What The Heck is This Keyword !

This Keyword [Basic Advance] Interview Question List !

COUNT: 17

Question	Level	Method Invocation Type	Solution

01.
 	
let arr = [1, 2, 3];
 	
arr.push(function sum ( ) {
console.log(this)
})
 	
arr[3]( );
	Basic	Method Invocation Type	The arr object itself

02.
 	
const income = {
skills: 100,
monthly () { console.log (this.skills + 100) },
yearly: () => console.log(this.skilss + 100)
}
 	
income.monthly();
income.yearly();
	Basic	Method Invocation Type And Arrow Function Type	Output: 200, NaN

03.
 	
class Animal {
 	
static getName(name) {
this.name = name;
console.log(this)
return this.name;
}
 	
constructor(name = "Tiger") {
this.name = name
}
 	
}
 	
const animal = new Animal("Lion");
console.log(animal.getName("Shark"))
	Advanced	Classes and How Static Method works inside a class ?	Output: Error ! [TypeError]
			
			Reason: When you create a statc method on a class.
			It's available only on Class itself. Meaning those static methods won't be available on class instance !!!
			
			You can call it by this only i.e ClassName.methodName ( )
			Here in the example, it will be like Animal.getName("Shark")
			And this in this case will be Class Animal.

04.
 	
function User (firstName, lastName) {
this.firstName = firstName;
this.lastName = lastName;
}
 	
User.getFullName = function ( ) {
console.log(this);
return `${this.firstName} ${this.lastName}`
}
 	
let user = new User("Hare", "Krishna")
 	
user.getFullName( );
User.getFullName( );
	Advanced	Constructor Invocation Type	Output: TypeError, "undefined undefined"
			
			Reason: The reason is same as above question, The method is made available only on function User not on it's instance.
			Hence the first wil result in TypeError.
			
			Whereas in second case, this will point to function User whole defination hence resulting in undefined undefined.
			
			How to Fix it ?
			
			Solution : User.prototype.getFullName = function ( ) {
			console.log(this);
			return `${this.firstName} ${this.lastName}`
			}

04.
 	
let obj = {
user: 'Sham',
getName: function( ) { console.log (this.user) }
}
 	
const getData = obj.getName;
getData( );
	Intermediate	Function Invocation Type	Output: undefined
			
			It will point to this window or undefined in strict mode

05.
 	
let user = {
email: "abc@tokopedia.com",
updateEmail: email => {
this.email = email;
}
}
 	
user.updateEmail("xyz@gmail.com");
 	
console.log(user.email);
		Intermediate		Arrow Function Invocation Type		**Output: abc@tokopedia.com**
						
						Reason: Arrow function doesn't have it own this		

06.
 	
let group = {
 	
title: "Our Group",
students: ["John", "Pete", "Alice"],
 	
showList() {
this.students.forEach(function(student) {
console.log(this, this.student);
});
}
 	
};
 	
group.showList();
		Intermediate		Function Invocation In callback And Method Invocation In showList		**Output: Window, undefined**
						
						Reason: As inside the callback function which is forEach it's a function invocation type. Hence this can be either window or undefined.
						
						NOTE: In strict mode (this keyword) it will be undefined		

07.
 	
let group = {
 	
title: "Our Group",
students: ["John", "Pete", "Alice"],
 	
showList( ) {
this.students.forEach(
student => console.log(this)
);
}
 	
};
 	
group.showList( );
		Intermediate		Arrow Function Type		**Output: object [given object]**
						
						Reason: As arrow function doesn't have it's own this, they will take it from its outer lexical enviornment		

08.
 	
let user = {
firstName: "Karan",
sayHi( ) { console.log(this) }
}
 	
setTimeout(user.sayHi, 1000);
 	
Alternatively, we can have do ---
 	
let user = {
firstName: "Karan",
sayHi( ) { console.log(this) }
}
 	
setTimeout(function ( ) {
user.sayHi( )
}, 1000);
		Intermediate		Function Invocation Type And Method Invocation Type		**Output: Window object both in strict and non-strict mode, Object**
						
						Reason: The callback fucntion will treat user.sayHi as function invocation instead of method invocation.
						
