

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a blue gradient background, resembling a circuit board or a neural network.

JS BASICS 2

ARRAYS AND STRINGS

- Basic functionality
- Inbuilt array methods – push, pop, shift, unshift, splice, sort
- Inbuilt string methods – length, indexOf, includes, slice, substring, substr, toLowerCase, toUpperCase
- String concatenation
- For-of loops

SLICE METHOD

- Slice and return elements from given starting index till the ending index
- Does not modify the original string

```
i 1 let p = "Apple"
i 2 let q = p.slice(2,5)
i 3 console.log(q)
4
5
i 6 let a = "abcdef"
i 7 let b = a.slice(1,-3)
8
i 9 console.log(b)
10 |
```

```
ple
bc
Hint: hit control+c anytime to enter REPL.
> |
```

SUBSTRING METHOD

- Same as slice but does not accept negative arguments
- If you omit the second parameter, substring() will slice out the rest of the string.

```
1 let str = "Hello World"
2 let sub1 = str.substring(2,6)
3 console.log(sub1)
4
5 let sub2 = str.substring(4)
6 console.log(sub2)
```

```
llo
o World
Hint: hit control+c anytime to enter REPL.
> 
```

SUBSTR METHOD

- Slices the string using given starting index and length

```
1 let str = "Hello World";  
2 let sub1 = str.substr(2,6);  
3 console.log(sub1);  
4  
5 let sub2 = str.substr(6,3);  
6 console.log(sub2);
```

```
llo Wo  
Wor  
Hint: hit control+c anytime to enter REPL.  
> 
```

HIGHER ORDER FUNCTIONS

- Functions that accept another function as argument or that return another function
- Eg - Filter, map, sort, reduce

FUNCTION THAT ACCEPTS OTHER FUNCTIONS AS ARGUMENTS

```
1  function formalGreeting() {  
2      console.log("How are you?");  
3  }  
4  function casualGreeting() {  
5      console.log("What's up?");  
6  }  
7  function greet(type, greetFormal, greetCasual) {  
8      if(type === 'formal') {  
9          greetFormal();  
10     } else if(type === 'casual') {  
11         greetCasual();  
12     }  
13 }  
14  
15 greet('formal', formalGreeting, casualGreeting)  
16 greet('casual', formalGreeting, casualGreeting)
```

FUNCTION THAT RETURNS ANOTHER FUNCTION

```
1  function adder(x) {  
2    return (y) => {  
3      return x + y;  
4    }  
5  }  
6  
7  const fourAdder = adder(4)  
8  console.log(fourAdder(5))  
9  
10 const tenAdder = adder(10)  
11 console.log(tenAdder(20))
```

9

30

FILTER FUNCTION

- The `filter()` method creates an array filled with all array elements that pass a test (provided as a function).

```
1  let arr = [1, 5, 4, 3, 6, 8, 7, 2]
2
3  let even = arr.filter((x) => {
4    return x % 2 == 0;
5  })
6
7  let odd = arr.filter((x) => {
8    return x % 2 == 1;
9  })
10
11 console.log(even)
12 console.log(odd)
```

```
[ 4, 6, 8, 2 ]
[ 1, 5, 3, 7 ]
```

MAP FUNCTION

- The `map()` method creates a new array with the results of calling a function for every array element.

```
1  let arr = [1, 5, 4, 3, 6, 8, 7, 2]
2
3  let squares = arr.map((x) => {
4    return x * x;
5  })
6
7  console.log(squares)
```

```
[
  1, 25, 16, 9,
  36, 64, 49, 4
]
```

REDUCE FUNCTION

- The `reduce()` method executes a reducer function (that you provide) on each element of the array, resulting in single output value.

```
1  let arr = [1, 2, 3, 4]
2
3  let totalSum = arr.reduce((accumulate, currentValue) => {
4    |   return accumulate + currentValue;
5  })
6
7  console.log(totalSum)
```

10

OBJECTS

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

```
11 let person = {  
12     name: 'Spiderman',  
13     age: 20,  
14     villains : ['Green Goblin', 'Doc Ock', 'Venom', 'Vulture']  
15 }
```

