



**BTK**  
**AKADEMİ**

# Programlama

Doç. Dr. Zafer CÖMERT



# Bölüm 13

## Fonksiyonlar

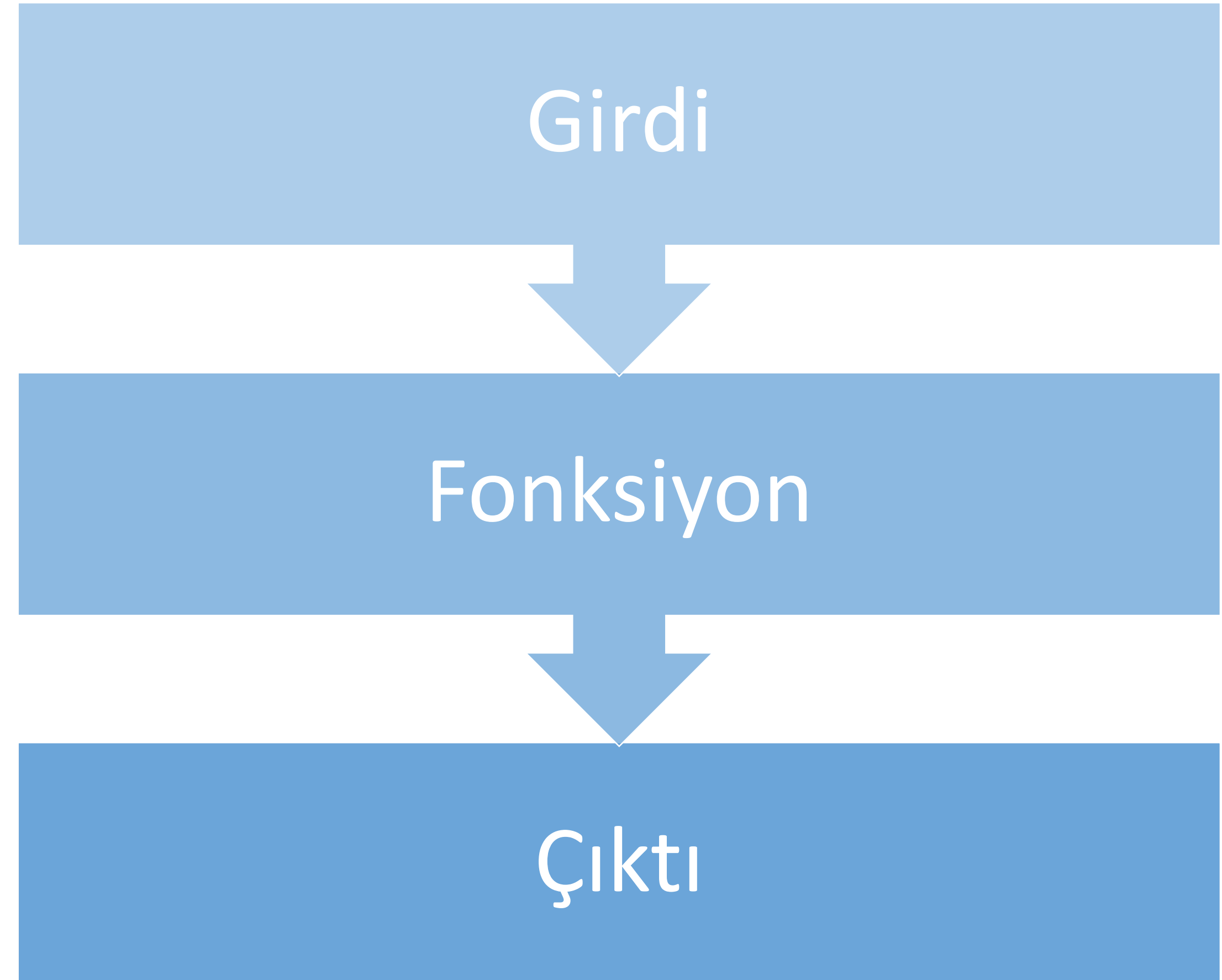
# Giriş

## İçerik

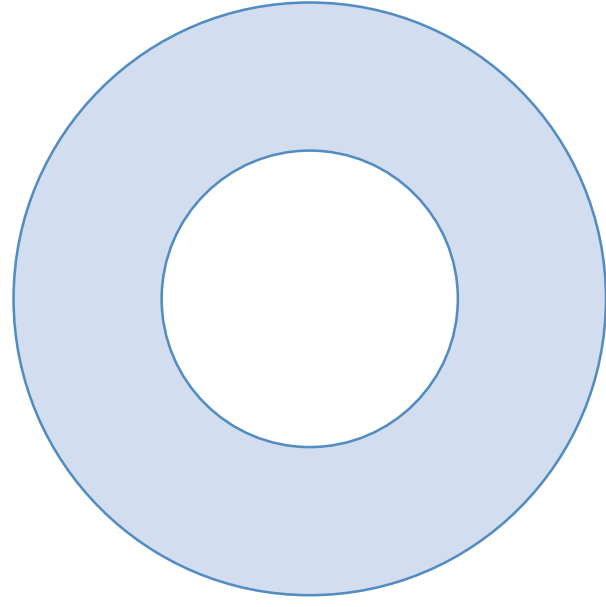
- Fonksiyonlara Giriş
- Fonksiyon Tanımama
- Fonksiyon Çağırımı
- Parametre ve Argüman Türleri
- Fonksiyonların Kapsamı ve Değişkenler
- Fonksiyon için Dokümantasyon
- Fonksiyonlar ve Dönüş Değeri
- Fonksiyonlarda Hata Yönetimi
- Fonksiyonlar ile Problem Çözümü
- Fonksiyonların Diğer Fonksiyonlar ile Kullanımı
