**Java Programcıları için ECMAScript (Javascript)**

**Oğuz KARAN**

Javascript ilk olarak Netscape firması tarafından geliştirilmiş ve ilk olarak “Livescript” ismi verilmiştir. 1997 yılında Ecma tarafından standartlaştırılmış ve “EcmaScript” ismini almıştır. Ecmascript Ecma-262 dökümanı ile standartlaştırlmıştır. Ecmascript aslında bir programlama dilidir. En çok Web ortamında kullanıldığından bir çok browser tarafından neredeyse standart kabul edilen teknolojiler eklenmiştir.

**Ecmascript Programlama Dili**

Ecmascript yorumlayıcı ile çalışan bir dildir. Herhangi bir derleme işlemi olmadığından aşağı seviyeli bir kod üretilmez. Ancak bazı yorumlayıcılar hızlı çalışma açısından ara kodlar üretebilir. Ancak programcı için ES yorumlayıcı ile çalışan bir dildir. Bu durum programcının yaptığı hataların hemen görülememesine sebep olabilir. Ecmascript çıktığından beri en çok web ortamında ön yüzde kullanılmaktadır. Ancak son yıllarda geliştirilen teknolojilerle server tarafta çalışan uygulamalar da ecmascript kullanılarak geliştirilebilmektedir.

**Türler:**

Türler şu şekildedir:

- number

- string

- boolean

- null

- undefined

- Symbol

- function

number türü IEEE 754 formatına göre sayıların tutulduğu bir türdür. Tamsayı ya da gerçek sayı ayrımı yapılmamıştır. Tüm sayılar bu formatta tutulur.

string bir değer türü olarak alınmıştır. Tüm dillerde olduğu gibi yazı işlemlerinde kullanılmaktadır.

boolean türü klasik olarak doğru yanlış işlemleri için düşünülmüştür.

null türü özellikle referans türleri için düşünülmüştür ve hiç bir nesneyi göstermeyen tür olarak algılanabilir.

undefined henüz değer almamış bir değişkenin türüdür.

Symbol türü ES6 ile eklenmiş ve sembol tanımlamakta kullanılan bir türdür.

number, string ve boolean türlerini sarmalayan (wrapper) sırasıyla Number, String ve Boolean isimli nesneler (object) bulunmaktadır.

function türü bir fonksiyonu temsil eden türdür. İçerisine fonksiyon adresi atanmış değişkenler function türündendir

**ECMAScript'de Değişkenler**

ECMAScript'de genel olarak 4 yerde değişken bildirimi yapılabilir.

1. Global olarak dosya içerisinde tüm nesne ve fonksiyonların dışında

2. Yerel değişken olarak bloklar içerisinde

3. Fonksiyonların veya metotların parametre değişkenleri olarak

4. Nesnelerin ve sınıfların veri elemanları olarak

Global değişkenler içerisinde değer atanarak doğrudan isimleriyle bildirilebilirler. Bu durumda bildirildikleri yerden o dosyanın sonuna kadar görülebilir olurlar:

**a** = 10  
  
***console***.log(**a**)

Doğrudan bildirilen değişkenlerin nerede bildirildiğinin önemi yoktur. Yani bir fonksiyon içerisinde bildirilseler bile global değişken olurlar

Global değişkenler içerisine değer atanmadan bildirilirse undefined değerini alırlar:

**var a**  
  
***console***.log(**a**)

***Anahtar notlar:*** *ES6 ile birlikte let anahtar sözcüğü ile de değişken bildirimi yapılabilmektedir.*

ES'de değişkenlerin içerisinde tutulan değerlerin türleri değişebilir:

**a** = **"ankara"** *//string***a** = 10 *//number*

***Anahtar notlar:*** *Yukarıdaki tanım daha iyi açıklamak içindir. Aşağı seviyede bu şekilde olmaz.*

Bir değişkenin yerel olması için kesinlikle var ya da let anahtar sözcükleri ile bildirilmesi gerekir:

**function** *main*()  
{  
 **var** a = **"ankara"**;  
  
 *foo*();  
}  
  
**function** *foo*()  
{  
 ***console***.log(a) // error  
}  
  
*main*()

Bu cümlenin tersi her zaman doğru değildir. Yani her var veya let olarak bildirilen değişken yerel olmayabilir.

Nesnenin veri elemanları daha sonra ele alınacaktır.

**let ve var Anahtar Sözcüğü Arasındaki Farklar**

let ile bildirilen yerel değişkenler ile var ile bildirilen yerel değişkenler arasındaki fark, let değişkenler blok faaliyet alanına sahiptir, var değişkenler fonksiyon faaliyet alanına sahiptir. Yani var değişkenler bir fonksiyon içerisinde bildirilmişlerse içteki blokta bile olsalar, bildirildikleri yerden bldirildikleri bloğa kadar görülebilirler. let değişkenler ise bildirildiği yerden bildirildiği bloğun sonuna kadar görülebilirler:

***function main()  
{  
 foo();  
}  
  
function foo()  
{  
 {  
 var a = 10  
 }  
  
 console.log(a) //error  
}  
  
main()***

***function main()  
{  
 foo();  
}  
  
function foo()  
{  
 {  
 let a = 10  
 }  
  
 console.log(a)  
}  
  
main()***

let ve var değişkenlerin global olması durumunda faaliyet alanı bakımından farkları yoktur. ES’de mümkün olduğunca let kullanılmalı. Doğrudan bildirilen değişkenlerin ise hiç kullanılmaması tavsiye edilir.

Aşağıdaki örneği inceleyiniz:

***function main()  
{  
 foo();  
}  
  
function foo()  
{  
 {  
 let a = 20   
 var a = 10 //error  
  
 }  
  
  
 console.log(a)  
}  
  
main()***

Yukarıdaki örnekte let yerine var kullanılsaydı bir error oluşmazdı:

***function main()  
{  
 foo();  
}  
  
function foo()  
{  
 {  
 var a = 20  
 var a = 10  
 }  
  
  
 var a = 30  
 console.log(a)  
}  
  
main()***

Peki sizce bu kod ne kadar anlamlı?

**Değişken İsimlendirme Kuralları**

ES’ de değişken isimlendirme kuralları Java programlama dili ile aynıdır.

***Anahtar Notlar:*** *Alttire karakterinin Java 9 ile birlikte tek başına değişken ismi olması yasaklanmıştır. Ancak ES'de bu geçerlidir.*

**Global Fonksiyonlar**

Global fonksiyonlar dosya içerisinde hiç bir nesneye/sınıfa ait olmayan fonksiyonlardır. Fonksiyon bildirimleri aynı zamanda tanımlamadır.

Global fonksiyonlar function anahtar sözcüğü ile bildirilirler.

**function** *foo*()  
{  
 ***console***.log(**"foo"**)  
}

Fonksiyonlar parametrik yapıları nasıl olursa olsun istenildiği kadar argüman ile çağrılabilirler:

*foo*()  
*foo*(20)  
*foo*(10, 20, 30)

Fonksiyonlar geri dönüş değeri olmasa da geri dönüş değeri varmış gibi kullanılabilir. Bu durumda geri dönüş değeri olarak undefined değeri elde edilecektir.

**function** *foo*()  
{  
 ***console***.log(**"foo"**)  
}

let ***a*** = *foo*(20)  
  
***console***.log(***a***)

Geri dönüş değeri için yine return deyimi kullanılır. Return deyiminin tek başına kullanılması durumunda fonksiyondan çıkılır ve değer döndürülmez:

**function** *foo*(val)  
{  
 **if** (val > 0)  
 **return  
  
 return** val \* val  
}  
  
**let *a*** = *foo*(-20)  
  
***console***.log(***a***)

Yukarıdaki örnekte fonksiyon akış durumuna göre undefined ya da argümanın karesine döner. ES kullanan programcının "her fonksiyonun zaten bir geri dönüş değeri var içeride yazılan koda göre geri dönüş değeri anlamlanacak" bakış açısıyla doğru kodu yazması gerekir. Yani fonskiyonun geri dönüş değeri olacaksa akışın her noktasında geri dönülmesi anlamlıdır.

ES'de bir fonksiyon parametre değişkeni alabilir. Bu durumda parametre değişkenlerinin yalnızca ismi yazılır:

**function** *square*(val)  
{  
 **return** val \* val  
}  
  
**let *res*** = *square*(-20)  
  
***console***.log(***res***)

Fonksiyonlara bakıldığında parametrelerin türlerinin anlaşılması her zaman mümkün olmayabilir. Bu durumda iki yaklaşım kullanılabilir:

1. Fonksiyonlar yazılırken parametreler için herhangi bir tür kontrolü yapılmaz. İyi bir dökümantasyonla kullanan programcıya bilgi verilir. Yanlış argüman geçilmesinden programcı sorumludur. Örneğin:

**function** *square*(val)  
{  
 **return** val \* val  
}  
  
**let *res*** = *square*(**"ankara"**)  
  
***console***.log(***res***)

Yukarıdaki kodda square fonksiyonuna geçilen argüman string türünden olduğu için NaN değeri elde edilir. Bu da fonksiyonu çağıranın sorumluluğundadır.

2. Fonksiyonların parametre değişkenlerinin türleri çağrılma sırasında kontrol edilebilir. Bu yaklaşım sürekli kontrol gerektirdiğinden çalışmayı yavaşlatabilir. Örneğin:

**function** *square*(val)  
{  
 **if** (**typeof**(val) != **"number"**)  
 **throw *Error***(**"unsupported argument"**)

**return** val \* val  
}  
  
**let *res*** = *square*(**"ankara"**)  
  
***console***.log(***res***)

Yukarıdaki örnekte istenen türden argüman için de kontrol yapılır. Bu da iyi bir teknik olmayabilir.

ES'de function overloading kavramı yoktur. Aynı isimde iki tane fonksiyon yazılırsa yukarıdan aşağıya son yazılanı çağrılırken geçerli kabul eder:

**function** *foo*()  
{  
 ***console***.log(**"foo1"**)  
}  
  
*foo*()  
  
**function** *foo*()  
{  
 ***console***.log(**"foo2"**)  
}  
  
*foo*()

Fonksiyonlar parametre sayısı önemli olmaksızın teorik olarak istenildiği sayıda argümanla çağrılabilirler. Bu durumda programcı fonksiyon içerisinde isterse argümanları kontrol edebilir:

**function** *main*()  
{  
 **let** result = *sum*(10, 20, 30)  
  
 ***console***.log(result)  
  
 result = *sum*(10, 20)  
  
 ***console***.log(result)  
}  
  
**function** *sum*()  
{  
 **let** result = 0;  
  
 **for** (**let** index **in *arguments***)  
 result += ***arguments***[index];  
  
 **return** result;  
}  
  
*main*()

Yukarıda, her metot içerisine iliştirilmiş olan arguments dizisi kullanılmıştır. Bu konu ileride ele alınacaktır.

**Sabitler**

Program içerisinde doğrudan yazılan değerlere sabit denilmektedir. Sabitler ES içerisinde çeşitli kategorilere ayrılmaktadır. Sabitler sayı biçimindeyse nokta içersin ya da içermesin “number” türündendir. Ayrıca boolean türden *true* ve *false* isimli sabitler de bulunmaktadır. Standartlara göre string atomları da sabit (string literal) olarak ele alınmaktadır. Sayısal sabitlerin çeşitli sayı sistemlerinde gösterilişi vardır:

**function** *main*()  
{  
 **let** a = 0xA  
  
 ***console***.log(a)  
  
 a = 0o12  
  
 ***console***.log(a)  
  
 a = 10  
  
 ***console***.log(a)  
  
 a = 0b1010  
  
 ***console***.log(a)  
}  
  
  
*main*()

ES programlama dilinde karakter sabiti yoktur. Tek tırnak içerisindeki atomlar da string türündendir. Ayrıca önceden tanımlanmış ters bölü sabitleri de (escape sequence) ES' de geçerlidir. Escape sequence karakterlerin listesi şu şekildedir:

|  |  |
| --- | --- |
| \b | Backspace |
| \t | Tab |
| \n | Line feed |
| \v | Vertical Tab |
| \f | Form feed |
| \r | Carriage return |
| \” | Double quote |
| \’ | Single quote |
| \\ | Back slash |
| \` | Backtic |

**Temel Operatörler**

Belirli bir işleme yol açan ve bir değer üreten atomlara operatör denir. Operatörler sınıflandırılabilir:

**Aritmetik Operatörler**

Aritmetik operatörler klasik 4(dört) işlem operatörleri ile işaret + ve – operatörleri ve ++ ve -- operatörleridir. Bu operatörlerin kullanımı genel olarak Java ile aynıdır. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
 **let** b = a++  
 **let** c = ++a  
  
 *writeln*(a) *//12  
 writeln*(b) *//10  
 writeln*(c) *//12*}  
  
*main*()

**Karşılaştırma Operatörleri**

Temel karşılaştırma operatörleri 8 tanedir: ==, !=, ===, !==, >, <, >=, <=

Burada eşitlik testi yapan operatörler bir çok programlama dilinden farklıdır.

== ve != operatörleri soyut karşılaştırma yaparlar. == ve != operatörleri kullanılırken otomatik tür dönüştürme kuralları uygulanır. Bu kurallar ileride ele alınacaktır.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = **"10"**;  
 **let** b = 10;  
  
 *writeln*(a == b);  
}  
  
  
*main*()

Bu operatörler kullanılırken dikkatli olunmalıdır. Tam anlamıyla eşitlik karşılaştırması (strictly equal or strictly not equal) === ve !== operatörleriyle yapılabilir. Bu operatörler karşılaştırılan türler birbirinden farklıysa kesinlikle false üretirler:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = **"10"**;  
 **let** b = 10;  
  
 *writeln*(a === b);  
}  
  
  
*main*()

Programcı öncelikle bu operatörleri düşünmeli, soyut eşitlik karşılaştırma operatörleri gerektiğinde kullanılmalıdır. Ancak === ve !== operatörlerinin fazladan kontrol yaptığı da unutulmamalıdır.

**Mantıksal Operatörler**

Mantıksal operatörlerden and ve or operatörleri klasik kısa devre davranışına sahiptir. Bu operatörler ifadenin değerini kısa devre yapılmadığında üretilen değeri değiştirmeden en kısa yoldan hesaplamaya çalışırlar. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** result = *foo*() || *bar*() && *tar*()  
  
 *writeln*(result);  
}  
  
**function** *foo*()  
{  
 *writeln*(**"foo"**);  
 **return true**;  
}  
  
**function** *bar*()  
{  
 *writeln*(**"bar"**);  
 **return false**;  
}  
**function** *tar*()  
{  
 *writeln*(**"tar"**);  
 **return true**;  
}  
  
  
*main*()

Bitsel and ve or operatörleri boolean türü ile işleme sokulduğunda kısa devre özelliği olmayan and ve or operatörleri gibi davranırlar:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** result = *foo*() | *bar*() & *tar*();  
  
 *writeln*(result);  
}  
  
**function** *foo*()  
{  
 *writeln*(**"foo"**);  
 **return true**;  
}  
  
**function** *bar*()  
{  
 *writeln*(**"bar"**);  
 **return false**;  
}  
**function** *tar*()  
{  
 *writeln*(**"tar"**);  
 **return true**;  
}  
  
  
*main*()

Burada elde edilen sonuç number türündendir.

