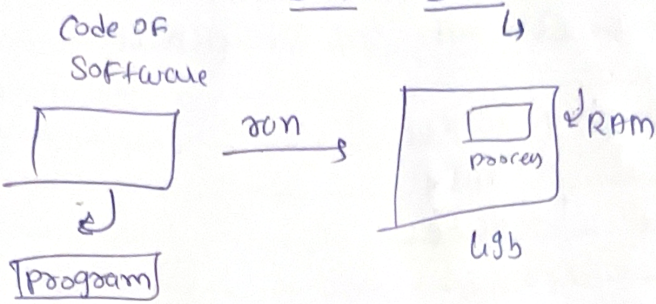
Basic of JS



Program in a Running state is called Process

keywords

↳ in a programming lang.. few words are reserved by the language for its own purpose. We cannot change there meaning or purpose them for a different use case.

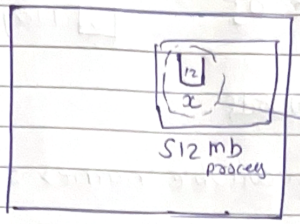
For example in JS few keywords are

- ↳ let
- ↳ For
- ↳ Const
- ↳ iF
- ↳ else
- ↳ return
- ↳ etc

EXPT. NO. NAME: Page No. Date: YOUVA

How can we store some values inside a program

↳ to store values we can use Variables.



Variables are memory buckets that stores our values and has name as ~~to~~ wells.

Javascriptfile → .js

Ex index.js
demo.js

↳ How to create a variables in js?

3 ways to create Variables

(1) Using Var keyword

Syntax

↳ var <varable name> = <value>;

For Ex

var marks = 90; 90 marks

var score = 5; 5 score

Teacher's Signature:

(2) Using let keywords

Syntax → let <variable name> = <value>;

Ex let age = 24; | 24 | age

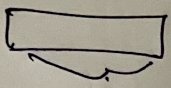
let Flag = 0; | 0 | Flag

(3) Using const keywords

syntax → const <variable name> = <value>;

Ex const x = 100;

Semicolomn



one line of JS code equal to one instruction that we want to give to the computer.

One complete instruction is called Statement

At the end of every statement in JS we can put a semicolom (but it is optional)

EXPT. NO.

NAME:

Naming

Page No.

Date:

YOUVA

+ Rules For Variable Creation.

(1) Variables can contain Small Alphabates Capital Alphabates, digits, Underscore (_) (&)

2) we can not have space or any other Special char.

3) we should give meaning full names So that reader can identify the names property.

4) Variables name cannot start with a digit, but it can ~~then~~ have digits in between or at last.

Ex let ajems = 10; X Wrong Syntax

(5) we cannot use keywords as variables names

Teacher's Signature:

① console.log( )

This function takes some data on input and then displays them in the output

→ if we have 2 different console.log( ), then both of them displays / prints output in different lines.

we can also pass multipies values together console.log() & all of them in a same line.

Ex console.log(10, "starting", age);

* what all values we can store in JS ??

(1) Number → 10, -3, 2.6, 3.414, 100, 1000, ... etc

(2) String → text
(1) " " (a pair of double quotes)
(2) ' ' (single quotes)
(3) ` ` (Use of backtick)

(3) boolean → True or False. } keywords
0/1
Ex- var cond=true; ↳ This are all small values

(4) Undefined → key word only
↳ something not defined yet but mab be defined later.

Var status = Undefined

EXPT NO | NAME:

Page No.
Date

YOUVA

5) objects :- If we have to somehow store key value pairs then we can use objects

<key, values>

```
{
    name : Sanket
    Company : Oracle
    position = "Software Devl"
}

{ name : "Jaymeen"
  Company = "Phonepay"
  position = "Apm"
}
```

key will be unique

Primitive and non-primitive

↓ types which are atomic in nature

Ex → numbers

↓ types which are composition of other types

Ex → object

```
User = { name = "jaymeen"
         age : 24
         posts : {
             createdAt : "Jun 12, 2024"
             text : "my first post"
         }
         gender : "MALE"
         :
       }
```

(6) NULL → It actually represent empty values

* Special chars

"The new apple iphone"
"has been launched"

4 new line consider they have to launch

"The new apple iphone \n has been

\n → escape signal

\t ⇒ tab

* Commets

↳ Commets are a piece of code that is for documentation purpose.

