**1. Which statement creates a new object using the Person constructor?**

**A.** var student = construct Person;

**B.** var student = new Person();

**C.** var student = construct Person();

**D.** var student = Person();

**2. When would 'results shown' be logged to the console?**

let modal = document.querySelector('#result');

setTimeout(function () {

modal.classList.remove('hidden');

}, 10000);

console.log('Results shown');

**A.** immediately

**B.** after 10 second

**C.** after 10000 seconds

**D.** after results are received from the HTTP request

**3. What is the result in the console of running the code shown?**

let Storm = function () {};

Storm.prototype.precip = 'rain';

let WinterStorm = function () {};

WinterStorm.prototype = new Storm();

WinterStorm.prototype.precip = 'snow';

let bob = new WinterStorm();

console.log(bob.precip);

**A.** Storm()

**B.** is not defined

**C.** 'snow'

**D.** 'rain'

**4. What is the result in the console of running this code?**

function logThis() {

this.desc = 'logger';

console.log(this);

}

new logThis();

**A.** {desc: "logger"}

**B.** undefined

**C.** window

**D.** function

**5. For the following class, how do you get the value of 42 from an instance of X?**

class X {

get Y() {

return 42;

}

}

var x = new X();

**A.** x.get('Y')

**B.** x.Y

**C.** x.Y()

**D.** x.get().Y

**6. Your code is producing the error: TypeError: Cannot read property 'reduce' of undefined. What does that mean?**

**A.** You are calling a method named reduce on an object that's has a null value.

**B.** You are calling a method named reduce on an empty array.

**C.** You are calling a method named reduce on an object that's declared but has no value.

**D.** You are calling a method named reduce on an object that does not exist.

**7. What is the result in the console of running the code shown?**

var start = 1;

function setEnd() {

var end = 10;

}

setEnd();

console.log(end);

**A.** 0

**B.** 1

**C.** 10

**D.** ReferenceError : end is not defined

**8. What will this code log in the console?**

function sayHello() {

console.log('hello');

}

console.log(sayHello.prototype);

**A.** an object with a constructor property

**B.** undefined

**C.** 'hello'

**D.** an error message

**9. Which method cancels event default behavior?**

**A.** stop()

**B.** cancel()

**C.** prevent()

**D.** preventDefault()

**10. Which method is called automatically when an object is initialized?**

**A.** create()

**B.** new()

**C.** constructor()

**D.** init()

**Interview Questions**

**1. What is the difference between a class and an object in JavaScript?**

**Class:** Human **Object:** Man, Woman

**Class:** Fruit **Object:** Apple, Banana, Mango, Guava wtc.

**Class:** Mobile phone **Object:** iPhone, Samsung, Moto

**Class:** Food **Object:** Pizza, Burger, Samosa

**2. Why are classes important in OOP? How do they help developers write better code?**

**3. Can you provide some examples of using inheritance in JavaScript?**

**4. What do you understand by polymorphism?**

[**https://kamer-dev.medium.com**](https://kamer-dev.medium.com/oopnin-4-ana-prensibi-encapsulation-inheritence-abstraction-polymorphism-712ed2fbac7e#:~:text=Polymorphism%2C%20(%C3%A7ok%20bi%C3%A7imlilik)%20methodlar%C4%B1n,davranacak%20%C5%9Fekilde%20bir%20esneklik%20vermektir.)

**Polymorphism (Çok Biçimlilik)**

Polymorphism, (çok biçimlilik) methodların objeye göre farklı çıktılar üretmesi veya farklı işler yapmasıdır. Yani alışageldiğimiz gibi methodlara sabit görevler vermek yerine onlara çok biçimli (polimorf) davranacak şekilde bir esneklik vermektir.



**Polymorphism Örnek 1 (Method Overriding)**

İlk vereceğim örnek klişe ama -bence- konuyu en iyi anlatan örnek.

**5. What is encapsulation?**

# Encapsulation (Kapsülleme)

Türkçe’de **kapsülleme** veya **sarmalama**olarak bilinen encapsulation prensibi class’ın property’lerini korumaya almasıdır. Bu korumayı sağlamak için private ve protected access modifier’larını (erişim belirteci) kullanırız. Eğer bir property’i tanımlarken private kullanırsak o class dışında hiçbir yerden erişim sağlanamaz. Protected kullanırsak da yalnızca subclass’lar ve aynı package’da bulunan classlar tarafından erişilir. Encapsulation prensibinin maksadı class’a ait property’lere her classın istediği gibi erişmesini ve değiştirmesini engellemektir. Erişim kısıtlanmayacaksa bile erişimi getter/setter methodları gibi yöntemlerle kontrol altına almaktır. Zaten encapsulation prensibine gözümüzün en aşina olduğu yer getter/setter methodlarıdır.