Aslında bitsel operatörler numeric türler için bitsel işlemlerde kullanılır. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
 **let** b = 11  
 **let** c = a & b  
  
 *writeln*(**`c=**${c}**`**)  
  
 c = a | b  
 *writeln*(**`c=**${c}**`**)  
}  
  
  
*main*()

**Atama Operatörü**

Atama operatörü iki operandlı bir operatördür. Atama işleminde birinci operandın türü atanan ifadenin türüne göre değişebilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10;  
 **let** b = **true**;  
  
 *writeln*(**typeof** (a));  
 *writeln*(**typeof** (b));  
  
 a = b;  
  
 *writeln*(**typeof** (a));  
 *writeln*(**typeof** (b));  
}  
  
*main*()

ES'de ismine işlemli atama denilen bir grup operatör bulunmaktadır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10;  
 **let** s = **"11"**

a += s; *// a = a + s  
 writeln*(a)  
}  
  
*main*()

İşlemli atama operatörleri kodun yazımını ve okunabililiği basitleştirir. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
 **let** b = 20  
 **let** c = 2  
  
 a \*= b + c *//a = a \* (b + c)  
   
 writeln*(a)  
}  
  
*main*()

**Virgül Operatörü**

Bu operatör iki operandlı (binary) araek (infix) durumunda bir operatördür. Virgül operatörüne ilişkin ifadede önce birinci operandına ilişkin ifade, sonra da ikinci operandına ilişkin ifade yapılır. Operatör öncelik tablosunun en sonunda olan operatördür. Ürettiği değer ikinci operandına ilişkin ifadenin değeridir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *foo*()  
{  
 *writeln*(**"foo"**)  
  
 **return** 20  
}  
  
**function** *bar*()  
{  
 *writeln*(**"bar"**)  
 **return** 10  
}  
  
**function** *main*()  
{  
 **let** a = (*foo*(), *bar*())  
  
 *writeln*(a)  
}  
  
  
*main*()

Virgül operatörü operatör öncelik tablosunun en alt seviyesinde olduğundan yukarıdaki işlemde atama işleminin en son yapılması için parantez zorunludur. Virgül operatörünün önce birinci operandının yapılacağı garanti altındadır. Örneğin:

**function writeln(a)  
{  
 console.log(a)  
}  
  
function foo()  
{  
 writeln("foo")  
  
 return 20  
}  
  
function bar()  
{  
 writeln("bar")  
 return 10  
}  
  
function tar()  
{  
 writeln("tar")  
 return 12  
}  
  
function main()  
{  
 let a = (foo(), bar(), tar())  
  
 writeln(a)  
}  
  
  
main()**

**typeof Operatörü**

Bu operatör operandı olan ifadenin türünü string olarak üretir. Parantezli veya parantezsiz kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
  
 *writeln*(**typeof** a)  
  
 a = **"ankara"** *writeln*(**typeof**(a))  
}  
  
  
*main*()

Diğer operatörler ileride ele alınacaktır.

**Tür dönüşümleri**

|  |  |  |  |
| --- | --- | --- | --- |
| **Değer** | **Number** | **String** | **Boolean** |
| false | 0 | "false" | false |
| true | 1 | "true" | true |
| 0 | 0 | "0" | false |
| 1 | 1 | "1" | true |
| "0" | 0 | "0" | true |
| "000" | 0 | "000" | true |
| "1" | 1 | "1" | true |
| NaN | NaN | "NaN" | false |
| Infinity | Infinity | "Infinity" | true |
| -Infinity | -Infinity | "-Infinity" | true |
| "" | **0** | "" | false |
| "20" | 20 | "20" | true |
| "twenty" | NaN | "twenty" | true |
| [ ] | **0** | "" | true |
| [20] | **20** | "20" | true |
| [10,20] | NaN | "10,20" | true |
| ["twenty"] | NaN | "twenty" | true |
| ["ten","twenty"] | NaN | "ten,twenty" | true |
| function(){} | NaN | "function(){}" | true |
| { } | NaN | "[object Object]" | true |
| null | **0** | "null" | false |
| undefined | NaN | "undefined" | false |

**Kontrol Deyimleri**

Akışın kontrolü için kullanılan deyimlere kontrol deyimleri denir.

**if Deyimi**

if deyimi parantez içerisindeki ifadenin true ya da false olmasına göre akışı yönlendiren bir deyimdir. Burada parantez içerisindeki ifade boolean türden olmasa bile otomatik tür dönüşümü kurallarına göre boolean türüne dönüştürülür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = **"ankara"  
  
 if** (a)  
 *writeln*(**"Doğru"**)  
 **else** *writeln*(**"Yanlış"**)  
}  
  
  
*main*()

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
  
 **if** (a)  
 *writeln*(**"Doğru"**)  
 **else** *writeln*(**"Yanlış"**)  
}  
  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı ikinci dereceden denklemin katsayılarına göre köklerini bulup ekrana yazdıran findRoots fonksiyonunu yazınız.

**Açıklama:** Klavyeden değer okuma NodeJS’ ye özgü olduğundan klavye işlemleri burada ele alınmayacaktır. Karekök alma işlemi için Math.sqrt kullanılabilir

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *findRoots*(a, b, c)  
{  
 **let** delta = b \* b - 4 \* a \* c  
  
 **if** (delta > 0) {  
 **let** sqrtDelta = ***Math***.sqrt(delta)

**let** x1 = (-b + sqrtDelta) / (2 \* a)  
 **let** x2 = (-b - sqrtDelta) / (2 \* a)  
  
 *writeln*(**`x1=**${x1}**, x2=**${x2}**`**)  
 }  
 **else if** (delta == 0) {  
 **let** x = -b / (2 \* a)  
  
 *writeln*(**`x1=x2=**${x}**`**)  
 }  
 **else** *writeln*(**"Gerçek kök yok"**)  
}  
  
**function** *main*()  
{  
 *findRoots*(1, -3, -18)  
 *findRoots*(1, 1, 1)  
 *findRoots*(1, 4, 4)  
 *findRoots*(1.4, -3.5, -1.8)  
}  
  
  
*main*()

Diğer bir çözüm:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *findRoots*(a, b, c)  
{  
 **let** delta = b \* b - 4 \* a \* c  
  
 **if** (delta >= 0) {  
 **let** sqrtDelta = ***Math***.sqrt(delta)  
 **let** x1 = (-b + sqrtDelta) / (2 \* a)  
 **let** x2 = (-b - sqrtDelta) / (2 \* a)  
  
 *writeln*(**`x1=**${x1}**, x2=**${x2}**`**)  
 }  
 **else** *writeln*(**"Gerçek kök yok"**)  
}  
  
**function** *main*()  
{  
 *findRoots*(1, -3, -18)  
 *findRoots*(1, 1, 1)  
 *findRoots*(1, 4, 4)  
 *findRoots*(1.4, -3.5, -1.8)  
}  
  
  
*main*()

**Döngü Deyimleri**

ES'de döngü deyimleri genel olarak üçe ayrılır. Ancak ES6 ile birlikte bir döngü deyimi daha eklenmiştir:

1. while döngüleri

2. for döngüsü

3. for-in döngüsü

4. for-of döngüsü

**while Döngüleri**

while döngü deyimleri iki gruba ayrılır:

- Kontrolün başta yapıldığı while döngü deyimi:

- Kontrolün sonda yapıldığı while döngü deyimi:

while döngüsü dendiğinde genelde kontrolün başta yapıldığı while döngü deyimi anlaşılır. Kontrolün sonda yapıldığı while döngü deyimi genel olarak "do-while döngü deyimi" biçiminde söylenir.

**Kontrolün başta yapıldığı while döngü deyimi:**

Bu deyimin genel biçimi şöyledir:

while (<ifade>)

<deyim>

Bu deyimde parantez içerisindeki ifade true ise (boolean türü dışında dönüşüm işleminden sonra) döngü yinelenir. while döngü deyiminde parantez içerisinde ifadenin doğru olup olmadığına akış while döngü deyimine geldiğinde de bakılır. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}

**function** *main*()  
{  
 **let** i = 0  
 **let** n = 10  
  
 **while** (i < n) {  
 *writeln*(**`i=**${i}**`**)  
 ++i;  
 }  
}  
  
*main*()

while döngü deyimi ile n-kez yinelenen aşağıdaki gibi yazılan bir döngü bazı programcılar tarafından çok sevilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
**function** *main*()  
{  
 **let** n = 4  
  
 **while** (n--) // **while** (n-- > 0)  
 *writeln*(**`n=**${n}**`**)   
  
 *writeln*(**`after loop:n=**${n}**`**)  
}  
*main*()  
  
Yukarıdaki döngüde n değerinin döngünün sonunda eski değerinde olmadığına dikkat ediniz.

**Sınıf Çalışması:** Parametresi ile aldığı number türden bir sayının basamak sayısını döndüren getDigitsCount fonksiyonunu yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getDigitsCount*(val)  
{  
 **if** (!val)  
 **return** 1  
  
 **let** count = 0  
  
 **while** (val != 0) {  
 ++count  
 val = *parseInt*(val / 10)  
 }  
  
 **return** count  
}  
  
**function** *main*()  
{  
 *writeln*(*getDigitsCount*(123))  
}  
  
*main*()

***Anahtar Notlar:*** *ES' de number türü IEEE 754 formatında olduğundan bölme işleminden sonra sayının noktadan sonraki kısmının atılması için parseInt global fonksiyonu kullanılmalıdır. Aslında parseInt fonksiyonu yalnızca sayının noktadan sonraki kısmını atmak için kullanılmaz. Bu fonksiyon genel olarak bir türü tamsayı olarak elde etmek için kullanılabilir. Örneğin yazıdan sayı yapmak için de bu fonksiyon kullanılır.*

**Sınıf Çalışması:** Parametresi ile aldığı tamsayının tersini döndüren getReverse isimli fonksiyonu yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getReverse*(val)  
{  
 **let** reverse = 0  
  
 **while** (val != 0) {  
 reverse = reverse \* 10 + val % 10  
 val = *parseInt*(val / 10)  
 }  
   
 **return** reverse  
}  
  
**function** *main*()  
{  
 *writeln*(*getReverse*(123))  
 *writeln*(*getReverse*(-123))  
}  
  
*main*()

while döngü deyimi ile tipik sonsuz döngü kalıpları:

**while** (**true**) {  
 *//...*}

**while** (1) {  
 *//...*}

**Kontrolün sonda yapıldığı while döngü deyimi (do-while)**

Bu deyimin genel biçimi:

do

<deyim>

while (<ifade>)

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getDigitsCount*(val)  
{  
 **let** count = 0  
  
 **do** {  
 ++count  
 val = *parseInt*(val / 10)  
 } **while** (val != 0)  
  
 **return** count  
}  
  
**function** *main*()  
{  
 *writeln*(*getDigitsCount*(0))  
 *writeln*(*getDigitsCount*(-345))  
}

*main*()

***Anahtar Notlar:*** *Bir sayının basamak sayısı döngü veya yazı kullanmadan aşağıdaki gibi de bulunabilir. Aşağdaki örnekte Math sınıfının abs ve log10 metotları kullanılmıştır. Bu metotların çalışması Java/C# ile aynıdır. Ayrıca kullanılan koşul operatörü de yine Java/C# ile aynıdır:*

*function writeln(a)  
{  
 console.log(a)  
}  
  
function getDigitsCount(val)  
{  
 return val === 0 ? 1 : parseInt(Math.log10(Math.abs(val))) + 1  
}  
  
function main()  
{  
 writeln(getDigitsCount(0))  
 writeln(getDigitsCount(-345))  
}  
main()*

**for döngü deyimi**

for döngü deyimi de Java' da olduğu gibi güçlü bir döngü deyimidir. Bu deyimin genel biçimi şöyledir:

for ([1.kısım];[2.kısım]; [3.kısım])

<deyim>

Bu döngüde kısımlar zorunlu değildir. Kısımlar boş bırakılmış olsa bile iki tane noktalı virgül zorunludur.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(**`i=**${i}**`**)  
}  
  
*main*()

Kısımların çalışma mantığı Java’dakine çok benzer. 2. kısmın boş bırakılması yine sonsuz döngü anlamına gelmektedir. Ancak okunabilirlik açısından sonsuz döngü kalıbı tüm kısımların boş bırakılması biçiminde yazılır:

**for (;;) {**

**//...**

**}**

for döngü deyimi içerisinde virgül operatörü kullanılarak deyim zenginleştirilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **for** (**let** i = 0, k = 10; i < 20 && k >= 0; ++i, k -= 2)  
 *writeln*(**`(**${i}**,** ${k}**)`**);  
}  
  
*main*()

for döngü deyimi zenginleştirilebilir. Güçlü bir deyimdir. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}

**function** *main*()  
{  
 **for** (**let** i = 0, k = 10; i < 20 && k >= 0; *writeln*(**`(**${i}**,** ${k}**)`**), ++i, k -= 2)  
 ;  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi aldığı bir tamsayının asal olup olmadığını test eden isPrime fonksiyonunu yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *isPrime*(val)  
{  
 **if** (val <= 1)  
 **return false  
  
 let** halfVal = *parseInt*(val / 2)  
  
 **for** (**let** i = 2; i <= halfVal; ++i)  
 **if** (val % i == 0)  
 **return false  
  
 return true**}  
  
**function** *main*()  
{  
 **for** (**let** n = -10; n <= 100; ++n)  
 **if** (*isPrime*(n))  
 *writeln*(n)  
  
 *writeln*(*isPrime*(1000003) ? **"Asal"** : **"Asal değil"**)  
}  
  
*main*()  
  
Yukarıdaki teknik kısmen kötüdür. Aşağıdaki teoremle fonksiyon daha etkin yazılabilir.

***Teorem (Euclid):*** *Bir sayı karekökünden daha küçük olan asal sayıların hiçbirine bölünemiyorsa asaldır. Aksi durumda asal değildir.*

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *isPrime*(val)  
{  
 **if** (val <= 1)  
 **return false  
  
 if** (val % 2 == 0)  
 **return** val == 2  
  
 **if** (val % 3 == 0)  
 **return** val == 3  
  
 **if** (val % 5 == 0)  
 **return** val == 5  
  
 **if** (val % 7 == 0)  
 **return** val == 7  
  
 **for** (**let** i = 11; i \* i <= val; i += 2)  
 **if** (val % i == 0)  
 **return false  
  
 return true**}  
  
**function** *main*()  
{  
 **for** (**let** n = -10; n <= 100; ++n)  
 **if** (*isPrime*(n))  
 *writeln*(n)  
  
 *writeln*(*isPrime*(1000003) ? **"Asal" :** **"Asal değil"**)  
}  
  
*main*()  
  
Diğer bir çözüm:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *isPrime*(val)  
{  
 **if** (val <= 1)  
 **return false  
  
 if** (val % 2 == 0)  
 **return** val == 2  
  
 **if** (val % 3 == 0)  
 **return** val == 3  
  
 **if** (val % 5 == 0)  
 **return** val == 5  
  
 **if** (val % 7 == 0)  
 **return** val == 7  
  
 **let** sqrtVal = *parseInt*(***Math***.sqrt(val))  
   
 **for** (**let** i = 11; i <= sqrtVal; i += 2)  
 **if** (val % i == 0)  
 **return false  
  
 return true**}  
  
**function** *main*()  
{  
 **for** (**let** n = -10; n <= 100; ++n)  
 **if** (*isPrime*(n))  
 *writeln*(n)  
  
 *writeln*(*isPrime*(1000003) ? **"Asal"** : **"Asal değil"**)  
}  
  
*main*()

**break Deyimi**

break deyiminin genel biçimi şu şekildedir:

break [etiket-ismi];

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **for** (**let** i = 20; i < 40; ++i) {  
 **if** (i % 12 == 0)  
 **break**

*writeln*(i);  
 }  
}  
  
*main*()  
  
break deyiminin etiketli biçimi kullanılarak içiçe döngülerden çıkış sağlanabilir. Zaten bunun için tasarlanmıştır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **EXIT\_LOOP**:  
 **for** (**let** i = 20; i < 40; ++i) {  
 **for** (**let** k = 21; k < 50; ++i) {  
 **if** (i % 12 == 0 && k < 25)  
 **break EXIT\_LOOP** *writeln*(**`(**${i}**,** ${k}**)`**);  
 }  
 }  
  
 ***console***.log(**"Tekrar yapıyor musunuz?"**)  
}  
*main*()

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **EXIT\_LOOP**:  
 **for** (**let** i = 20; i < 40; ++i) {  
 **EXIT\_SECOND\_LOOP**:  
 **for** (**let** j = 20; j < 40; ++j) {  
 **for** (**let** k = 21; k < 50; ++i) {  
 **if** (j > 22)  
 **break EXIT\_SECOND\_LOOP  
  
 if** (i % 12 == 0 && k < 25)  
 **break EXIT\_LOOP** *writeln*(**`(**${i}**,** ${j}**,** ${k}**)`**);  
 }  
 }  
 }  
  
 ***console***.log(**"Tekrar yapıyor musunuz?"**)  
}  
  
*main*()

**continue Deyimi**

continue deyimi döngünün adımını sonlandırmak için kullanılabilir. continue deyiminin genel biçimi:

continue [etiket ismi];

continue deyimi break deyimi kadar kullanılmasa da bazı durumlarda okunabilirliği artırdığı için tercih edilebilir.

