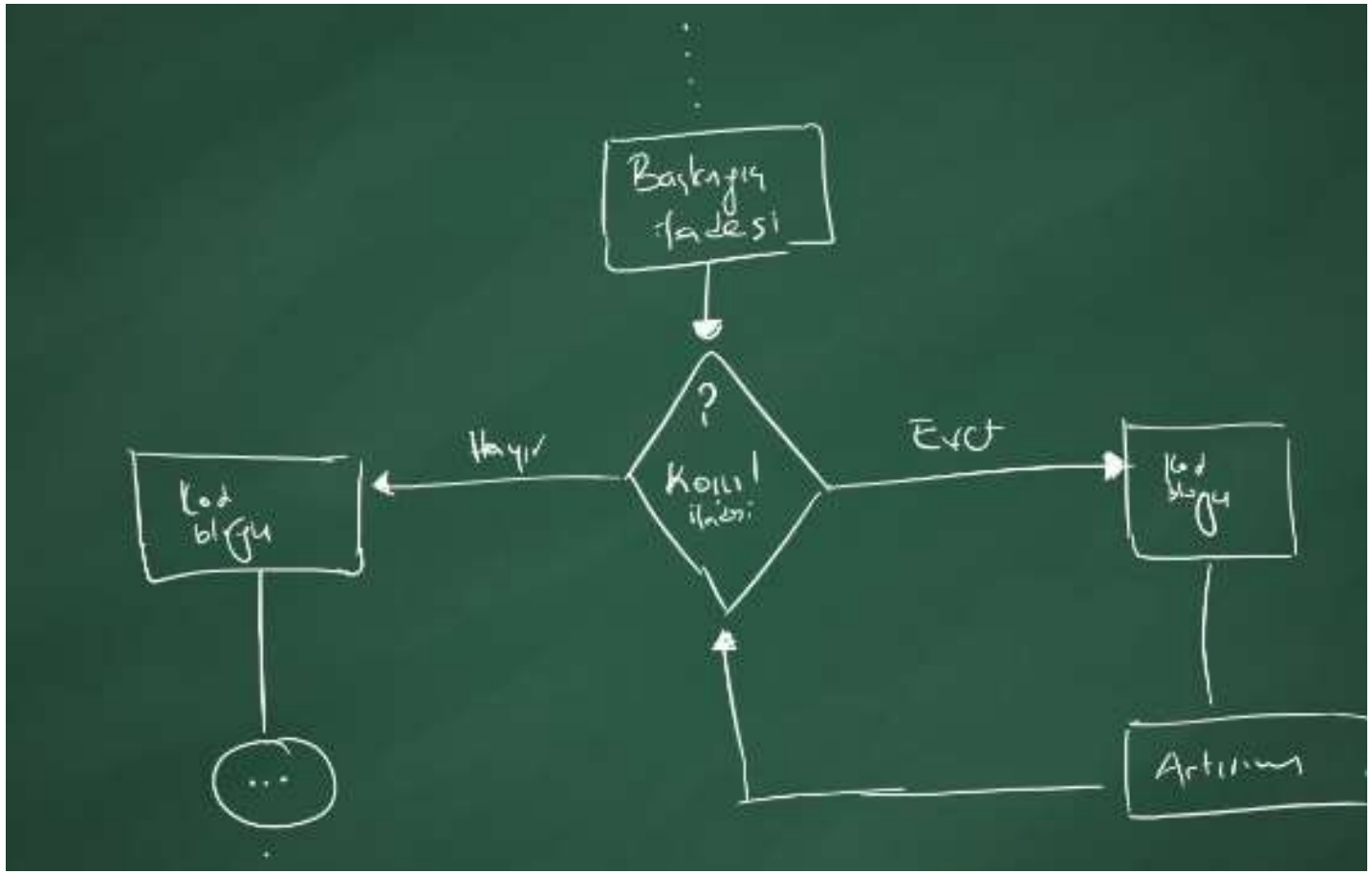

Başlangıç ifadesi
?
Koşul ifadesi
Hayır
Evet
...

```
static void Main(string[] args)
{
    int sayac = 10;
    while (sayac>0)
    {
        Console.WriteLine("{0,-3} {1,-3}",sayac,sayac*sayac);
        sayac += 1;
    }
    Console.ReadKey();
}
```