- Alt Programlar

# Fonksiyon

- Programlama dillerinde fonksiyonlar, bir programın mantıksal olarak bölümlere ayrılmasını ve kod tekrarının azaltılmasını sağlayan temel yapı taşlarıdır.
- Fonksiyonlar sayesinde karmaşık problemler daha küçük, yönetilebilir parçalara bölünerek çözülür.



# Fonksiyonların Temel Avantajları



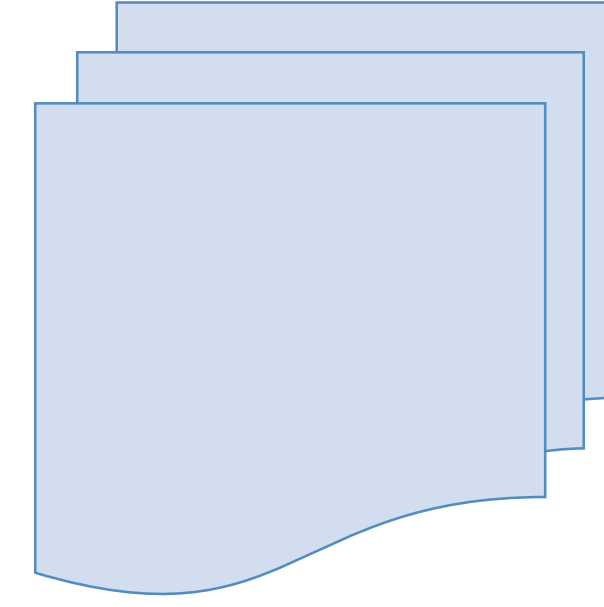
## Tekrar kullanılabilirlik

- Bir kez yazılan fonksiyon farklı yerlerde tekrar tekrar kullanılabilir.



