

# İÇİNDEKİLER

## I. BÖLÜM ETİK, GÜVENLİK VE TOPLUM PROBLEM ÇÖZME VE ALGORİTMALAR

### 1. ETİK, GÜVENLİK VE TOPLUM

1.1. Etik Değerler .....	17
1.2. Bilişim Teknolojileri ve İnternet Kullanımında Dikkat Edilmesi Gereken Etik İlkeler .....	17
1.2.1. Fikrî Mülkiyet .....	17
1.2.2. Erişim .....	19
1.2.3. Gizlilik .....	19
1.2.4. Doğruluk .....	19
1.2.5. İnternet Etiği .....	22
1.3. Bilgi Güvenliği .....	23
1.3.1. Bilgi Güvenliğine Yönelik Tehditler .....	24
1.3.2. Sayısal Dünyada Kimlik ve Parola Yönetimi .....	25
1.3.3. Kişisel Bilgisayarlarda ve Ağ Ortamında Bilgi Güvenliği .....	26

### 2. PROBLEM ÇÖZME VE ALGORİTMALAR

2.1. Problem Çözme Kavramları ve Yaklaşımlar .....	29
2.1.1. Programlama Nedir? .....	29
2.1.2. Program Nedir? .....	30
2.1.3. Hata Ayıklama Nedir? .....	31
2.1.4. Günlük Hayatta Problem Çözme .....	31
2.1.5. Problem Çözme Süreci .....	32
2.1.5.1. Tıltı, Kaz ve Mısır Çuvalı .....	32
2.1.5.2. Sudoku .....	37
2.1.5.3. Dikdörtgeni Parçalara Ayırma .....	39
2.1.5.4. Engelli Yollar .....	39
2.1.5.5. Hanoi Kulesi .....	40

### 3. PROBLEM ÇÖZME SÜRECİ

3.1. Problem Çözme Teknikleri .....	44
3.1.1. Her Zaman Bir Planınız Olsun .....	44
3.1.3. Problemi Küçük Parçalara Ayırın .....	44
3.1.4. Önce Bildiklerinizden Yola Çıkın .....	45
3.1.5. Problemi Basitleştirin .....	45
3.1.6. Benzerlikleri Arayın .....	46
3.1.7. Deneme Yapın .....	46
3.1.8. Asla Vazgeçmeyin .....	46
3.2. Problem Çözme Adımları .....	46
3.3. Problem Türleri .....	48
3.4. Bilgisayarlar ile Problem Çözme .....	48
3.5. Problem Çözme Kavramları .....	49
3.6. Veri Türleri .....	49
3.6.1. Sayısal Veri .....	49

3.6.2. Alfanümerik/Karakter Veri	50
3.6.3. Mantıksal Veri	50
3.7. Bilgisayar Veriyi Nasıl Saklar?	52
3.8. Sabit ve Değişkenler	52
3.9. Fonksiyonlar	53
3.10. Operatörler	55
3.11. İşlem Önceliği	55
3.12. İfade ve Eşitlikler	56

#### 4. PROBLEM ÇÖZME YAKLAŞIMLARI

4.1. Bilgisayar ile Nasıl İletişim Kurulur?	59
4.2. Çözümün Düzenlenmesi	59
4.2.1. Problemin Analiz Çizelgesi	60
4.2.2. Etkileşim Çizelgesi Geliştirme	60
4.2.3. GSÇ Çizelgesi	61
4.2.4. Algoritmalar	61
4.2.5. Akış Şemaları	61
4.3. Algoritma Yönergeleri ve Akış Şeması Sembollerİ	62
4.4. Haricî ve Dâhilî Dokümantasyon .....	63
4.5. Çözümün Programlanması/Kodlanması .....	63

