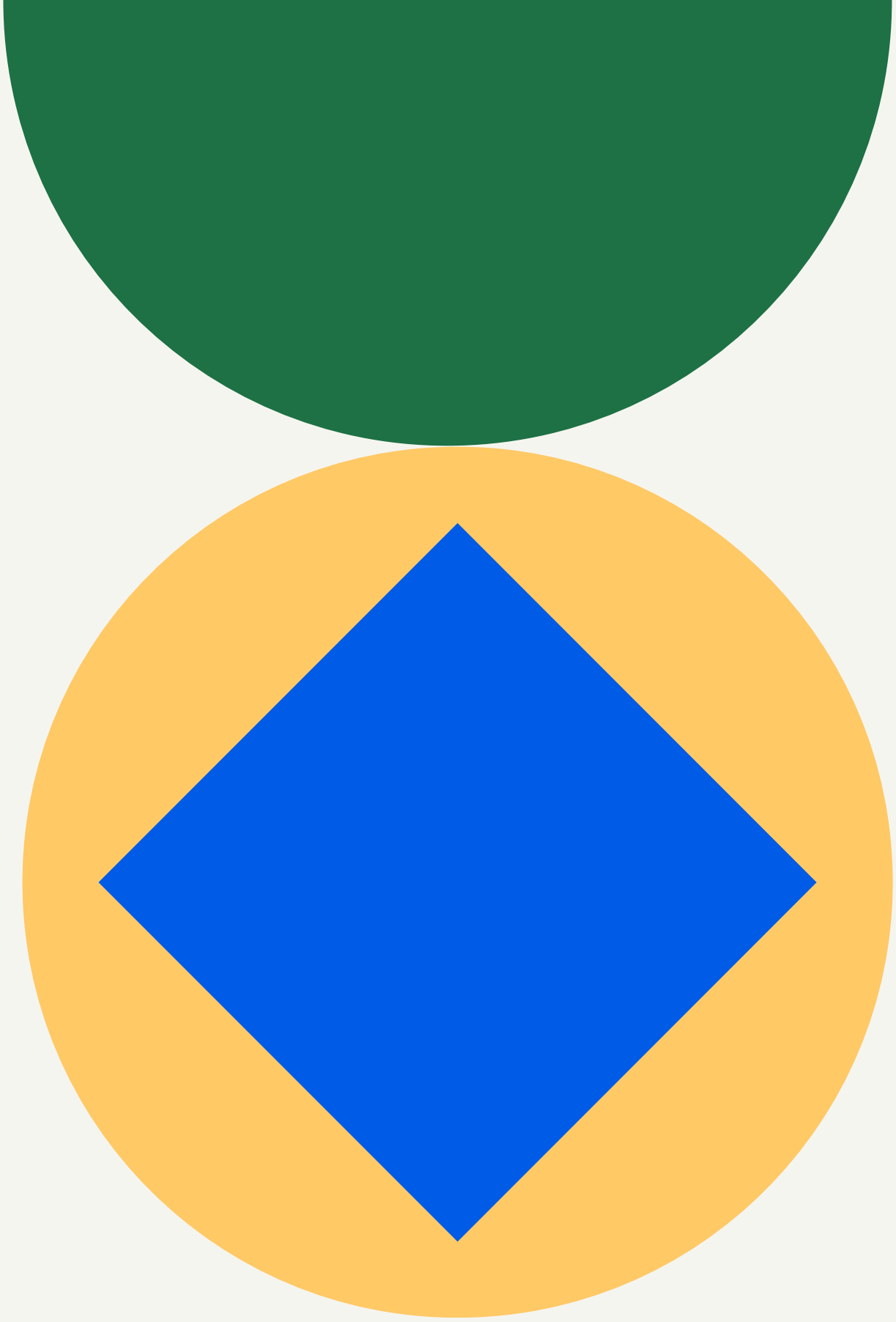


Teknik Mülakat Sorular





JavaScript nedir ? Kullanım alanları neledir?

JavaScript, web sayfalarını inaktif hale getirmek için kullanılan bir programlama dilidir.

Bu dil, istemci tarafı (client-side) web geliştirme için kullanılır. Senkron bir şekilde sunucu ile iletişime geçilmesi ve web sayfası içeriğinin değiştirilmesi gibi işlevler sağlanır. HTML ve CSS ile birlikte kullanılarak dinamik web sayfaları oluşturulmasına olanak tanır.

== ve === arasındaki fark nedir?

- == operatörü , veri türlerini önemsemeden iki değerin eşit olup olmadığını kontrol eder.
- === operatörü , aynı veri tipine sahip oldukları için iki değerin eşit olup olmadığını kontrol eder.

```
let a = [1, 2, 3]
let b = [1, 2, 3]
let c = a

console.log(a === b) // false
console.log(a === c) // true
```

`==` ve `===` arasındaki fark nedir?

- `a === b` returns **false**. This is because `a` and `b` are different objects in memory even though they look identical.
- `a === c` returns **true**. This is because `a` and `c` point to the same object in memory.

Reference types `==` işleci ile `===` işleci arasındaki fark, primitive types ile aynıdır:

- `==` operatörü , karşılaştırılan öğeleri *ortak bir veri türüne dönüştürür*. Bir Reference türünü primitive bir türle karşılaştırmaya çalışırsanız, tür zorlaması başvurusu primitive bir türe dönüştürmeye çalışır.
- `===` operatörü tür zorlamasını hiç kullanmaz. Karşılaştırılan iki değerin ikisi de Reference türü değilse, otomatik olarak false değerini döndürür .



Scope nedir?

JavaScript'de değişkenlerin kapsamı (scope) hangi bölgede tanımlandığına bağlıdır. Doğru kapsamı kullanarak, kodunuzun daha organize, daha anlaşılır ve daha az hata ile yazılması sağlanabilir.



Scope Türleri

1. Global Scope: Global kapsamdaki değişkenler, kodun herhangi bir yerinden erişilebilir ve tüm fonksiyonlar tarafından kullanılabilir.
2. Local Scope: Yerel kapsamdaki değişkenler, yalnızca tanımlandıkları fonksiyon içinde erişilebilir ve dışarıdan erişilemezler.

Scope Türleri

3. Function Scope: Fonksiyon içinde tanımlanan değişkenler, sadece o fonksiyon içinde erişilebilirler ve fonksiyon dışındaki kod tarafından erişilemezler.

4. Block Scope: Blok kapsamı, *let* ve *const* anahtar kelimeleri ile tanımlanan değişkenlerin kapsamını ifade eder. Blok kapsamı, herhangi bir blok (örneğin, *if*, *for*, *while* gibi) içinde tanımlanan değişkenlerin kapsamını ifade eder. Blok içinde tanımlanan değişkenler, sadece o blok içinde erişilebilirler ve blok dışındaki kod tarafından erişilemezler.

