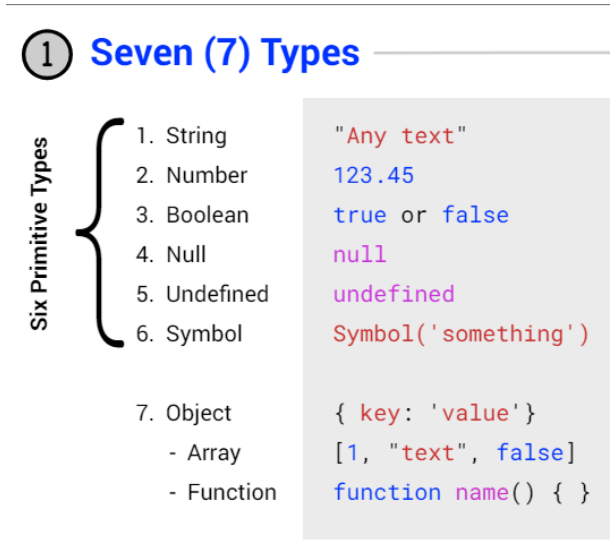
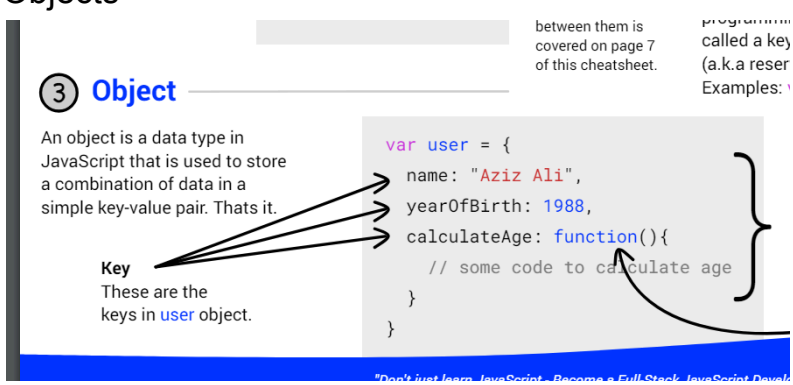
Phase 2nd

1. Proper Layout Structure
2. Firebase deployment
3. Team practice in GitHub
4. Responsive UI on every screen.
5. Increasing the speed of the website
6. Cumulative layout shift and SEO-friendly code.
7. Javascript Cheatsheet topics.

a. Data Types



b. Objects



c. Array

d. Functions

4 Function

A function is simply a bunch of code bundled in a section. This bunch of code ONLY runs when the function is called. Functions allow for organizing code into sections and code reusability.

Using a function has ONLY two parts. (1) Declaring/defining a function, and (2) using/running a function.

Name of function
Thats it, its just a name you give to your function. Tip: Make your function names descriptive to what the function does.

Return (optional)
A function can optionally spit-out or "return" a value once its invoked. Once a function returns, no further lines of code within the function run.

```
// Function declaration / Function statement
function someName(param1, param2){

    // bunch of code as needed...
    var a = param1 + "love" +  param2;
    return a;
}

// Invoke (run / call) a function
someName("Me", "You")
```

Parameters / Arguments (optional)
A function can optionally take parameters (a.k.a arguments). The function can then use this information within the code it has.

Code block
Any code within the curly braces { ... } is called a "block of code", "code block" or simply "block". This concept is not just limited to functions. "if statements", "for loops" and other statements use code blocks as well.

e. Operators

6 Operators

Full list of JavaScript operators https://developer.mozilla

Operators are reserved-words that perform action on values ar

Arithmetic
.. + .. Add
.. - .. Subtract
.. * .. Multiply
.. / .. Divide
.. % .. Remainder
.. ** .. Exponential

Assignment
.. = .. Assign value
.. += .. Add then assign
.. -= .. Subtract then assign
.. *= .. Multiply then assign

Logical
.. || .. Or
.. && .. And

Equality
.. === .. Equality
.. == .. Equality with coercion

Conversion
+ .. Convert to number
- .. Convert to number then negate it
! .. Convert to boolean then inverse it

Relational / Comparison
.. >= .. Greater than or equal to
.. <= .. Less than or equal to
.. != .. Not equal after coercion
.. !== .. Not equal

Increment / Decrement
..++ Postfix increment
..-- Postfix decrement

++.. Prefix increment
--.. Prefix increment

Others
typeof ..
.. instanceof ..
(..)
...spread-operator
.
..[..]
new ..
delete ..
(.. ? .. : ..)

f. Coercion

7 Coercion

When trying to compare different "types", the JavaScript engine attempts to convert one type into another so it can compare the two values.