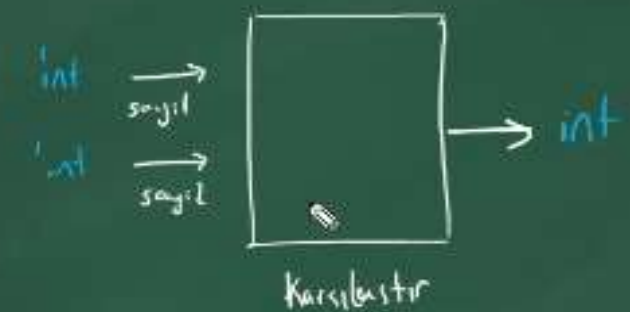
```
static void Main(string[] args)
{
    int a = Convert.ToInt32(Console.ReadLine());
    int b = Convert.ToInt32(Console.ReadLine());

    for (int i = a; i <= b; i++)
    {
        if (i  == 100)
            break;
        Console.Write("{0,3}",i);
    }

    Console.ReadKey();
}
```

```
static void Main(string[] args)
{
    int n = 5;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Console.Write(" {0},{1} ",i+1,j+1);
        }
        Console.WriteLine();
    }

    Console.ReadKey();
```

```
static void Main(string[] args)
{
    int n = 5;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j <= i ; j++)
        {
            Console.Write(" * ");
        }
        Console.WriteLine();
    }

    Console.ReadKey();
}
```

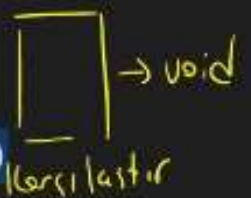
Metotlar

- Metot / Fonksiyon / Yordam / Altyordam
- Erişim belirteci — public, internal, protected, private...
  - Dönüş tipi — void
  - Metot Adı
  - Parametre Listesi
- static

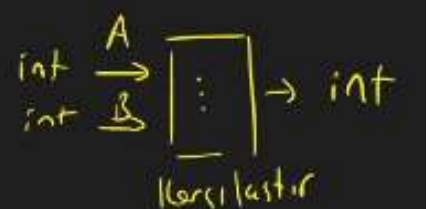



```
int Karsilastir(int, int)
```

```
static void Main(string[] args)
{
    Karsilastir(3,5);
    Console.WriteLine("Metot bitti.");
    Console.ReadKey();
}

public static void Karsilastir(int sayi1, int sayi2)
{

}
```




```
static void Main(string[] args)
{
    Karsilastir(3,5);
    Console.WriteLine("Metot bitti.");
    Console.ReadKey();
}

public static int Karsilastir(int A, int B)
{
    return 0;
}
```



```
static void Main(string[] args)
{
    int buyuk = Karsilastir(3,5);
    Console.WriteLine(buyuk);
    Console.ReadKey();
}

public static int Karsilastir(int A, int B)
{
    if (A>B)
    {
        return A;
    }
    else
    {
        return B;
    }
}
```

```
        var x = KareAl(3);
        double y = KareAl(x);

        Console.WriteLine(x);
        Console.WriteLine(y);

        Console.ReadKey();
    }

    public static int Karsilastir(int A, int B)...

    static double KareAl(double sayi)
    {
        double kare = sayi * sayi;
        return kare;
```

```
def my_function():
  print("Hello from a function")
my_function()
```

Argümanlar

Bilgi, işlevlere bağımsız değişken olarak iletilebilir.
Bağımsız değişkenler, işlev adından sonra parantez içinde belirtilir. İstediğiniz kadar argüman ekleyebilirsiniz, sadece virgülle ayırın.
Aşağıdaki örnek, tek bağımsız değişkenli (fname) bir işleve sahiptir.
İşlev çağrıldığında, işlevin içinde tam adı yazdırmak için kullanılan bir ad iletiriz:

```
def my_function(name):
  print(name + " Geldi")

my_function("Emel")
my_function("Tarık")
my_function("Nermin")
```

