Comm & Exam

FRIDAY CESC LIMITED JAVASCRIPT BASICS

JANUARY

① There are (3) ways in which we can store some data in our Ram.

- var
- let
- const.

② For clean JS code, always put ";".

③ Variable Naming Convention :-

Right :	Wrong :-
Small alphabets Capital alphabets Underscore Dollar digits → Not at start (wrong) → In between or end (allowed)	No spaces other special characters.

W Official JS documentation :- ECMA international

④ Values we can store in JS :-

- Number :- 10, 20, -3, 14.15
- String :- "Name"; 'Subho', `backtick`
- Boolean :- true / false
  1 / 0
- Undefined: Something not defined yet, but maybe defined later.

```
var status = undefined.
```

FEBRUARY 2017

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

SATURDAY 20 17

3rd Week • 014-351

JANUARY 14

⊙ Objects :- If we have to somehow store Key-value pairs.

Ex: { name : "subhadip", company : "coinbase", position : "SWE" }

This object for one user.

There can be multiple users.

Keys will be unique.

09.00

10.00

11.00

12.00

01.00

02.00

03.00

04.00

05.00

06.00

07.00

⑤ Types of Datatype :-

Primitive (which are atomic in nature) :- No.s

Non-primitive (which are comporition of other types) :- objects.

⊙ Null :- It actually represents empty value.

let a; → undefined.

let b = 10;
b = null;

value is now empty.

SUNDAY

Special characters :-

⊙ multiline string :- (A) " Your phone \n has been launched "

\n → one break / new line character.

(B) \t → tab.

2017 MONDAY

16 4th Week • 016-349
JANUARY

JANUARY 2017

Wk	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

09.00

(7) Comments :- (documentation purpose)

Single line comment :- // ✓

10.00

Multi line comment :- /* ..... */ ✓

11.00

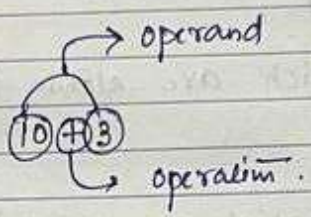
(8) Arithmatic operations :- / operators :-

01.00

(A) arithmatic operators

02.00

+ , - , / , * ; % , **



03.00

(B) assignment operators

04.00

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

05.00

06.00

(9) Relational Operators :- / Comparison operators

Always return bool

07.00

- ⊙ < less than
- ⊙ > Greater
- ⊙ <=
- ⊙ >=

(10) Logical operators :-

AND (&&)

x	y	x AND y
1	1	1
0	0	0
0	1	0
1	0	0

OR (||)

x	y	x OR y
1	1	1
0	0	0
1	0	1
0	1	1

NOT (!)

X	O/P
1	0
0	1

XOR (^)

x	y	x ^ y.
0	0	0
1	1	0
0	1	1
1	0	1

{ When both (x) ; (y) has same value, x ^ y will give 0 }

(H) Type Conversion :- To Boolean (argument)

(A) ⊙ undefined :- false.

⊙ null :- false.

⊙ Number :- True (except for +0, -0, NAN → return false)

⊙ String :- (If empty → return false)
Else → return true.

⊙ Symbol :- return true

⊙ Object :- return true.

(B) Coercion (type interconversion) :-

→ 10 + + 6 —O/P→ (6) ?? | 10 + + "Subha" → "Subha"

⊙ In a (AND) Gate if 1st I/P is (False) → then it doesn't evaluate 2nd I/P and immediately returns 1st I/P.

⊙ If 1st I/P is (True) → then the 2nd I/P has to be evaluated. 2nd I/P is returned.

2017 WEDNESDAY

18 4th Week • 018-347 JANUARY

JANUARY 2017

WK	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

∴ 10 && 6

↳ (10) is true.

∴ (&&) checks 2nd I/P (6)

and returns 2nd I/P

∴ O/P is (6).

⊙ (10 > 6) && (6 < 7)

O/P

(6 < 7) ✓

true ✓

Because <, >, <=, >= are

relational operators and it returns a Boolean value.

OR Gate

If 1st I/P is true → return 1st I/P.

If 1st I/P is false → return 2nd I/P.