Type coercion priority order:
1. String
2. Number
3. Boolean

```
2 + "7";   // "27"
true - 5   // -4
```

Coercion in action
Does this make sense?

g. Conditional Statements.

8 Conditional Statements

Conditional statements allow our program to run specific code only if certain conditions are met. For instance, lets say we have a shopping app. We can tell our program to hide the "checkout" button if the shopping cart is empty.

If -else Statement: Run certain code, "if" a condition is met. If the condition is not met, the code in the "else" block is run (if available.)

```
if (a > 0) {
  // run this code
} else if (a < 0) {
  // run this code
} else {
  // run this code
}
```

Ternary Operator: A ternary operator returns the first value if the expression is truthy, or else returns the second value.

```
(expression)? ifTrue: ifFalse;
```

Switch Statement: Takes a single expression, and runs the code of the "case" where the expression matches. The "`break`" keyword is used to end the switch statement.

```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice1:
    // run this code
    break;

  default:
    // run this code
}
```

h. Loops

For loop

```
for (initial-expression; condition; second-expression){
  // run this code in block
}
```

While loop

```
while (i<3){
  // run this code in block
  i++;
}
```

Do while loop

```
do {
  // run this code in block
  i++;
} while (i<3);
```

i. Dom

Learn JavaScript Correctly (Video course) https://ilovecoding.org/courses/js2

14 DOM - Document Object Model

Query/Get Elements

```
// Preferred way:
document.querySelector('css-selectors')
document.querySelectorAll('css-selectors', ...)

// Old ways, and still work:
document.getElementsByTagName('element-name')
document.getElementsByClassName('class-name')
document.getElementById('id')
```

Create / clone Element

```
document.createElement('div')
document.createTextNode('some text here')
node.cloneNode()
node.textContent = 'some text here'
```

Add node to document

```
parentNode.appendChild(nodeToAdd)
parentNode.insertBefore(nodeToAdd, childNode)
```

Get Element Details

Modify Element

```
node.style.color = 'red'
node.style.padding = '10px',
node.style.fontSize = '200%'

node.setAttribute('attr-name', 'attr-value')
node.removeAttribute('attr-name')
```

Get and Modify Element Class

```
node.classList
node.classList.add('class-name', ...)
node.classList.remove('class-name', ...)
node.classList.toggle('class-name')
node.classList.contains('class-name')
node.classList.replace('old', 'new')
```

Remove Node

```
parentNode.removeChild(nodeToRemove)
// Hack to remove self
nodeToRemove.parentNode.removeChild(nodeToRemove)
```

j. Array method

```
const thing = "some text";
```

```
const num = 123.45;
```

String

Google 'Mozilla String' to find the docs

```
.concat()
.charAt()
.indexOf()
.startsWith()
.endsWith()
.split()
.slice()
```

Number

Google 'Mozilla Number' to find the docs

```
.toFixed()
.toPrecision()
.toString()
```

Boolean

Google 'Mozilla Boolean' to find the docs

```
.toString()
```

Array

Google 'Mozilla Array' to find the docs

```
.filter()
.map()
.find()
.every()
.some()
.sort()
.slice()
.splice()
.reduce()
.forEach()
```

Full list of builtin objects in JavaScript visit https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

k. Promise

```
// (A) Create a promise
const p = new Promise((resolve, reject)=>{
  // Do some async task
  setTimeout(()=>{
    if(condition){
      resolve('Successful login');
    } else {
       reject('Login failed');
    }
  }, 2000)
})
```

```
// (B) Using a promise
p.then((res)=>{
  console.log(res)
})
.catch((err)=>{
  console.log(err)
})
```

Note: 90% of the time you will be working with pre-existing promises. The step of "Creating a promise" would be done for you either by a library, framework or environment you are using. Examples of promises: fetch

I. Constructor.

```
// Defining a Constructor
function Car(make, model, year){
  this.make = make;
  this.model = model;
  this.year = year;

  this.setMiles = function(miles){
    this.miles = miles
    return miles;
  }
}

// Using a constructor
const car1 = new Car('Toyota', 'Prius', 2016);
const car2 = new Car('Hyundai', 'Sonata', 2018);

// Adding method to the constructor prototype
Car.prototype.age = function(){
  return (new Date()).getFullYear() - this.year;
}
```

After the cheat sheet is over they practice the following tasks.

1. Todo
2. Form validation in HTML and Javascript

React js

1. Introduction of React
2. React UI
3. Using React Bootstrap and Tailwind css
4. NPM Library practice(Slick slider, swiper slider)
5. Hooks practice.
    a. UseState

    b. UseEffect
    c. UseRef(email js)
6. React Router dom
7. Form validation
8. Regex in form validation
9. Map method
10. Props
11. Folder structure of react.