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** sum = 0  
  
 **for** (**let** i = 1; i <= 100; ++i) {  
 **if** (i % 2 == 0)  
 **continue** sum += i  
 }  
  
 ***console***.log(**`Toplam:**${sum}**`**)  
}  
  
*main*()

Etiketli continue deyimi ile bir döngünün adımı içerisindeyken onu kapsayan döngüden devam edilmesi sağlanabilir. Etiketli continue deyimi yapısal programlamada dolayısıyla nesne yönelimli programlamada da tercih edilmemektedir.

**switch Deyimi**

switch deyimi Java'daki klasik switch deyimine çok benzemekle beraber aslında farklılıklar da içermektedir. switch deyiminin Java programlama dilindeki bir takım kısıtları ES'de yoktur.

switch deyiminin aşağı düşme (fall through) özelliği vardır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** val = 1  
  
 **switch** (val) {  
 **case** 1:  
 *writeln*(**"Bir"**)  
 **case** 2:  
 *writeln*(**"İki"**)  
 **break**  
 **case** 3:  
 *writeln*(**"İki"**)  
 **break**  
 **default**:  
 *writeln*(**"default"**)  
 }  
  
}  
  
*main*()

switch deyimi number türü için yani noktalı sayılar için de kullanılabilir. Ancak eşitlik karşılaştırması yaptığı için dikkatli olunmalıdır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let a = 0.1  
 let b = 0.2  
 let val = a + b  
  
 switch (val) {  
 case 0.3:  
 *writeln*("Bir")  
 break  
 case 2.5:  
 *writeln*("İki")  
 break  
 case 3:  
 *writeln*("İki")  
 break  
 default:  
 *writeln*("default")  
 }  
  
}  
  
*main*()

***Anahtar Notlar:*** *Gerçek sayılar için doğrudan eşitlik karşılaştırmada yuvarlama hataları olabileceğinden her zaman doğru sonuç elde edilmez. Eşitlik karşılaştırma işlemi için çeşitli yöntemler kullanılabilir. Örneğin:*

***function*** *writeln(a)  
{* ***console****.log(a)  
}****function*** *main()  
{* ***let*** *val = 0.1 + .2* ***let*** *a = .3  
  
 writeln(****`val=****${val}****`****)* ***if*** *(****Math****.abs(val - a) < 0.00001)  
 writeln(****"Eşit"****)* ***else*** *writeln(****"Eşit değil"****)  
}  
  
main()*

switch deyiminin case bölümlerinin sabit ifadesi olması gerekmez:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
 **let** b = 10  
}  
  
**function** *main*()  
{  
 **let** val = 1  
 **let** a = 1  
  
 **switch** (val) {  
 **case** a:  
 *writeln*(**"Bir"**)  
 **break  
 case** 2:  
 *writeln*(**"Iki"**)  
 **break  
 case** 3:  
 *writeln*(**"Uc"**)  
 **break**;  
 **default**:  
 *writeln*(**"default"**)  
 }  
  
}  
  
*main*()

Örnek bir menu programı. Program bir html içerisinde çalışabilecek şekilde tasarlanmıştır:

**function** *main*()  
{  
 **EXIT\_LOOP**:  
 **for** (;;) {  
 **var** result = *myprompt*();  
  
 **switch** (result) {  
 **case "1"**:  
 *writeln*(**"Ekle"**);  
 **break**;  
 **case "4"**:  
 **break EXIT\_LOOP**;  
 **default**:  
 *writeln*(**"Hala secmediniz"**);  
 **break**;  
 }  
 }  
  
}  
**function** *myprompt*()  
{  
 **var** msg = **"1.Ekle\n"**;  
 msg += **"2.Guncelle\n"**;  
 msg += **"3.Sil\n"**;  
 msg += **"4.Cikis\n"**;  
 msg += **"Secenek:"**;  
  
 **return** *prompt*(msg);  
}  
**function** *writeln*(val)  
{  
 ***document***.write(val + **"</br>"**);  
}  
  
**function** *write*(val)  
{  
 *write*(val);  
}

**Object Türü**

Object ES' de bir referans türünü temsil eden en temel türdür. Bir object türü yaratmak için:

- new operatörü kullanılabilir

- Küme parantezi kullanılabilir. Buna ilkdeğer verme sentaksı da denebilir. JSON formatı tamamen buna dayanır. Zaten JSON Javascript Object Notation demektir

- ES6 ile birlikte belirli koşullar altında new operatörsüz de nesne yaratılabilir. Nesnenin buna uygun olarak bildirilmiş olması gerekir

Bir Object referansı ve nokta operatörü ile bir isim yazıldığında bu isim bir property anlamına gelir. Bu isim daha sonra kullanılmak üzere Object türünde bildirilmiş olur:

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** device = **new *Object***()  
  
 device.**id** = 1  
 device.**name** = **"test"** device.**port** = 1234  
 device.**ip** = **"192.168.2.234"** *displayDevice*(device)  
}  
  
**function** *displayDevice*(dev)  
{  
 *writeln*(**`Device Id:**${dev.**id**}**`**)  
 *writeln*(**`Device name:**${dev.**name**}**`**)  
 *writeln*(**`Device port:**${dev.**port**}**`**)  
 *writeln*(**`Device IP:**${dev.**ip**}**`**)  
}  
  
*main*()  
  
Object daha basit olarak sadece {} ile de elde edilebilir. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** device = {}  
  
 device.**id** = 1  
 device.**name** = **"test"** device.**port** = 1234  
 device.**ip** = **"192.168.2.234"** *displayDevice*(device)  
}  
  
**function** *displayDevice*(dev)  
{  
 *writeln*(**`Device Id:**${dev.**id**}**`**)  
 *writeln*(**`Device name:**${dev.**name**}**`**)  
 *writeln*(**`Device port:**${dev.**port**}**`**)  
 *writeln*(**`Device IP:**${dev.**ip**}**`**)  
}  
  
*main*()

Küme parantezi ile de Object’e ilk değer verme sentaksı biçiminde de yapılabilir. Bu durumda property elemanları küme parantezi içerisinde atanabilir. Burada mutlaka atama yapılmalıdır. Bu sentaksta atama işlemi için = kullanılmaz. : kullanılmalıdır.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** device = {  
 **id**: 1,  
 **name**: **"test"**,  
 **port**: 1234,  
 **ip**: **"192.168.2.234"** }  
  
 *displayDevice*(device)  
  
 device.**port** = 5765  
  
 *displayDevice*(device)  
}  
  
**function** *displayDevice*(dev)  
{  
 *writeln*(**`Device Id:**${dev.**id**}**`**)  
 *writeln*(**`Device name:**${dev.**name**}**`**)  
 *writeln*(**`Device port:**${dev.**port**}**`**)  
 *writeln*(**`Device IP:**${dev.**ip**}**`**)  
}  
  
*main*()

Property elamanları tek tırnak veya çift tırnaklar arasında yazılabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let device = {  
 "id": 1,  
 'name': "test",  
 'port': 1234,  
 'ip': "192.168.2.234"  
  
 }  
  
 *displayDevice*(device)  
}  
  
function *displayDevice*(dev)  
{  
 *writeln*(`Device Id:${dev.id}`)  
 *writeln*(`Device name:${dev.name}`)  
 *writeln*(`Device port:${dev.port}`)  
 *writeln*(`Device IP:${dev.ip}`)  
}  
  
*main*()

Bir object içerisinde fonksiyon tanımlaması yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** device = {  
 **id**:1,  
 **name**:**"test"**,  
 **port**:1234,  
 **ip**:**"192.168.2.234"**,  
 display: **function** ()  
 {  
 *writeln*(**`Device Id:**${**this**.**id**}**`**)  
 *writeln*(**`Device name:**${**this**.**name**}**`**)  
 *writeln*(**`Device port:**${**this**.**port**}**`**)  
 *writeln*(**`Device IP:**${**this**.**ip**}**`**)  
 }  
 }  
  
 device.display()  
 device.**port** = 5765  
 device.display()  
  
 *displayDevice*(device)  
}  
  
**function** *displayDevice*(dev)  
{  
 *writeln*(**`Device Id:**${dev.**id**}**`**)  
 *writeln*(**`Device name:**${dev.**name**}**`**)  
 *writeln*(**`Device port:**${dev.**port**}**`**)  
 *writeln*(**`Device IP:**${dev.**ip**}**`**)  
}  
  
*main*()

Burada display değişkeni aslında function türündendir. function türü olduğundan artık fonksiyon çağırma operatörü ile kullanılabilmektedir.

Object'in bir elemanına global bir fonksiyonun da ismi verilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *displayDevice*()  
{  
 *writeln*(**`Device Id:**${**this**.**id**}**`**)  
 *writeln*(**`Device name:**${**this**.**name**}**`**)  
 *writeln*(**`Device port:**${**this**.**port**}**`**)  
 *writeln*(**`Device IP:**${**this**.**ip**}**`**)  
}  
  
  
**function** *main*()  
{  
 **let** device = {  
 **id**: 1,  
 **name**: **"test"**,  
 **port**: 1234,  
 **ip**: **"192.168.2.234"**,  
 display: *displayDevice* }  
  
 device.*display*()  
 device.**port** = 5765  
 device.*display*()  
}  
  
*main*()

Burada *displayDevice* fonksiyonu içerisinde this referansı kullanıldığına dikkat ediniz.

Aslında herhangi bir değişken de fonksiyon türünde olabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *display*(val)  
{  
 let isEven = function() {  
 return val % 2 == 0  
 }  
  
 *writeln*(isEven() ? "Çift" : "Tek")  
}  
  
function *main*()  
{  
 *display*(10)  
 *display*(11)  
}  
  
*main*()

Burada isEven değişkenin function türündendir. Yukarıdaki örnekte adeta bir yerel fonksiyon tanımlanmıştır. isEven fonksiyonunun display fonksiyonun parameter değişkenini kullanabildiğine (capture) dikkat ediniz.

**Diziler**

ES’ de diziler daha fazla yeteneğe sahip olacak biçimde tasarlanmıştır. Yani ES’ de diziler yalın olarak düşünülmemelidir

Dizilere ilk değer verilmesi aşağıdaki gibi yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** names = [**"ali"**, **"veli"**, **"selami"**];  
  
 *writeln*(names)  
}  
  
  
*main*()

Bir dizi Array türü ile new operatörü kullanılarak yaratılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** names = **new *Array***()  
  
 *writeln*(names)  
}  
  
  
*main*()  
  
Bir diziye ekleme [] operatörü ile yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** names = **[]**  
  
 names[0] = **"ali"** names[1] = **"veli"** *writeln*(names)  
}  
  
  
*main*()  
  
Array nesnesi için aralara da yer ayrılabilir. Yani aşağıdaki örnekte değer atanmamış indeksteki elemanlar undefined değerinde olurlar:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** names = **new *Array***()  
  
 names[0] = **"ali"** names[3] = **"veli"** *writeln*(names[1])  
}  
  
  
*main*()

Array nesnesi için başlangıçta yer ayrılabilir. Bu durumda tüm elemanlar undefined değerinde olacaktır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = **new *Array***(10)  
  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(a[i])  
}  
  
  
*main*()

Array nesnesi for döngüsü ile dolaşılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4];  
  
 **for** (**let** i = 0; i < numbers.**length**; ++i)  
 *writeln*(numbers[i]);  
  
}  
  
*main*()

**for-in döngü deyimi**

Bu döngü deyimi ile bir dizi baştan itibaren sonuna kadar dolaşılabilir. Bu döngü deyiminde her adımda index numarası verilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4];  
  
 **for** (**let** i **in** numbers)  
 *writeln*(numbers[i])  
}  
  
  
*main*()

for-in döngü deyimi ile aslında dizinin diğer elemanları da gezilebilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4];  
  
 numbers.**x** = 90  
 numbers.**name** = **"ali"  
  
 for** (**let** i **in** numbers)  
 *writeln*(`${i}->${ numbers[i]}`)  
}  
  
  
*main*()

Dikkat edilirse for-in döngü deyimi ile elde edilen dizi nesnesinin property elemanları index olarak kullanıldığında property elemanlarının değerleri elde edilmektedir.

**for-of döngü deyimi**

ES6 ile birlikte for-of denilen bir döngü deyimi de eklenmiştir. Bu döngü deyiminde her adımda döngü değişkenine dizinin elemanı atanır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4];  
  
 **for** (**let** val **of** numbers)  
 *writeln*(val)  
}  
  
  
*main*()

for-of döngü deyimi ile yalnızca dizinin elemanları elde edilebilir. Dizi nesnesine ilişkin elemanlar ve değerleri elde edilemez:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let numbers = [1, 2, 3, 4];  
  
 numbers.x = 10  
 numbers.y = "ankara"  
  
 for (let val of numbers)  
 *writeln*(val)  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı bir diziyi tersyüz eden reverse isimli global bir fonksiyon yazınız ve test ediniz.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *reverse*(a)  
{  
 **let** halfLen = *parseInt*(a.**length** / 2)  
  
 **for** (**let** i = 0; i < halfLen; ++i) {  
 **let** temp = a[i]  
 a[i] = a[a.**length** - 1 - i]  
 a[a.**length** - 1 - i] = temp  
 }  
  
 **return** a  
}  
  
  
**function** *display*(a)  
{  
 **for** (**let** val **of** a)  
 *writeln*(val)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5];  
  
 *reverse*(numbers)  
 *display*(numbers)  
}  
  
  
*main*()  
  
Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *reverse*(a)  
{  
 **let** first = 0  
 **let** last = a.**length** - 1  
  
 **while** (first < last) {  
 **let** temp = a[first]  
 a[first++] = a[last]  
 a[last--] = temp  
 }  
  
 **return** a  
}  
  
**function** *display*(a)  
{  
 **for** (**let** val **of** a)  
 *writeln*(val)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5];  
  
 *display*(*reverse*(numbers))  
}  
  
*main*()

Her fonksiyonun içerisinde argümanların bulunduğu arguments isimli bir dizi referansı kullanılabilir. Böylece değişken sayıda argüman alan fonksiyonlarda kolay (ama dikkatli) olarak yazılabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *sum*()  
{  
 let total = 0  
  
 for (let val of arguments)  
 total += val  
  
 return total  
}  
  
function *main*()  
{  
 *writeln*(*sum*(10, 20))  
 *writeln*(*sum*())  
 *writeln*(*sum*(10, 20, 30))  
}  
  
*main*()

**İsimsiz fonksiyonlar ve Lambda ifadeleri**

Bilindiği gibi ES’de fonksiyonlar function anahtar sözcüğü ile bildirilir. function anahtar sözcüğünden sonra isim yazılmazsa bu tarz fonksiyonlara isimsiz fonksiyonlar (anonymous functions) denir.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** *add* = **function** (a, b)  
 {  
 **return** a + b  
 }  
  
 *writeln*(**typeof** *add*)  
 *writeln*(*add*(10, 20))  
}  
  
*main*()

İsimsiz fonksiyonlar bir değişkene atanabilir. Bu durumda bu atanan değişkenin türü function olur.