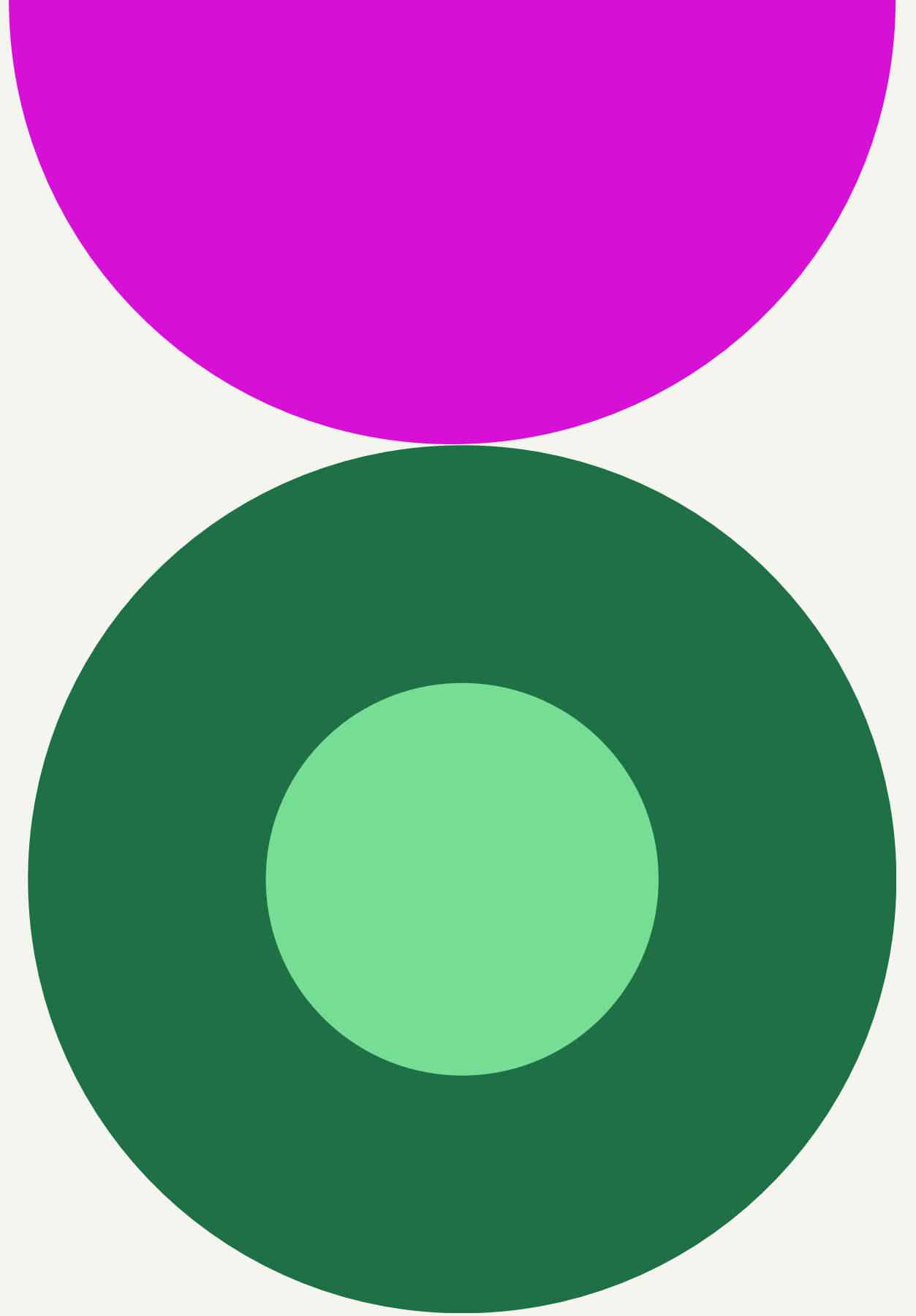
JavaScript'te olay döngüsü nedir ve nasıl çalışır?

Kullanıcının bir web sayfasındaki etkileşimlerini yakalamak ve bu etkileşimlere yanıt vermek için kullanılan bir işlem döngüsüdür.

Bu döngü, bir web sayfasındaki olayları dinler ve olaylar gerçekleştiğinde, belirtilen eylemleri gerçekleştirir.

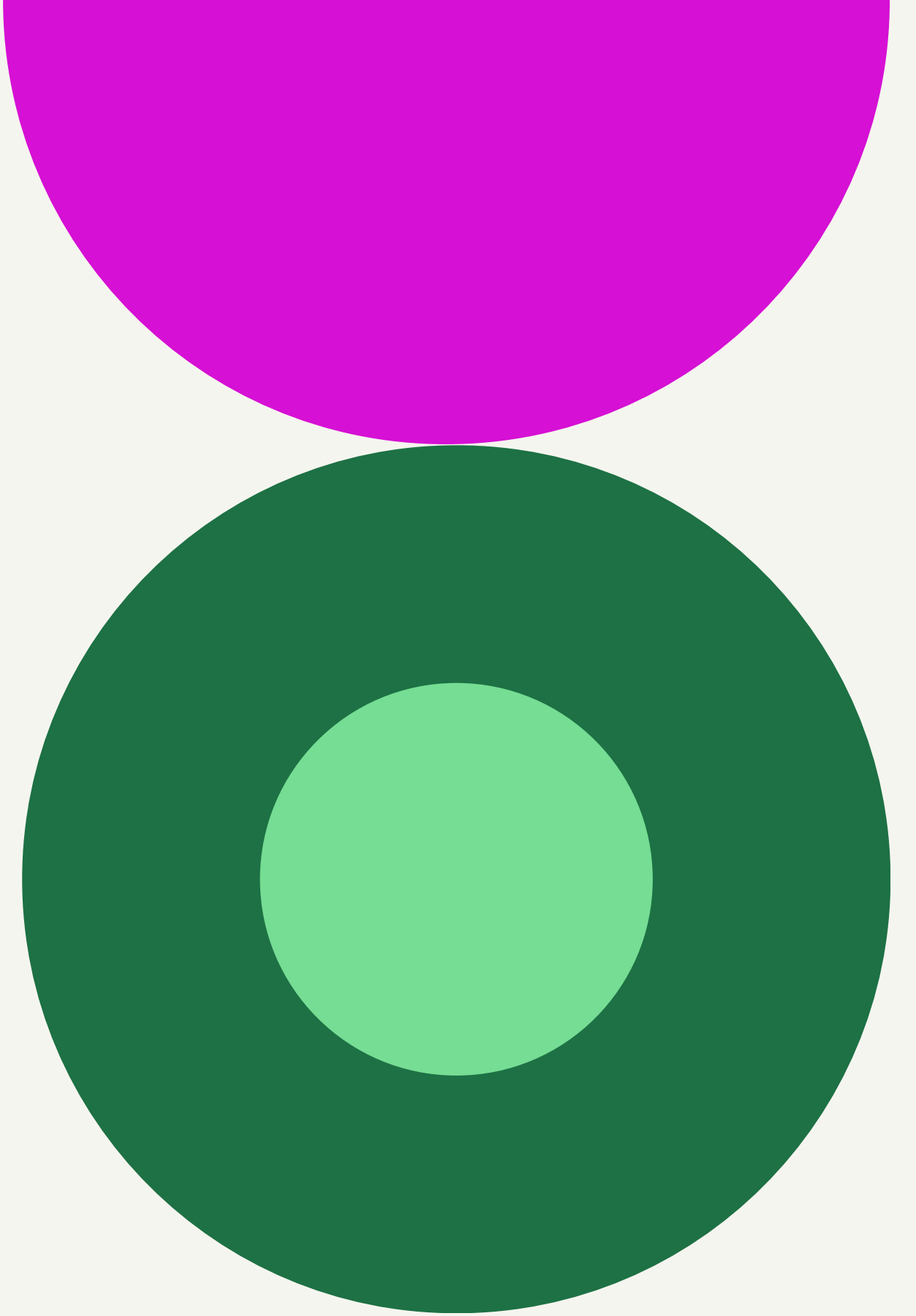
JS'de olay döngüsü, sayfa etkileşimlerine dinamik olarak yanıt vermek için kullanılır.

