
3 box['size'] = 9;  
4  
5 box['0'] = 'meow';  
6  
7  
8 box['size']; // 9  
9  
10 box[0]; // 'meow'
```



The Rules Don't Change



Iteration(for loop)

Indexes

```
1 var box = [];  
2  
3 box['size'] = 9;  
4 box['0'] = 'meow';  
5 box.push('Whooohoo!');  
6  
7 for(var i = 0; i < box.length; i++){  
8   console.log(i); //??  
9 }
```

Values

```
1 var box = [];  
2  
3 box['size'] = 9;  
4 box['0'] = 'meow';  
5 box.push('Whooohoo!');  
6  
7  
8 for(var i = 0; i < box.length; i++){  
9   console.log(box[i]); //??  
10 }
```



Iteration(for each loop)

Indexes

Values

```
1 var box = [];  
2  
3 box['size'] = 9;  
4  
5 box['0'] = 'meow';  
6  
7 for(var k in box){  
8  
9   console.log(k); // ??  
10  
11 }
```

```
1 var box = [];  
2  
3 box['size'] = 9;  
4  
5 box['0'] = 'meow';  
6  
7  
8 for(var k in box){  
9  
10   console.log(box.k); // ??  
11  
12 }
```



Native Properties



```
1 var box = [];  
2  
3 box['size'] = true;  
4  
5 box['0'] = 'meow';  
6  
7  
8  
9  
10 box.length; //??
```

```
1 var box = [];  
2  
3 box['0'] = 'meow';  
4 box[3] = {'babyBox': true};  
5  
6  
7  
8  
9 box['length']; //??
```



3



```
1 var box = [];  
2  
3 box['0'] = 'meow';  
4 box[1] = 'Whooheooo!';  
5 box[3] = {'babyBox': true};  
6  
7  
8  
9 box[box.length]; //??
```

```
1 var box = [];  
2  
3 box['0'] = 'meow';  
4 box[1] = 'Whooheooo!';  
5  
6  
7  
8 box[box.length - 1]; //??
```



Nesting




first initialize

```
1 var box = {};  
2  
3 box['innerBox'] = {};
```

then assign


```
1 var box = {};  
2  
3 box['innerBox'] = {};  
4  
5 box['innerBox']['full'] = true;
```





```
1 var box = {};  
2  
3 box.innerBox = {};  
4  
5 box.innerBox.full = true;  
6  
7  
8 var myInnerBox = box.innerBox;  
9  
10 myInnerBox; //??
```





```
1 var box = {};  
2  
3 box.innerBox = {};  
4  
5 box.innerBox.babyBox = {};  
6  
7  
8 box.innerBox['babyBox']; //??  
9  
10 box.innerBox['babyBox'].says = "What's up!?";
```

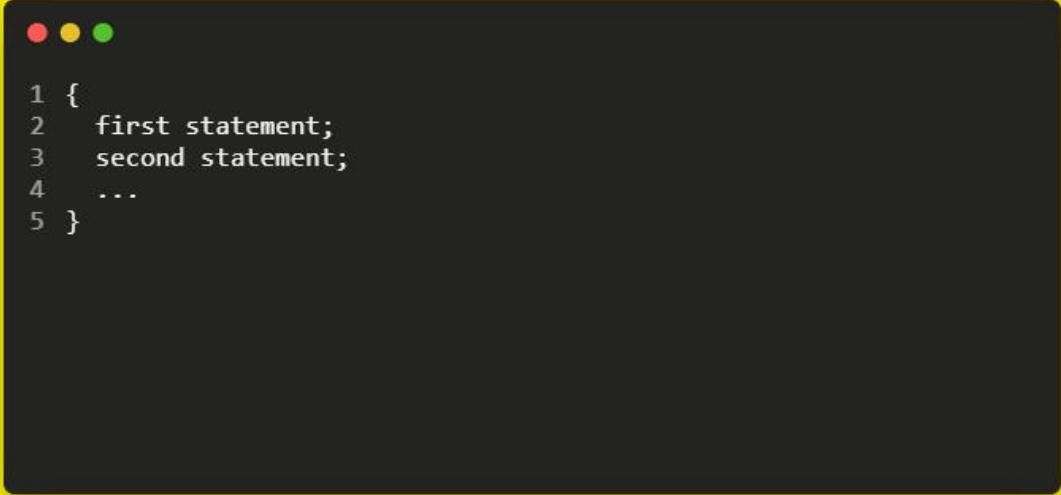


Control Flow



Block

The block statement per se doesn't change the control flow but is used to group statements. The block is set by a pair of curly brackets.