İsimsiz fonksiyonlar arka planda çağrılacak (callback) fonksiyonlar biçiminde başka bir fonksiyona argüman olarak geçilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *forEach*(array, fn)  
{  
 **for** (**let** elem **of** array)  
 fn(elem)  
}  
  
**function** *main*()  
{  
 **let** a = [1, 2, 3, -6]  
  
 *forEach*(a, **function** (elem) {  
 *writeln*(**`**${elem \* elem}**`**)  
 })  
  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 *forEach*(a, **function** (elem) {  
 *writeln*(elem)  
 })

*writeln("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
 forEach(a, writeln)*  
}  
  
*main*()

Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *forEach*(array, fn)  
{  
 **for** (**let** elem **of** array)  
 fn(elem)  
}  
  
**function** *main*()  
{  
 **let** a = [1, 2, 3, -6]  
  
 **let** *displaySquare* = **function** (elem) {  
 *writeln*(**`**${elem \* elem}**`**)  
 }  
  
 *forEach*(a, *displaySquare*)  
  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
  
 **let** *display* = **function** (elem) {  
 *writeln*(elem)  
 }  
 *forEach*(a, *display*)  
}  
  
*main*()

Fonksiyonlar Object oluşturmak için de kullanılabilir. Yani ES’ de bir Object function anahtar sözcüğü ile de oluşturulabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *WeatherInfo* = **function** (id, place, country, lat, lng, description) {  
 **this**.**id** = id  
 **this**.**place** = place  
 **this**.**country** = country  
 **this**.**lat** = lat  
 **this**.**lng** = lng  
 **this**.**description** = description  
 **this**.toString = **function** () {  
 **return `**${**this**.**country**}**:**${**this**.**place**}**:**${**this**.**lat**}**;**${**this**.**lng**}**,** ${**this**.**description**}**`** }  
}  
  
**function** *main*()  
{  
 **let** wi = **new** *WeatherInfo*(1, **"Mecidiyeköy"**, **"TR"**, 23.456, 41.234, **"Bulutlu"**)  
  
 *writeln*(wi.toString())  
}  
  
*main*()

Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (name, price, stock)  
{  
 **this**.**name** = name  
 **this**.**price** = price  
 **this**.**stock** = stock  
 **this**.getTotal = **function** () {**return this**.**stock** \* **this**.**price** }  
}   
  
**function** *main*()  
{  
 **let** p1 = **new** *Product*(**"laptop"**, 4000, 34)  
 **let** p2 = **new** *Product*(**"mouse"**, 40, 349)  
  
 *writeln*(**`Total:**${p1.getTotal()}**`**)  
 *writeln*(**`Total:**${p2.getTotal()}**`**)  
}  
  
*main*()

Yukarıdaki kodlarda aslında bir Object için constructor (ctor) yazılmıştır. Bu fonksiyonlar new operatörü ile nesne yaratılırken çağrılır.

***Anahtar Notlar:*** *ES’ de de türler kategori olarak değer türleri ve referans türleri şeklinde iki gruba ayrılır. Referans türleri içerisinde adresler tutulur. Object türleri referans türleridir. Örneğin:*

***function*** *writeln(a)  
{* ***console****.log(a)  
}****let*** *Product =* ***function*** *(name, price, stock)  
{* ***this****.****name*** *= name* ***this****.****price*** *= price* ***this****.****stock*** *= stock* ***this****.getTotal =* ***function*** *() {****return this****.****stock*** *\** ***this****.****price*** *}* ***this****.toString =* ***function*** *() {* ***return `****${****this****.****name****}****:****${****this****.****stock****}* ***\**** *${****this****.****price****}****=****${****this****.****stock*** *\** ***this****.****price****}****`*** *}  
}****function*** *main()  
{* ***let*** *p1 =* ***new*** *Product(****"laptop"****, 4000, 34)* ***let*** *p2 = p1  
  
 writeln(p1.toString())  
 writeln(p2.toString())  
  
 p2.****stock*** *\*= 2  
  
 writeln(p1.toString())  
 writeln(p2.toString())  
}  
  
main()*

***Anahtar Notlar:*** *Aslında referans türleri ES terminolojisinde mutable type olarak geçer. Yani aslında herşeyin adresi tutulur. Yalnızca tutulan adresin türü değişikliğe izin vermiyorsa immutable, veriyorsa mutable type olarak ele alınır. Yani aslında yukarıda ”referans türleri içerisinde adresler tutulur” cümlesi basitleştirmek için yazılmıştır. Temel türler immutable, Object türleri ise mutable type olarak ele alınır.*

Fonksiyonlar kendilerinden önce bildirilen değişkenlerin hepsini yakalayabilirler (capture):

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *foo*(b)  
{  
 **let** a = 10  
  
 **let** *f* = **function** ()  
 {  
 ***console***.log(**`a=**${a}**`**)  
 ***console***.log(**`b=**${b}**`**)  
 }  
  
 *f*()  
}  
  
**function** *main*()  
{  
 *foo*(20)  
  
}  
  
*main*()

Fonksiyonlar yakaladıkları değişkenleri değiştirebilirler:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *foo*(b)  
{  
 **let** a = 10  
  
 **let** *f* = **function** ()  
 {  
 ***console***.log(**`a=**${a}**`**)  
 a \*= 2  
 ***console***.log(**`b=**${b}**`**)  
 b -= 3  
 }  
  
 *f*()  
 *f*()  
}  
  
**function** *main*()  
{  
 *foo*(20)  
  
}  
  
*main*()

**Lambda İfadeleri**

Lambda ifadeleri fonksiyonel programlama tekniği ile kod yazımı için matematikten programlamaya aktarılmıştır. Lambda ifadeleri anonim fonksiyonların gelişmiş biçimi gibi düşünülebilir. Lambda ifadeleri ile çok karmaşık kodlar yazılabilir. Ancak biz kullanılan genel biçimlerini ele alacağız. Lambda ifadelerinin genel biçimleri:

1. (<değişken listesi>) => ifade

2. (<değişken listesi>) => {..}

3. <değişken> => ifade

4. <değişken> => {…}

5. () => ifade

6. () => {…}

Lambda ifadeleri function türündendir. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** *f* = (a, b) => a + b

*writeln*(*f*(10, 20))  
}  
  
*main*()

Lambda ifadelerinde kullanılan parametre değişkenleri yalnızca o ifade içerisinde görülebilirdir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** *f* = (a, b) => a + b  
 *writeln*(*f*(10, 20))  
  
 *writeln*(**`a=**${a}**`**) *//error ~ exception*}  
  
*main*()

Lambda ifadeleri kendisinden önce bildirilen değişkenlerin hepsini yakalabilirler:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *foo*(c)  
{  
 **let** *f* = (a, b) => a + b + c

*writeln*(*f*(10, 20))  
}  
  
**function** *main*()  
{  
 *foo*(3)  
}  
  
*main*()

***Anahtar Notlar:*** *ES’de lambda ifadelerine bazı kaynaklarda “oklu fonksiyonlar (arrow functions)” da denilmektedir.*

Aşağıdaki örnekte generate fonksiyonu iki tane fonksiyon (callback) almaktadır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *generate*(n, supplier, consumer)  
{  
 while (n--)  
 consumer(supplier())  
}  
  
function *main*()  
{  
 let min = 20  
 let max = 40  
  
 *generate*(10, () => *parseInt*(Math.random() \* (max - min) + min), val => *writeln*(val))  
  
 *writeln*("//////////////////////////////")  
  
 let x = 1  
  
 *generate*(5, () => x++, *writeln*)  
}  
  
*main*()

***Anahtar Notlar:*** *ES’ de genel olarak global olarak yazılan alt programlara fonksiyon, sınıflar (object’ler) içerisinde yazılan alt programlara ise metot denir. Ancak metotlara da fonksiyon denmesi yanlış değildir. Ancak bir fonksiyona metot denmesi yanlış anlaşılabilir.*

**Array nesnesi**

Array türüne ait bir çok yararlı metot bulunmaktadır. Bu metotların bir kısmı fonksiyonel programlama tekniği ile de uyumlu olarak kullanılabilmektedir. Array türü ile bir çok veri yapısına yönelik işlem yapılabilmektedir.

Array türü istenirse stack gibi kullanılabilir. Array türüne ait push ve pop isimli fonksiyonlar bulunmaktadır.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = []  
  
 a.push(10)  
 a.push(20)  
  
 *writeln*(a.pop())  
 *writeln*(a.pop())  
}  
  
*main*()

Array türünün sort fonksiyonu ile sıralama yapılabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = [-3, 9, -7, 8, 6, 5]  
  
 a.sort()  
 *writeln*(a)  
}  
  
*main*()

Parametresiz sort fonksiyonu değerleri yazıya çevirerek sıralama yapar. Sıralama doğal sıralamadır. Doğal sıralama artan sırada (ascending) sıralamak anlamına gelmektedir. sort sıralamanın nasıl yapılacağına ilişki bir fonksiyon (callback) alabilmektedir. Bu konu ileride ele alınacaktır.

Array nesnenin unshift ve shift metotları ile eleman eklemek ve çıkarmak mümkündür. unshift metodu başa ekleme yapar, shift metodu baştan elemanı döndürür ve siler:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = **new *Array***()  
  
 **for** (**let** i = 0; i < 10; ++i)  
 numbers.unshift(i)  
  
 *writeln*(numbers)  
 *writeln*(**`Last shifted:**${numbers.shift()}**`**)  
 *writeln*(numbers)  
}  
  
*main*()

Aşağıdaki kodda bir Array unshift ve shift kullanarak boşaltılmış ve her adımda elemanı elde edilmiştir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = **new *Array***()  
  
 **for** (**let** i = 0; i < 10; ++i)  
 numbers.unshift(i)  
  
 **while** (numbers.**length** > 0) {  
 *writeln*(numbers.shift())  
 }  
}  
  
*main*()

Aşağıdaki kodda bir Array push ve shift kullanarak boşaltılmış ve her adımda elemanı elde edilmiştir. Bir FIFO (First In First Out) kuyruk sistemi oluşturulmuştur:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = **new *Array***()  
  
 **for** (**let** i = 0; i < 10; ++i)  
 numbers.push(i)  
  
 **while** (numbers.**length** > 0) {  
 *writeln*(numbers.shift())  
 }  
}  
  
*main*()

Bir Object türünden dizi de oluşturulabilir ve for-in ya da for-of döngüsü ile dolaşılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `**${**this**.**name**}**:**${**this**.**stock** \* **this**.**price**}**`**}  
  
**function** *main*()  
{  
 **let** products = [  
 { **name**: **"laptop"**, **stock**: 6, **price**: 4345, toString: *toString* },  
 { **name**: **"mouse"**, **stock**: 40, **price**: 70, toString: *toString* },  
 { **name**: **"motherboard"**, **stock**: 5, **price**: 500, toString: *toString* },  
 ]  
  
 **for** (**let** index **in** products)  
 *writeln*(products[index].*toString*())  
}  
  
*main*()

for-of döngüsü ile aşağıdaki gibi de yazılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `**${**this**.**name**}**:**${**this**.**stock** \* **this**.**price**}**`**}  
  
**function** *main*()  
{  
 **let** products = [  
 { **name**: **"laptop"**, **stock**: 6, **price**: 4345, toString: *toString* },  
 { **name**: **"mouse"**, **stock**: 40, **price**: 70, toString: *toString* },  
 { **name**: **"motherboard"**, **stock**: 5, **price**: 500, toString: *toString* },  
 ];  
  
 **for** (**let** product **of** products)  
 *writeln*(product.*toString*());  
}  
  
*main*()

sort metodunun function tür parametreli versiyonu ile sıralama işlemi istenildiği şekilde yapılabilir. sort metodu parametresi ile aldığı fonksiyonun geri dönüş değerine göre artan sırada (ascending) sıralama yapar. Bu fonksiyona karşılaştırma fonksiyonu denir. Fonksiyonun geri dönüş değeri negatif ise birinci parametresi ikinci parametresinden küçük kabul edilir, pozitif ise birinci parametre ikinci parametreden büyük kabul edilir, sıfır ise eşit kabul edilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `**${**this**.**name**}**:**${**this**.**price**} **\*** ${**this**.**stock**} **=** ${**this**.**stock** \* **this**.**price**}**`**}  
  
**function** *displayProducts*(products)  
{  
 **for** (**let** p **of** products)  
 *writeln*(p.toString())  
}  
  
**function** *main*()  
{  
 **let** products = [  
 { **name**: **"motherboard"**, **stock**: 5, **price**: 500, toString: *toString* },  
 { **name**: **"laptop"**, **stock**: 6, **price**: 4345, toString: *toString* },  
 { **name**: **"mouse"**, **stock**: 40, **price**: 70, toString: *toString* }  
 ]  
  
 *displayProducts*(products)  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 products.sort(**function** (p1, p2) {**return** p1.**price** - p2.**price**})  
  
 *displayProducts*(products)  
}  
  
*main*()

Aynı örnek lambda ifadesi kullanılarak da yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `**${**this**.**name**}**:**${**this**.**price**} **\*** ${**this**.**stock**} **=** ${**this**.**stock** \* **this**.**price**}**`**}  
  
**function** *displayProducts*(products)  
{  
 **for** (**let** p **of** products)  
 *writeln*(p.toString())  
}  
  
**function** *main*()  
{  
 **let** products = [  
 { **name**: **"motherboard"**, **stock**: 5, **price**: 500, toString: *toString* },  
 { **name**: **"laptop"**, **stock**: 6, **price**: 4345, toString: *toString* },  
 { **name**: **"mouse"**, **stock**: 40, **price**: 70, toString: *toString* }  
 ]  
  
 *displayProducts*(products)  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 products.sort((p1, p2) => p1.**price** - p2.**price**)  
  
 *displayProducts*(products)  
}  
  
*main*()

Aynı yöntemle büyükten küçüğe sıralama karşılaştırma fonksiyonu değiştirilerek yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `**${**this**.**name**}**:**${**this**.**price**} **\*** ${**this**.**stock**} **=** ${**this**.**stock** \* **this**.**price**}**`**}  
  
**function** *displayProducts*(products)  
{  
 **for** (**let** p **of** products)  
 *writeln*(p.toString())  
}  
  
**function** *main*()  
{  
 **let** products = [  
 { **name**: **"motherboard"**, **stock**: 5, **price**: 500, toString: *toString* },  
 { **name**: **"laptop"**, **stock**: 6, **price**: 4345, toString: *toString* },  
 { **name**: **"mouse"**, **stock**: 40, **price**: 70, toString: *toString* }  
 ]  
  
 *displayProducts*(products)  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 products.sort((p1, p2) => p2.**price** – p1.**price**)  
  
 *displayProducts*(products)  
}  
  
*main*()

Array nesnesinin slice fonksiyonu ile dizinin belirli bir aralığındaki elemanlar yine Array olarak alınabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** sliced = numbers.slice(2, 3) *//[2, 3)  
  
 writeln*(sliced)  
}  
  
  
*main*()

slice fonksiyonuna negatif index numaraları verilebilir. Bu durumda index numaralarını son elemana göre almaktadır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** sliced = numbers.slice(-3, -1) *//[2, 4)  
  
 writeln*(sliced)  
}  
  
*main*()

Aslında negatif indeks değerleri için dizinin uzunluğu ile toplandığında elde edilen aralıktaki değerler verilmiş olur.

slice fonksiyonu alt bir dizi elde edilemezse boş bir dizi döndürür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** sliced = numbers.slice(30, 3)  *writeln*(sliced)  
}  
  
*main*()

Array nesnesinin indexOf fonksiyonu ile aranacak elemanın index numarası elde edilebilir. Eğer bulunamazsa -1 değerine döner:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** index = numbers.indexOf(2)  
  
 *writeln*(index)  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı bir dizi ve dizi içerisindeki eleman için, o elemanın dizide kaç tane olduğunu döndüren getCount isimli fonksiyonu yazınız ve test ediniz. Test için ilk değer verilmiş bir dizi kullanabilirsiniz.