Argüman-Parametre Sayısı

Varsayılan olarak, bir işlev doğru sayıda bağımsız değişkenle çağrılmalıdır. Yani, işleviniz 2 argüman bekliyorsa, işlevi 2 argümanla çağırmalısınız, daha fazla veya daha az değil.

```
  print(name + " " + surname)
my_function("Emel", "uzun")
```

Dinamik Argümanlar, *args

İşlevinize kaç bağımsız değişkenin iletileceğini bilmiyorsanız *, işlev tanımında parametre adından önce bir ekleyin. Bu şekilde, işlev bir dizi bağımsız değişken alır ve *öğelere* buna göre erişebilir:

Argüman sayısı bilinmiyorsa, *parametre adından önce a ekleyin:

```
def my_function(*kids):
  print("The youngest child is " + kids[1])

my_function("Emel", "Tahsin", "Latif")
```

Anahtar Kelime Argümanları

Anahtar = *değer* söz dizimiyle de bağımsız değişkenler gönderebilirsiniz .
Bu şekilde argümanların sırası önemli değildir.

```
def my_function(per3, per2, per1):
  print("Seçilen " + per3)

my_function(per1 = "Emel", per2 = "Tobias", per3 = "Leman")
```

Rastgele Anahtar Sözcük Argümanları, **kwargs

İşlevinize kaç tane anahtar sözcük bağımsız değişkeninin aktarılacağını bilmiyorsanız **, işlev tanımında parametre adından önce iki yıldız işareti: ekleyin.

Bu şekilde, işlev bir bağımsız değişkenler *sözlüğü* alır ve buna göre öğelere erişebilir:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
def my_function(country = "Norway"):
  print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Argüman Olarak Bir Liste Geçirmek

Bir işleve herhangi bir veri türü argümanı gönderebilirsiniz (dize, sayı, liste, sözlük vb.) ve işlev içinde aynı veri türü olarak ele alınacaktır.

Örneğin, argüman olarak bir Liste gönderirseniz, işleve ulaştığında yine bir Liste olacaktır:

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Dönüş Değerleri

Bir işlevin bir değer döndürmesine izin vermek için şu `return` ifadeyi kullanın:

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Geçiş Bildirimi

`function`tanımlar boş olamaz, ancak herhangi bir nedenle `function`içeriği olmayan bir tanımınız varsa, `pass` hata almamak için ifadeyi girin.

```
def myfunction():
  pass
```

özyineleme

Python ayrıca işlev özyinelemesini de kabul eder; bu, tanımlı bir işlevin kendisini çağırabileceği anlamına gelir.
Özyineleme, ortak bir matematik ve programlama kavramıdır. Bu, bir fonksiyonun kendisini çağırdığı anlamına gelir. Bu, bir sonuca ulaşmak için veriler arasında geçiş yapabileceğiniz anlamına gelir.
Geliştirici, asla sonlanmayan veya aşırı miktarda bellek veya işlemci gücü kullanan bir işlevi yazmaya kaymak oldukça kolay olabileceğinden, özyineleme konusunda çok dikkatli olmalıdır. Bununla birlikte, doğru yazıldığında özyineleme, programlamaya çok verimli ve matematiksel olarak zarif bir yaklaşım olabilir.
Bu örnekte `tri_recursion()` , kendisini ("recurse") çağırmak için tanımladığımız bir işlevdir. Veri olarak, her yinelememizde azalan (`-1`) `k` değişkenini kullanıyoruz
. Özyineleme, koşul 0'dan büyük olmadığında (yani 0 olduğunda) sona erer.
Yeni bir geliştirici için bunun tam olarak nasıl çalıştığını anlamak biraz zaman alabilir, öğrenmenin en iyi yolu onu test etmek ve değiştirmektir.

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```