# FRONT-END


# JAVASCRIPT


## FONKSİYONLAR

# FONKSİYONLAR (FUNCTIONS)

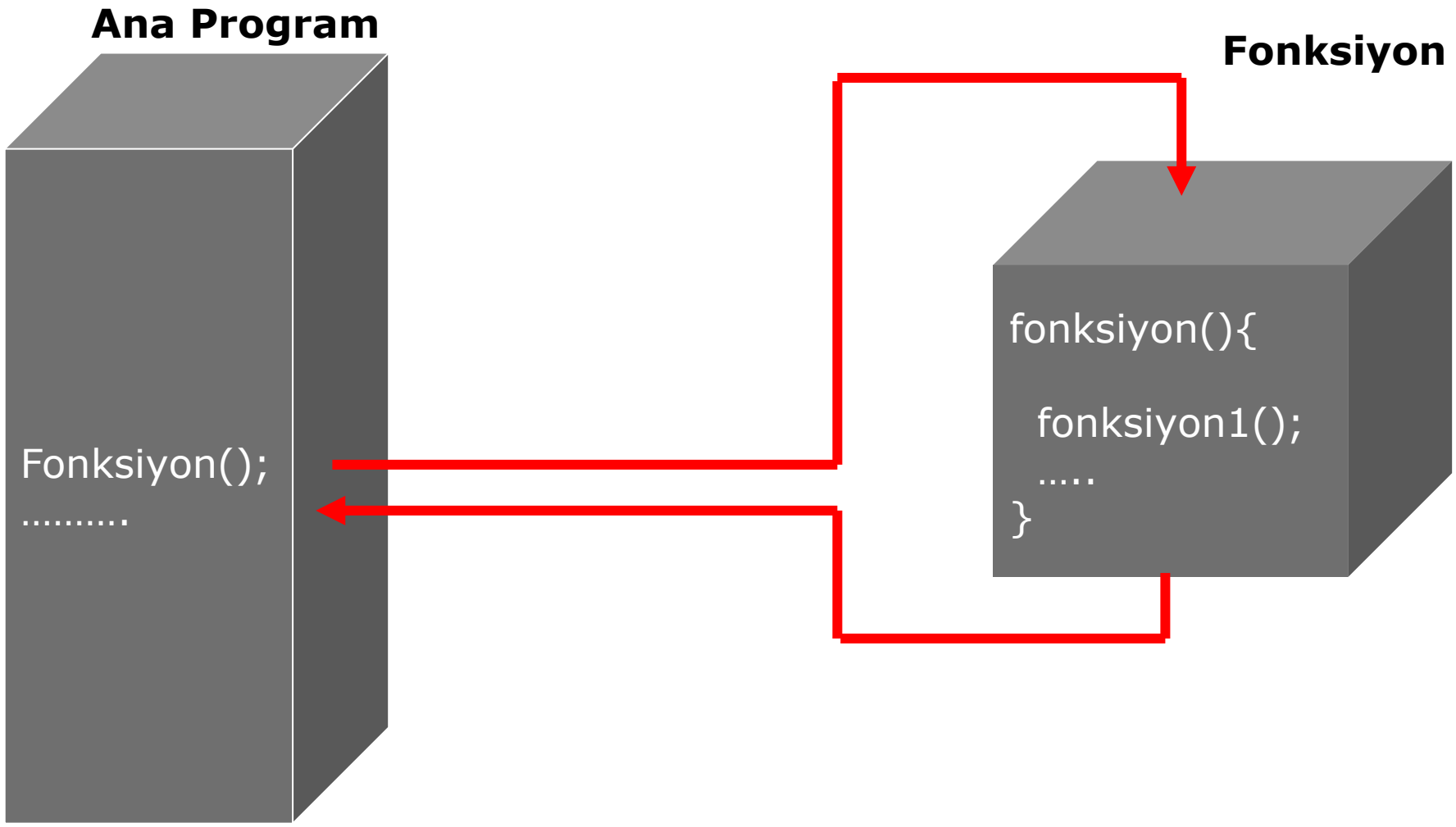
- Program içerisinde bazı kod kısımlarının birçok kere çalışması gerekebilir. (Tekrar-tekrar)

- Büyük problemler, küçük parçalara bölünerek kolay çözülür. (Böl-Parçala-Fethet)

- Hata ayıklama küçük ölçekte daha kolaydır. Küçük parçalara yoğunlaşmak daha kolaydır. (Kolay hata tespiti)

- Parçalara ayrılan problem çok sayıda insan tarafından paralel olarak çözülebilir (Takım çalışması).