Çözüm:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getCount*(array, val)  
{  
 **let** count = 0  
 **let** index = -1  
  
 **while** ((index = array.indexOf(val, index + 1)) != -1)  
 ++count  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5, 2, 5, 6, 2, 7]  
 **let** count = *getCount*(numbers, 2)  
  
 *writeln*(**`count=**${count}**`**)  
}  
  
*main*()

Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getCount*(array, val)  
{  
 **let** count = 0  
 **let** index = 0  
  
 **while** ((index = array.indexOf(val, index)) != -1) {  
 ++count  
 ++index  
 }  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5, 2, 5, 6, 2, 7]  
 **let** count = *getCount*(numbers, 2)  
  
 *writeln*(**`count=**${count}**`**)  
}  
  
*main*()

Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getCount*(array, val)  
{  
 **let** count = 0  
  
 **for** (**let** index = -1; (index = array.indexOf(val, index + 1)) != -1; ++count)  
 ;  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5, 2, 5, 6, 2, 7]  
 **let** count = *getCount*(numbers, 2)  
  
 *writeln*(**`count=**${count}**`**)  
}  
  
*main*()

Ya da örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getCount*(array, val)  
{  
 **let** count = 0  
  
 **for** (**let** index = 0; (index = array.indexOf(val, index)) != -1; ++count, ++index)  
 ;  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5, 2, 5, 6, 2, 7]  
 **let** count = *getCount*(numbers, 2)  
  
 *writeln*(**`count=**${count}**`**)  
}  
  
*main*()

Array nesnesinin find fonksiyonu ile belirli koşula uygun olan ilk eleman bulunabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** result = numbers.find(val => val % 2 == 0)  
  
 *writeln*(result)  
}  
  
*main*()

find fonksiyonu koşula uygun herhangi bir eleman yoksa undefined değerini döndürür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 23, 3, 4, 5]  
 **let** result = numbers.find(val => val % 2 == 0)  
  
 *writeln*(result == **undefined** ? **"Bulunamadı"**: result)  
}  
  
*main*()

find fonksiyonu aşağıdaki gibi de kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** result = numbers.find((val, index)=> val % 2 == 0 && index > 2)  
  
 *writeln*(result)  
}  
  
*main*()

Aşağıdaki gibi bir find global fonksiyonu da yazılabilir:

**function** main()

{

**let** numbers = [1, 2, 3, 4, 5];

**let** result = myfind(numbers, (val, index, a) => val %2 == 0 && index > 2);

writeln(result);

}

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *myfind*(a, pred)  
{  
 **for** (**let** i **in** a)  
 **if** (pred(a[i], i, a))  
 **return** a[i]  
  
 **return undefined**}  
  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
 **let** result = *myfind*(numbers, (val, index)=> val %2 == 0 && index > 2)  
  
 *writeln*(result)  
}  
  
*main*()

myFind fonksiyonu nesneye de eklenebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *myFind*(pred)  
{  
 **for** (**let** i = 0; i < **this**.**length**; ++i)

**if** (pred(**this**[i], i, **this**))  
 **return this**[i]  
  
 **return undefined**}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
  
 numbers.myFind = *myFind* **let** result = numbers.myFind((val, index, a) => val %2 == 0 && index > 2)  
  
 *writeln*(result)  
}  
  
*main*()

Array nesnesinin findIndex fonksiyonu ile belirli bir koşula uyan ilk elemanın index numarası elde edilebilir. Eğer koşula uyan eleman bulunamazsa -1 döner:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 2, 3, 4, 5]  
  
 **let** index = numbers.findIndex(val => val % 2 == 0)  
  
 *writeln*(index != -1 ? **`**${index} **numaralı indekste bulundu`** : **"Bulunamadı"**)  
}  
  
*main*()

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Person* = **function** (id, name) {  
 **this**.**id** = id  
 **this**.**name** = name  
 **this**.toString = **function** () {  
 **return this**.**name** }  
}  
  
**function** *main*()  
{  
 **let** people = [**new** *Person*(1, **"oğuz"**), **new** *Person*(2, **"muhammed"**), **new** *Person*(3, **"oğuzhan"**)]  
 **let** index = people.findIndex(per => per.**id** == 2)  
  
 *writeln*(index != -1 ? **`**${index} **numaralı indekste** ${people[index].toString()} **bulundu`** : **"Bulunamadı"**)  
}  
  
*main*()

Array nesnesinin every metodu dizinin tüm elemanları belirli koşula uyarsa true bir tane bile uymayan eleman varsa false değeri ile döner:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [10, 20, 30, 4, 58]  
  
 **if** (numbers.every(val => val % 2 == 0))  
 *writeln*(**"Tüm sayılar çift"**)  
 **else** *writeln*(**"en az bir tane çift olmayan sayı var"**)

}  
  
*main*()

Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *SeatInfo* = **function** (number) {  
 **this**.**number** = number  
 **this**.**isEmpty** = **true**}  
  
**function** *main*()  
{  
 **let** seats = [**new** *SeatInfo*(1), **new** *SeatInfo*(2), **new** *SeatInfo*(3)]  
  
 seats.forEach(s => s.**isEmpty** = **false**)  
  
 **if** (seats.every(s => s.**isEmpty** == **false**))  
 *writeln*(**"Uçak dolu"**)  
 **else** *writeln*(**"Boş koltuk var"**)  
}  
  
*main*()

Array nesnesinin some fonksiyonu belirlenen koşula uyan hiç eleman yoksa false değerini döndürür. Koşula uyan en az bir tane eleman varsa true döner:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [1, 21, 30, 41, 5]  
  
 **if** (numbers.some(val => val % 2 == 0))  
 *writeln*(**"En az bir tane çift sayı var"**)  
 **else** *writeln*(**"Hiç çift yok"**)}  
  
*main*()

Örnek:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *SeatInfo* = **function** (number) {  
 **this**.**number** = number  
 **this**.**isEmpty** = **true**}  
  
**function** *main*()  
{  
 **let** seats = [**new** *SeatInfo*(1), **new** *SeatInfo*(2), **new** *SeatInfo*(3)]  
  
 seats.forEach(s => s.**isEmpty** = **false**)  
  
 seats[0].**isEmpty** = **true  
  
 if** (seats.some(s => s.**isEmpty** == **true**))  
 *writeln*(**"Boş koltuk var"**)  
 **else** *writeln*(**"Uçak dolu"**)  
}  
  
*main*()

Aşağıdaki örnekte bitmiş ürün olup olmadığı sorgulanmaktadır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (name, price, stock)  
{  
 **this**.**name** = name  
 **this**.**price** = price  
 **this**.**stock** = stock  
 **this**.getTotal = **function** () {**return this**.**stock** \* **this**.**price** }  
}  
  
**function** *main*()  
{  
 **let** products = **new *Array***()  
  
 products.push(**new** *Product*(**"laptop"**, 4000, 34))  
 products.push(**new** *Product*(**"mouse"**, 40, 349))  
 products.push(**new** *Product*(**"klavye"**, 30, 0))  
 products.push(**new** *Product*(**"kalem"**, 40, 34))  
  
 *writeln*(products.some(p => p.**stock** <= 0) ? **"Stokta bitmiş ürün var"** : **"Tüm ürünler mevcut"**)  
}  
*main*()

Aşağıdaki örnekte ilk bitmiş ürün aranmaktadır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
let *Product* = function (name, price, stock)  
{  
 this.name = name  
 this.price = price  
 this.stock = stock  
 this.getTotal = function () {return (this.stock > 0 ? this.stock : 0) \* this.price }  
 this.toString = function (){return `${this.name}:${this.getTotal()}`}  
}  
  
  
function *main*()  
{  
 let products = new Array()  
  
 products.push(new *Product*("laptop", 4000, 34))  
 products.push(new *Product*("mouse", 40, 349))  
 products.push(new *Product*("klavye", 30, -6))  
 products.push(new *Product*("kalem", 40, 34))  
  
 let index = products.findIndex(p => p.stock <= 0)  
  
 if (index != -1)  
 *writeln*(`${products[index].toString()} ürünü stokta yok`)  
 else  
 *writeln*("Tüm ürünler mevcut")  
}  
*main*()

Aşağıdaki örnekte stokta olmayan tüm ürünler başka bir diziye atılarak listelenmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (name, price, stock)  
{  
 **this**.**name** = name  
 **this**.**price** = price  
 **this**.**stock** = stock  
 **this**.getTotal = **function** () {**return this**.**stock** \* **this**.**price** }  
}  
  
**function** *main*()  
{  
 **let** products = **new *Array***()  
  
 products.push(**new** *Product*(**"laptop"**, 4000, 0))  
 products.push(**new** *Product*(**"mouse"**, 40, 349))  
 products.push(**new** *Product*(**"klavye"**, 30, 0))  
 products.push(**new** *Product*(**"kalem"**, 40, 34))  
  
 **let** productsNotInStock = **new *Array***(products.**length**)  
  
 **let** index = -1  
   
 **while** (**true**) {  
 index = products.findIndex((p, i) => i >= index + 1 && p.**stock** <= 0)  
  
 **if** (index == -1)  
 **break** productsNotInStock.push(products[index])  
 }  
  
 productsNotInStock.forEach(p => *writeln*(p.**name**))  
}  
*main*()

Array nesnesinin reduce metodu ile dizinin elemanları kullanılarak bir işlem yapılabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [2, 4, 6, 8]  
 **let** result = numbers.reduce((sum, val) => sum + val)  
  
 *writeln*(result)  
}  
  
*main*()

reduce fonksiyonunun ikinci parametresi biriktirilecek değerin ilk başlayacağı değeri alan parametredir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [2, 4, 6, 8]  
 **let** result = numbers.reduce((sum, val) => sum + val, 6)  
  
 *writeln*(result)  
}  
  
*main*()

Aşağıdaki örnekte dizi içerisindeki yazılar birleştirilmiştir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** numbers = [**"ankara"**, **"istanbul"**, **"izmir"**, **"muğla"**]  
 **let** result = numbers.reduce((sum, val) => sum + val, **"şehirler:"**)  
  
 *writeln*(result)  
}  
  
*main*()

Basit bir myReduce fonksiyonu aşağıdaki gibi yazılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *myReduce*(f, val) {  
 **let** result = 0  
  
 **if** (***arguments***.**length** >= 2)  
 result = val  
  
 **for** (**let** i = 0; i < **this**.**length**; ++i)  
 result = f(result, **this**[i], i, **this**)  
  
 **return** result  
}  
  
**function** *main*()  
{  
 **let** cities = [**"istanbul"**, **"ankara"**, **"izmir"**]  
  
 cities.myReduce = *myReduce* **let** result = cities.myReduce((sum, val) => sum + val, **"şehirler:"**)  
  
 *writeln*(result);  
  
}  
  
*main*()

filter metodu parametresi ile aldığı koşula uyan elemanlardan oluşan diziyi verir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let cities = ["ankara", "istanbul", "izmir", "muş", "bolu"]  
 let result = cities.filter(city => city.length > 4)  
  
 *writeln*(result)  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı bir dizinin, parametresi aldığı callback predicate fonksiyon için koşula uyan elemanlarının sola koşula uymayanların ise saga atan ve ilk koşula uymayan elemanın indeks numarasını döndüren partition isimli global fonksiyonu yazınız.

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *gswap*(a, i, j)  
{  
 let temp = a[i]  
  
 a[i] = a[j]  
 a[j] = temp  
}  
  
function *partition*(array, pred)  
{  
 let partitionIndex = 0;  
  
 while (partitionIndex < array.length && pred(array[partitionIndex]))  
 partitionIndex++  
  
 if (partitionIndex == array.length)  
 return partitionIndex  
  
 for (let i = partitionIndex + 1; i < array.length; ++i)  
 if (pred(array[i]))  
 *gswap*(array, i, partitionIndex++)  
  
 return partitionIndex  
}  
  
function *getRandomIntArray*(n, min, max)  
{  
 let a = new Array(n);  
  
 for (let i = 0; i < n; ++i)  
 a[i] = *parseInt*(Math.random() \* (max - min) + min)  
  
 return a  
}  
  
function *main*()  
{  
 let a = *getRandomIntArray*(7, 1, 100)  
 *writeln*(a)  
 let pi = *partition*(a, val => val % 2 == 0)  
 *writeln*(a)  
 *writeln*(`Partition Index:${pi}`)  
}  
  
*main*()

**Temel Türleri Sarmalayan Nesneler**

ES’ de temel türleri sarmalayan (wrapper) nesneler bulunur. Örneğin string türü için *String* sarmalayan nesnesi, number türü için *Number* sarmalayan nesnesi kullanılabilir. Temel türlerin metotları olamayacağından yararlı işlemler için sarmalayan sınıflar düşünülmüştür. Örneğin:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** a = 10  
  
 *writeln*(**typeof** a)  
  
 **let** msg = a.toString(2)

*writeln*(**typeof** a)

*writeln*(msg)  
}  
  
*main*()

Burada a nın gösterdiği tür için toString fonksiyonu çağrılırken sarmalanır ve *Number* türü olarak işleme girer. Burada a nın gösterdiği tür değişmez. İşlem öncesi otomatik tür dönüşümü ile Number nesnesi olarak sarmalanır.

**String Nesnesi**

ES’ de string türünü sarmalayan *String* isimli bir nesne bulunmaktadır. String nesnesi Java’ da olduğu gibi immutable’dır. Yani String nesnesi içerisinde bulunan yazı üzerinde değişiklik yapılamaz. Değişiklik yapan metotlar yeni bir String nesnesi döndürürler. String nesnesinin yararlı fonksiyonları vardır.