```
1 {  
2   first statement;  
3   second statement;  
4   ...  
5 }
```



Break

Break as the name implies, breaks. Breaks what? Breaks the statement or in normally the cases break the loop.

```
1 for (let i = 0; i < 5; i++) {  
2   if (i === 3) {  
3     break;  
4   }  
5   console.log(i);  
6 }  
7 //excepted output : 0,1,2
```

0



Output?

```
1 for (let i = 0; i < 10; i++) {  
2   if(i === 5){  
3     continue;  
4   }  
5   console.log(i);  
6 }
```



Continue

So if break, ends the loop what the continue statement does? That's right, it jumps the loop in that iteration and continues to the next iteration.

In other words, when the continue condition is met, it will not run or print it and it goes straight to the next iteration.

0

```
1 for (let i = 0; i < 10; i++) {  
2   if(i === 5) {  
3     continue;  
4   }  
5   console.log(i);  
6 }  
7 /*excepted output:  
8 0  
9 1  
10 2  
11 3  
12 4  
13 6  
14 7  
15 8  
16 9*/
```



Output?

```
1 var music = [  
2   "placebo",  
3   "smashing Pumpkins",  
4   "pearl jam",  
5   "ornatos violeta",  
6   "feromona"  
7 ];  
8 for (var i = 0; i < music.length; i++) {  
9   if (music[i] === "ornatos violeta") {  
10    continue;  
11   }  
12   console.log(music[i]);  
13 }
```



if-else

```
1 var music = [  
2   "placebo",  
3   "smashing Pumpkins",  
4   "pearl jam",  
5   "ornatos violeta",  
6   "feromona"  
7 ];  
8 if (music.includes("placebo")) {  
9   console.log(true);  
10 } else if (music[0] === "feromona") {  
11   console.log(false);  
12 } else {  
13   console.log(okay);  
14 }
```



switch - case

So a switch statement is used when you have to use many if statements in the same function for example.

```
1 let name = "professor"
2 switch (name){
3   case "professor":
4     console.log("you teach well");
5     break;
6   case "artist":
7     console.log("I like your painting");
8     break;
9   case "singer":
10    console.log("I love your voice");
11    break;
12  default:
13    console.log("what you do?");
14 }
```

0



Loops

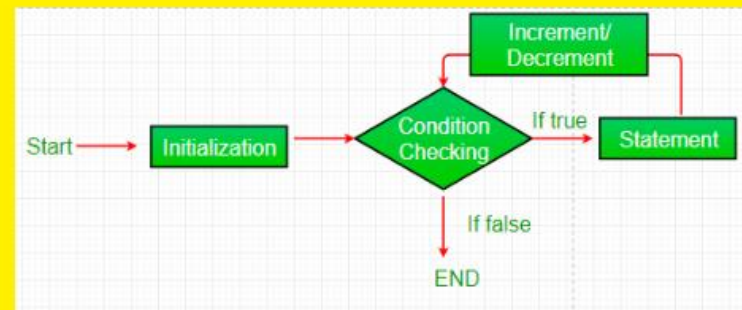



Iterative Method



for loop

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.