# FONKSİYONLAR (FUNCTIONS)



# FONKSİYON TANIMLANMA (1.YÖNTEM)

**Parametre Listesi**

```
function fonsiyonİsmi(parametre1, parametre2,...){
  // işlemler
  // .....
  return dönüşDeğeri;
}
fonksiyonİsmi(); // fonksiyon çağrılıyor.
fonksiyonİsmi(); // fonksiyon çağrılıyor
```

**Fonksiyon Gövdesi**

- Javascript'te fonksiyonlar bir kaç farklı şekilde tanımlanabilmektedir. Bu yöntem klasik tanımlamadır.
- Bir fonksiyon bir kere tanımlanır ve program içerisinde istenildiği kadar çağrılabilir.
- Fonksiyonun tanımlanması çağrılmadan önce de yapılabilir sonra da yapılabilir.
- Bir Fonksiyon, argüman (parametre) almak zorunda değildir.
- Bir fonksiyon bir sayı döndürmek zorunda değildir. İhtiyaca göre fonksiyon düzenlenebilir.

# ÖRNEK-1

```
function yaşYazdır() {
  console.log(`Benim adım Mehmet ve Ben ${2021 - 1979} yaşındayım`);
}

yaşYazdır();
yaşYazdır();
yaşYazdır();
```

```
Benim adım Mehmet ve Ben 42 yaşındayım
Benim adım Mehmet ve Ben 42 yaşındayım
Benim adım Mehmet ve Ben 42 yaşındayım
>
```

- Fonksiyonumuz sadece Mehmet ve Yaşını konsola yazdırıyor.
- Farklı isimleri ve farklı yaşları yazdıracak şekilde daha modüler bir hale nasıl getirilir?

- **ÇÖZÜM:** Argüman kullanmak

# ÖRNEK-2

```
function yaşYazdır(ad, tarih) {
  console.log(`Benim adım ${ad} ve Ben ${2021 - tarih} yaşındayım`);
}

yaşYazdır('Murat', 1990);
yaşYazdır('John', 1980);
yaşYazdır('Ayşe', 1994);
```

```
Benim adım Murat ve Ben 31 yaşındayım
Benim adım John ve Ben 41 yaşındayım
Benim adım Ayşe ve Ben 27 yaşındayım
```

- Argümanlar ile fonksiyonumu daha kullanışlı bir hale geldi.
- Ancak, fonksiyonda hesapladığımız yaş bilgisini ana programda kullanmak istersek ne yapabiliriz?

- **ÇÖZÜM:** Dönüş değeri kullanmak

# ÖRNEK-3

```
function yaşYazdır(ad, doğumTarihi) {
  const yaş = 2021 - doğumTarihi;
  console.log(`Benim adım ${ad} ve Ben ${yaş} yaşındayım`);
  return yaş;
}

const yaşMurat = yaşYazdır('Murat', 1990);
const yaşJohn = yaşYazdır('John', 1980);
const yaşAyşe = yaşYazdır('Ayşe', 1994);

console.log(`Yaşların ortalaması = ${(yaşMurat + yaşJohn + yaşAyşe) / 3}`);
```

```
Benim adım Murat ve Ben 31 yaşındayım
Benim adım John ve Ben 41 yaşındayım
Benim adım Ayşe ve Ben 27 yaşındayım
Yaşların ortalamaması = 33
```

- Fonksiyonumuz sonuçları ekrana yazdırıyor ve hesapladığı bir değeri (**yaş**) ana programa döndürüyor.
- NOT: yaş değişkenin değeri ana programa döner.

# ÖRNEK-4

- 4 işlem hesap makinasını fonksiyonlar ile tekrardan yazalım.

```
const sayı1 = Number(prompt("1.sayıyı giriniz:"));
const işlem = prompt("İşlemi giriniz (+, -,?,/): ");
const sayı2 = Number(prompt("2.sayıyı giriniz:"));

if (işlem === "+") {
        sonuç = topla(sayı1, sayı2);
} else if (işlem === "-") {
        sonuç = çıkar(sayı1, sayı2);
} else if (işlem === "*") {
        sonuç = çarp(sayı1, sayı2);
} else if (işlem === "/") {
        sonuç = böl(sayı1, sayı2);
} else {
        alert("yanlış işlem");
}
console.log(`${sayı1} ${işlem} ${sayı2} = ${sonuç}`);
```

```
function topla(sayı1, sayı2) {
  return sayı1 + sayı2;
}

function çıkar(sayı1, sayı2) {
  return sayı1 - sayı2;
}

function çarp(sayı1, sayı2) {
  return sayı1 * sayı2;
}

function böl(sayı1, sayı2) {
  return sayı1 / sayı2;
}
```