String nesnesinin length property elemanı ile içerisinde tutuğu yazının karakter sayısı elde edilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"ankara"** *writeln*(**`length:**${s.**length**}**`**)  
}  
  
*main*()

length property elemanına değer atanamaz. Atanma durumunda error olup olmayacağı garanti değildir. Ancak length property elemanının değişmeyeceği garanti altındadır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let s = "ankara"  
  
 s.length = 70  
  
 *writeln*(s)  
 *writeln*(s.length)  
}  
  
*main*()

***Anahtar Notlar:*** *Array nesnesi immutable bir tür olmadığından length property elemanına değer atanabilir. Örneğin dizinin eleman sayısı büyütülebilir. Bu durumda yeni elemanlar undefined değerinde olurlar:*

*function writeln(a)  
{  
 console.log(a)  
}  
  
function main()  
{  
 let a = [1, 2]  
  
 for (let val of a)  
 writeln(val)  
  
 writeln("//////////////")  
  
 a.length = 7  
  
 for (let val of a)  
 writeln(val)  
}  
  
main()*

*Benzer şekilde length property elemanı ile dizinin boyutu küçültülebilir:*

*function writeln(a)  
{  
 console.log(a)  
}  
  
function main()  
{  
 let a = [1, 2, 4]  
  
 for (let val of a)  
 writeln(val)  
  
 writeln("//////////////")  
  
 a.length = 2  
  
 for (let val of a)  
 writeln(val)  
}  
  
main()*

String sınıfının charAt metodu ile herhangi bir indekteki karakter elde edilebilir.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"ankara"  
  
 for** (**let** i **in** s)  
 *writeln*(s.charAt(i))  
}  
  
*main*()

charCodeAt metodu ile verilen bir index numarasındaki karakterin karakter tablosundaki sıra numarası elde edilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"oğuz"  
  
 for** (**let** i **in** s)  
 *writeln*(s.charCodeAt(i))  
}  
  
*main*()

concat fonksiyonu ile iki string birleştirilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** name = **"oğuz"  
 let** surname = **" karan"  
  
 let** fullName = name.concat(surname)  
  
 *writeln*(fullName)  
}  
  
*main*()

ES’ de de Java da olduğu gibi + operatörü ile de birleştirme yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** name = **"oğuz"  
 let** surname = **" karan"  
  
 let** fullName = name + surname  
  
 *writeln*(fullName)  
}  
  
*main*()

Aşağıdaki örnekte concat metoduna number türü geçirilmiş ve otomatik olarak string türüne dönüşmüştür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** degree = 3.4  
 **let** msg = **"Bugün hava "**.concat(degree).concat(**" derece"**)  
  
 *writeln*(msg)  
}  
  
*main*()

Yukarıdaki kod aşağıdaki gibi de yazılabilirdi:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** degree = 3.4  
 **let** msg = **"Bugün hava "** + degree + **" derece"** *writeln*(msg)  
}  
  
*main*()

indexOf metodu ile bir yazı içerisinde başka bir yazı aranabilir. Bu durumda metot yazı bulunursa ilk bulunan yazının ilk karakterinin index numarası, bulunamazsa -1 değeri döndürür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** text = **"Bugün hava çok güzel. Bu çok güzel havada ders mi yapılır"  
 let** s = **"çok"  
 let** index = text.indexOf(s)  
   
 *writeln*(index != -1 ? **"Aranan yazı "** + index + **" numaralı indekste bulundu"** : **"Bulunamadı"**)  
}  
  
*main*()

indexOf metoduna ikinci argüman olarak aramaya hangi indeksten başlayacağı bilgisi geçilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** text = **"Bugün hava çok güzel. Bu çok güzel havada ders mi yapılır"  
 let** s = **"çok"  
 let** index = text.indexOf(s, 26)  
   
 *writeln*(index != -1 ? **"Aranan yazı "** + index + **" numaralı indekste bulundu"** : **"Bulunamadı"**)  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı birinci yazının içerisinde parametresi ile aldığı ikinci yazıdan kaç tane olduğunu döndüren getStringCount isimli fonksiyonu yazınız ve test ediniz. Fonksiyon büyük-küçük harf duyarlı olarak yazılacaktır.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getStringCount* (text, searchText)  
{  
 **let** count = 0  
  
 **for** (**let** index = 0; (index = text.indexOf(searchText, index)) != -1; ++count, ++index)  
 ;  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** text = **"Bugün hava çok güzel. Bu çok güzel havada ders mi yapılır"  
 let** s = **"çok"  
 let** count = *getStringCount*(text, s)  
  
 *writeln*(count)  
}  
  
*main*()

Fonksiyon aşağıdaki gibi de yazılabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *getStringCount* (text, searchText)  
{  
 let count = 0  
  
 for (let index = -1; (index = text.indexOf(searchText, index + 1)) != -1; ++count)  
 ;  
  
 return count  
}  
  
function *main*()  
{  
 let text = "Bugün hava çok güzel. Bu çok güzel havada ders mi yapılır"  
 let s = "çok"  
 let count = *getStringCount*(text, s)  
  
 *writeln*(count)  
}  
  
*main*()

toUpperCase ve toLowerCase fonksiyonları sırasıyla büyük harfe ve küçük harfe dönüştürülmüş yeni String döndürürler:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *getStringCount*(text, searchText, ignoreCase)  
{  
 **if** (ignoreCase) {  
 text = text.toLowerCase()  
 searchText = searchText.toLowerCase()  
 }  
  
 **let** count = 0  
  
 **for** (**let** index = -1; (index = text.indexOf(searchText, index + 1)) != -1; ++count)  
 ;  
  
 **return** count  
}  
  
**function** *main*()  
{  
 **let** text = **"Bugün hava çok güzel. BuÇok güzel havada ders mi yapılır"  
 let** s = **"çok"  
 let** count = *getStringCount*(text, s, **true**)  
  
 *writeln*(count)  
}  
  
*main*()

startsWith ve endsWith metotları sırasıyla bir String’in bir yazı ile başlayıp başlamadığını veya bitip bitmediğini test etmek için kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *concatIfNotStarts*(s1, s2)  
{  
 **return** s1.startsWith(s2) ? s1 : s2 + s1  
}  
  
**function** *concatIfNotEnds*(s1, s2)  
{  
 **return** s1.endsWith(s2) ? s1 : s1 + s2  
}  
  
**function** *main*()  
{  
 **let** url = **"www.csystem.org"  
 let** email = **"oguzkaran@csystem.org"** url = *concatIfNotStarts*(url, **"http://"**)  
 email = *concatIfNotEnds*(email, **".org"**)  
  
 *writeln*(url)  
 *writeln*(email)  
}  
  
*main*()

fromCharCode metodu karakter sıra numarasından karakteri bir String olarak elde etmek için kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** code = 67  
  
 *writeln*(***String***.fromCharCode(code))  
}  
  
*main*()

includes metodu bir yazı içerisinde parametresi aldığı yazının bulunup bulunmadığını test etmek için kullanılabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 let str = "Bugün hava çok güzel"  
 let s = "az"  
  
  
 *writeln*(str.includes(s) ? "Yazı var" : "Yazı yok")  
}  
  
*main*()

repeat metodu bir yazıdan istenildiği miktarda birleştirilmiş yeni bir yazı elde etmek için kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
  
**function** *main*()  
{  
 **let** s = **"ali"  
  
 let** str = s.repeat(10)  
  
 *writeln*(str)  
}  
  
*main*()

slice, substr ve substring fonksiyonları bir yazı içerisinden belirli aralıktaki yazıyı elde etmekte kullanılır. substring fonksiyonu belirli bir indeksten itibaren ikinci parametresi ile aldığı indekse kadar yazıyı döndürür.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"ankara"  
  
 let** str = s.substring(2, 5) *//[2, 5)  
  
 writeln*(str)  
}  
  
*main*()

substr fonksiyonun ikinci parametresi birinci parametresi ile verilen indeksten itibaren kaç tane karakterin alınacağını belirtir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"ankara"  
 let** str = s.substr(2, 3)  
  
 *writeln*(str)  
}  
  
*main*()

slice fonksiyonu substr fonksiyonu gibidir. Ancak negatif indeks değerleri için de kullanılabilir. Negatif indeks için yazının karakter uzunluğu ile negatif indeks değeri toplanarak elde edilen indeks değerinden itibaren yazı elde edilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"ankara"  
 let** str = s.slice(-4) *//length - 4  
  
 writeln*(str)  
}  
  
*main*()

split metodu verilen ayraç veya ayraçlara gore yazıyı parçalar ve yazının parçalarından oluşan diziyi döndürür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"Bugün hava çok güzel"  
 let** words = s.split(**" "**)  
  
 **for** (**let** word **of** words)  
 *writeln*(word)  
}  
  
*main*()

split metodu düzenli ifade (regular expression) alarak da çalışabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"Bugün.hava.çok, güzel"  
 let** words = s.split(/[ .]/)  
  
 **for** (**let** word **of** words)  
 *writeln*(word)  
}  
  
*main*()

Düzenli ifade / ve / arasında verilmelidir. Birden fazla karakter ayraç olarak kullanılacaksa [] arasında yazılmalıdır. Yukarıdaki örnekte ayraçlar yanyana yazıldığında yanyana yazılan ayraçlar arasındaki boş string de diziye eklenir. Boş string’in eklenmemesi için ]’den sonra + konmalıdır. Bu işlem düzenli ifadeler için “buradaki karakterlerden bir veya birden fazlası” anlamına gelmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"Bugün.hava.çok, güzel"  
 let** words = s.split(/[ .]/)  
  
 **for** (**let** word **of** words)  
 *writeln*(word)  
}  
  
*main*()

[ veya ] ayraç olarak kullanılacaksa ters bölü ile yazılmalıdır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** s = **"Bugün]hava.çokgüzel"  
 let** words = s.split(/[ .\]]+/)  
  
 **for** (**let** word **of** words)  
 *writeln*(word)  
}  
  
*main*()

**Sınıf Çalışması:** Parametresi ile aldığı bir dizinin elemanlarını aralarında ikinci parametresi ile aldığı başka bir string olacak şekilde birleştirilmiş string ile geri dönden join isimli fonksiyonu yazınız ve test ediniz.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *join*(array, delim)  
{  
 **let** s = array.reduce((result, s) => result + s + delim, **""**)  
  
 **return** s.substr(0, s.**length** - delim.**length**)  
}

**function** *main*()  
{  
 **let** names = [**"oğuz"**, **"ali"**, **"kaan"**, **"necati"**]  
  
 *writeln*(*join*(names, **":"**))  
}  
  
*main*()

Array sınıfının join metodu ile yukarıdaki fonksiyon yazılmadan da birleştirme işlemi yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** names = [**"oğuz"**, **"ali"**, **"kaan"**, **"necati"**]  
  
 *writeln*(names.join(**":"**))  
}  
  
*main*()

ES6 ile birlikte String nesnesine raw isimli bir fonksiyon eklenmiştir. raw fonksiyonu aldığı yazının içerisindeki escape sequence karakterlerin kendi anlamında kullanılmasını engeller:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 *writeln*(String.raw`c:\test\names.txt`)  
}  
  
*main*()

**Date nesnesi**

ES’ de tarih zaman işlemleri için Date isimli bir nesne bulunmaktadır. Date nesnesi yaratmak için ctor çeşitli argümanlar ile kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***()  
  
  
 *writeln*(now.toString())  
}  
  
*main*()

Date nesnesi parametresiz olarak yaratılırsa çalışılan sistemin o anki tarih-zaman bilgisi elde edilir. toString metodu tarih-zamana ilişkin standart yazı bilgisini döndürür. Bu zaman sistemin zamanıdır. Date nesnesinin ctor’u yıl, ay, gün, saat, dakika, saniye, milisaniye değerlerini parametre olarak alabilir. Zamana ilişkin parametrelerin default değerleri sıfırdır, tarihe ilişkin parametrelerin default değerleri ay için 1(bir) ve gün için de 1(bir) değeridir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***(2019, 2, 27, 19, 26, 34, 234)  
  
 *writeln*(now.toString())  
}  
  
*main*()

***Anahtar Notlar:*** *Date nesnesi ay bilgisini [0, 11] aralığında alır. Buna dikkat edilmelidir.*

Date sınıfının ctor’una milisaniye değeri de parameter olarak geçilebilir.

Date nesnesinin getDay fonksiyonu tarihin haftanın hangi gününe geldiği bilgisini verir.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***(2019, 1, 24)  
  
 *writeln*(now.getDay())  
}  
  
*main*()

getDay fonksiyonu haftanın günlerini [0, 6] aralığında sıfır pazar ve 6 cumartesi olacak şekilde değer verir.

getDate fonskiyonu ile o aya ilişkin gün bilgisi elde edilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***(2019, 1, 24)  
  
 *writeln*(now.getDate())  
}  
  
*main*()

getYear fonksiyonu deprecated olmuştur. Bu fonksiyonu yerine getFullYear fonksiyonu kullanılmalıdır:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***(2019, 3, 30)  
  
 *writeln*(now.getFullYear())  
}  
  
*main*()

***Anahtar Notlar:*** *ES’ de deprecated olan fonksiyonlar kesinlikle kullanılmamalıdır. Bazı ES motorları deprecated fonksiyonları silebilir. Bu durumda kod taşınabilir olmaz.*

getTime fonksiyonu 1.1.1970 gece yarısı ile o tarih arasındaki milisaniye değerini döndürür:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **let** now = **new *Date***()  
  
 *writeln*(now.getTime())  
}  
  
*main*()

Tüm get fonksiyonların set fonksiyonu karşılığı da bulunmaktadır.

**Sınıf Çalışması:** 17.08.1999 senesinde 03:02:00 zamanında meydana gelen deprem ile şu anki zaman arasında tam olarak kaç yıl geçtiğini hesaplayan programı yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **const** DAY\_FACTOR = 1000 \* 60 \* 60 \* 24 \* 365  
 **let** eqDate = **new *Date***(1999, 7, 17, 3, 2)  
 **let** now = **new *Date***()  
  
 **let** days = now.getTime() - eqDate.getTime()  
  
 days /= DAY\_FACTOR  
  
 *writeln*(days)  
}  
  
*main*()

Date nesnesinin now fonksiyonu 1.1.1970 ile çağrıldığı zaman arasında milisaniye değerini döndürür. Yukarıdaki örnek now fonksiyonu ile de yapılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **const** DAY\_FACTOR = 1000 \* 60 \* 60 \* 24 \* 365  
 **let** eqDate = **new *Date***(1999, 7, 17, 3, 2)  
   
 **let** days = ***Date***.now() - eqDate.getTime()  
  
 days /= DAY\_FACTOR  
  
 *writeln*(days)  
}  
  
*main*()

**Sınıf Çalışması:** Aşağıdaki elemanları içeren bir Person nesnesi oluşturunuz.

* Ad soyad
* T.C. Kimlik No
* Doğum tarihi

Person nesnelerinden oluşan bir dizi yaratınız. Bu diziyi parametre olarak alıp kişilerin yaş ortalamasını bulan averageAges isimli global fonksiyonu yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Person* = **function** (name, citizenId, birthDate) {  
 **const** DAY\_FACTOR = 1000 \* 60 \* 60 \* 24 \* 365  
 **this**.**name** = name  
 **this**.**citizenId** = citizenId  
 **this**.**birthDate** = **new *Date***(birthDate.getTime())  
 **this**.getAge = **function** () {  
 **return** (***Date***.now() - **this**.**birthDate**.getTime()) / DAY\_FACTOR  
 }  
}  
  
**function** *averageAges*(people)  
{  
 **return** people.reduce((total, p) => total + p.getAge(), 0) / people.**length**}  
  
**function** *main*()  
{  
 **let** people = [  
 **new** *Person*(**"Ekrem"**, **"1234567756"**, **new *Date***(1994, 8, 5)),  
 **new** *Person*(**"Oğuz"**, **"1234567758"**, **new *Date***(1976, 8, 10)),  
 **new** *Person*(**"Elif"**, **"1234567752"**, **new *Date***(1995, 6, 20)),  
 **new** *Person*(**"Fethi"**, **"1234567750"**, **new *Date***(1996, 7, 15))  
 ]  
 *writeln*(*averageAges*(people))  
}  
  
*main*()

Date nesnesinin diğer fonksiyonları dökümanlardan incelenebilir.

