

İç içe döngüler, ızgara ve animasyon

Bir döngü bir sıra çizer. İki döngü bir tahta çizer.

Satır × sütun

Dama tahtası

frameCount ile hareket

sin ve cos

Debug

SORU

**Bir `for` ile bir sıra kare çizdik.
Satranç tahtasını nasıl çizeriz?**

8 satır var, her satırda 8 kare. 64 tane `rect` mi yazacağız?



Döngünün içine bir döngü daha

		sutun →			
		0	1	2	3
satır →	0				
	1				
	2				
	3				

Sarı hücre: satır 1, sutun 2

Izgara (grid)

Satır ve sütunlardan oluşan, hücrelere bölünmüş düzlem. Oyun tahtalarının ve seviye tasarımının temel yapısıdır.

- **Dış döngü** satırları gezer.
- **İç döngü** o satırın sütunlarını gezer.
- Konum: `rect(sutun * boyut, satır * boyut, ...)`

Önce konsolda: hücreler hangi sırayla geziliyor?

sketch.js

Kopyala

Düzenle

Büyüt

```
1 function setup() {  
2   createCanvas(200, 100);  
3  
4   for (let satir = 0; satir < 3; satir++) {  
5     for (let sutun = 0; sutun < 3; sutun++) {  
6       console.log("hücre (" + satir + "," + sutun + ")");  
7     }  
8   }  
9 }  
10  
11 function draw() {  
12   background(240);  
13 }
```

● Konsol

Çalıştır

Büyüt

```
hücre ( 0,0)  
hücre ( 0,1)  
hücre ( 0,2)  
hücre ( 1,0)  
hücre ( 1,1)  
hücre ( 1,2)  
hücre ( 2,0)  
hücre ( 2,1)  
hücre ( 2,2)
```

İç döngü, dış döngünün **her turunda** baştan başlar. Bu yüzden önce bir satırın bütün sütunları, sonra bir sonraki satır gelir.

Aynı sıra, bu kez tuvalde

● Çıktı

Çalıştır

Büyüt

	0	1	2	3	4	5
0	1	2	3	4	5	6
1						
2						
3						

`satir = 0`

dış döngü: yavaş değişir

`sutun = 5`

iç döngü: hızlı değişir

çizilen hücre: 6 / 24

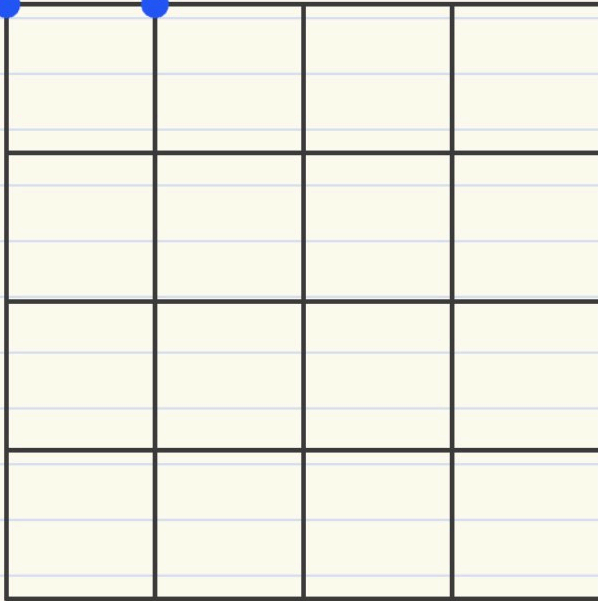
İç döngü bir satırı baştan sona bitirmeden dış döngü bir sonraki satıra geçmez. Bir kitabı okurken gözünüzün izlediği sıranın aynısı.

Takıldığınızda kâğıt kalem

Çıktı

Çalıştır

Büyüt

 $(0, 0)$ $(40, 0)$ 

Döngü başlığını oturtamıyorsanız birkaç hücrenin köşesini elle yazın (hücre 40 piksel). Önce üst satırı, sonra sol sütunu; kural kendiliğinden çıkar.

Örüntüden döngüye

sketch.js

Kopyala

Büyüt

```
1 // Takılınca: birkaç hücreyi tek tek yazıp örüntüyü görmek
2 rect(0, 0, 40, 40);
3 rect(40, 0, 40, 40);
4 rect(80, 0, 40, 40);
5 // ... satırda X +40; alt satırda Y +40, X yeniden 0'dan
6
7 // Örüntü oturunca aynı kural for ile:
8 for (let satir = 0; satir < 4; satir++) {
9   for (let sutun = 0; sutun < 4; sutun++) {
10     rect(sutun * 40, satir * 40, 40, 40);
11   }
12 }
```

Amaç kodu kâğıtta bitirmek değil, örüntüyü görünür kılmak. Bu bir zorunluluk da değil; yalnızca takıldığınızda başvuracağınız bir yol.

Sektörde de böyle çalışılır

Kernighan'ın uyarısı

Hata ayıklamak, kodu yazmaktan çoğu zaman daha zordur. Baştan karmaşık yazmak yerine adım adım ilerleyin.

Ördeğe anlat

Rubber duck debugging: kodu yüksek sesle, adım adım, sessiz bir nesneye anlatmak. Tutarsızlık anlatırken ortaya çıkar.

Kâğıt ve tahta

Stüdyolarda ve oyun jam'lerinde karalama ve gri kutu prototip yaygındır. "Satır bitti, sütun başa döndü" diye iz sürmek de bir tür debug'dır.

SORU

**Günlük hayatta “1. satır, 1. sütun”
deriz. Programda neden 0’dan
başlıyoruz?**

Konsoldaki ilk hücre (0, 0) idi. 1’den başlatsak ne değişirdi?



Çünkü formül sade kalıyor

0'dan başlayınca

Kopyala

Büyüt

```
1 for (let satir = 0; satir < 3; satir++) {  
2   for (let sutun = 0; sutun < 3; sutun++) {  
3     rect(sutun * 40, satir * 40, 40, 40);  
4   }  
5 }
```

1'den başlayınca

Kopyala

Büyüt

```
1 for (let satir = 1; satir <= 3; satir++) {  
2   for (let sutun = 1; sutun <= 3; sutun++) {  
3     rect((sutun - 1) * 40, (satir - 1) * 40, 40, 40);  
4   }  
5 }
```

1'den başlarsak her konum hesabına `- 1` eklemek gerekir. 0'dan başlayınca `sutun * 40` yeter ve `satir < 3` koşulu doğal okunur.

