Coercion [Basic - Advance] Interview Questions

COUNT: 20

Question	Level	Type	Solution

01.
 	
console.log(+true);
 	
console.log(!"Javascript");
	Basic	Urinary Operator
		Logical Operator
			Output: 1, false
			
			Reason: + will try to convert it to number and ! will try to convert it into a boolean value

02.
 	
const num = 10;
const num_1 = new Number(10);
const num_2 = 10;
 	
console.log(num == num_1)
console.log(num === num_1)
	Basic	Constructor Function	**Output: true, false**
			
			Reason: Since its a abstract equaltiy in first case, Js will try to do coercion, i.e non-primitive to primitve value resulting in true.
			
			Whereas in second case, because of strict equality since type and value are not consistent. It will result false.
			
			NOTE: typeof new Number(10) is "object"

03.
 	
function sum (a, b) {
return a + b;
}
 	
console.log(sum (10, "10"))
	Basic	String Concatination	**Output: "1010"**
			
			Reason: String concatination is happening as one of the operand is a string.

04.
 	
console.log(10 + 20 + '10');
console.log(parseInt('10 + a'));
	Basic	String Concatination	**Output: '3010', 10**
			
			Reason: Js precedence will go from right to left.
			Hence 10 + 20 ---> 30 + '10' ---> '3010'
			
			b. parseInt will look for number in the start until it finds any invalid string that can't be coerced.

05. 99["toString"].length + 1	Advance	toString Implementation	**Output: 2** **Reason**: Here, we are not calling toString method, Hence it will be like function.length which will gives the number of argument it accept. And bydefault toString(radix) accept 1 argument Hence 1 + 1 = 2

06.
 	
const str = new String("JS");
 	
console.log(str === "JS");
console.log(str == "JS");
	Intermediate	new String () Implementation	**Output: false, true**
			
			Reason: as string created with new result in Object
			Hence === gives false, while == gets coerced.

07.
 	
console.log(false == []);
console.log(false == ![]);
		Intermediate		Array values type		**Output: true, true**
						
						Reason: Empty array corrosponds to truthy value
						Also, when in first case when false == [] , The comparison operator will convert the [] to "" empty string. Hence false == "" will comes to be true, as empty string is a falsey value.
						
						Now since the types are different i.e false == ""
						They will be converted to number i.e Boolean(false) and Boolean(""). Both will result in 0
						
						And finally 0 == 0 will be true.

08. console.log(undefined.toString( )) console.log(null.valueOf( ))		Advance		Types		**Output: TypeError** **Reason**: Not everything in Js is objects.

09.
 	
const num1 = Number( );
const num2 = Number(undefined);
 	
console.log(num1, num2);
		Intermediate		Number() works		**Output: 0, NaN**
						
						Reason: By default if no argument is passed in Number() it returns 0 and if you try to convert undefined to number it will give NaN as mentioned in specs.

10.
 	
let age = '51';
let age2 = '51';
 	
age+= 1;
age2++;
 	
console.log(age, age2)
		Intermediate		post increment, string		**Output: '511', 52**
						
						Reason: when you add a string to a number it will give string only i.e '51' + 1 will give '511'
						
						Wheras postincrement on string will give a number

11. console.log(1 + +"2" + 3);		Beginner		Number()		**Output: "33"** **Reason**: +"2" will convert it to number 2 Hence 1 + 2 = 3 then 3 + "3" = 33
12. console.log("" == [ ]);		Intermediate		How array transform		**Output: true** **Reason**: Coercion of an empty array will be empty string Hence, "" == "" will corrosponds to false == false which comes out to be true
13. const obj = { x: 10, valueOf ( ) { } } console.log(10 - obj)		Advance		ToPrimitive Algorithum **Subtract Operation** calls hint with "**number**"		**Output: NaN** **Reason**: The valueOf is returning undefined, and hence 10 - undefined will result in NaN

14.
 	
const obj = {
toString( ) { return 5 },
valueOf( ) { return { } }
}
 	
console.log(9 - obj);
		Advance		ToPrimitive Algorithum
				Subtract Operation calls
				hint with "**number**"
						Output: 4
						
						Reason: toString() returns 5
						Hence 9 - 5 is 4
						
						NOTE: valueOf will be called first, but since it returns object i.e non-primitve value the Js will toString, as it looks for a primitve value

15.
 	
const obj = {
toString( ) { return { } },
valueOf( ) { return { } }
}
 	
console.log(10 - obj);
		Advance		ToPrimitive Algorithum
				Subtract Operation calls
				hint with "**number**"
						Output: TypeError [Can't convert object to primitive value]
						
						Reason: As both toString and valueOf returns an object
						It throws TypeError

16.
 	
const obj = { };
 	
console.log("100" + obj);
 	
console.log(100 + obj);
 	
console.log(obj - 100);
		Advance		ToPrimitive Algorithum
				Addition Operation calls
				with no hint
						Ouput: "100[object Object]" , "100[object Object]", NaN
						
						Reason: In addition it will simply concatinate whereas in subtraction it will result in NaN

17. console.log(Number(" "));		Medium		Explicit Number ( ) Type Conversion		**Ouput: 0** **Reason**: The string will get trimmed from right and left by js resulting in empty string
18. console.log("2" > "12");		Medium		Comparison Operator Rules		**Output: true** **Reason**: Here both are of same type, hence none of them get converted to number, Js basically use the lexo graphical algorithum for string to check which one is greater.

19.
 	
console.log(null == 0);
 	
console.log(null == undefined);
 	
console.log(undefined == NaN)
 	
console.log(NaN == 0)
		Advance		Sweet Couple		**Output: false, true, false, false**
						
						Reason: null and undefined are called sweet couple. They equal only each other and to no other value.
						
						Similary NaN is special type which doesn't equal anything Not to even itself !

20.
 	
console.log(" -9 " - 2);
 	
console.log(undefined + 1);
 	
console.log(" \t \n " - 2)
		Advance		ToNumber Abstract Operation		**Output**: -11 , NaN , -2
						
						Reason: subtraction always convert to number, removing white spaces from start and end !
						Whereas if it was + the trimming will not occur in above cases. Try and see it for yourself !