**Rasgele Sayı Üretimi**

ES’ de rasgele sayı üretimi için bir nesne bulunmamaktadır. Rastgele sayı üretimi Math nesnesinin random isimli fonksiyonu ile yapılabilir. Random fonksiyonu [0, 1) aralığında rasgele belirlenmiş bir sayı üretir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *main*()  
{  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(***Math***.random())  
}  
  
*main*()

Belirli aralıkta rasgele sayılar programcı tarafından yazılan fonksiyonlar ile elde edilebilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *randomBoolean*()  
{  
 **return** *randomInt*(0, 2) == 1}  
  
**function** *randomInt*(min, max) *//[min, max)*{  
 **return** *parseInt*(*randomNumber*(min, max))  
}  
  
**function** *randomNumber*(min, max) *//[min, max)*{  
 **return *Math***.random() \* (max - min) + min  
}  
  
**function** *main*()  
{  
 **let** min = 10, max = 20  
  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(*randomInt*(min, max))  
  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(*randomNumber*(min, max))  
  
 *writeln*(**"\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(*randomBoolean*())  
}  
  
*main*()

**Sınıf Çalışması:** Bir paranın yazı gelme olasılığını yaklaşık olarak hesaplayan program yazınız.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *randomBoolean*()  
{  
 **return** *randomInt*(0, 2) == 1}  
  
**function** *randomInt*(min, max) *//[min, max)*{  
 **return** *parseInt*(*randomNumber*(min, max))  
}  
  
**function** *randomNumber*(min, max) *//[min, max)*{  
 **return *Math***.random() \* (max - min) + min  
}  
  
**function** *getTailProb*()  
{  
 **let** n = 100000  
 **let** sum = 0  
 **for** (**let** i = 0; i < n; ++i)  
 sum += *randomInt*(0, 2)  
  
 **return** sum / n  
}  
  
**function** *main*()  
{  
 *writeln*(*getTailProb*())  
}  
  
*main*()

**Sınıf Çalışması:** 1.1.1900 ile 31.12.2100 tarihleri arasında rastgele geçerli bir tarih üreten getRandomDate isimi fonksiyonu yazınız ve test ediniz.

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *randomBoolean*()  
{  
 **return** *randomInt*(0, 2) ? **true** : **false**}  
  
**function** *randomInt*(min, max) *//[min, max)*{  
 **return** *parseInt*(*randomNumber*(min, max))  
}  
  
**function** *randomNumber*(min, max) *//[min, max)*{  
 **return *Math***.random() \* (max - min) + min  
}  
  
**function** *getRandomDate*(minYear, maxYear)  
{  
 **let** ms1 = **new *Date***(minYear, 0, 1).getTime()  
 **let** ms2 = **new *Date***(maxYear,11, 31).getTime() + 1  
  
 **return new *Date***(*randomInt*(ms1, ms2))  
}  
  
**function** *main*()  
{  
 *writeln*(*getRandomDate*(1900, 2100).toString())  
}  
  
*main*()

**Sınıflar**

ES ile nesne yönelimli programlama ES6 öncesinde de yapılabilmekteydi. Bunun için bir object ilk değer verme sentaksı ya da bir fonksiyon bildirimi kullanılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (id, name, stock) {  
 **this**.**id** = id  
 **this**.**name** = name  
 **this**.**stock** = stock  
 **this**.toString = () => **"["** + **this**.**id** + **"]"** + **this**.**name** + **"-"** + **this**.**stock**  
}  
  
**function** *main*()  
{  
 **let** p = **new** *Product*(1234, **"laptop"**, 10)  
  
 *writeln*(p.toString());  
  
}  
  
*main*()

Örneğin bir nesne türünden dizi bildirimi de yapılabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (id, name, stock) {  
 **this**.**id** = id  
 **this**.**name** = name  
 **this**.**stock** = stock  
 **this**.toString = () => **"["** + **this**.**id** + **"]"** + **this**.**name** + **"-"** + **this**.**stock**  
}  
  
**function** *main*()  
{  
 **let** products = [  
 **new** *Product*(1234, **"laptop"**, 10),  
 **new** *Product*(12345678, **"mouse"**, 11),  
 **new** *Product*(12345, **"klavye"**, 10)]  
  
 **for** (**let** p **of** products)  
 *writeln*(p.toString())  
}  
  
*main*()

Burada new operatörü programcının nesne bildiriminde kullanılan fonksiyon da this dönülmediği için zorunludur. Eğer nesne bildiriminde kullanılan fonksiyon this dönerse new operatörü kullanılmasa da olur. Fakat bu şekilde nesne yaratılması this dönülse bile her javascript yorumlayıcısında çalışmayabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**let** *Product* = **function** (id, name, stock) {  
 **this**.**id** = id;  
 **this**.**name** = name;  
 **this**.**stock** = stock;  
 **this**.toString = () => **"["** + **this**.**id** + **"]"** + **this**.**name** + **"-"** + **this**.**stock**;  
  
 **return this**;  
};  
  
**function** *main*()  
{  
 **let** products = [  
 **new** *Product*(1234, **"laptop"**, 10),  
 *Product*(12345678, **"mouse"**, 11),  
 *Product*(12345, **"klavye"**, 10)];  
  
 **for** (**let** p **of** products)  
 *writeln*(p.toString());  
}  
  
*main*()

ES ile object içerisindeki değişkenler iki tırnak ile de bildirilebilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**function** *toString*()  
{  
 **return `[**${**this**.**id**}**]**${**this**.**name**}**-**${**this**.**stock**}**`**}  
  
**function** *main*()  
{  
 **let** product = {  
 **"id"**: 12345,  
 **"name"**: **"laptop"**,  
 **"stock"**: 10,  
 **"toString"** : *toString* }  
  
 *writeln*(product.*toString*())  
}  
  
*main*()

ES6 ile birlikte artık bir sınıf bildirimi, türetme, çok biçimlilik gibi kavramlar da dile dahil edilmiştir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**class** Product {  
 constructor(id, name, stock)  
 {  
 **this**.**Id** = id;  
 **this**.**Name** = name;  
 **if** (***arguments***.**length** >= 3)  
 **this**.**Stock** = stock;  
 **else  
 this**.**Stock** = 0;  
 }  
  
 toString()  
 {  
 **return "["** + **this**.**Id** + **"]"** + **this**.**Name** + **"-"** + **this**.**Stock**;  
 }  
}  
  
**function** *main*()  
{  
 **let** p = **new** Product(1, **"laptop"**, 10)  
  
 *writeln*(p.toString())  
  
 p.**Stock** -= 3  
  
 *writeln*(p.toString())  
}  
  
*main*()

Burada Product sınıfı içerisinde bildirilen constructor (ctor) içerisinde aynı zamanda property elemanları da (sınıfın veri elemanları gözüyle de bakılabilir) tanımlanmıştır. Sınıflar içerisinde bildirilen metotlarda (fonksiyonlar) function anahtar sözcüğü kullanılmaz.

ES6 ile bir sınıf bildirimi içerisinde sınıfın elemanlarına erişim için get ve set property leri yazılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**class** Point {  
 constructor(x, y)  
 {  
 **if** (arguments.**length** > 2)  
 **throw new *Error***(**"Argument error"**)  
  
 **let** status = arguments.**length** != 2  
  
 **this**.**\_x** = status ? 0 : x;  
 **this**.**\_y** = status ? 0 : y;  
 }  
  
 **get** x() { **return this**.**\_x**; }  
  
 **set** x(value) { **this**.**\_x** = value;}  
  
 toString()  
 {  
 **return `(**${**this**.**\_x**}**,** ${**this**.**\_y**}**)`** }  
}  
  
**function** *main*()  
{  
 **let** p = **new** Point(100, 100)  
  
 *writeln*(p.toString())  
  
 p.x = 200  
  
 *writeln*(p.toString())  
}  
  
*main*()

ES6 ile birlikte bir sınıfın static fonksiyonu da olabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**class** Point {  
 constructor(x, y)  
 {  
 **if** (arguments.**length** > 2)  
 **throw new *Error***(**"Argument error"**)  
  
 **let** status = arguments.**length** !== 2  
  
 **this**.**\_x** = status ? 0 : x;  
 **this**.**\_y** = status ? 0 : y;  
 }  
  
 **static** *distance*(x1, y1, x2, y2)  
 {  
 **return *Math***.sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2))  
 }  
  
 **get** x() { **return this**.**\_x**; }  
  
 **set** x(value) { **this**.**\_x** = value;}  
  
 toString()  
 {  
 **return `(**${**this**.**\_x**}**,** ${**this**.**\_y**}**)`** }  
}  
  
**function** *main*()  
{  
 **let** x1 = 100  
 **let** y1 = 100  
 **let** x2 = 50  
 **let** y2 = 50  
  
 *writeln*(Point.*distance*(x1, y1, x2, y2))  
}  
  
*main*()

ES6 ile birlikte bir sınıf başka bir sınıftan da türetilebilmektedir. Burada Java programlama dilindeki gibi extends anahtar sözcüğü kullanılmaktadır.

ES’ de türemiş sınıf ctor’u içerisinde taban sınıfın ctor’unu çağırmak programcının sorumluluğundadır. Programcı bunu super ctor sentaksı ile yapabilir. Bu sentaks ES’ye gore zorunludur. Bazı ES motorları super ctor sentaksı kullanılmadan da türetmeye izin verse de super ctor sentaksı mutlaka kullanılmalıdır. super anahtar sözcüğü taban sınıfın diğer fonksiyonlarına da ulaşmak için kullanılabilmektedir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**class** A {  
 constructor()  
 {  
 *writeln*(**"A constructor"**)  
 }  
 foo() { *writeln*(**"A foo"**)}  
}  
  
**class** B **extends** A {  
 constructor()  
 {  
 **super**()  
 *writeln*(**"B.constructor"**)  
 }  
 foo()  
 {  
 *writeln*(**"B.foo"**)  
 **super**.foo()  
 }  
}  
  
**function** *main*()  
{  
 **let** b = **new** B()  
}  
  
*main*()

Yukarıdaki örnekte foo fonksiyonu aslında override edilmiş gözüyle bakılabilir.

ES6 çoklu türetmeyi desteklemez.

Aşağıdaki örneği inceleyiniz:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
**class** Circle {  
 constructor(r)  
 {  
 **if** (arguments.**length** > 1)  
 **throw new *Error***(**"radius can not be negative"**);  
  
 **this**.**\_r** = arguments.**length** === 0 ? 0 : ***Math***.abs(r)  
 }  
 **get** r() { **return this**.**\_r**; }  
 **set** r(value) { **this**.**\_r** = ***Math***.abs(value); }  
 **get** area() { **return *Math***.**PI** \* **this**.**\_r** \* **this**.**\_r**; }  
 **get** perimeter() { **return** 2 \* ***Math***.**PI** \* **this**.**\_r**; }  
}  
  
**class** Point {  
 constructor(x, y)  
 {  
 **if** (arguments.**length** > 2)  
 **throw new *Error***(**"Argument error"**);  
  
 **this**.**\_x** = arguments.**length** === 0 ? 0 : x  
 **this**.**\_y** = arguments.**length** !== 2 ? 0 : y  
 }  
  
 **static** *distance*(x1, y1, x2, y2)  
 {  
 **return *Math***.sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2))  
 }  
  
 **get** x() { **return this**.**\_x**; }  
 **set** x(value) { **this**.**\_x** = value; }  
  
 **get** y() { **return this**.**\_y**; }  
 **set** y(value) { **this**.**\_y** = value; }  
  
 offset(dx, dy)  
 {  
 **this**.**\_x** += dx  
 **this**.**\_y** += dy  
 }  
  
 toString() { **return "{X="** + **this**.**\_x** + **", Y="** + **this**.**\_y** + **"}"**}  
}  
  
  
**class** AnalyticalCircle **extends** Circle {  
 constructor(r, a, b)  
 {  
 **if** (arguments.**length** > 3)  
 **throw new *Error***(**"Argument error"**);  
  
 **if** (arguments.**length** !== 0)  
 **super**(r)  
 **else  
 super**.r = 0  
  
 **if** (arguments.**length** === 1) {  
 **this**.**\_center** = **new** Point(0, 0)  
 }  
 **else if** (arguments.**length** === 2) {  
 **this**.**\_center** = **new** Point(a.X, a.Y)  
 }  
 **else** {  
 **this**.**\_center** = **new** Point(a, b);  
 }  
 }  
  
 **get** center() { **return new** Point(**this**.**\_center**.x, **this**.**\_center**.y) }  
 **set** center(p) { **this**.**\_center** = **new** Point(p.**x**, p.**y**)}  
  
 **get** x() { **return this**.**\_center**.x }  
 **set** x(value) { **this**.**\_center**.x = value}  
  
 **get** y() { **return this**.**\_center**.y; }  
 **set** y(value) { **this**.**\_center**.y = value; }  
  
 offset(dx, dy)  
 {  
 **this**.**\_center**.offset(dx, dy)  
 }  
}  
  
  
**function** *main*()  
{  
 **let** c = **new** AnalyticalCircle(3, 100, 100)  
  
 *writeln*(**"Yarıçap:"** + c.r)  
 *writeln*(**"Alan:"** + c.area)  
 *writeln*(**"Merkez:"** + c.center.toString())  
  
 c.offset(30, -45)  
  
 *writeln*(**"Yarıçap:"** + c.r)  
 *writeln*(**"Alan:"** + c.area)  
 *writeln*(**"Merkez:"** + c.center.toString())  
}  
  
*main*()

Rasgele sayı üretimi için aşağıdaki gibi bir sınıf yazılabilir:

**function** *writeln*(a)  
{  
 ***console***.log(a)  
}  
  
class Random {  
 static *nextDouble*(min, max) *//[min, max)* {  
 if (min >= max)  
 throw new Error("min must be less than max")  
  
 return Math.random() \* (max - min) + min  
 }  
  
 static *nextInt*(min, max) *//[min, max)* {  
 if (min >= max)  
 throw new Error("min must be less than max")  
  
 return Math.floor(Math.random() \* (max - min) + min)  
 }  
  
 static *nextBoolean*()  
 {  
 return this.*nextInt*(0, 2) === 1;  
 }  
}

**function** *main*()  
{  
 **let** r = **new** Random()  
  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(r.nextInt(1, 100))  
  
  
 *writeln*(**"\*\*\*\*\*\*\*\*\*"**);  
  
 **for** (**let** i = 0; i < 10; ++i)  
 *writeln*(r.nextBoolean())  
  
}  
  
*main*()

ES'de tohum değerini dışarıdan alabilen bir rasgele sayı üreticisi yoktur. Bu durumda programcının yazması gerekir.