```
1 for (i = 0; i < 10; i++) {  
2   console.log(i);  
3 }
```



for-in loop

JavaScript also includes another version of for loop also known as the for..in Loops. The for..in loop provides a simpler way to iterate through the properties of an object. This will be more clear after leaning objects in JavaScript. But this loop is seen to be very useful while working with objects.



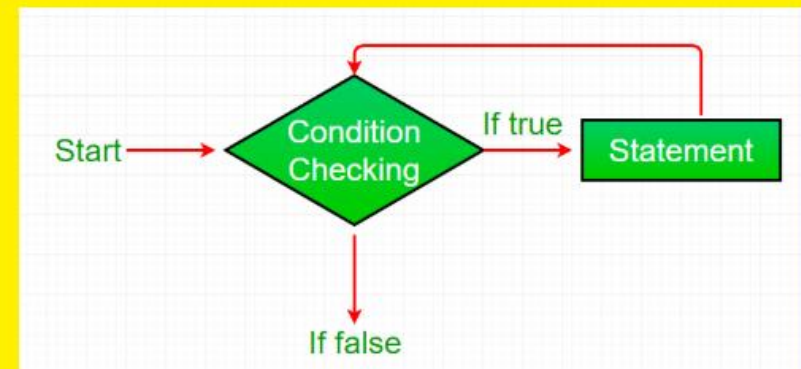



```
1 var box = {};  
2  
3 box['material'] = "cardboard";  
4 box[0] = 'meow';  
5 box['^&*'] = "testing 123";  
6  
7  
8 for(var key in box){  
9   console.log(key); //??  
10 }
```



while loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.



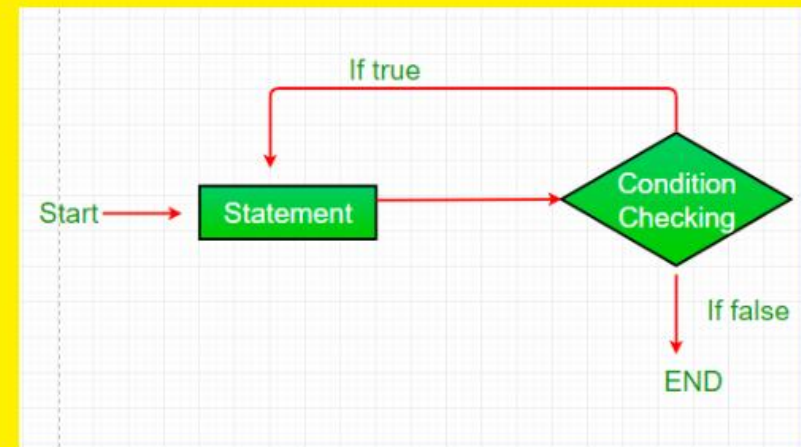



```
1 var x = 1;  
2  
3  
4 while (x <= 4) {  
5   console.log(x);  
6   x++;  
7 }
```



do-while loop

do-while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.





```
1 var x = 1;  
2  
3  
4 do {  
5   console.log(x);  
6   x++;  
7 } while (x <= 4);
```



Infinite loop