#### 5. PROGRAMLAMA YAPISI

5.1. Programlama Yapısına Giriş	65
5.1.1. Göstergeler	65
5.1.2. Modüller ve İşlevleri	72
5.1.3. Bağlılık ve Birleşim	72
5.1.4. Yerel ve Global Değişkenler	73
5.1.5. Parametreler	74
5.1.6. Dönen Değerler	75

#### 6. DOĞRUSAL MANTIK YAPISI İLE PROBLEM ÇÖZME

6.1. Doğrusal Mantık Yapısı	78
6.2. Çözüm Üretilmesi	78
6.2.1. Problem Analiz Çizelgesi	79
6.2.2. Etkileşim Çizelgesi	79
6.2.3. GSÇ Çizelgesi	80
6.2.4. Birleşim Çizelgesi ve Veri Sözlüğü	80
6.2.5. Algoritma ve Akış Şemaları	81
6.2.6. Çözümün Test Edilmesi:	82
6.3 Özet	82

#### 7. KARAR MANTIK YAPISI İLE PROBLEM ÇÖZME

7.1. Karar Mantık Yapısı	84
7.1.1. Tek Koşullu Yapılar	85
7.1.2. Çok Koşullu Karar Yapıları	86
7.1.3. İç İçe Karar Yapıları	86
7.2. Düz Mantık Kullanımı	87
7.3. Pozitif Mantık Kullanımı	89

7.4. Negatif Mantık Kullanımı	90
7.5. Mantık Dönüşümleri	90
7.6. Hangi Mantık Yapısı?	91
7.7. Karar Tabloları	93

## 8. DÖNGÜ YAPISI İLE PROBLEM ÇÖZME

8.1. Döngü Mantık Yapısı	96
8.2. Arttırma	96
8.3. Biriktirme	97
8.4. While/While End Döngüsü	97
8.5. Repeat/Until Döngüsü	99
8.6. Otomatik Sayaç Döngüsü	101
8.7. İç İçe Döngüler	103
8.8. Göstergeler	105
8.9. Öz Yineleme	105

## II. BÖLÜM PYTHON İLE PROGRAMLAMANIN TEMELLERİ

### 1. PYTHON İLE PROGRAMLAMANIN TEMELLERİ

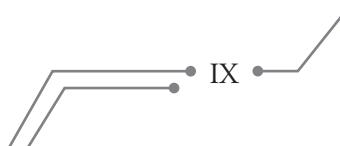
1.1. Yazılım Geliştirme Süreci	109
1.1.1 Yazılım	109
1.1.2. Yazılım Geliştirme Ortamları	110
1.1.3. Editörler	110
1.1.4. Derleyiciler	111
1.1.5. Yorumlayıcılar	111
1.1.6. Hata Ayıklayıcılar	112
1.1.7. Yanaylaçlar	112
1.1.8. Bütünleştirilmiş Geliştirme Ortamları	112
1.2. Neden Python?	114
1.3. Python Sürümleri	114

### 2. DEĞERLER VE DEĞİŞKENLER

2.1. Tam Sayı ve Diziler	116
2.2. Değişkenler ve Atama	119
2.3. Reel Sayılar	122
2.4. Belirteçler	124

### 3. İFADELER VE ARİTMETİK İŞLEMLER

3.1. Sabit Değerler	127
3.2. Python'da Sık Kullanılan Aritmetik İkili Operatörler	127
3.3. Karışık Türülü İfadeler	128
3.4. Operatör Önceliği ve Birleşim	128



3.5. İfadeleri Biçimlendirme	128
3.6. Yorumlar	129
3.7. Hatalar	129
3.7.1. Çalışma Zamanı Hataları	130
3.7.2. Mantık Hataları	130
3.8. Aritmetik Örnekler	130
3.9. Aritmetik İfadeler	131