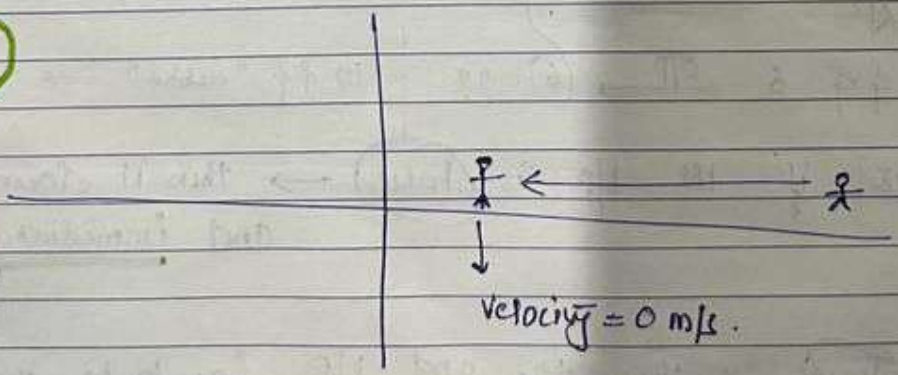
(12) Numbers (0, -0, NaN) :-, Infinity, -Infinity.

-0 :-

↓

directional quantity

magnitude direction.



FEBRUARY 2017

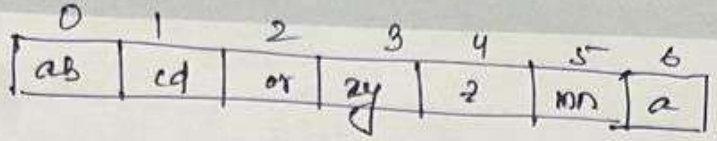
W	M	T	W	T	F	S	S
6			1	2	3	4	5
7	6	7	8	9	10	11	12
8	13	14	15	16	17	18	19
9	20	21	22	23	24	25	26
10	27	28					

THURSDAY 2017

4th Week • 019-346

JANUARY 19

NaN :- Not a Number



09.00 Return Bucket No. in which 'def' string is present

10.00 We will return a invalid No.

11.00 ∴ If there is a situation where you're bound to return a

12.00 no. but there is no valid no. to return then we use

01.00 NaN ∴ type of NaN → always Number.

02.00 ∴ anywhere we have to return invalid no. ✓

03.00 undefined | null → NaN

04.00 Imp.

05.00 (*) Only primitive value which is not equal to itself → NaN

06.00 10 < Infinity → true ✓

(15) **Bitwise operators :-**

- ⊙ & → Bitwise and
- ⊙ | → " OR
- ⊙ ^ → " XOR
- ⊙ ~ → " NOT

✉ @ ☎

2017 FRIDAY

20 JANUARY

4th Week • 020-345

(14)

Equality operators :-

09.00

10.00 `==` → abstract equality op.

11.00 `===` → strict equality op.

12.00

`==` → Checks the type of both operands.

01.00 ① If type is same then it calls `===`.

02.00

03.00 ② If types are not same then type conversion occurs (coercion).

04.00 & then comparison is done.

05.00

06.00

`===` → Checks the type of both operands.

07.00 ① If types are different, it returns false

② If types are same, then value comparison occurs.

Imp

∴ "==" will do coercion.

"===" will never do coercion.

eg ✉ @ ☎ `1 == "1"`

string to no.

`1 == 1` → `1 === 1` ⇒ true o/p

no. no.

Eg2 `1 == "One"`

imp `1 == NaN`

→ at this point type of (1) and type of (NaN) will be same.

∴ `1 === NaN` (type is same, value comparison via user)

∴ returns false.

Eg3 `NaN === NaN`

NaN is the only no. that is not equals to itself.

(15) Type of :- type of "1" → string. ✓ type of NaN → num

type of false → boolean ✓

" " 1 → Number ✓

" " undefined → undefined ✓

Explain. " " null → object ✓

(16) Conditionals :-

Syntax

```
if (condition) {
} else if (condition2) {
} else {
}
```

2017 MONDAY

23 5th Week • 023-342 JANUARY

JANUARY 2017

WK	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

check odd / even

```
if (27 % 2 == 0) {
    console.log("Even")
} else {
    console.log("Odd");
}
```

```
let x = 10; y = 20; z = 30
```

valid statement

Valid triangle (a, b, c) → sides.

a+b > c
b+c > a
a+c > b

→ all 3 needs to be true

if any one of them is false, whole thing is false.

```
if (a+b > c && b+c > a && a+c > b) {
    console.log("Valid triangle")
} else {
    console.log("Not")
}
```

```
if (a == b && b == c && a == c) {
    Equilateral
} else if (a != b && b != c && a != c) {
    Scalene
} else {
    Isosceles
}
```