Bu döngü, web sayfalarını daha interaktif ve kullanıcı dostu hale getirir.



JavaScript'te, olay döngüsü şu şekilde çalışır:

1. Kullanıcı bir web sayfasında bir etkileşim gerçekleştirir.(Bu etkileşim, bir tuşa basma, fare tıklama, form gönderme veya sayfa yüklemesi gibi bir olay olabilir.)
2. Tarayıcı, olayı yakalar ve olay nesnesini oluşturur. Bu nesne, olayla ilgili bilgileri içerir (örneğin, fare tıklamasının koordinatları veya formun gönderildiği veriler).
3. Tarayıcı, olay nesnesini JavaScript motoruna iletir ve olay döngüsü başlar.
4. JavaScript motoru, olayı dinleyen tüm kodları çalıştırır. Bu kodlar, olaya yanıt vermek için yazılmış fonksiyonlar olabilir.
5. Eğer bir fonksiyon, olay nesnesindeki bilgilere göre bir eylem gerçekleştiriyorsa, tarayıcıda o eylem gerçekleşir. Örneğin, fare tıklaması bir bağlantıyı açabilir veya form gönderimi bir sayfayı yenileyebilir.
6. Olay döngüsü sona erer ve tarayıcı, sonraki etkileşim için hazır hale gelir.





JavaScript'te closure nedir ve nasıl/neden kullanılır?

JavaScript'te closure (kapanış), bir iç fonksiyonun, dış fonksiyonun kapsamındaki değişkenleri ve parametreleri yakalayabilmesi anlamına gelir. Bu, iç fonksiyonun kapsamı dış fonksiyonun kapsamına dahil edildiği için mümkündür.



JavaScript'te closure nedir? (Lexical Environment)

- JavaScript'te bir fonksiyonun bir iç fonksiyonu dışında bir kapsama (scope) ile birlikte kullanılmasıdır.
- Bu, bir fonksiyonun, iç fonksiyonun kapsamındaki değişkenlere erişebileceği anlamına gelir, ancak bu değişkenlerin dışarıdan erişilebilir olmaması anlamına gelir.



JavaScript'te closure nedir? (Lexical Environment)

- Closure, fonksiyonlara özgü bir özelliktir ve fonksiyonları daha esnek ve güçlü hale getirir.
- Bir fonksiyon içinde tanımlanan bir değişken, fonksiyonun çalışması bittikten sonra da hafızada tutulur.
- Dışarıdan erişilemez, ancak iç fonksiyonlar bu değişkenlere erişebilir.



JavaScript'te closure nedir? (Lexical Environment)

- Bunun temelinde, JavaScript'in Lexical Environment (Sözdizimi Çevresi) yapısı yatmaktadır.
- Lexical Environment, bir fonksiyonun çalıştığı ortamı temsil eder ve bu ortamda tanımlanan değişkenler ve fonksiyonlar dahil olmak üzere tüm yerel değişkenleri içerir.
- Bir iç fonksiyonun kapsamı, Lexical Environment yapısı nedeniyle, ana fonksiyonun kapsamına erişebilir.



JavaScript'te closure nedir? (Lexical Environment)

- Closure, fonksiyonların bir özellik olarak kullanılabildiği birçok farklı senaryoda kullanılabilir.
- Örneğin, bir fonksiyonun bir iç fonksiyonu aracılığıyla bir değişkeni güncellemesi veya bir olay dinleyicisi olarak kullanılması gibi durumlarda kullanılabilir



JavaScript'te closure nedir? (Lexical Environment)

- İç fonksiyonun local scope'u ile birlikte global scope da JavaScript'te kullanılan kapsamlardan biridir.
- Global scope, herhangi bir fonksiyonun içinde veya herhangi bir blokun içinde değil, en dışarıdaki kapsamı ifade eder. Global scope'ta tanımlanan değişkenlere, tüm fonksiyonlar ve bloklar tarafından erişilebilir.

JavaScript'te closure nedir? (Lexical Environment)

- Bir fonksiyon içinde tanımlanan değişkenler local scope'ta, yani fonksiyonun kapsamında kalırken, global scope'ta tanımlanan değişkenler tüm kod bloklarında ve fonksiyonlarda erişilebilir. Ancak, global değişkenlerin kullanımı, kodun anlaşılabilirliğini ve bakımını zorlaştırabilir ve global değişkenlerin yanlışlıkla başka yerlerde değiştirilmesine neden olabilir. Bu nedenle, global değişken kullanımı, mümkün olduğunca sınırlanmalı ve yerel değişkenler tercih edilmelidir.

Closure kullanmanın birkaç nedeni şunlar olabilir:

- **Gizlilik:** Closure, bir fonksiyonun kapsamındaki değişkenleri ve fonksiyonları dışarıdan erişilemez hale getirir. Bu nedenle, closure, özel bir bilgi veya işlevselliği saklamak için kullanılabilir. Bu, bir değişkenin veya fonksiyonun belirli bir kapsamda saklanması ve yalnızca ihtiyaç duyulduğunda kullanılabilmesi anlamına gelir.
- **Kapsam yönetimi:** Closure, değişkenlerin ömrünü yönetmek için kullanılabilir. Örneğin, bir değişkenin ömrünü yalnızca bir fonksiyon çalıştığı sürece sınırlamak mümkündür.



Closure kullanmanın birkaç nedeni şunlar olabilir:

- Fonksiyonlar arası veri paylaşımı: Closure, bir fonksiyondan diğerine veri aktarmak için kullanılabilir. Bir fonksiyondan diğerine veri aktarmanın başka yolları da vardır, ancak closure, bu işlemi güvenli ve ölçeklenebilir bir şekilde yapmak için kullanılabilir.
- Callback fonksiyonları: Callback fonksiyonları, bir fonksiyonun başka bir fonksiyona parametre olarak geçirilmesi ve çağırılmasıdır. Callback fonksiyonlarının kullanımı, closure'ların kullanımıyla birleştirildiğinde, asenkron programlama için çok güçlü bir araç haline gelebilir.



Closure kullanmanın birkaç nedeni şunlar olabilir:

- Bellek yönetimi: Closure, gereksiz bellek tüketimini önlemek için kullanılabilir. Örneğin, bir değişkenin kullanımının tamamlanması durumunda bellekteki yerini temizlemek için closure kullanılabilir.
- Closure'lar, JavaScript'in daha gelişmiş ve güçlü programlama özelliklerini kullanarak daha ölçeklenebilir ve yönetilebilir kodlar oluşturmanıza olanak tanır.

Bu örnekte, greeting fonksiyonu, name adlı bir parametre alır ve message adlı bir değişken tanımlar. Fonksiyon, içindeki displayName fonksiyonunu döndürür. displayName fonksiyonu, dış fonksiyonun kapsamındaki name ve message değişkenlerine erişebilir ve bu değişkenleri kullanarak bir mesaj yazdırır.