↳ The programming language will just avoid comment during execution.

EXPT NO: NAME: Page No: Date: YOUVA

In JS, there are two ways

Single line Comment → // → Double forward Slash

multiline Comments → /* */ → multiple

Operands → values on which use word to do the operation

Operators

There are different types of operators

(1) Arithmetic operators:

+ → addition
- → substraction
/ → division
% → remainder
* → multiplication

Ex

operands

10 (+) 3

operator

Ex

```
let x=10;
let y=3;
```

13 → `console.log(x+y)` // addition
7 → `console.log(x-y)` // substraction
30 → `console.log(x*y)` // multiplication
3.33 → `console.log(x/y)` // Division
1 → `console.log(x%y)` // Remainder
9 → `console.log(y**2)` // power

Output

(2) Assignment operators

=, +=, -=, *=, /=, %=

let a=10;

→ assignment
→ = just directly assigns the value on R.HS to L.H.S

a+=2

→ a+=2 means → a=a+2
let say previous of a was 10, then if you do a+=2, → a=a+2
→ on LHS use take old values of a & add 2 to it.

Ex

```
// Assignment operator
let a=10;
a+=10;
console.log(a);
```

EXPT NO. | NAME: | Page No.: | Date: | YOUVA

Relational operators

<, >, <=, >=

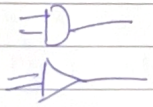
operand 1 (<, >, <=) operand 2

(10 < 12) — True

(5 < 2) — False

logical operators

Boolean logic gate



Input → logic

true
false

Ex AND Gate, OR GATE, NOT GATE etc

(&&) (||) (!)

AND

x	y	x AND y
T	T	T
F	F	F
T	F	F
F	T	F

OR

x	y	x or y
T	T	T
T	F	T
F	T	T
F	F	F

NOT

I	Out
T	F
F	T

```
Console.log ( True && False )        output: False
Console.log ( (10>5) && (6<3) )      False
                ↓        ↓
              True     False
```

Teacher's Signature:

Qn = what values are falsy in JS ??

null
undefined
"" (empty)
+0
-0
NaN
false

} Most apart from this everything is truly.

→ Coercion (Type interconversion)

→ In a and gate, if the first input is false, then it doesn't evaluate the second input one immediately return the first input as well as if first input is true, then the second input has to be evaluated then second input is return.

→ In a or gate, if the first input is true, then it doesn't evaluate the second input and immediately return the first input.

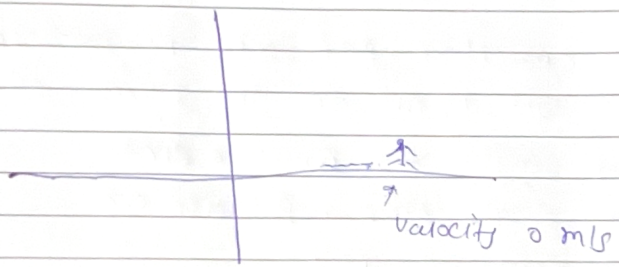
whenever,

if the first input is false then it returns the second input.

EXPT. NO. NAME: Page No. Date: YOUVA

Numbers → 0
-0 } all three case numbers
directional Num

magnitude direction



Nan → Not a Numbers

0	1	2	3	4	5	6
as	df	or	xy	2	mn	a.

↳ Return the Bucket Number in which df string is present

↳ if there is a situation where you're bound to return a number, but there is no valid posible no to return, then we use **NaN**.

10/3 → ____

10/NaN → ____

Teacher's Signature:

Q= # which is the only primitive value which can is not equal to its self ??

Ans → NaN

Bitwise operator

↳ 5 → (101) binary representation of 5

bitwise operators performs the corresponig operation bit by bit on the given operand.

& → single ~~&~~ → Bitwise and
| → single pipe → Bitwise or
^ → bitwise xor
~ → bitwise not

equality operators

== → called as Abstract equality operator
=== → called strict equality operator.

== → it check the type of both operands
↳ if type is same, then it called ===
↳ if types are not same then type conversion occurs coercion & then comparison is done.

===→ it also check type of both values and both operator.