Aynı alışkanlığı dizilerde de göreceksiniz: ilk eleman `dizi[0]` 'dır.

8 × 8 dama tahtası: sabitler ve tuval

sketch.js · satır 1-8

Kopyala

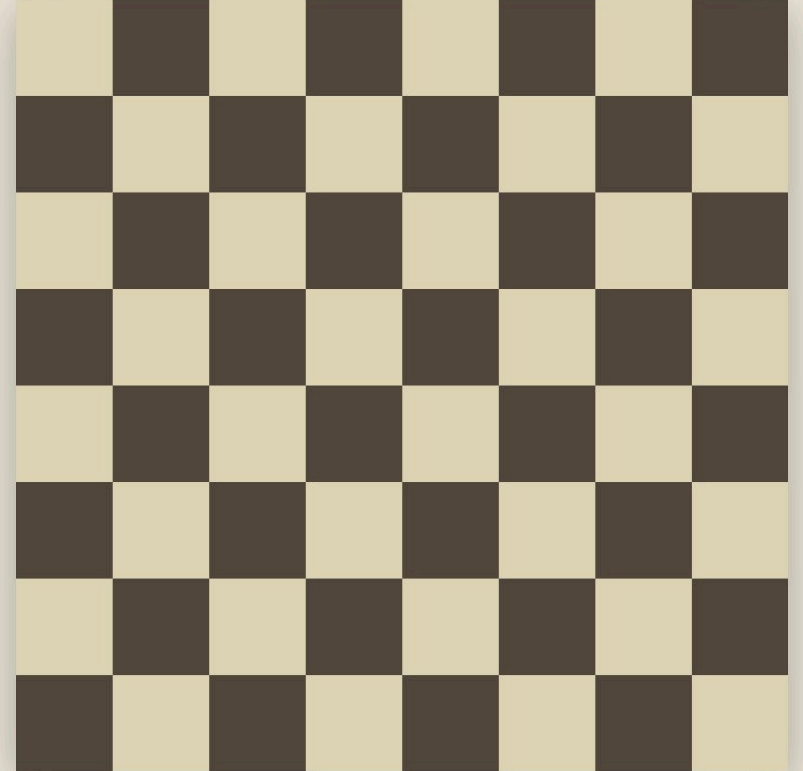
Büyüt

```
1 const HUCRE_BOYUTU = 50;  
2 const SUTUN_SAYISI = 8;  
3 const SATIR_SAYISI = 8;  
4  
5 function setup() {  
6   createCanvas(SUTUN_SAYISI * HUCRE_BOYUTU, SATIR_SAYISI * HUCRE_BOYUTU);  
7   noStroke();  
8 }
```

Çıktı

Çalıştır

Büyüt



Ölçüler en üstte, sabit olarak duruyor. Tuvalin boyutu da onlardan hesaplanıyor: $8 \times 50 = 400$.

8 × 8 dama tahtası: iki döngü

sketch.js · satır 10-24

Kopyala

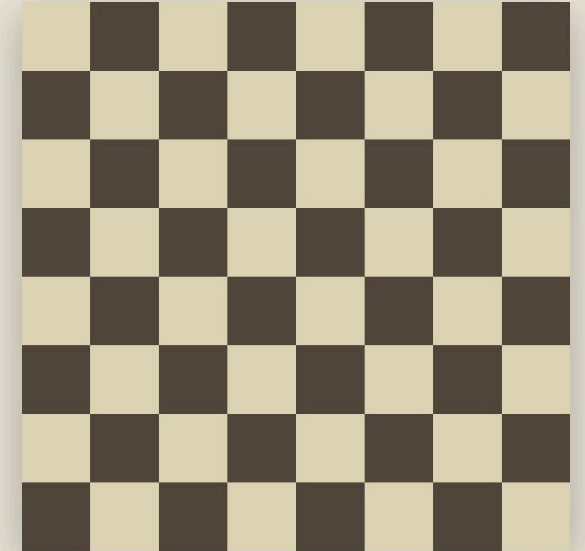
Büyüt

```
10 function draw() {  
11   background(40, 44, 52);  
12  
13   for (let satir = 0; satir < SATIR_SAYISI; satir = satir + 1) {  
14     for (let sutun = 0; sutun < SUTUN_SAYISI; sutun = sutun + 1) {  
15       // Satır + sütun toplamı tek sayıysa açık, çift sayıysa koyu kare (dama deseni)  
16       if ((satir + sutun) % 2 === 0) {  
17         fill(220, 210, 180);  
18       } else {  
19         fill(80, 70, 60);  
20       }  
21       rect(sutun * HUCRE_BOYUTU, satir * HUCRE_BOYUTU, HUCRE_BOYUTU, HUCRE_BOYUTU);  
22     }  
23   }  
24 }
```

Çıktı

Çalıştır

Büyüt



Renk nasıl dönüşümlü oluyor?

satir	sutun	toplam	% 2	Renk
0	0	0	0	açık
0	1	1	1	koyu
0	2	2	0	açık
1	0	1	1	koyu
1	1	2	0	açık

`(satir + sutun) % 2 === 0`

- Toplam çiftse kalan 0: açık kare.
- Toplam tekse kalan 1: koyu kare.
- Yan hücreye ya da alt hücreye geçince toplam 1 artar; renk değişir.

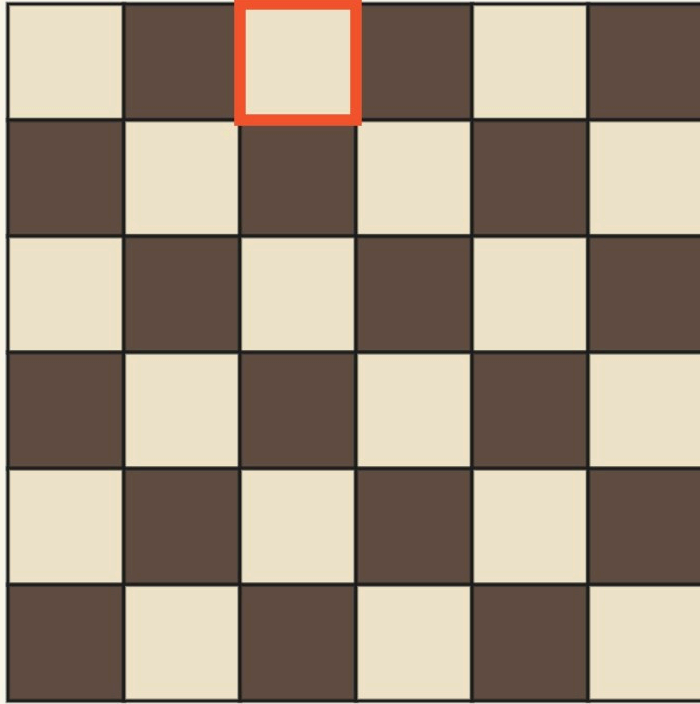
Burada üç şey birleşti: döngü, `%` ve `if / else`.