FEBRUARY 2017

WK	M	T	W	T	F	S	S
6			1	2	3	4	5
7	6	7	8	9	10	11	12
8	13	14	15	16	17	18	19
9	20	21	22	23	24	25	26
10	27	28					

TUESDAY 2017

5th Week • 024-341

JANUARY 24

09.00

(17) Loops:-

```
let i = 0;
while (i <= 10) {
    console.log(i);
    i += 1;
}
```

While loop Syntax

10.00

11.00

12.00

01.00

02.00

03.00

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

For loop Syntax

04.00

05.00

07.00

(18) Functions:- Syntax

```
function myNewFunction (ip1, ip2, ip3) {
    // logic
    return somevalue;
}
```

Function to check if a number is even or odd.

```
function checkNumber (number) {
    if (number % 2 == 0) {
        return even;
    } else {
        return odd;
    }
}
```

```
let x = 10;
if (checkNum(2)) {
    return even
} else {
    return odd
}
```

2017 WEDNESDAY
25 5th Week • 025-340 JANUARY

JANUARY 2017

WK	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

Note: console.log(" ")
↳ This function returns undefined and prints on the console.

x = console.log("Subhadip")
↳ This will print "Subhadip" on the console and also stores the value undefined in x

Inbuilt fune

⊙ Math.sqrt(100)

In JS, if you don't return something it automatically returns undefined for you.

Eg:-

```
function welcome(name){
    console.log('Hello', name, "welcome to JS");
}

let value = welcome("Subhadip")
console.log(value)
```

Hello Subhadip welcome to JS
undefined.

∵ When we wrote

```
let value = welcome("Subhadip");
```

Hello Subhadip welcome to JS got printed in the console because we used a console.log() statement.

FEBRUARY 2017

W	M	T	W	T	F	S	S
6			1	2	3	4	5
7	6	7	8	9	10	11	12
8	13	14	15	16	17	18	19
9	20	21	22	23	24	25	26
10	27	28					

THURSDAY 2017

5th Week • 026-339

JANUARY 26

Again,

∵ console.log() has a return type of "undefined".

∴ "undefined" got stored in x.

∴ The value of x in console.log(x) is undefined.

09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
07.00

```
function add (x,y) {
    return x+y;
}
c = add (10,20)
console.log (c);
```

parameters

parameters are defined

arguments.

arguments are passed.

```
c = add (10, 20, 30)
console.log (c)
```

→ 30

The function add will only process the first two arguments

(19) Unary operator :- An operator which has only one operand.

(20) i++ vs ++i :- [illegible]

i++ :- Postfix unary operator.

++i :- prefix unary operation.

Both of them are increment operator when not assigned to any variable. ✓

✉ @ ☎

① assign the old value of (i) to a variable.

② Then increments old value by 1

① Increments old value of i by 1

② assigns new value of i to a variable.

2017 FRIDAY

27 5th Week • 027-338

JANUARY

JANUARY 2017

Wk	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

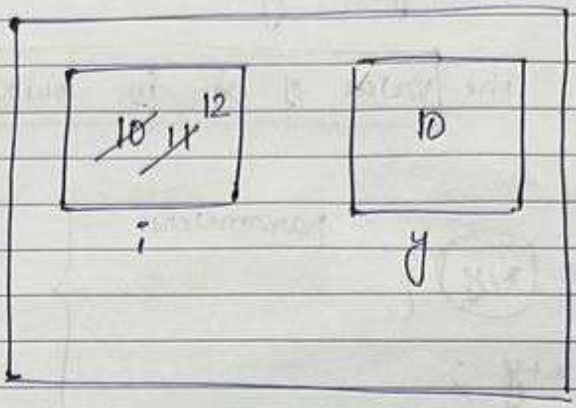
(A) (i) let i = 10

(ii) let y = i++

(iii) console.log (i, y) → 11, 10

(iv) i++

(v) console.log (i, y) → 12, 10



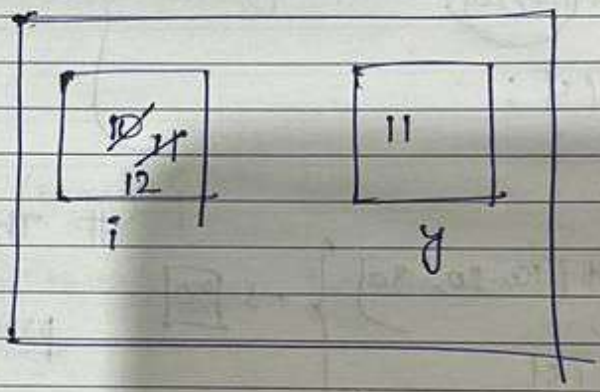
(B) (i) let i = 10

(ii) let y = ++i

(iii) console.log (i, y) → 11, 11

(iv) ++i

(v) console.log (i, y) → 12, 11



Explanation :- (A)

i. i = 10

ii let y = i++ → Postfix unary operator

old value of (i) gets assigned to y

∴ i = 10 ; y = 10

Now, old value of (i) get incremented by 1

i = 11 , y = 10

iii console.log (i, y) → 11, 10

FEBRUARY 2017

SU	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

(10) i++

(i) gets incremented by 1.