Bu örnekte closure kullanarak, greeting fonksiyonu her çağrıldığında farklı bir name parametresi alabilir ve her seferinde farklı bir sayHello fonksiyonu döndürebilir. Bu, fonksiyonların özelleştirilmesine ve tekrar kullanılmasına olanak tanır.

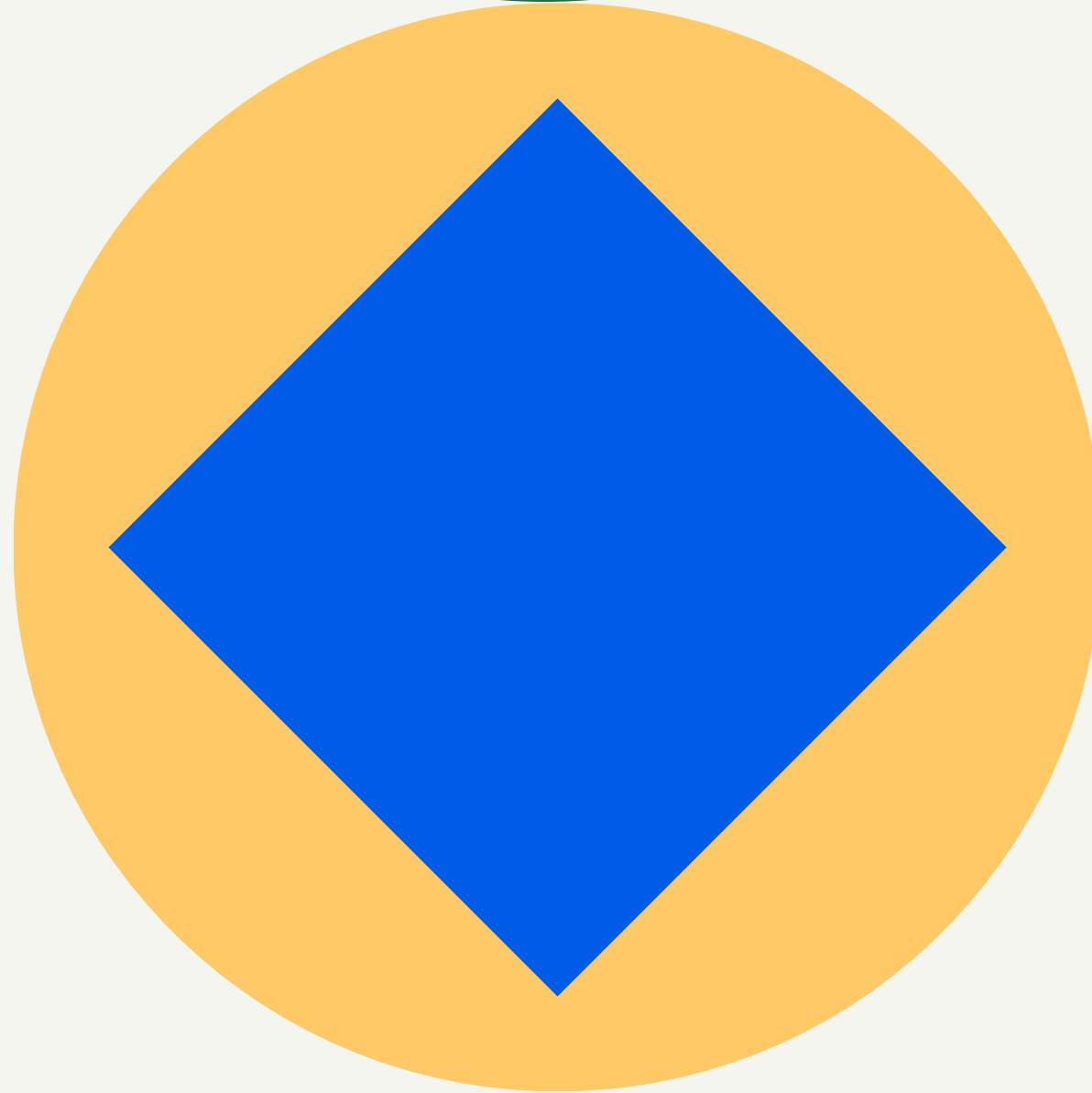
```
function greeting(name) {  
    var message = "Hello, ";  
  
    function displayName() {  
        console.log(message + name);  
    }  
  
    return displayName;  
}  
  
var sayHello = greeting("John");  
sayHello(); // Hello, John
```

JavaScript'te null ve undefined arasındaki fark nedir?

JavaScript'te undefined, bir değişkenin bildirildiği ancak henüz bir değer atanmamış olduğu anlamına gelir. null ise bir atama değeridir, değişkene değersiz bir gösterim olarak atanabilir. typeof null bize nesne döndürür.



JavaScript'te synchronous ve asynchronous kod arasındaki fark nedir?