Bir hücrenin üstüne gelin

● Çıktı

Çalıştır

Büyüt



satir = 0 sutun = 2

$x = \text{sutun} * 50 = 100$

$y = \text{satir} * 50 = 0$

$(\text{satir} + \text{sutun}) \% 2$

$= 2 \% 2 = 0$

kalan 0 → açık kare

Komşu hücreye geçtiğinizde toplam 1 değişir, kalan 0'dan 1'e ya da 1'den 0'a döner. Renklerin dönüşümlü olmasının tek sebebi bu.

İç içe döngü kaç tur atar?

Dış: satır	İç: sütun	Bu satırda çizilen
0	0 → 7 (8 tur)	8 kare
1	0 → 7 (8 tur)	8 kare
...	...	...
7	0 → 7 (8 tur)	8 kare

$$8 \times 8 = 64$$

Toplam tur, dış tur sayısı ile iç tur sayısının çarpımıdır.

Uzaylı filosu: 5 satır × 11 sütun

● Çıktı

Çalıştır

Büyüt



Space Invaders'ın (1978) filosu bir ızgaradır. Tetris'in kuyusu 10 × 20, satranç tahtası 8 × 8, Mayın Tarlası'nın büyük boyu 30 × 16 hücredir; hepsi iç içe iki döngüyle çizilir.

İki döngüye aynı adı vermeyin

sketch.js

Kopyala

Büyüt

```
1 for (let i = 0; i < 3; i++) {  
2   for (let i = 0; i < 3; i++) {  
3     rect(i * 40, i * 40, 40, 40);  
4   }  
5 }
```

Gölgeleme

İç döngünün `i` 'si, dış döngünün `i` 'sini gölgeler. JavaScript hata vermez; kod çalışır ama sonuç sessizce bozulur.

Satır ve sütun için `satır` ve `sutun`, genel sayaçlar için `i` ve `j` kullanın.

İç içe döngü + `continue` : yalnızca köşegen

sketch.js

Kopyala

Düzenle

Büyüt

```
1 function setup() {  
2   createCanvas(200, 100);  
3  
4   for (let satir = 0; satir < 4; satir++) {  
5     for (let sutun = 0; sutun < 4; sutun++) {  
6       if (satir !== sutun) {  
7         continue; // diyagonal değilse bu hücreyi atla  
8       }  
9       console.log("diyagonal:", satir, sutun);  
10    }  
11  }  
12 }  
13  
14 function draw() {  
15   background(240);  
16 }
```

● Konsol

Çalıştır

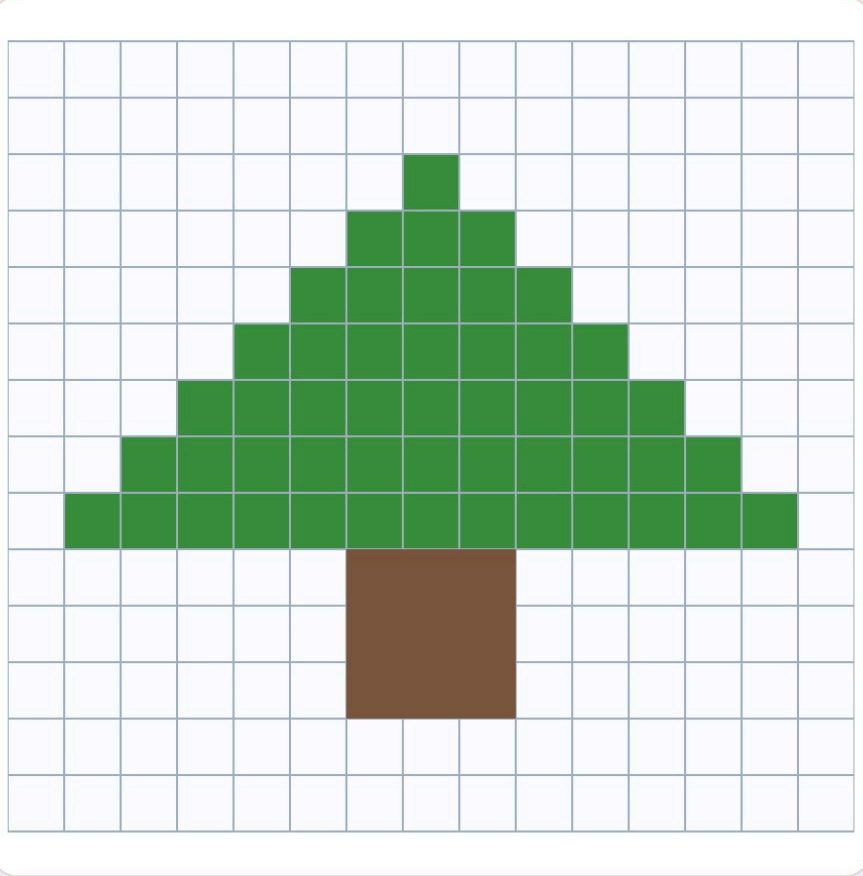
Büyüt

```
diyagonal: 0 0  
diyagonal: 1 1  
diyagonal: 2 2  
diyagonal: 3 3
```

`satir !== sutun` ise o hücre atlanır. Geriye (0,0), (1,1), (2,2), (3,3) kalır.