i = 12; y = 10

(v) console.log (i, y) → 12, 10

(iii) i = 10

(iii) y = ++i → Prefix unary operator.

(i) gets increment by 1 unit.

i = 11

(i) gets assigned to y.

∴ i = 11; y = 11

(iii) console.log (i, y) → 11, 11

(iv) ++i → i = 12; y = 11

(v) console.log (i, y) → 12, 11

Imp

In Postfix (i++) → You first assign then increment.

In Prefix (++i) → First increment, then assign.

SUNDAY 29

{The same logic undergoes for i-- & --i}.

(c) + (Unary operator) :- This unary operator tries to convert a variable to a number.

let x = "22";

let y = +x;

type of (y); → Number } type of y → Number.

type of x → String.

2017 MONDAY

30 6th Week • 030-335 JANUARY

JANUARY 2017

We	M	T	W	T	F	S	S
1	30	31					1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29

09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
07.00

④ Unary (-)

x = "22"

y = -x

It also converts the operand to a number but makes the result -ive.

ie ② will remain as it is.

y will be (-22) → Number.

Note

∴ For Unary (+) and Unary (-) ; x will remain "22"

type of (x) will be string.

x = "22"

y = (-x) / (+x) ← but

y = -22 y = +22

if typeof y → Number.

If we are just doing +x or -x without assignment, the value of x & type of x will not change.

Other unary operators :-

type of

④ Note 2

Unary + / - will convert anything to a number. If it can't it will return NaN.

* If Unary + or Unary - is not able to convert it to a no. it will return NaN

eg) let x = "Rishi"

let y = -x;

console.log(y)

→ NaN.

eg let x = "-22"

let y = -x } → y o/p :- -(-22) = 22

21

Do while Loop :- (Can be substituted with while itself)

→ Not imp

```
let y = 10;
while (y < 5) {
    y++;
}
console.log(y); → 10
```

false

In Do-while it doesn't matter if the condition is T|F from the start we will execute the (Do) block (once) at least then go | not go in the while loop.

Syntax :-

```
let y = 10;
do {
    y++;
} while (y < 5);
```

y: 10 → 11

this piece of code will run once

then it may | may not go inside while loop.

```
console.log(y) → 11
```

Eg

```
let y = 5
do {
    console.log(y);
    y++;
} while (y < 10);
```

y: 5 6 7 8 9

O/P

y: 10 9 8

2017 WEDNESDAY

01 FEBRUARY

6th Week • 032-333

FEBRUARY 2017

Wk	M	T	W	T	F	S	S
6			1	2	3	4	5
7	6	7	8	9	10	11	12
8	13	14	15	16	17	18	19
9	20	21	22	23	24	25	26
10	27	28					

(22) Ternary operator :- ? :

Syntax :-

```
let y = ((condition) ? (exp1) : (exp2));
```

If condition is true ⟶ evaluate exp1

Whatever is the result return that to y

false ⟶ evaluate exp2

Whatever is the result return that to y.

Eg

```
let y = ((10 > 5) ? (10) : (5));
```

O/P

10.

Eg

```
let y = ((10 < 5) ? (10) : (5));
```

O/P

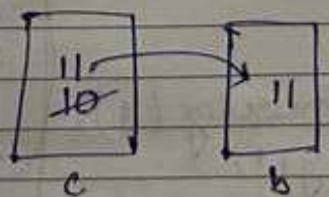
5.

Eg.

```
let c = 10;
let b = ((3 < 1) ? (2-3) : (++c));
```

F

```
let b = ++c
```




```
console.log(b, c);
```

O/P

11 11

THURSDAY 2017

6th Week • 033-332

FEBRUARY 02

Note!

When you compare NaN with any number:

(A) 4 > NaN → false

(B) 4 < NaN → false

(C) 4 == NaN → false.

ie you always get false

✓ But, if we do type 4 == type of NaN

IMP.

↳ this will return true

Because NaN is a invalid number.