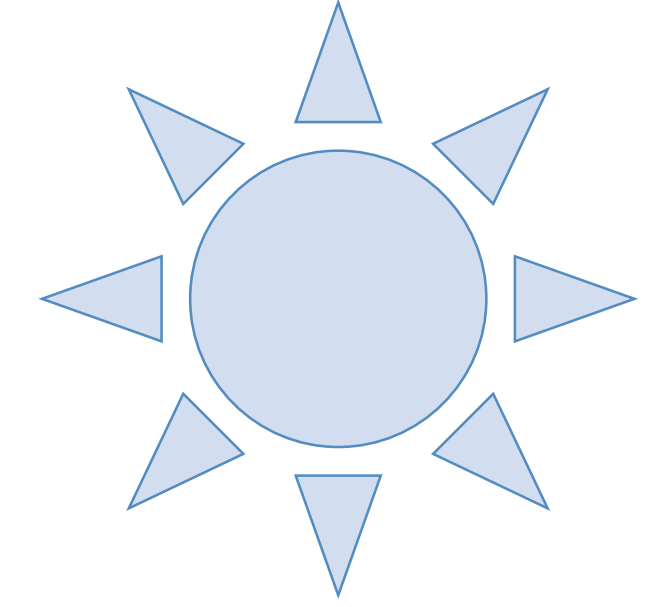
## Kodun okunabilirliğini artırır

- Uzun programlar daha kısa ve anlaşılır parçalara ayrılmış olur.



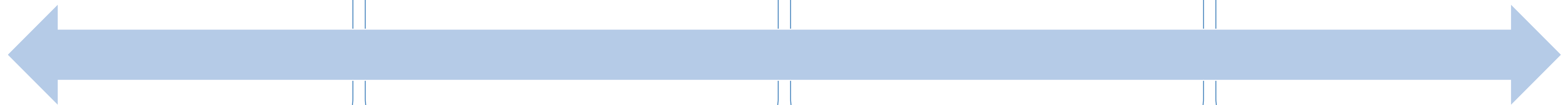
## Hata ayıklamayı kolaylaştırır

- Hatalar daha küçük birimlerde daha kolay bulunur ve düzeltilir.