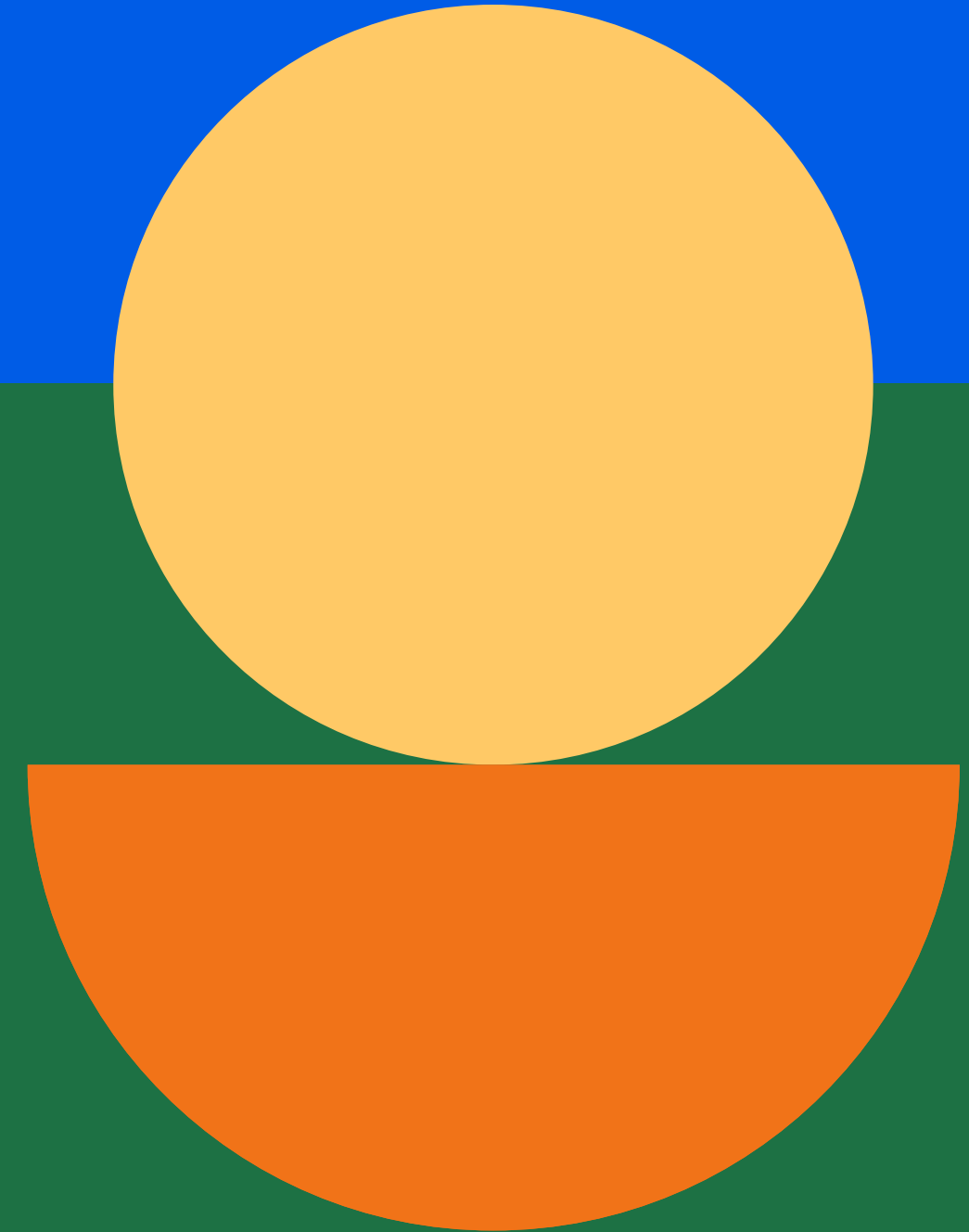


- Synchronous kod blokları, kod satırı satırı çalıştırılır ve bir satırın tamamlanması beklenirken, asynchronous kod blokları, kodun devam ederken diğer işlemlerle birlikte çalışır. Synchronous kod blokları, kod sırasına uygun bir şekilde çalışırken, asynchronous kod blokları, kodun çalışması bitmeden önce başka bir işlem yapmak için bir fonksiyonun çağırılmasına izin verir.
- Asynchronous kod genellikle callback fonksiyonları veya Promise objeleri kullanılarak gerçekleştirilir. Bu yöntemler, bir işlem tamamlandığında veya bir hata oluştuğunda çağrılacak fonksiyonları belirtmenizi sağlar.
- Synchronous kodlar, özellikle küçük uygulamalarda, kodun okunması ve anlaşılması daha kolaydır, ancak büyük uygulamalarda veya yoğun işlem gerektiren uygulamalarda, asynchronous kodlar daha uygun olabilir.

JavaScript kodunda nasıl hata ayıklarsınız?

JavaScript kodunda hataları ayıklamak için kullanabileceğiniz bazı yöntemler vardır.

- `console.log()` fonksiyonunu belirli noktalara yerleştirerek, ilgili değişkenlerin değerlerini veya işlevlerin çağrıldığını görebilirsiniz.
- **Debugger**, kodunuzu satır satır çalıştırmanıza ve değişkenlerin değerlerini görmenize olanak tanır
- **try-catch** blokları kullanarak: Bu yöntem, programınızda bir hata oluştuğunda, hatayı yakalamak ve işlemek için kullanılır.



JavaScript geliřtirmede kaçınılması gereken bazı yaygın tuzaklar nelerdir?

- Deęiřken tanımlamalarında var kullanmak: var anahtar kelimesi, bir deęiřkeni birden çok kez tanımlamaya ve aynı isimle farklı deęiřkenler oluřturmaya izin verir. Bu, beklenmedik sonuçlara ve hatalara neden olabilir. Bunun yerine let ve const anahtar kelimelerini kullanarak deęiřkenleri tanımlamak daha doęru bir yöntemdir.



JavaScript geliřtirmede kaçınılması gereken bazı yaygın tuzaklar nelerdir?

- Global deęiřken kullanımı: Global deęiřkenler, tüm kodda erişilebilir olduğundan, bu tür deęiřkenlerin kullanımı, kodunuzda bir deęiřiklik yapmanın dięer kod parçalarını etkilemesine neden olabilir. Bunun yerine, yerel deęiřkenler kullanmak daha güvenlidir.



JavaScript geliřtirmede kaçınılması gereken bazı yaygın tuzaklar nelerdir?

- Döngülerde sonsuz döngüler oluşturmak: Döngüler, işlemleri yinelemek için kullanılır, ancak döngülerde bir hata yaparsanız, sonsuz döngüler oluşturabilirsiniz. Bu, tarayıcınızın veya bilgisayarınızın donmasına neden olabilir. Bu nedenle, döngülerde mutlaka bir çıkış koşulu belirleyin.

JavaScript geliřtirmede kaçınılması gereken bazı yaygın tuzaklar nelerdir?

- İřlevlerde yan etkiler oluřturmak: İřlevlerin yan etkileri, iřlevin bir deęiřkeni deęiřtirmesi, bir dosyaya yazması veya bir aę isteęi yapması gibi iřlemlerdir. Bu yan etkiler, programın daha öngörülemeyen bir řekilde davranmasına neden olabilir. Bu nedenle, iřlevlerin yan etki yaratmayacak řekilde tasarlanması daha iyi bir uygulama yöntemidir.



JavaScript geliřtirmede kaçınılması gereken bazı yaygın tuzaklar nelerdir?

- İç içe fonksiyon çağırmak: İç içe fonksiyonlar, kodun daha karmaşık hale gelmesine neden olabilir ve okunabilirliği azaltabilir. Bu nedenle, mümkün olduğunca basit bir fonksiyon yapısı kullanmak daha iyidir.

Bu tuzakların kaçınılması, daha güvenli, okunaklı ve güvenilir bir JavaScript kodu yazmanıza yardımcı olabilir.