Hücre hücre bir çam ağacı

🕒 30 dk



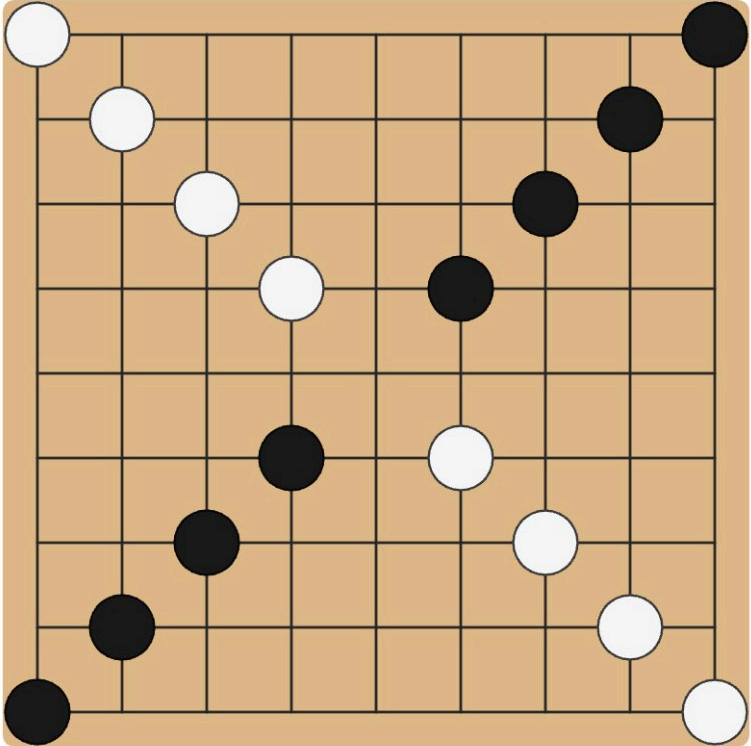
1. **Izgara:** yatay ve dikey çizgiler (`line` + `for`).
2. **Taç:** üstte ortada 1 hücre; her alt satırda sola ve sağa birer hücre büyür: 1, 3, 5... `triangle` kullanmayın; hücreleri `rect` ile boyayın.
3. **Gövde:** yeşilin altında, 3 hücre genişliğinde kahverengi bir `rect` .
4. Ağaç tuvalde yatay ve dikey ortalansın.

İpucu

Orta sütun `mid = floor(SUTUN / 2)` . Yeşil satır `r` için aralık `mid - r ... mid + r` . Sıra: ızgara → yeşil → gövde; `background` bir kez.

9 × 9 Go tahtası

🕒 30 dk



1. **Izgara:** 9 yatay + 9 dikey çizgi. Her yön için bir `for` yeter.
2. **Beyaz taşlar:** ana köşegen, `satir === sutun`.
3. **Siyah taşlar:** diğer köşegen, `satir + sutun === 8`.
4. İki köşegenin kesiştiği (4, 4) boş kalsın: `continue` ya da `if`.
5. Taşlar çizgilerin **kesişiminde** dursun; çap hücreden biraz küçük.

İpucu

`const N = 9;` ve `const BOYUT = 40;` gibi sabitler kullanın. Önce boş tahtayı çizin, sonra taş döngüsünü ekleyin.

SORU

Arkadaşınızın tahtasında çizgiler taşların üstünden geçiyor.

Beyaz taşların üstünde ızgara çizgileri görünüyor. Sizce kodda ne ters gitti?

İpucu: tuvalde üstte görünen şekil hangisidir?



SORU

Geçen bölümde tek bir daireyi kaydirdık. Beşini birden nasıl kaydırırız?

Elimizde iki araç var: aynı frame'de birçok şekil çizen `for` ve zamanı sayan `frameCount`.



Döngü çoğaltır, frameCount hareket ettirir

sketch.js

Kopyala

Düzenle

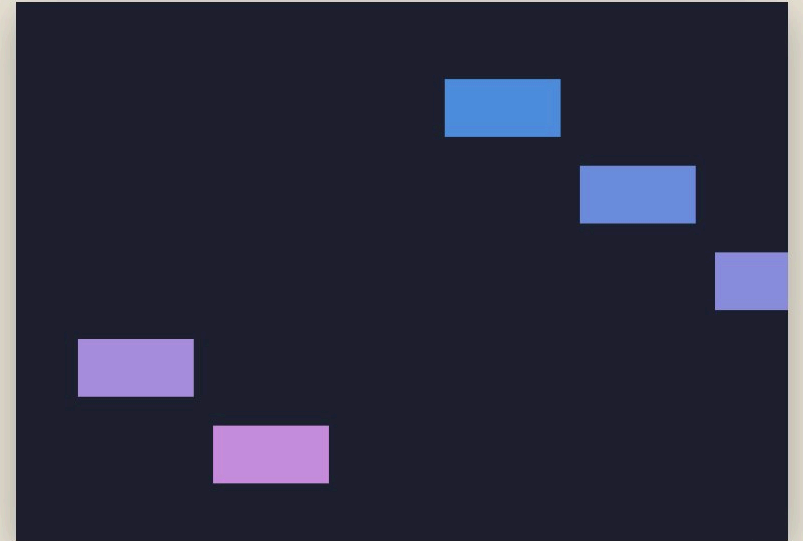
Büyüt

```
1 const SERIT_SAYISI = 5;
2
3 function setup() {
4   createCanvas(400, 280);
5   noStroke();
6 }
7
8 function draw() {
9   background(30, 34, 48);
10
11   for (let i = 0; i < SERIT_SAYISI; i++) {
12     // Her şerit farklı yatay ofsetle başlar; hepsi aynı hızda kayar
13     let x = (frameCount * 2 + i * 70) % width;
14     let y = 40 + i * 45;
15
16     fill(80 + i * 30, 140, 220);
17     rect(x, y, 60, 30);
18   }
19 }
```

Çıktı

Çalıştır

Büyüt



Formülü parçalayalım

sketch.js

Kopyala

Büyüt

```
let x = (frameCount * 2 + i * 70) % width;
```

`frameCount * 2`

Hız. Her frame 2 piksel sağa.

`i * 70`

Offset. Her şerit bir öncekinden 70 piksel ileride başlar.

`% width`

Sonucu 0 ... width - 1 arasında tutar. Kenara gelen şerit soldan yeniden girer.

Çekirdek fikir: döngü = aynı frame'de birden fazla nesne · `frameCount` = zaman.