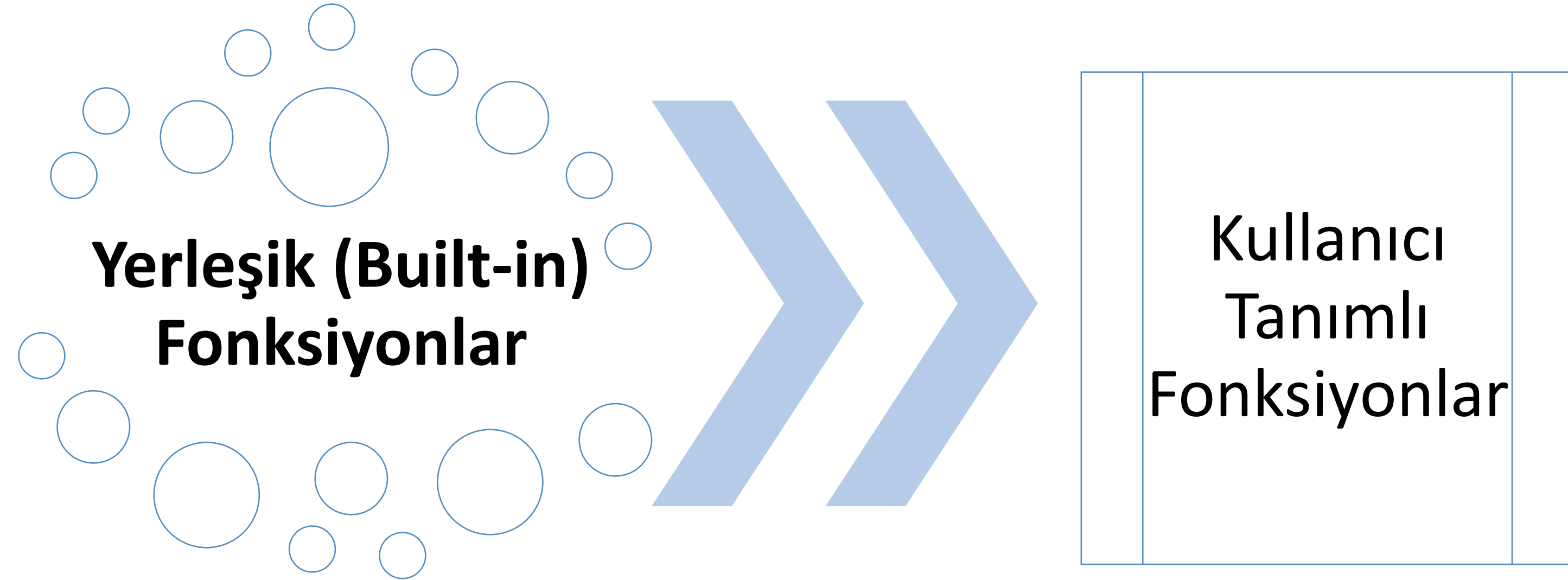


## Bakımı kolaydır

- Fonksiyonlardaki bir değişiklik tüm kullanımları etkiler, kodu merkezi bir yerden kontrol edebilirsiniz.



# Fonksiyon Türleri



Programlama dili tarafından önceden tanımlanmış ve doğrudan kullanılabilen fonksiyonlardır.

Programcının kendisinin tanımladığı ve özel görevler için yazdığı fonksiyonlardır.

# Fonksiyon

- Bir fonksiyonu, kahve yapan bir makineye benzetebiliriz.
- Her seferinde kahve yapmak için aynı işlemleri elle yapmak yerine, sadece makinenin düğmesine basarız.
- Fonksiyonlar da bu şekilde çalışır: Görev tanımlanır, ihtiyaç duyulduğunda çağrılır.



# Fonksiyon Tanımlama



```
1 def function_name():  
2     # Fonksiyonun yapacağı işlemler  
3     print("Bu bir fonksiyondur.")
```



```
1 def greet(name):  
2     print(f"Merhaba {name}, hoş geldin!")
```



# Fonksiyon Tanımlama

## Parametresiz Fonksiyon Çağrısı



```
1 def say_hello():  
2     print("Merhaba!")  
3  
4 # Fonksiyon çağrılır  
5 say_hello()
```

## Parametrelili Fonksiyon Çağrısı



```
1 def square(number):  
2     return number * number  
3  
4 result = square(4)  
5 print("Sonuç:", result) # Çıktı: Sonuç: 16
```

# Parametre ve Argüman Türleri

## Pozisyonel (Sıralı) Argümanlar



```
1 def print_info(name, age):  
2     print(f"{name} {age} yaşındadır.")  
3  
4 print_info("Ayşe", 25)
```

## Anahtar Kelime (Keyword) Argümanlar



```
1 def print_info(name, age):  
2     print(f"{name} {age} yaşındadır.")  
3  
4 print_info(age=25, name="Ayşe")
```

# Parametre ve Argüman Türleri

- Fonksiyon tanımında bir parametreye varsayılan bir değer atanabilir.
- Bu durumda çağrılırken o parametre verilmezse, varsayılan değer kullanılır.



```
1 def greet(name="Misafir"):  
2     print(f"Merhaba {name}!")  
3  
4 greet()           # Merhaba Ziyaretçi!  
5 greet("Ali")      # Merhaba Ali!
```

# Parametre ve Argüman Türleri

## \*args



```
1 def add(*numbers):
2     total = 0
3     for n in numbers:
4         total += n
5     print("Toplam:", total)
6
7 add(3, 5, 7) # Toplam: 15
```

## \*\*kwargs



```
1 def print_details(**info):
2     for key, value in info.items():
3         print(f"{key}: {value}")
4
5 print_details(name="Zeynep", age=30, city="Ankara")
```

# Fonksiyonların Kapsamı ve Değişkenler

## Yerel Değişken



```
1 def message():
2     name = "Ayşe"
3     print("Merhaba", name)
4
5 message()
6 # print(name) # HATA: 'name' fonksiyon dışında tanımsızdır.
```

## Global Değişken



```
1 name = "Zeynep"
2
3 def display():
4     print("Merhaba", name) # Global değişken kullanılıyor
5
6 display()
```