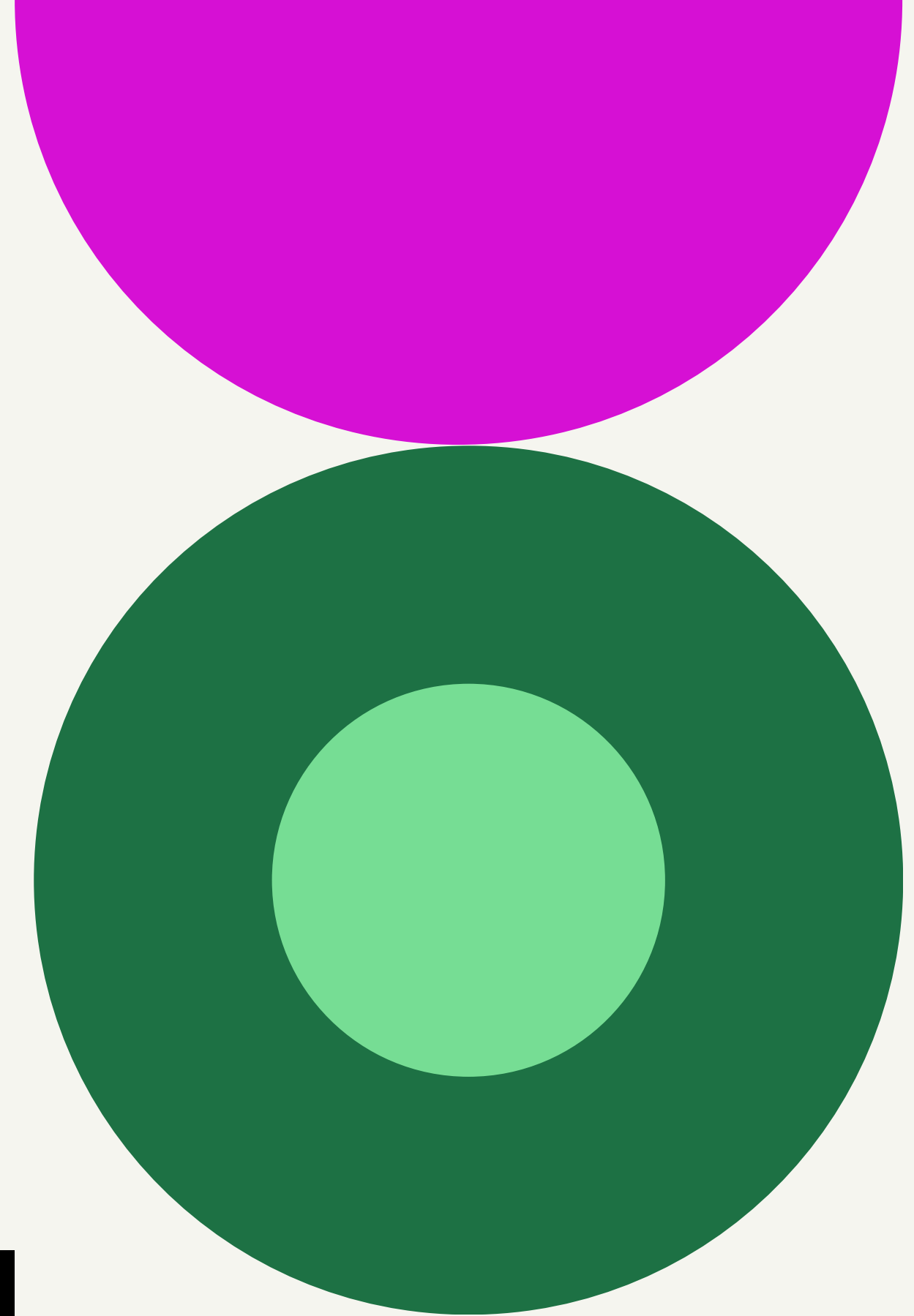
JavaScript kullanarak nasıl çerez oluşturabilirsiniz?

Çerezler, web sitelerinin kullanıcı bilgilerini tarayıcılarda depolamak için kullandığı küçük veri dosyalarıdır. Bu veriler, kullanıcıların siteye giriş yapma bilgileri, tercihleri veya gezinme geçmişi gibi çeşitli bilgileri içerebilir.

JavaScript cookie oluşturma, okuma ve silme işlemi için "document.cookie" özelliği kullanılır. Bu özellik, açılan web sayfasında tanımlı bütün çerezleri temsil eder.

Ayrıca, kullanıcı bilgilerini çerezlerde depolarken, güvenlik önlemlerini de almanız gerektiğini unutmayın.

```
document.cookie = "username=John";
```



JavaScript'de this nedir?

JavaScript'te nesnelerin olduğu gibi fonksiyonların da aldığı özellikler bulunuyor. Bir fonksiyon çağırıldığında "this" özelliğini almakta, bu "this" ise fonksiyonu çağıran nesnemizin değeridir.

this, kullanıldığı yere ve fonksiyonun çağırılma şekline göre farklı değer içermekle birlikte her zaman bir nesneyi refer eder ve genellikle fonksiyon veya metod içinde kullanılır. Fonksiyon dışında yani global scope'da da kullanılabilir ama dikkat edilmesi gereken konu; sınıfımızı strict mode'da (JavaScript'te bir kod bloğunu sıkı bir şekilde işlemek için kullanılan bir özelliktir. Katı Mod, kodun hatalarını saptamayı ve bu hataları engellemeyi amaçlar.) çalıştırıyorsak "this", undefined olacaktır.

JavaScript'de this nedir?

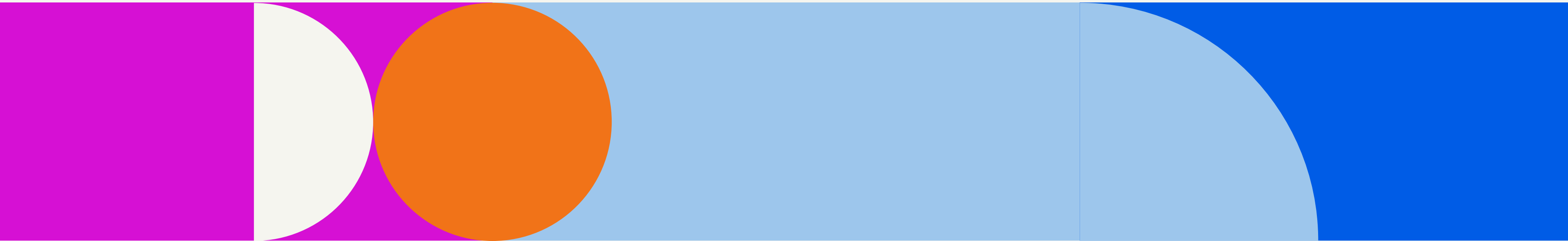
- 01 Global bağlamda: this, global nesneyi ifade eder. Browser'da bu, window nesnesi olarak ifade edilir.
- 02 Fonksiyon bağlamında: this, fonksiyonu çağıran nesneyi ifade eder.
- 03 Method bağlamında: this, methodu çağıran nesneyi ifade eder. Bir method, bir nesne içinde bir fonksiyon olarak tanımlanır ve bu nedenle this, methodun çağrıldığı nesneyi ifade eder.
- 04 Constructor bağlamında: this, yeni oluşturulan nesneyi ifade eder. Constructor, yeni bir nesne oluşturmak için kullanılan özel bir fonksiyondur.
- 05 Event bağlamında: this, olayın tetiklendiği nesneyi ifade eder.

JavaScript'te 'use strict' nedir ve nasıl etkinleştirilebilir?

JavaScript use strict anahtar kelimesi kullanım amacı kodun katı kurallı olarak çalıştırılacağını belirtir. Katı kurallı kullanımda değişken oluşturulmadan kullanmaya izin verilmez.

JavaScript kodlarının katı kurallı olarak yorumlanması için kod veya fonksiyon başına "use strict"; yazmak yeterli olacaktır.

Katı kural tanımı kod başına yazılırsa tüm kodlar katı kurallı olarak çalışacaktır.