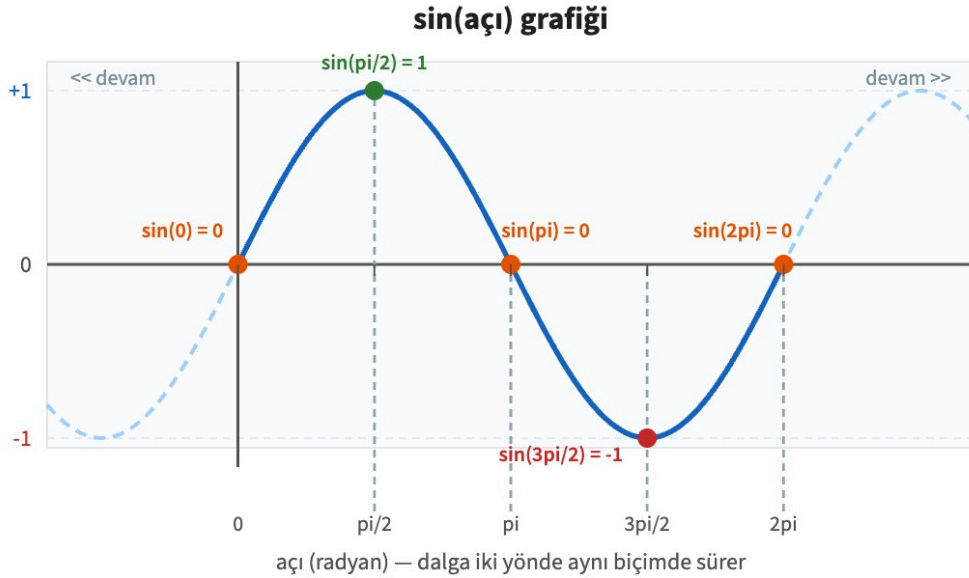
SORU

Şeritler düz gidiyor. Onları dalga gibi yukarı aşağı nasıl oynatırız?

Zamanla artıp azalan, sonra yine artan bir sayıya ihtiyacımız var.



$\sin()$: -1 ile +1 arasında salınan sayı



$\sin(\text{açı})$

Verilen açının sinüs değerini döndürür. Sonuç her zaman -1 ile +1 arasındadır; açı arttıkça değer dalga gibi yükselir ve alçalır.

Animasyonda $\sin$ 'i "zamanla -1 ... +1 arasında gidip gelen bir sayı üretici" gibi düşünün.

Sayıyı piksele çevirmek

sketch.js

Kopyala

Büyüt

```
let y = 150 + sin(frameCount * 0.05 + i) * 40;
```

Parça	Aralık	Anlamı
sin(...)	-1 ... +1	Ham salınım
* 40	-40 ... +40	Genlik: kaç piksel oynasın?
150 + ...	110 ... 190	Merkez: hangi yükseklikte?

+ i her daireye farklı bir **faz** verir: hepsi aynı anda aynı yükseklikte olmaz, dalga oluşur.

Yatay kayma + dikey salınım

sketch.js

Kopyala

Düzenle

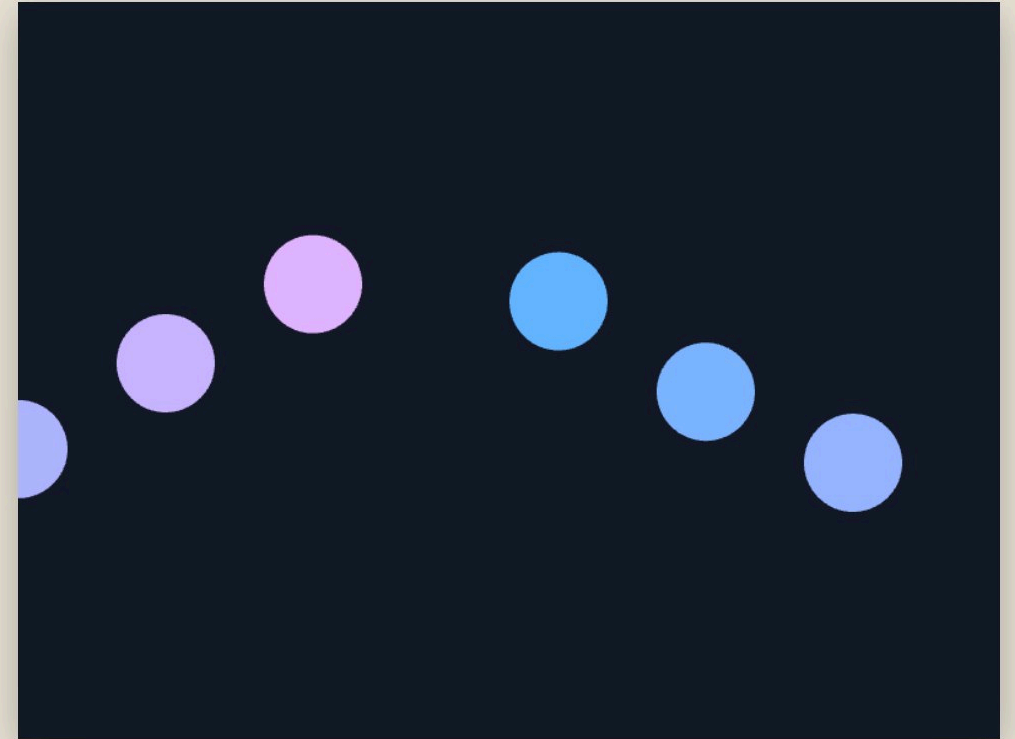
Büyüt

```
1 const DAIRE_SAYISI = 6;
2 const DAIRE_YARICAP = 20;
3
4 function setup() {
5   createCanvas(400, 300);
6   noStroke();
7 }
8
9 function draw() {
10   background(20, 25, 40);
11
12   for (let i = 0; i < DAIRE_SAYISI; i = i + 1) {
13     // Yatay: şerit örneğindeki gibi - yalnızca % ve toplama
14     let x = (frameCount * 2 + i * 60) % width;
15     // Dikey: sin ile yumuşak salınım
16     let y = 150 + sin(frameCount * 0.05 + i) * 40;
17
18     fill(100 + i * 25, 180, 255);
19     ellipse(x, y, DAIRE_YARICAP * 2, DAIRE_YARICAP * 2);
20   }
21 }
```

Çıktı

Çalıştır

Büyüt



`sin()` oyunlarda her yerde

● Çıktı

Çalıştır

Büyüt

BAŞLAMAK İÇİN TIKLA

$$y = 150 + 14 * \sin(t + i * 0.7)$$

$$\text{alfa} = 150 + 105 * \sin(t * 2)$$



$$x = 470 + 90 * \sin(t * 0.5)$$



$$\text{nefes: boy} = 1 + 0.05 * \sin(t * 1.5)$$

Süzülen altınlar, gidip gelen platform, yanıp sönen yazı, nefes alan karakter: dördü de aynı kalıp. `orta + genlik * sin(zaman)`.

Açı birimi: radyan

p5.js'te `sin`, `cos`, `rotate` açığı varsayılan olarak **radyan** kabul eder.