# SORU

- Bir sayıyı parametre olarak alan ve bu sayının tek veya çift olduğunu hesaplayıp sonucu ana programa döndüren fonksiyonu yazınız.
- Fonksiyon sonucu **TEK** veya **ÇİFT** olarak döndürmelidir ve sonuç ana programda ekrana bastırılmalıdır.

```
function tekMi(x) {
      return x % 2 ? "TEK" : "ÇİFT";
}

const sayı = Number(prompt("Sayıyı giriniz:"));
console.log(`${sayı} ${tekMi(sayı)}`);
```

# FONKSİYON TANIMLAMA 2.YÖNTEM(EXPRESSION)

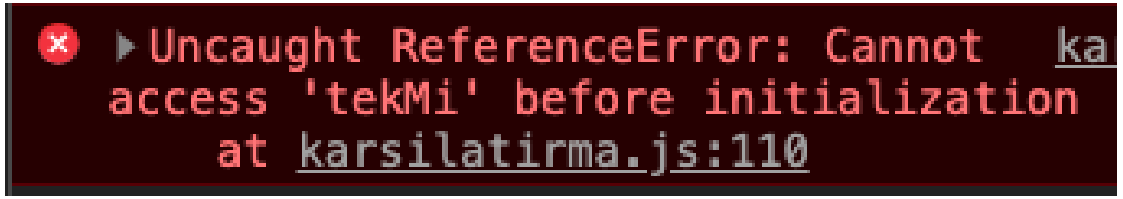
- Javascript'te fonksiyonlar ifade (**expression**) olarak da tanımlanabilmektedir.
- Bu yöntemde, fonksiyonlar **isimsizdir** (**anonymous**) ve bir **değişkene atanırlar**. Dolayısıyla fonksiyonun bir dönüş değeri olmalıdır.
- Bu değişken, fonksiyon olarak kullanılır.

```
// Function expression
const tekMi = function (x){
      return x % 2 ? "TEK" : "ÇİFT";
}

const sayı = Number(prompt("Sayıyı giriniz:"));
console.log(`${sayı} ${tekMi(sayı)}`);
```

# FONKSİYON TANIMLAMA 2.YÖNTEM(EXPRESSION)

- Bu yöntemde şekilde görüldüğü gibi fonksiyon tanımlanmadan önce çağrılırsa JS hata verecektir.
- Dolayısıyla **expression** yöntemini kullanmak için önce fonksiyonu tanımlamak sonra çağırmak gerekir.

```
const sayı = Number(prompt("Sayıyı giriniz:"));
console.log(`${sayı} ${tekMi(sayı)}`);

// Function expression
const tekMi = function (x){
      return x % 2 ? "TEK" : "ÇİFT";
}
```



# FUNCTION EXPRESSION AVANTAJI

- Fonksiyonları **expression** yöntemi ile kullanmanın klasik yöntem (**declaration**) ile kullanmaya nazaran **2 avantajı** bulunmaktadır.
  1. Programcıyı, önce fonksiyonların tanımlanması, sonra kullanılmasına zorladığı için aslında daha düzenli ve daha anlaşılır kod yazmaya olanak sağlamaktadır.

  2. Fonksiyonların ve değerlerin değişkenlerde saklanmasını gerektirmektedir. Bu da daha sade bir kodlama demektir.

NOT: Her iki yöntem de yaygın bir şekilde kullanılabilmektedir. Bu yüzden 2 yönteme de aşina olmakta fayda vardır.

# FONKSİYON TANIMLAMA 3.YÖNTEM(ARROW)

- **Arrow** (Ok) fonksiyonları fonksiyon tanımlamada yaygın bir şekilde kullanılan bir diğer yöntemdir.

- 2. Yöntem gibi **expression** tanımlamaya benzemektedir.

**Arrow fonksiyonu tanımlama**

```
let fonkAdı = (arg1, arg2, .., argN) => expression
```

**Expression Yöntemi ile fonksiyon tanımlama**

```
let fonkAdı = function(arg1, arg2, ..., argN) {
    return expression;
};
```

- **Arrow** fonksiyonları, expression Yöntemi ile fonksiyon tanımlamanın kısa yolu gibidir.

- **Tek satırlık** fonksiyon tanımlamak için çok elverişlidir.