# Fonksiyon İçi Dokümantasyon

- Kodun anlaşılabilirliğini artırır.
- Başka biri kodunuzu okurken ne yaptığını kolayca anlayabilir.
- `help()` fonksiyonu ile otomatik yardım alınmasını sağlar.
- IDE'ler (örneğin VS Code, PyCharm) docstring'leri otomatik olarak gösterir.

```
1 def add(a, b):  
2     """  
3     Bu fonksiyon iki sayıyı toplar ve sonucu döndürür.  
4  
5     Parametreler:  
6     a (int, float): Birinci sayı  
7     b (int, float): İkinci sayı  
8  
9     Dönüş:  
10    int veya float: Toplam değer  
11    """  
12    return a + b  
13  
14 result = add(4, 6)  
15 print("Toplam:", result) # Toplam: 10
```

# Fonksiyonlarda Hata Yönetimi

- Gerçek hayattaki programlarda her zaman beklenmeyen durumlar ortaya çıkabilir.
- Örneğin:
- Kullanıcı geçersiz bir değer girebilir,
- Dosya açılamayabilir,
- Sıfıra bölme gibi matematiksel hatalar olabilir.

try

except

raise


else

finally

# Özel Hata Yakalama

try

except



```
1 def divide(a, b):
2     try:
3         return a / b
4     except ZeroDivisionError:
5         return "Sıfıra bölme hatası!"
6
7 print(divide(10, 2)) # 5.0
8 print(divide(10, 0)) # Sıfıra bölme hatası!
```



# Genel Hata Yakalama

try

except



```
1 def write():
2     try:
3         name = input("Adınızı girin: ")
4         print("Hoş geldin", name.upper())
5     except:
6         print("Beklenmeyen bir hata oluştu.")
```

# Genel Hata Yakalama

try

except

else

finally

```
1 def check_value(value):
2     try:
3         number = int(value)
4     except ValueError:
5         print("Lütfen sayısal bir değer giriniz.")
6     else:
7         print("Girilen sayı:", number)
8     finally:
9         print("Kontrol tamamlandı.")
10
11 check_value("42")
12
13 # Girilen sayı: 42
14 # Kontrol tamamlandı.
15
16 check_value("merhaba")
17 # Lütfen sayısal bir değer giriniz.
18 # Kontrol tamamlandı.
```

# Genel Hata Yakalama


try

except

else

finally

raise



```
1 def factorial(n):
2     if n < 0:
3         raise ValueError("Negatif sayı için faktoriyel tanımsızdır.")
4     result = 1
5     for i in range(1, n + 1):
6         result *= i
7     return result
8
9 print(factorial(5))    # 120
10 print(factorial(-3))  # ValueError: Negatif sayı için faktoriyel tanımsızdır.
```

# Lambda Fonksiyonları

- Lambda fonksiyonları, tek satırlık, isimsiz fonksiyonlardır. Özellikle kısa işlemler için kullanışlıdır.



```
1 square = lambda x: x * x
2 print(square(4)) # 16
```

# map()

- Tüm liste elemanlarına bir fonksiyonu uygular.



```
1 numbers = [1, 2, 3, 4]
2 squares = list(map(lambda x: x**2, numbers))
3 print(squares)  # [1, 4, 9, 16]
```

# filter()

- Bir koşula uyan elemanları süzer.



```
1 numbers = [1, 2, 3, 4]
2 odds = list(filter(lambda x: x % 2 != 0, numbers))
3 print(odds) # [1, 3]
```

# reduce()

- Tüm elemanları bir işlemle birleştirir (kütüphane import gerekir).



```
1  from functools import reduce
2
3  numbers = [1, 2, 3, 4]
4  total = reduce(lambda x, y: x + y, numbers)
5  print(total)  # 10
```

# Alt Yordamlar ve Ana Yordam



```
1  def greet():  
2      print("Merhaba!")  
3  
4  if __name__ == "__main__":  
5      greet()
```



# Teşekkürler

ZAFER CÖMERT  
Öğretim Üyesi