Sabit	Derece	Anlamı
<code>HALF_PI</code>	$\approx 90^\circ$	Çeyrek tur
<code>PI</code>	$\approx 180^\circ$	Yarım tur
<code>TWO_PI</code>	$\approx 360^\circ$	Tam tur

Dönüşüm	p5.js
Derece → radyan	<code>radians(90)</code> → $\text{PI}/2$
Radyan → derece	<code>degrees(HALF_PI)</code> → 90

`angleMode(DEGREES)` ile bütün sketch dereceye çevrilebilir; o zaman `sin(90)` yazılır. Örneklerimiz radyanla kalıyor.

SORU

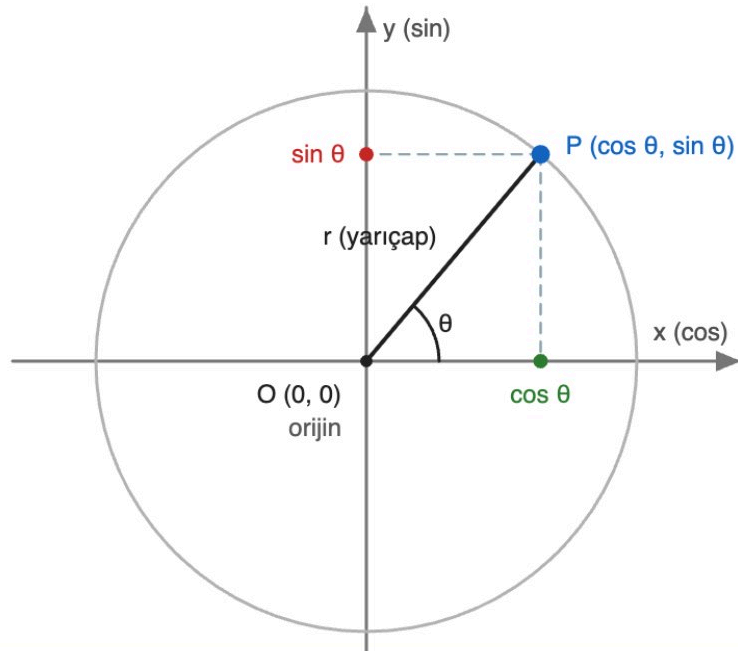
20 köşeli düzgün bir çokgeni nasıl çizeriz?

20 köşenin 20 çift koordinatını tek tek hesaplamak istemiyoruz.
Köşelerin ortak özelliği ne?



Bütün köşeler aynı çemberin üstünde

Birim çember — $\cos(x)$ ve $\sin(y)$



$$x = \text{merkezX} + r * \cos(\theta)$$

$$y = \text{merkezY} + r * \sin(\theta)$$

$r = \text{yarıçap}$ · birim çemberde $r = 1$

- Köşeler çember üzerinde **eşit açıyla** dizilir.
- Tam tur TWO_PI . 20 köşe için adım: $\text{TWO_PI} / 20$ (18°).
- i . köşenin açısı: $i * \text{adım}$.

Açıdan koordinata

$$x = \text{merkezX} + r * \cos(\text{açı})$$

$$y = \text{merkezY} + r * \sin(\text{açı})$$

$\cos$ yatay, $\sin$ dikey izdüşümü verir; ikisi de $-1 \dots +1$ arasındadır.

for ile düzgün yirmigen

sketch.js

Kopyala

Düzenle

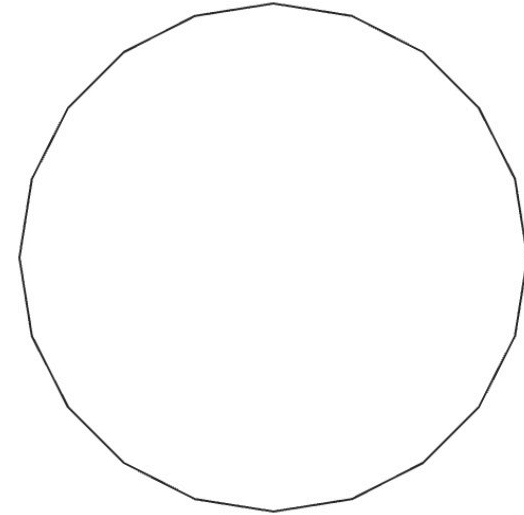
Büyüt

```
1 const KOSE = 20;  
2 const R = 120;  
3  
4 function setup() {  
5   createCanvas(400, 400);  
6 }  
7  
8 function draw() {  
9   background(255);  
10  translate(width / 2, height / 2);  
11  
12  noFill();  
13  stroke(40);  
14  beginShape();  
15  for (let i = 0; i < KOSE; i++) {  
16    let aç1 = -HALF_PI + i * TWO_PI / KOSE;  
17    let x = R * cos(aç1);  
18    let y = R * sin(aç1);  
19    vertex(x, y);  
20  }  
21  endShape(CLOSE);  
22 }
```

Çıktı

Çalıştır

Büyüt



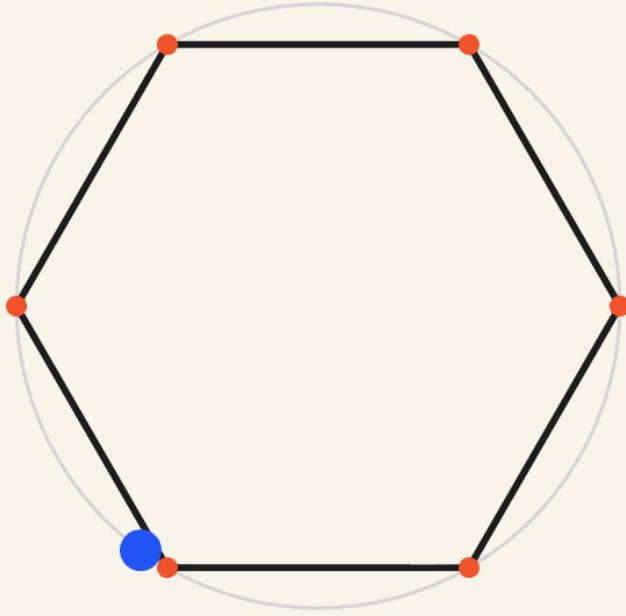
KOSE 'yi 3 yapın: üçgen. 6 yapın: altıgen. $-\text{HALF_PI}$ ilk köşeyi tepeye, saat 12 yönüne taşır.