# FONKSİYON TANIMLAMA KARŞILAŞTIRMA

## Arrow fonksiyonu

```
const topla = (a, b) => a + b;

alert( topla(1, 2) ); // 3
```

## Function expression

```
const topla = function(a, b){
        return a + b;
};

alert( topla(1, 2) ); // 3
```

## Function declaration

```
function topla(a,b){
        return a+b;
}

alert( topla(1, 2) ); // 3
```

# FONKSİYON TANIMLAMA (ARROW)

**ÖRNEK1:**

```
const yaşHesapla = (doğumTarihi) => 2022 - doğumTarihi;
alert(yaşHesapla(1979));
```

**ÖRNEK2:**

```
let selamVer = () => alert("Merhaba Arkadaşlar");
selamVer();
```

**ÖRNEK3:**

```
const üsAl = (taban,üs) => taban**üs;
console.log(üsAl(2,3));
```

**ÖRNEK4:**

```
const tekMi = (x) => x % 2 ? "TEK" : "ÇİFT";
console.log(tekMi(8));
```

# FONKSİYON TANIMLAMA (ÇOK SATIRLI ARROW)

**ÖRNEK1:**

```
const topla = (a, b) => {
      const sonuç = a + b;
      return sonuç;
};

alert(topla(3, 2) ); // 5
```

- Fonksiyon satır sayısı birden fazla ise **süslü parantez** kullanmalıyız.
- Eğer süslü parantez kullanıldı ise **return** de kullanmalıyız.

**NOT:** Eğer **tek satırlık** bir fonksiyon yazılacaksa **arrow** fonksiyonu yazmak çok daha avantajlı. Ama satır sayısı fazla ise o zaman diğer yöntemler ile çok benzer oluyor.

**ÖNEMLİ**: **Arrow** fonksiyonlarında **this** anahtar kelimesi kullanılamıyor. İleride bu konuya ayrıntılı değineceğiz.

# FONKSİYON TANIMLAMA

**SORU:** Bir dairenin alanını hesaplayan fonksiyonu **arrow** fonksiyon olarak yazınız. Yarıçap **prompt** ile girilmeli ve sonuç ana programda yazdırılmalıdır.

```
const r = +prompt("Yarıçapı Giriniz");

const alan = (r) => Math.PI * r * r;

console.log(`Alan(${r}): ${alan(r)}`);
```

**ÖNEMLİ**: **prompt** fonksiyonun önündeki **+** klavyeden girilen sayının **string** kabul edilmesini engellemektedir.

# FONKSİYON TANIMLAMA

**SORU:** Doğum tarihini parametre olarak alan ve ana programa yaşı hesaplayıp döndüren fonksiyonu yazınız.

```
const yaşYazdır =(doğumTarihi)=>`Yaşım ${new Date().getFullYear()-doğumTarihi}`;

alert(yaşYazdır(2000));
```

**Alternatif yol**

```
const yaşYazdır = (doğumTarihi) => {
        const yaş = new Date().getFullYear() - doğumTarihi ;
        return `Yaşım ${yaş}`;
}
alert(yaşYazdır(2000));
```

# FONKSİYONUN BAŞKA FONKSİYONU ÇAĞIRMASI

- Bir fonksiyon içerisinde bir başka fonksiyonun çağrılması mümkündür.

```
const meyveDilimle = (meyveSayısı) => meyveSayısı * 4;

const meyveSuyuHazırla = function (portakal, elma, muz) {
        const pDilimSayısı = meyveDilimle(portakal);
        const eDilimSayısı = meyveDilimle(elma);
        const mDilimSayısı = meyveDilimle(muz);
        const meyveSuyu = `Meyve Suyu ${pDilimSayısı} dilim portakal,
    ${eDilimSayısı} dilim elma ve ${mDilimSayısı} dilim muz'dan oluşmaktadır`;

        return meyveSuyu;
};

console.log(meyveSuyuHazırla(2, 1, 3));
console.log(meyveSuyuHazırla(4, 2, 1));
```

# FONKSİYONLARDA SCOPE KAVRAMI

- Değişkenler tanımlandığı konuma (scope) göre geçerliliği değişebilmektedir.