```
1 for (var i = 5; i != 0; i -= 2) {  
2   console.log(i);  
3 }
```



Error handling



try-catch

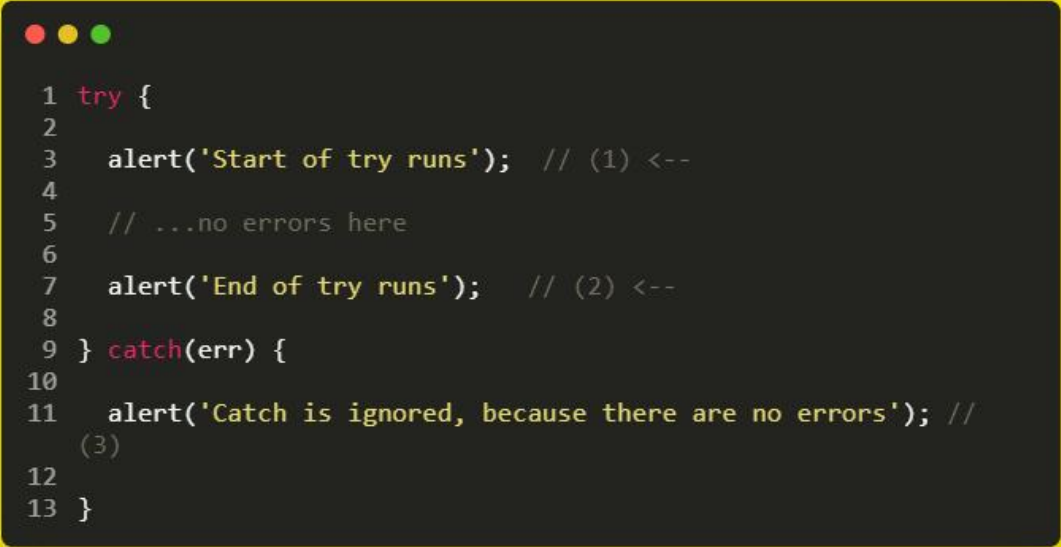
The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements try and catch come in pairs

```
1 try {  
2   Block of code to try  
3 }  
4 catch(err) {  
5   Block of code to handle errors  
6 }
```






```
1 try {  
2  
3   alert('Start of try runs'); // (1) <--  
4  
5   // ...no errors here  
6  
7   alert('End of try runs'); // (2) <--  
8  
9 } catch(err) {  
10  
11   alert('Catch is ignored, because there are no errors'); //  
    (3)  
12  
13 }
```





```
1 try {  
2  
3   alert('Start of try runs'); // (1) <--  
4  
5   lalala; // error, variable is not defined!  
6  
7   alert('End of try (never reached)'); // (2)  
8  
9 } catch(err) {  
10  
11   alert(`Error has occurred!`); // (3) <--  
12  
13 }
```





```
1 try {
2   lalala; // error, variable is not defined!
3 } catch(err) {
4   alert(err.name); // ReferenceError
5   alert(err.message); // lalala is not defined
6   alert(err.stack); // ReferenceError: lalala is not defined at
  (...call stack)
7
8   // Can also show an error as a whole
9   // The error is converted to string as "name: message"
10  alert(err); // ReferenceError: lalala is not defined
11 }
```



Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).



```
1 function name(parameter1, parameter2, parameter3) {
2   // code to be executed
3 }
4
5 function myFunction(p1, p2) {
6   return p1 * p2; // The function returns the product of p1
7   and p2
8 }
9
10 var x = myFunction(4, 3); // 12
11
12 function toCelsius(fahrenheit) {
13   return (5/9) * (fahrenheit-32);
14 }
15
16 console.log(toCelsius(43)); // 6.11
```



Classes



```
1 // Initializing a class definition
2 class Hero {
3     constructor(name, level) {
4         this.name = name;
5         this.level = level;
6     }
7
8     // Adding a method to the constructor
9     greet() {
10         return `${this.name} says hello.`;
11     }
12 }
13
14 const hero1 = new Hero('Varg', 1);
15
16 console.log(hero1.name); // Varg
```



Object Oriented Approach VS Functional



OOPS

Object-oriented programming based on the main features that are:

1. Abstraction: It helps in letting the useful information or relevant data to a user, which increases the efficiency of the program and make the things simple.
2. Inheritance: It helps in inheriting the methods, functions, properties, and fields of a base class in the derived class.
3. Polymorphism: It helps in doing one task in many ways with the help of overloading and overriding which is also known as compile-time and run-time polymorphism respectively.
4. Encapsulation: It helps in hiding the irrelevant data from a user and prevents the user from unauthorized access.

Functional

The function can be easily invoked and reused at any point. It also helps the code to be managed and the same thing or statements does not need to write again and again.



Thank You!

[Github](#) | [Website](#) | [LinkedIn](#) | [Instagram](#)