Köşe sayısını siz seçin

● Çıktı

Çalıştır

Büyüt



köşe sayısı = 6

her adımda açı: $360^\circ / 6 = 60.0^\circ$

$x = cx + r * \cos(\text{açı})$

$y = cy + r * \sin(\text{açı})$

mavi nokta: açı her frame biraz artıyor

Fareyi sağa götürdüğçe köşe sayısı artar; çokgen çembere yaklaşır. Mavi nokta aynı formülle çemberde dolaşır: oyunlardaki yörüngeler, saat ibreleri ve daire biçimli mermi dizilimleri böyle hesaplanır.

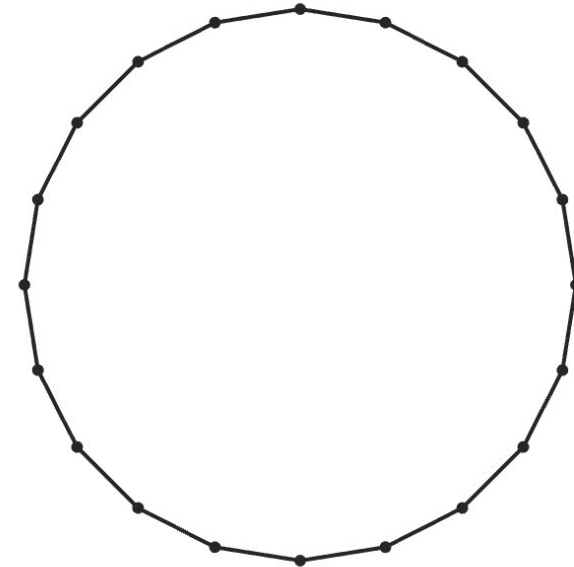
Aynı döngü `while` ile

sketch.js

Kopyala

Büyüt

```
1 let i = 0;
2 while (i < KOSE) {
3   let aç1 = -HALF_PI + i * TWO_PI / KOSE;
4   let x = R * cos(aç1);
5   let y = R * sin(aç1);
6   vertex(x, y);
7   i++;
8 }
```



20 eşit kenar. Yalnızca bir sabit değişince şekil değişir.

Örnek olay: XOX tahtası

Gelecek bölümde bir tic-tac-toe (XOX) oyunu yazacağız. 3×3 tahtayı döngülerle çizeceğiz.

1. 3×3 tahta için iç içe döngü `rect()` 'i kaç kez çağırır?
2. `HUCRE_BOYUTU` iki katına çıkarsa `createCanvas` boyutu nasıl etkilenir?
3. `frameCount` olmadan döngü yalnızca konum çeşitliliği mi sağlar, hareket mi?
4. Kayan şerit örneği ile `sin` 'li daire örneğinin ortak yanı ne?
5. Yirmigende `TWO_PI / 20` neyi temsil eder? `KOSE` 6 olursa şekil ne olur?

Yapay zekâyla: döngüden ızgara

 30-35 dk

1. 4×4 , 6×6 ya da 10×10 bir ızgara hedefi belirleyin.
2. Yapay zekâdan, `draw()` içinde iç içe `for` ile bu ızgarayı çizen kodu isteyin. Dama deseni için `(satır + sütun) % 2` kullanmasını, her satırı yorumla açıklamasını söyleyin.
3. Kodu Web Editor'da çalıştırın.
4. Hücre sayısının `satır × sütun` olduğunu `console.log` ile doğrulayın.

Debug: programı çalışırken izlemek

Debug (hata ayıklama)

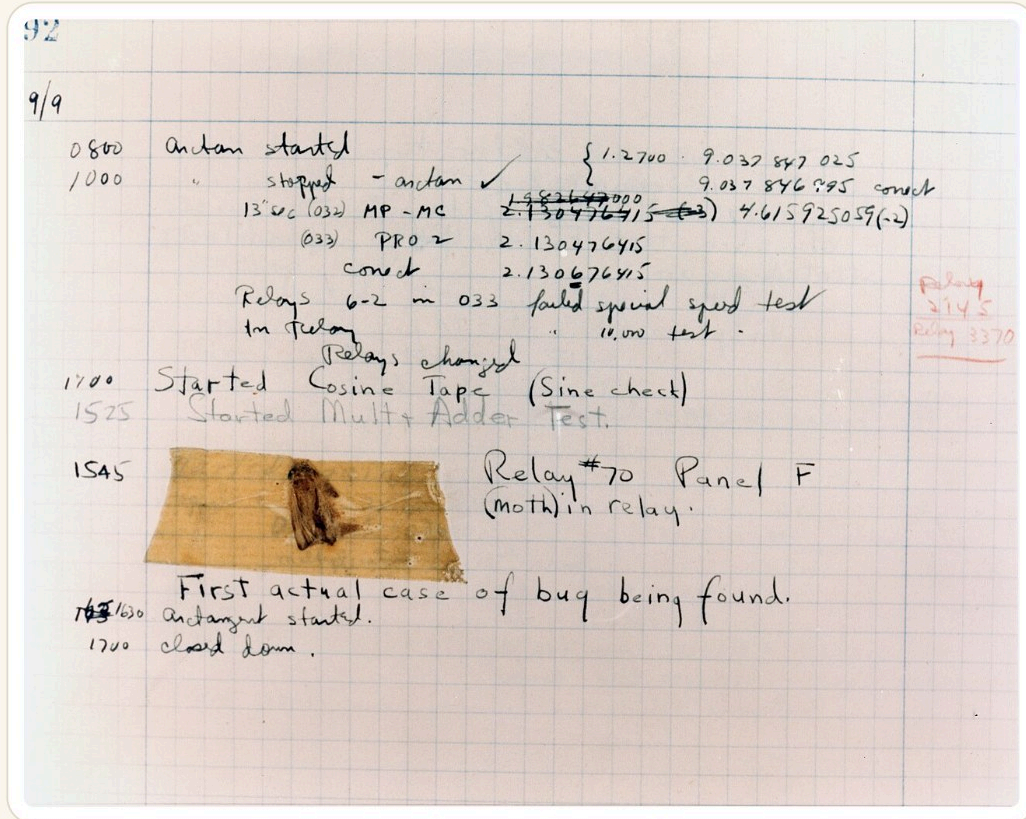
Programın çalışırken ne yaptığını kontrollü biçimde gözlemektir: nerede durdu, değişkenler o anda kaç, hangi dal seçildi?

“Çalışmıyor” demek yetmez; **neden** çalışmadığını görmek gerekir.

- p5.js tarayıcıda çalışır. Chrome, Firefox, Edge, Safari’nin **Geliştirici Araçları** (DevTools) bu işin doğal yeridir.
- **Breakpoint:** programı o satırda durdurur.
- **Step over:** bir satır ilerletir.
- **Scope / Watch:** değişkenlerin o anki değerini gösterir.