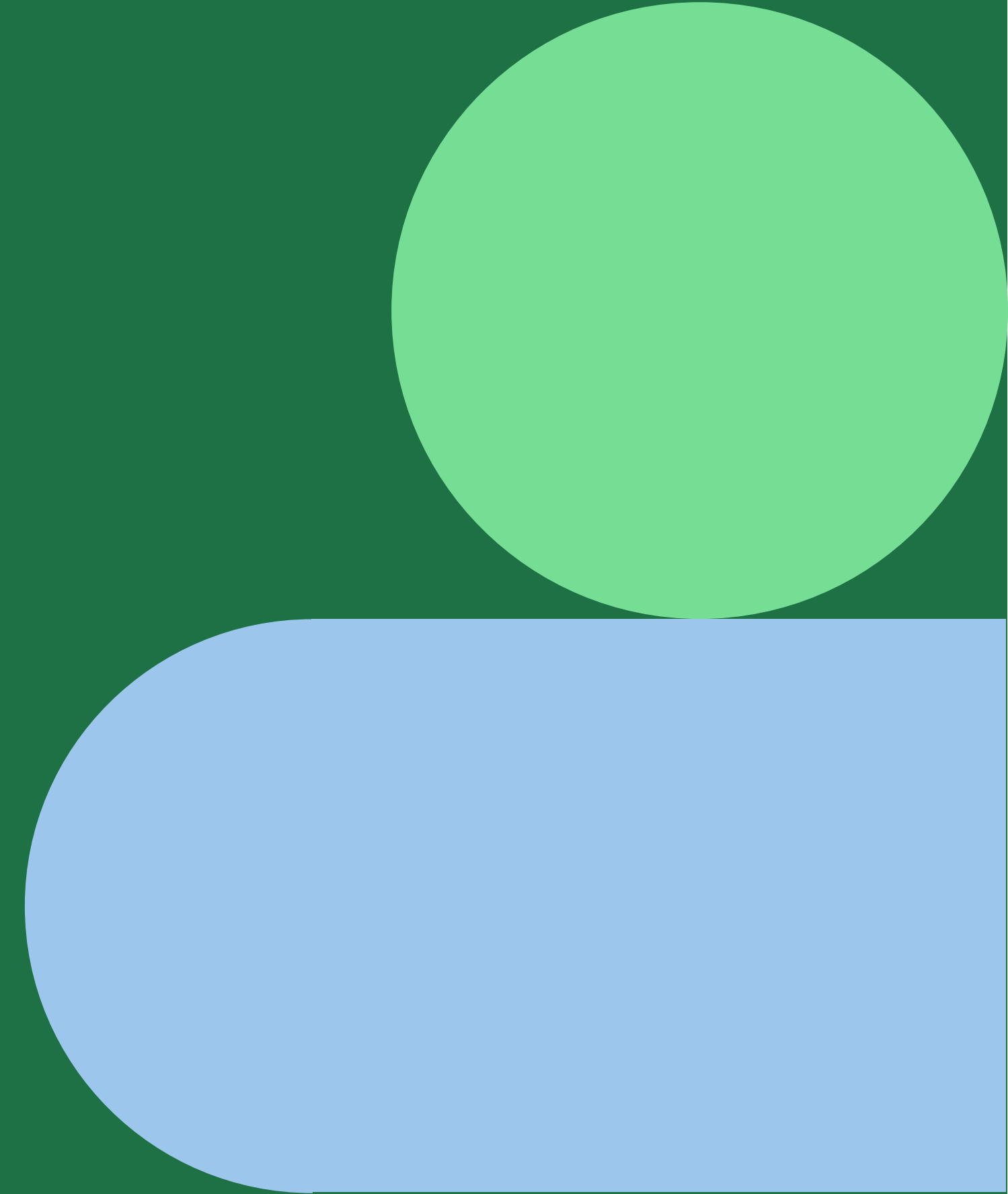


Zamanlayıcılar JavaScript'te nasıl çalışır?

Zamanlayıcılar, belirli bir zamanda bir kod parçasını çalıştırmak veya kodu belirli bir aralıkta tekrarlamak için kullanılır. Bu, `setTimeout`, `setInterval` ve `clearInterval` işlevleri kullanılarak yapılır.

`SetTimeout` (fonksiyon, gecikme) işlevi yukarıda bahsedilen bir gecikmeden sonra, belirli bir fonksiyonu çağıran bir zamanlayıcı başlatmak için kullanılır. `SetInterval` (fonksiyon, gecikme) yapılması ne zaman fonksiyon defalarca belirtilen gecikme sadece duraklamalara verilen işlevini yürütür. `ClearInterval` (id) işlevi durağına zamanlayıcı talimatını verir.

Zamanlayıcılar tek bir iş parçacığı içinde çalıştırılır ve bu nedenle olaylar kuyruğa girerek yürütülmeyi bekler.





ViewState ve SessionState arasındaki fark nedir?

ViewState, oturumdaki bir sayfaya özel olarak kullanılırken;

SessionState, tüm web uygulaması sayfalarında erişilebilen kullanıcıya özel verilere özeldir.

Void (0) kullanımı nedir?

Sayfanın yenilenmesini önlemek için Void (0) kullanılır.

void anahtar kelimesi, bir ifadeyi değer döndürmeden işlemek için kullanılır. 0 ifadesi de bir değerdir, ancak burada void anahtar kelimesi kullanıldığından değer döndürmez ve bu nedenle kullanılan 0 değeri önemsizdir.

Bu ifade, tarayıcıda mevcut sayfayı yenilemeden bir bağlantıya veya düğmeye tıklanmasına izin verir.





let-const-var farkı

- **var** anahtar kelimesi, aynı isimde birden fazla değişken tanımlanmasına izin verirken, **let** ve **const** anahtar kelimeleri aynı isimde sadece bir kez tanımlanabilir.
- **var** anahtar kelimesi, fonksiyon kapsamında veya global kapsamda tanımlanan değişkenlerde kullanılabilirken, **let** ve **const** anahtar kelimeleri blok kapsamında tanımlanan değişkenlerde kullanılır.

let-const-var farkı

- **let** anahtar kelimesi değişkenlere tekrar atama izni verirken, **const** anahtar kelimesi atama yapıldıktan sonra değişkenin değerinin değiştirilmesine izin vermez. **const** ile tanımlanan değişkenler sabit olarak kabul edilir ve değiştirilemez.
- **let** ve **const** anahtar kelimeleri daha güvenli kod yazmaya yardımcı olurken, **var** anahtar kelimesi daha esnek bir yapıya sahip olabilir.

- Promise bir işlemin sonucunu temsil eden bir Javascript nesnesidir.
- Promise nesnesi **Resolve** ve **Reject** adında iki tane parametre alır ve **olumlu** durumlarda Resolve ile belirtilen işlemlerin, **olumsuz** durumlarda da Reject ile belirtilen işlemlerin yapılacağına dair güvence verir.

Promise Nedir?



event loop nedir?

- JavaScript'de event loop, programın asenkron işlemlerini yöneten bir mekanizmadır. Event loop, JavaScript'in çalışma mantığında büyük bir rol oynar ve programın asenkron işlemlerinin düzgün bir şekilde çalışmasını sağlar.
- event loop, JavaScript'in asenkron işlemlerini yöneten bir mekanizmadır ve call stack, task queue ve microtask queue arasındaki etkileşim ile çalışır.



event loop' un temel bileşenleri:

1-Call stack: Programın çalıştığı sırayla her bir fonksiyonun çağrıldığı ve sonlandırıldığı bir veri yapısıdır.

Call stack'in bir fonksiyonu tamamlaması, bir sonraki fonksiyonun çalıştırılması için beklemesi gerektiği anlamına gelir.



event loop' un temel bileşenleri:

2-Task queue: Asenkron olarak işletilecek olan işlemlerin kuyruğu olarak düşünülebilir. Bu işlemler genellikle kullanıcı etkileşimleri, ağ istekleri gibi işlemlerdir.

İşlemler, event loop tarafından kontrol edilen bir kuyruğa eklenir ve call stack'in boş olması durumunda çalıştırılır.



event loop' un temel bileşenleri:

3-Microtask queue: Task queue'den farklı olarak, event loop içinde daha öncelikli olarak işletilen işlemlerin bulunduğu bir kuyruktur.