						NOTE: this is won't affected in setTimeout it will always point to window object !!!
								REASONING

09.
 	
let user = {
firstName: "Batman",
sayHi( ) {
console.log(this);
}
};
 	
setTimeout(( ) => user.sayHi( ), 1000);
 	
user = {
firstName: "Ironman",
sayHi( ) { console.log(this) }
};
		Advance		Method Invocation Type		**Output: { firstName: "Ironman", sayHi : function () {})**
						
						Reason: Simply we changed the value of user.
						
						NOTE: In setTimeout we are not passing the reference of user we are simply accessing the value of user		

10.
 	
function func ( ) {
console.log(this);
}
 	
let user = {
gun: func.bind(null)
};
 	
user.gun( );
 	
If you use 'use strict' at the top of the function
It will become this = null !
		Advance		Bind Invocation		**Output: Window object in non-strict mode and null in strict mode;**
						
						Reason: The value of this can't be string/number or null in non strict mode.
						
						In strict mode it can have any value.		

11.
 	
let user = {
firstName: "Karan",
sayHi ( ) { console.log(this) }
}
 	
user.sayHi.bind({name: "negi"})
user.sayHi.bind({name: "abc"})
 	
user.sayHi( )
		Beginner		Method Invocation		**Output: user Object**
						
						Reason: We are simply accessing the sayHi function of an object
						We are not using bind invocation as bind invocation requires the returned bind function to be called.
						
						i.e user.sayHi.bind({name: "negi"})( )		

12.
 	
function fun ( ) { console.log(this.number) };
 	
fun.number = 100;
 	
let bindFun = fun.bind({name: 'abc', number: 100 })
 	
console.log(bindFun.number);
		Intermediate		Bind Func Invocation		**Output: Undefined**
						
						Reason: The bind function returns a new function, Hence number property was not existed there.		

13. function sayHi (a, b, c) { console.log(a, b, c) }; let user = { name: 'Js' }; sayHi.apply(user, { 0: 'Hello', 1: "World", length: 2 }, { 0: 'Abc', 1: "Xyx", 2: "Hi", length: 3 }) sayHi.call(user, ...["Panda"]); sayHi.call(user, ...[{0:1 , 1: 2, 3:3, length:3}])		Advance		Apply Function Works		**Output: "Hello", "World", undefined** **And** **"P", "a", "n"** **And** **{ '0': 1, '1': 2, '3': 3, length: 3 } undefined undefined** **Reason**: apply function takes/accepts only array-like args !! Hence object mentioned in array-like form is acceptable here It will consider only 1st argument after this is passed as actual argument. The rest are ignored		
14. console.log([].join.call({0: 1, 1: 2, length: 3}))		Advance		Method Borrowing		**Output; '1,2,'** **Reason**: join method can accept array-like object/ iterable when called with call. Here the length is 3, Hence extra comma is at the end.		

15.
 	
function test ( ) {
console.log(this, typeof this)
}
 	
test.call("")
 	
How to preserve this
		Advance		How this behaves in Strict version		**Output: String { length: 0 } , "object"**
						
						Reason: If this is not an object i.e if it's primitive type, then this is ignored.
						
						If you want to preserve this you should have used 'use strict' at the top of the function.		

16.
 	
class User {
constructor(name) {
this.name = name;
}
 	
sayHi( ) {
console.log(this)
}
}
 	
let user = new User("karan")
user.sayHi( );
User.prototype.sayHi( );
User.sayHi( );
		Advance		Constructor Function Invocation		**Output: User { }, { sayHi: f sayHi( ), constructor: class User },**
						TypeError: User.sayHi is not a function
						
						Reason: When invoked with new , this belongs to the Class Object		

17.
 	
class Button {
constructor(value) {
this.value = value;
}
click = ( ) => {
console.log(this, this.value);
}
}
 	
let button = new Button("hello");
 	
setTimeout(button.click, 1000);
		Advance		Class Contructor
				Function Invocation
						Output: Button { click: f click( ), vaule: "hello" } ,"hello"
						
						Reason: Since the anonymous function has been passed in setTimout, and we know that anonymous function doesn't have it's own this, it will take from it's outer scope.
						
						The outer scope here is the Class Button function, Hence the value will point to "hello"		