İç içe `for`’da `satır` ve `sutun`’un tur tur değişmesini izlemek, çoğu zaman `console.log` yağmurundan daha öğreticidir.

Deftere yapıştırılan "bug"



9 Eylül 1947, Harvard Mark II bilgisayarı arıza yapar.

- Ekip makineyi açar: bir rölenin arasına güve sıkışmıştır.
- Güveyi kayıt defterine bantlayıp altına not düşerler: "Gerçek bir böceğin (bug) bulunduğu ilk vaka."

Sözcük daha eskiydi

Mühendisler arızalara zaten "bug" diyordu; espri de buradan geliyor. Hatayı ayıklamaya bugün de "debug" diyoruz.

Kaynak: [Wikimedia Commons](#), ABD Deniz Kuvvetleri arşivi (kamu malı)

Lastik ördek yöntemi

Kodum çalışmıyor. Sana satır satır anlatacağım.



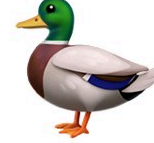
programcı, masadaki ördeğe

Burada `i` sıfırdan başlıyor, sonra iç döngüde yine `i` ... Bir dakika!



dördüncü satırda

...



ördek • tek kelime etmedi, hata bulundu

Kodu birine (ya da bir şeye) yüksek sesle anlatmak, ne yaptığınızı sandığınız ile gerçekte ne yaptığınız arasındaki farkı ortaya çıkarır.

Yapay zekâyla: tarayıcıda debug

 25-35 dk

1. Tarayıcınızı not edin. Küçük bir ızgara sketch'i açık olsun.
2. Sorun: "p5.js sketch'imi [tarayıcı] üzerinde debug etmek istiyorum. Sources / Debugger panelini nasıl açarım? Breakpoint, Step over, Step into, Continue ve Scope / Watch'u kısaca anlat."
3. En az bir breakpoint koyun; `satır` ve `sutun`'un tur tur değiştiğini görün.
4. **Hata avı:** yapay zekâdan içinde 1-2 mantık hatası olan kısa bir ızgara sketch'i isteyin; hatayı söylemesin. Debugger ile kendiniz bulun, sonra teşhisinizi doğrulatin.
5. Bir cümle yazın: "Debug bana gösterdi."

Yapay zekâyla: döngü ve animasyon

 15-20 dk

1. İç içe `for` ile 6×6 daire ızgarası isteyin: `ellipse(sutun * 60, satir * 60, 30)`.
2. Hareket ekletin: her dairenin y konumuna `sin(frameCount * 0.05 + satir + sutun) * 12` kadar kayma. Dalganın satır ve sütun boyunca ilerleyişini izleyin.
3. `sutun` yerine sabit bir sayı yazın: `ellipse(3 * 60, ...)`. Ne değişti? Aynısını `satir` için deneyin.
4. Bir cümle yazın: "Döngü değişkenini sabitleyince oldu; demek ki o değişken yi taşıyor."

Kod zamanı 30 dk

1. 10×10 bir ızgara çizin; hücrelerin rengi soldan sağa kırmızılaşsın, yukarıdan aşağı mavileşsin.
2. Dama tahtasında yalnızca köşegendeki hücreleri sarıya boyayın.
3. Kayan şeritler bu kez **yukarıdan aşağı** aksın.
4. `sin` 'li dairelerin yalnızca y'si değil, **çapı** da salınsın.
5. **Zorlu:** yirmigenin köşe sayısı her saniye 1 artsın: üçgenden başlayıp 20'ye çıksın, sonra başa dönsün.

console.log kaç kez çalışır?

sketch.js

Kopyala

Büyüt

```
1 for (let a = 0; a < 4; a++) {  
2   for (let b = 0; b <= a; b++) {  
3     console.log(a, b);  
4   }  
5 }
```

☐ A 16☐ B 12☒ C 10☐ D 4

İç döngünün sınırı dış değişkene bağlı: $1 + 2 + 3 + 4 = 10$. Çam ağacındaki üçgenin sırrı da bu.

Hücre boyutu 50. (satir = 2, sutun = 5) hücresinin sol üst köşesi nerede?

A (100, 250)

✓ (250, 100)

C (2, 5)

D (300, 150)

x sütundan, y satırdan gelir: `rect(sutun * 50, satir * 50, ...)`. İkisini karıştırmak en sık yapılan ızgara hatası.

$y = 200 + \sin(\text{frameCount} * 0.1) * 60$. **y** hangi aralıkta gezer?

A 0 ... 200

B -60 ... 60

☒ C 140 ... 260

D 200 ... 260

`sin` -1 ... +1 arasında. 60 ile çarpınca -60 ... +60; 200 eklenince 140 ... 260.

Merak edenler için kaynaklar

PATT VIRA

Nested Loop

İç içe döngüyle ızgara ve desenler.

pattvira.com > [intro-to-creative-coding](#) > [nested-loop](#)

THE CODING TRAIN

Nested Loops

İç içe döngünün adım adım kurulması.

thecodingtrain.com > [4-loops](#) > [2-nested](#)

CHROME DEVTOLS

Debug JavaScript

Kesme noktası koymak ve adım adım ilerlemek (İngilizce).

developer.chrome.com > [docs](#) > [devtools](#) > [javascript](#)

WIKİPEDIA

Rubber duck debugging

Lastik ördek yönteminin kısa hikâyesi (İngilizce).

en.wikipedia.org/wiki/Rubber_duck_debugging

Bugün ne öğrendik?

- ✓ İç içe döngüde dış sayaç satır, iç sayaç sütunu gezer; toplam tur ikisinin çarpımıdır.
- ✓ Konum: `x = sutun * boyut`, `y = satir * boyut`. O'dan saymak formülü sade tutar.
- ✓ `(satir + sutun) % 2` dama desenini üretir.
- ✓ Döngü çoğaltır, `frameCount` hareket ettirir, `% width` başa döndürür.
- ✓ `sin` -1 ... +1 arasında salınır; genlikle çarpılır, merkeze eklenir.
- ✓ Düzgün çokgen: açı adımı `TWO_PI / köşe`; köşeler `cos` ve `sin` ile bulunur.
- ✓ Takılınca: kâğıt kalem ya da tarayıcının debugger'ı.

SIRADAKİ DERS

Kodu parçalara ayırmak

Fonksiyonlar, parametreler ve return.

Sonraki sunum →

← Önceki