Since there is no fixed type for the data, objects can further contain objects and so on.


```
11 let person = {  
12   name: {  
13     first: 'Peter',  
14     last: 'Parker'  
15   },  
16   age: 20,  
17   villains : ['Green Goblin', 'Doc Ock', 'Venom', 'Vulture']  
18 }
```

- The properties of an object can be accessed using . (dot operator) or using [] (square brackets) operators.

```
1  let person = {  
2    name: {  
3      first: 'Peter',  
4      last: 'Parker'  
5    },  
6    age: 20,  
7    villains : ['Green Goblin', 'Doc Ock', 'Venom', 'Vulture']  
8  }  
9  
10 console.log(person.name.first)  
11 console.log(person.name.last)  
12  
13 console.log(person['age'])  
14 console.log(person['villains'])  
15
```

- Property values can be changed in the same manner as they are accessed.
- We can also create new properties at runtime.
- If a property already exists, its value is changed. If it doesn't a new entry is created for it.

```
1 let person = {  
2   name: {  
3     first: 'Peter',  
4     last: 'Parker'  
5   },  
6   age: 20,  
7   villains : ['Green Goblin', 'Doc Ock', 'Venom', 'Vulture']  
8 }  
9  
10 person.name = 'Peter Parker'  
11 person.girlfriend = 'Mary Jane Watson'  
12  
13 console.log(person)
```



```
{  
  name: 'Peter Parker',  
  age: 20,  
  villains: [ 'Green Goblin', 'Doc Ock', 'Venom', 'Vulture' ],  
  girlfriend: 'Mary Jane Watson'  
}
```


NEW AND THIS KEYWORDS

- The new operator allows us to create a new instance of a user-defined object type or of one of the built-in object types that has a constructor function.
- 'this' refers to the calling object.

- In the given code snippet, we create a new object of the given function using the 'new' keyword.
- 'batman' now contains an object of Superhero class.

```
16 function Superhero(name, age, villains) {  
17     this.name = name  
18     this.age = age  
19     this.villains = villains  
20 }  
21  
22 let batman = new Superhero('Batman', 30, ['Joker', 'Penguin', 'Deathstroke'])  
23 console.log(batman)
```

Note that the batman object has 'Superhero' written before the object definition. This indicates that it is an instance of Superhero.



```
Superhero {  
  name: 'Batman',  
  age: 30,  
  villains: [ 'Joker', 'Penguin', 'Deathstroke' ]  
}
```

PROTOTYPES

- Property inheritance.
- `Object.create()` function

```
30 let p = {  
31   a: 10  
32 }  
33  
34 let q = Object.create(p)  
35 q.b = 20  
36  
37 let r = Object.create(q)  
38 r.c = 30  
39  
40 console.log(p)  
41 console.log(q)  
42 console.log(r)  
43 console.log(r.c)  
44 console.log(r.b)  
45 console.log(r.a)  
46 console.log(r.__proto__ === q)
```

```
{ a: 10 }  
{ b: 20 }  
{ c: 30 }  
30  
20  
10  
true
```

SETTIMEOUT FUNCTION

- Executes the given function after a delay of specified time.
- The time argument is passed as number of milliseconds
- (1000 milliseconds = 1 second)

```
14  setTimeout(() => {  
15      |    console.log('Timeout after 5 seconds')  
16  },5000)  
17
```

SETINTERVAL FUNCTION

- Executes the given function infinitely after specified interval of time.
- The given example snippet will print 'hello' infinitely after every one second.

```
10 setInterval(() => {  
11     console.log('hello')  
12 }, 1000)
```

- The execution of setInterval function can be stopped by clearInterval function.

```
9  let count = 0;
10 let intervalId = setInterval(() => {
11     console.log('Hello ',count)
12     count++;
13     if (count >= 5) {
14         clearInterval(intervalId)
15     }
16 }, 1000)
```

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
```

CALL STACK

JavaScript Call Stack is a mechanism to keep track of the function calls.