1. Bir değişken **fonksiyon içerisinde** tanımlanmış ise sadece o fonksiyon içerisinde geçerlidir. (**function-scope**).

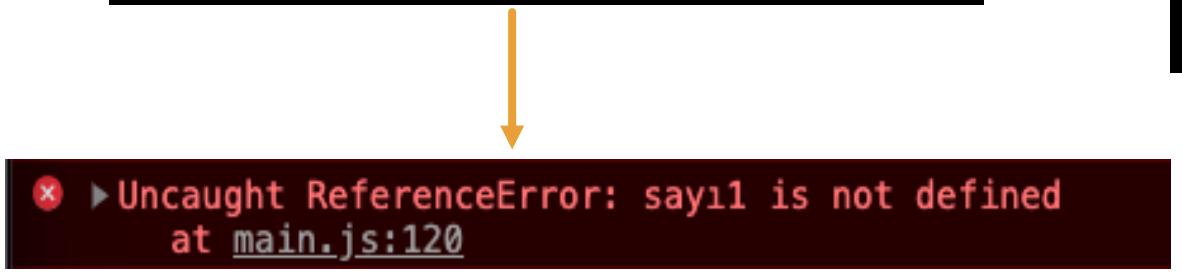
2. Değişken ana programda tanımlanmış ise tüm kod içerisinde geçerlidir (**global-scope)**

3. Sadece tanımlandığı alt alanda (blokta) geçerli olan değişkenlere **block-scope** değişkenler denilir.
   - **ES6** ile gelen bir özelliktir.

# FUNCTION-SCOPE VS GLOBAL SCOPE

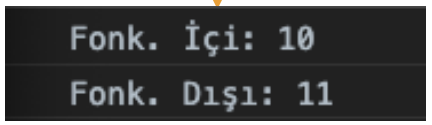
## Function-scoped

```
const fonk1 = function () {
        let sayı1 = 22;
        console.log(sayı1);
};
fonk1();
console.log(++sayı1);
```




## Global scope

```
let sayı2 = 5;
const fonk2 = function (){
        sayı2 = 10;
        console.log(`Fonk. İçi: ${sayı2}`);
}
fonk2();
console.log(`Fonk. Dışı: ${++sayı2}`);
```




**NOT:** Global scope'da değişkene fonksiyon içerisinden veya dışında erişilebilir. **Güvenlik** açısından sorun oluşturabilir.

# FUNCTION-SCOPE VS GLOBAL SCOPE

- **Global** ve **function** scope değişkenler aynı anda kullanılır ise:

```
let sayı3 = 3;
const fonk3 = function () {
  let sayı3 = 7;
  console.log(`Fonk. İçi: ${sayı3}`);
};
fonk3();
console.log(`Fonk. Dışı: ${++sayı3}`);
```

**Çıktı Ne Olur?**

```
Fonk. İçi: 7
Fonk. Dışı: 4
```

# BLOCK-SCOPE

```
const fonk4 = function (sayı4) {
        if (sayı4 < 0){
                let negatif = true;
        }
        console.log(negatif);
};
fonk4(-4);
```

```
▶Uncaught ReferenceError: negatif is not defined
    at fonk4 (main.js:153)
    at main.js:155
```

**NOT:**

- Block scope'da değişkene sadece tanımlandığı blok içerisinde (if bloğu, for bloğu v.b) içerisinden erişilebilir.
- Bu özellik **ES6** ile gelmiştir .
- Daha güvenilir ve okunabilir kod yazmaya olanak sağlamaktadır.

# SCOPE ÖZET

- Değişken tanımlaması yaparken mümkünse **en küçük scope** kullanmayı tercih etmek **hata ihtimalini** azaltacaktır.
- Bir değişken isim verirken **aynı ismi** defalarca farklı scope'larda güvenli bir şekilde kullanmanıza imkan tanıyacaktır.
- Bu sebeplerden dolayı önce **block**, sonra **function** en son olarak global scope kullanmayı tercih etmekte fayda vardır.
- Ama bu bir zorunluluk değildir. Kodlama tercihidir.
- Fonksiyonlar ile ilgili bazı İleri seviye konuları daha sonra ele alacağız.

# ÇALIŞMA SORULARI

**SORU1: Taban** ve **yükseklik** değerlerini parametre olarak alan ve bir üçgenin alanını hesaplayarak ana programa döndüren fonksiyonu yazınız.

**SORU2: kareAl**, **küpAl**, **üsAl** şeklinde üç adet farklı **arrow** fonksiyonu tanımlayın. Bu fonksiyonların ana programdan gereken parametreleri alarak sonuçları ana programa döndürmeli gerekmektedir.

**SORU3:** Yıl değerini parametre olarak alan ve bu yılın artık yıl olup olmadığını hesaplayarak sonucu ana programa döndüren fonksiyonu **function-expression** yöntemi ile yazınız. **NOT:** Yıl 4'e tam bölünüyorsa **VE** (100'e tam bölün**mü**yorsa **VEYA** 400'e tam bölünüyorsa) **artık yıldır** aksi takdirde değildir.