Bu kuyruk, Promise işlemleri ve MutationObserver gibi belirli işlemleri içerebilir. Microtask queue'deki işlemler, task queue'deki işlemlerden daha önceliklidir ve daha hızlı işlenir.

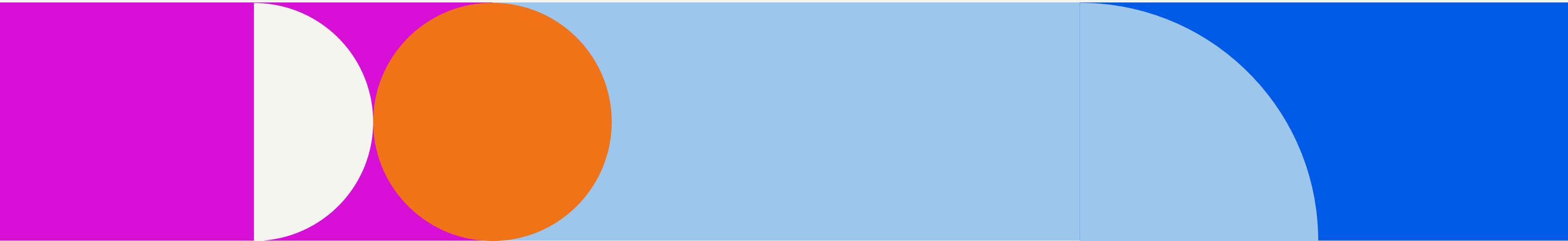


event loop,

- öncelikle call stack, task queue ve microtask queue arasındaki etkileşim ile çalışır.
- Call stack'deki bir işlem tamamlandığında, event loop task queue'ya bakar ve task queue'daki işlemlerden birini call stack'e ekler. Eğer task queue'da işlem yoksa, microtask queue'ya bakar ve microtask queue'daki işlemleri call stack'e ekler.
- Bu süreç, sürekli olarak devam eder ve programın asenkron işlemleri düzenli bir şekilde çalışır.

Hoisting nedir?

Hoisting, JavaScript'te deęişken ve fonksiyonların tanımlanmasının, kodun çalıştırılması öncesinde gerçekleştięi bir davranıştır. Bu nedenle, bir deęişken ya da fonksiyon, kodda tanımlanmadan önce de kullanılabilir. Ancak, hoisting sadece tanımlamanın yukarı taşınması (hoist) anlamına gelir, deęer atanması yukarı taşınmaz.



Bu kod çalıştırıldığında, undefined çıktısı alınır. Çünkü x değişkeni, tanımlanmadan önce kullanıldığı için hoisting işlemi ile tanımlanmış olsa da henüz bir değer atanmamıştır. Eğer değişken let veya const anahtar sözcükleriyle tanımlansaydı, hoisting işlemi gerçekleşmezdi.

Hoisting aynı şekilde fonksiyonlar için de geçerlidir. Bir fonksiyon tanımlandığında, fonksiyonun tamamı kodda yukarı taşınır ve bu nedenle fonksiyon, tanımlandığı yerden önce kullanılabilir.

Hoisting, JavaScript'in davranışlarından biri olduğu için, kodun anlaşılabilirliğini ve bakımını zorlaştırabilir. Bu nedenle, değişken ve fonksiyonların tanımlanmasının kodun başında yapılması, hoisting'in neden olduğu olası sorunları önlemek için iyi bir pratiktir.

```
console.log(x); // undefined  
var x = 5;
```

Currying nedir?

Currying, fonksiyon programlamada kullanılan bir tekniktir. Bu teknik, birden fazla parametresi olan bir fonksiyonu, her bir parametreyi alan bir dizi daha küçük fonksiyona dönüştürür. Daha sonra, bu küçük fonksiyonlar, sırayla çağrılabilir ve toplam fonksiyon çalıştırılabilir.

Bu teknik, özellikle fonksiyonların bileşimi gerektiren durumlarda faydalıdır. Fonksiyonları bir araya getirerek yeni fonksiyonlar oluşturabilirsiniz.

Currying, fonksiyonları daha esnek ve modüler hale getirir ve fonksiyonların yeniden kullanılabilirliğini artırır. Ayrıca, fonksiyonların parametrelerini ayrı ayrı ele alarak test etmeyi kolaylaştırır.

```
function add(x, y) {  
  return x + y;  
}  
  
// Currying  
function curryAdd(x) {  
  return function(y) {  
    return x + y;  
  }  
}  
  
const addFive = curryAdd(5);  
console.log(addFive(3)); // 8
```

bir fonksiyonu currying kullanarak yandaki kod örneğindeki gibi dönüştürebiliriz.

- add fonksiyonu, iki parametre alırken, curryAdd fonksiyonu birinci parametreyi alır ve geriye yeni bir fonksiyon döndürür.
- Bu dönen fonksiyon ise ikinci parametreyi alır ve toplama işlemini yapar. addFive fonksiyonu, curryAdd fonksiyonu ile 5 parametresi olarak oluşturulur ve daha sonra ikinci parametre olan 3 ile çağrılır. Sonuç olarak, addFive(3) ifadesinin değeri 8 olacaktır.

Throttling

- Bir işlevin sıklığını sınırlandırır. Belirli bir süre boyunca, yalnızca belirli aralıklarla işlev çağrılmasına izin verir.
- Bu, sayfa yüklemelerinde veya kullanıcının bir düğmeye çok hızlı tıkladığında çok yararlı olabilir.
- Bu durumlarda, işlevin çok sık çağrılması, sayfanın yavaşlamasına veya yanıt vermemesine neden olabilir.

Debounce

- Bir işlevin çağrılmasını erteler. Belirli bir süre boyunca, işlevin çağrılmasına izin vermez.
- Bu, özellikle kullanıcının bir giriş alanına veri girdiği durumlarda yararlıdır.
- Kullanıcının hızlı bir şekilde veri girişi yapması durumunda, işlevin her karakter için çağrılması yerine, kullanıcı veri girişini tamamladıktan sonra işlev çağrılabilir.



Throttling ve Debounce nedir? Arasındaki farklar nelerdir?

- JavaScript'te sıklıkla kullanılan iki tekniktir ve her ikisi de bir işlevin sıklığını kontrol etmek için kullanılır. İkisi arasındaki temel fark, hangi durumda işlevin çağrılacağıdır.
- Throttle ve debounce, bir işlevin sıklığını kontrol etmek için kullanılan tekniklerdir.
- Throttle, belirli aralıklarla işlev çağrılmasına izin verirken, debounce işlevin çağrılmasını erteler.
- Throttle daha sık kullanılırken, debounce daha az sıklıkla kullanılır.



Kaynakça

DAMLA ERVA KASAL

- 1
<https://kodsozluk.com/hackerman/yaygin-javascript-mulakat-sorulari>
- 2
<https://talentgrid.io/tr/javascript-mulakat-sorulari/>
- 3
<https://medium.com/@gizemnkorkmaz/closure-a086d757da58>
- 4
<https://medium.com/frontend-development-with-js/execution-context-lexical-environment-scope-ve-clousure-anlamak-407d1dba185c>

- 5
<https://www.codingem.com/double-equals-vs-triple-equals-javascript/>