**Hata Ayıklama (Error/Exception Handling)**

ES yorumlayıcı ile çalışan bir programlama dili olduğundan sentaks ve semantic hatalar da çalışma zamanı sırasında oluşur. Yani örneğin genel olarak akış sentaks ya da semantik olarak hatalı bir koda gelene kadar sentaks ve/veya semantic hata anlaşılamayabilir. Bu durum genel hatalı durumlar için bazı yorumlayıcılar tarafından anlaşılabilse de programcı buna dikkat etmelidir. Üstelik yorumlayıcıdan yorumlayıcıya sentaks ve semantic hataların anlaşılması değişkenlik gösterebilir:

function *writeln*(a)  
{  
 console.log(a)  
}

let a = 10  
  
if (a) {  
 *writeln*("Doğru")  
}  
else {  
 foo("Test")  
}

Burada akış if deyiminin doğru kısmını çalıştırdığından foo isminin var olup olmadığının önemi yoktur. Akış else kısmına gelirse error oluşur.

Bu durumda ES için 3 çeşit hata vardır denebilir: Sentaks hataları (Syntax errors), Çalışma zamanı hataları (Runtime Errors) ve Mantıksal hatalar (Logical Errors):

Sentaks hataları dilin sözdizimine yönelik hatalardır. Bu hatalar ES’ de sadece hatnın olduğu akışı etkiler (thread). Eş zamanlı (asenkron) diğer akışlar bundan etkilenmez. Bu hatalar yakalanıp işlenecek hatalardan değildir.

Çalışma zamanı hataları program çalışma zamanında olası hata durumlarını (exception) temsil eder. Programcının yakalayıp işleyeceği hatalardandır. Çalışma zamanında oluşan durumların her zaman hata olması gerekmez. Yakalanmaması durumunda yalnızca hatanın meydana geldiği akış sonlanır.

Mantıksal hatalar yakalanması mümkün olmayan hatalardır. Santaks ya da çalışma zamanı hataları gibi değil, script içerisinde mantıksal yapılan bir hatalı durumda meydana gelir. Bu durum genelde kullanılan ortamın iyi bilinmemesinden kaynaklanır.

ES’ de hata ayıklama try, catch, throw ve finally anahtar sözcükleri ile yapılır. ES’de herhangi bir tür fırlatılabileceği gibi genel olarak çalışma zamanı sırasında Error sınıfı türünden nesneler fırlatılır. ES’ de Error sınıfı dışında fırlatılabilecek bazı hata nesneleri de bulunur:

* EvalError: Bu hata nesnesi eval isimli global fonksiyonu tarafından fırlatılır

eval fonskiyonu paramtresi ile aldığı yazıyı ES olarak yorumlayıp çalıştırır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 try {  
 let expression = 'writeln("Merhaba"); console.log("nasılsın")'  
  
 *eval*(expression)  
 }  
 catch (e) {  
 *writeln*(e.toString())  
 }  
}  
  
  
*main*()

Burada eval fonskiyonuna verilen kod parçasında hata oluştuğunda yakalanabilir

* RangeError: Bu hata numeric bir değişkenin sınır değerleri dışında değer aldığında fırlatılır.
* ReferenceError: Bu hata geçersiz bir referansa erişim sağlandığında oluşur.
* SyntaxError: Herhangi bir sentaks hatası oluştuğunda fırlatılır. Çoğu yorumlayıcı yakalanmasına izin vermez
* TypeError: Bir değişkenin kullanımında türün hatalı olması durumunda fırlatılır:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
function *main*()  
{  
 try {  
 let a = "ankara"  
  
 a.toUpperCase()  
 a.foo()  
 }  
 catch (e) {  
 *writeln*(e.toString())  
 }  
}  
  
  
*main*()

Bu sınıflar dışında kullanılan ortama göre başka hata nesneleri de olabilmektedir.

Bir hata throw anahtar sözcüğü ile aşağıdaki şekillerde fırlatılabilir:

* throw new Error([message])
* throw([message])
* throw new <sınıf ismi>([message])

Örneğin:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class MatUtil {  
 static *log*(value)  
 {  
 if (value < 0)  
 throw new Error("Belirsiz")  
 if (value == 0)  
 throw new Error("Tanımsız")  
  
 return Math.log(value)  
 }  
}  
  
function *main*()  
{  
 *writeln*(MatUtil.*log*(0))  
}  
  
  
*main*()

Ya da örneğin:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class MatUtil {  
 static *log*(value)  
 {  
 if (value < 0)  
 throw("Belirsiz")  
 if (value == 0)  
 throw("Tanımsız")  
  
 return Math.log(value)  
 }  
}  
  
function *main*()  
{  
 *writeln*(MatUtil.*log*(0))  
}  
  
  
*main*()

Programcı isterse Error sınıfından türetme yaparak kendi hata sınıfını da yazabilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class MathError extends Error {  
 constructor(message, errCode)  
 {  
 super(message)  
 this.\_errCode = errCode  
 }  
  
 get errCode() {return this.\_errCode}  
}  
  
class MatUtil {  
 static *log*(value)  
 {  
 if (value < 0)  
 throw new MathError("Belirsiz", -1)  
 if (value == 0)  
 throw new MathError("Tanımsız", 0)  
  
 return Math.log(value)  
 }  
}  
  
function *main*()  
{  
 *writeln*(MatUtil.*log*(0))  
}  
  
*main*()

ES’ de akış error bakımından ele alınacaksa (handling) try bloğu içerisinde olmalıdır. try bloğu bir bir catch bloğu ve finally bloğu veya tak başına catch veya finally bloğu ile devam edebilir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class MathError extends Error {  
 constructor(message, errCode)  
 {  
 super(message)  
 this.\_errCode = errCode  
 }  
  
 get errCode() {return this.\_errCode}  
}  
  
class MatUtil {  
 static *log*(value)  
 {  
 if (value < 0)  
 throw new MathError("Belirsiz", -1)  
  
 if (value == 0)  
 throw new MathError("Tanımsız", -2)  
  
 return Math.log(value)  
 }  
}  
  
function *main*()  
{  
 try {  
 *writeln*(MatUtil.*log*(0))  
 }  
 catch (e) {  
 *writeln*(`Message:${e.message}, Error Code:${e.errCode}`)  
 }  
 finally {  
 *writeln*("finally")  
 }  
}  
  
*main*()

try-finally kullanımı:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class MathUtil {  
 static *log*(value)  
 {  
 if (value < 0)  
 throw("Belirsiz")  
  
 if (value == 0)  
 throw("Tanımsız")  
  
 return Math.log(value)  
 }  
}  
  
function *doWork*(value)  
{  
 try {  
 let result = MathUtil.*log*(value)  
  
 *writeln*(`result=${result}`)  
 }  
 finally {  
 *writeln*("doWork finally")  
 }  
}  
  
function *main*()  
{  
 try {  
 *doWork*(0)  
 }  
 catch (e) {  
 *writeln*(`Message:${e}`)  
 }  
 finally {  
 *writeln*("main finally")  
 }  
}  
  
*main*()

try bloklarının geri kalan çalışma sistematiği Java ile çok benzerdir.

***Anahtar Notlar:*** *Bilindiği gibi Java’ da exception sınıfları checked ve unchecked olmak üzere iki gruba ayrılır. ES’ de böyle bir durum yoktur.*

**Türlerin prototype elemanı**

Her türün prototype denilen bir elemanı vardır. Bu eleman sayesinde bir türe yeni bir eleman eklenebilmektedir:

Aşağıdaki örnekte s referansının gösterdiği nesneye count isimli bir metot eklenmiştir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class Sample {  
 constructor(value)  
 {  
 this.\_value = value  
 }  
  
 get value() {return this.\_value}  
}  
function *main*()  
{  
 let s = new Sample(10000)  
  
 s.count = function () {  
 return this.\_value == 0 ? 1 : *parseInt*(Math.log10(Math.abs(this.\_value)) + 1)  
 }  
  
 *writeln*(s.count())  
  
 let k = new Sample(123)  
  
 *writeln*(k.count()) //error  
  
  
}  
  
*main*()

prototype elemanı ile Sample sınıfına aşağıdaki gibi count metodu eklenmiştir:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
class Sample {  
 constructor(value)  
 {  
 this.\_value = value  
 }  
  
 get value() {return this.\_value}  
}  
function *main*()  
{  
 let s = new Sample(10000)  
  
 Sample.prototype.count = function () {  
 return this.\_value == 0 ? 1 : *parseInt*(Math.log10(Math.abs(this.\_value)) + 1)  
 }  
  
 *writeln*(s.count())  
  
 let k = new Sample(123)  
  
 *writeln*(k.count())  
}  
  
*main*()

Örneğin:

function *writeln*(a)  
{  
 console.log(a)  
}  
  
  
function *main*()  
{  
 let a = 10  
  
 Number.prototype.count = function () {  
 return this == 0 ? 1 : *parseInt*(Math.log10(Math.abs(this)) + 1)  
 }  
  
 *writeln*(a.count())  
 *writeln*((123).count())  
}  
  
*main*()

**Modüller**

Bir js dosyasına modül denir. ES6 dan önce ES’de bir js dosyasından başka bir js dosyasındaki bir ismi kullanmanın doğrudan bir yolu yoktu. Bu sebeple bazı ES kullanann teknolojiler bu işlemi yapabilecek araçları da barındırıyorlardı. Örneğin NodeJS teknolojisinde farklı modüller ile çalışmak ES6 olmasa da mümkündür.

***Anahtar Notlar:*** *ES6’ ya bazı durumlarda çıktığı yıl olan 2015 dolayısıyla ES2015 de denilmektedir. Örneğin NodeJS terminolojisinde modüller iki gruba ayrılır: CommonJS moduller ya da NodeJS moduller, ES2015 veya ES6 modüller.*

***Anahtar Notlar:*** *ES2015 ile birlikte modül dosyalarının uzantılarının mjs biçiminde seçilmesi yaygınlaşmıştır. Bu uzantı okunabilirliği artırdığı bazı teknolojilerde kendi modül sistemi yerine ES2015 modül kullanıldığı durumu da ayrıştırmış olur.*

***Anahtar Notlar:*** *ES6 ile**birlikte bazı durumların kısıtlı bir biçimde kullanılmasını veya doğrudan kullanılamaması gibi kuralların devreye girmesine “strict mode” denilmektedir. ES6’nın ilk zamanlarında bu şekilde bildirim gerekebiliyordu. Ancak şu an ES6’dan sonraki sürümler yaygın olarak kullanıldığında bu bildirime çoğu zaman gerek yoktur. Zaten kısıtlar vardır ve kuralları da belirlidir.*

Bir modül içerisinde kullanılan bir isme başka bir modülden doğrudan eirişilemez. Bunun için modül içerisinde bir isim export anahtar sözcüğü ile bildirilmelidir. export anahtar sözcüğün iki kullanımı vardır: isimlerin export edilmesi (named exports), varsayılan export (default export)

**Modül içerisinden export edilmesi (named exports)**

Bu kullanımın genel biçimi:

export <isim>

veya

export {<isim listesi>}

export function *writeln*(msg)  
{  
 console.log(msg)  
}

export let x = 10

Yukarıdaki export değişkenler aynı script içerisinde ise aşağıdaki gibi bir export bildirimi yapılabilir:

function *writeln*(msg)  
{  
 console.log(msg)  
}  
  
let x = 10  
  
export {*writeln*, x}

Aşağıdaki örnekte bir sınıf export edilmiştir:

export class Random {  
 static *nextDouble*(min, max) *//[min, max)* {  
 if (min >= max)  
 throw new Error("min must be less than max")  
  
 return Math.random() \* (max - min) + min  
 }  
  
 static *nextInt*(min, max) *//[min, max)* {  
 if (min >= max)  
 throw new Error("min must be less than max")  
  
 return Math.floor(Math.random() \* (max - min) + min)  
 }  
  
 static *nextBoolean*()  
 {  
 return this.*nextInt*(0, 2) === 1;  
 }  
}

**Varsayılan export**

Bir modülün ismi zaten export edilmiştir. Ancak programcı isterse bir modülü default bir isimle de export edebilir. Bi modül içerisinde bir tane default yapılabilir:

export default *writeln*

**Modüllerin import edilmesi**

Bir modül içerisinde başka bir modülde export edilen bir elemanı kullanabilmek içim import anahtar sözcüğü kullanılır. Bir modül içerisinde birden fazla import bildirimi olabilir. Bildirim sırasının önemi yoktur. Import bildirimleri iki şekilde kullanılabilir: isimlerin import edilmesi, default export edilen modüllerin import edilmesi

İsimlerin import edilmesinin genel biçimi:

import {isim listesi} from <modül ismi>

Örneğin:

import {*writeln*} from "./util.mjs"  
import {Random} from "./Random.mjs"

Tüm isimler aşağıdaki gibi yıldız (\*) (asterisk oeprator) ile takma isim (alias) aşağıdaki gibi bildirilebilir:

import \* as util from "./util.mjs"

import bildirimde {} içerisindeki isimlere de takma isim verilebilir:

import {Random as R} from "./Random.mjs"

default export edilen modüllerin import edilmesi aşağıdaki gibi yapılabilir:

import *writeln* from "./util.mjs"

**Bazı yararlı global fonksiyonlar**

ES’ de NodeJS gibi teknolojilerde de çok sık kullanılan bazı global fonksiyonlar bulunur.

**setTimeout Fonksiyonu**

Bu fonksiyon verilen bir süre sonunda verilen bir arkaplan (callback) fonksiyonun çağrılmasını sağlar. Süre milisaniye cinsinden verilir:

import {*writeln*} from "./util.mjs"  
  
function main()  
{  
 *setTimeout*(() => {*writeln*("Süre doldu")}, 3000)  
  
 *writeln*("main bitti")  
}  
  
main()

Bu işlem asenkron yapılmaktadır. Yani varolan akış devam eder. Timeout ayrı bir akış olarak çalışmaktadır.

Set edilmiş olan bir timeout clearTimeout isimli bir fonksiyon ile devre dışı bırakılabilir:

import {*writeln*} from "./util.mjs"  
  
function main()  
{  
 let timeout = *setTimeout*(() => *writeln*("Süre doldu"), 4000)  
  
 let ms = *parseInt*(Math.random() \* 10000)  
  
 *writeln*(ms)  
  
 *setTimeout*(() => *clearTimeout*(timeout), ms)  
  
 *writeln*("main bitti")  
}  
  
main()

import {*writeln*} from "./util.mjs"  
  
function main()  
{  
 *setInterval*(() => *writeln*("."), 1000)  
  
 *writeln*("main bitti")  
}  
  
main()

Set edilmiş olan bir interval clearInterval isimli bir fonksiyon ile iptal edilebilir:

import {*writeln*} from "./util.mjs"  
  
function main()  
{  
 let interval = *setInterval*(() => *writeln*("."), 1000)  
  
 *setTimeout*(() => *clearInterval*(interval), 8000)  
  
 *writeln*("main bitti")  
}  
  
main()

**Sınıf Çalışması:** Saniyede bir çalışan bir interval içerisinde rasgele sayılar üretilsin. 3 tane asal sayı geldiğinde interval’I durduran program yazınız

import {*writeln*} from "./util.mjs"  
import {Random} from "./Random.mjs"  
import {*isPrime*} from "./numberutil.mjs"  
  
let count = 0  
let randomGeneratorInterval = *setInterval*(randomGeneratorIntervalCallback, 1000)  
  
function randomGeneratorIntervalCallback()  
{  
 *writeln*("Generating number...")  
 let val = Random.*nextInt*(1, 100)  
  
 if (*isPrime*(val)) {  
 *writeln*(`value:${val}`)  
 ++count  
 }  
  
 if (count === 3)  
 *clearInterval*(randomGeneratorInterval)  
}