#### 4. KOŞULLU DURUMLAR

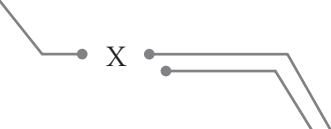
4.1. Boolean İfadesi	133
4.2. Python'da İlişkisel Operatörler	133
4.3. "if" İfadesi	133
4.4. "if/else" İfadesi	134
4.5. Birleşik Boolean İfadesi	135
4.6. Pass İfadesi	136
4.7. Kayan Noktalı Eşitlik	136
4.8. İç İçe Koşul İfadeleri	136
4.9. Çok Yönlü Karar İfadeleri	137
4.10. Çok Yönlü ve Zincirleme Durum İfadeleri	138
4.11. Koşullu İfadeler	139
4.11.1. Koşullu İfadelerde Hatalar	139
4.12. Mantık Karmaşası	140

#### 5. DÖNGÜLER

5.1. Döngü Yapıları	143
5.2. For Döngüsü	143
5.2.1. For Döngüsü İçin Söz Dizimi	144
5.2.2. For Döngüsü İçin Farklı Örnekler	144
5.3. İç İçe Döngüler	147
5.4. İç İçe 3'lü Döngü	150
5.5. While Döngüsü	150
5.5.1. While Döngüsü İçin Söz Dizimi	150
5.5.2. While Döngüsü İçin Akış Şeması	151
5.6. Belirli ve Belirsiz Döngüler	154
5.7. Döngü'den Çıkma Komutları	155
5.7.1. Break İfadesi	155
5.7.2. Continue İfadesi	156
5.7.3. While/else ve for/else	157
5.8. Döngü Örnekleri	158

#### 6. FONKSİYONLAR 1

6.1. Neden Fonksiyonlar?	162
6.2. Fonksiyon Nedir?	162
6.3. Fonksiyonlara Giriş	162
6.3.1. sqrt() Fonksiyonu	163
6.3.1.1. sqrt() Fonksiyonunun Farklı Kullanımları	163
6.3.2. Fonksiyonların Bölümleri	164
6.3.3. Parametresiz Fonksiyonlar	165



6.3.4. Değer Döndürmeyen Fonksiyonlar	165
6.4. Fonksiyon ve Modüller	166
6.5. Yerleşik İşlevler	167
6.6. Standart Matematik Fonksiyonları	168
6.7. time Fonksiyonları	170
6.8. Rastgele Sayılar	172

## 7. FONKSİYON YAZMA

7.1. Fonksiyon Kavramı	175
7.1.1. Fonksiyon Tanımlama	175
7.1.2. Fonksiyon Yazma	175
7.1.3. Fonksiyon Çağırma	175
7.2. Fonksiyon Kullanımı Örnekleri	176
7.3. Değer Gönderme ile İlgili Olası Sorunlar	178
7.4. Çoklu Değer Gönderme Örnekleri	178
7.4.1. Grafik Ortamda Beşgen Çizimi	178
7.4.2. En Büyük Ortak Çarpan Fonksiyonu	179
7.5. Yerel Değişken	180
7.6. Fonksiyon Yazarken Fonksiyon Sıralamasını Belirleme	181
7.6.1. Girilen İki Değerin En Büyük Ortak Bölünü	181
7.7. Parametre Gönderme	182
7.8. Fonksiyon Yazarken Tanımlayıcı Bilgileri Ekleme	183
7.9. Fonksiyon Örnekleri	184
7.9.1 Asal Sayıların Bulunması	184
7.9.3 Kısıtlı Veri Girişisi	186
7.9.4 Ağaç Çizimi	187

## 8. FONKSİYONLAR 2

8.1. Global Değişkenler	189
8.2. Örnekler	189
8.2.1. Hesap Makinesi Örneği	189
8.2.2. Varsayılan (Default) Parametreler	191
8.2.3. Varsayılan ve Diğer Parametreler	191
8.2.4. Gelişmiş Poligon Çizimi	192
8.2.5. Öz Yineleme	193
8.2.6. Öz Yineleme Olmadan Faktöriyel Hesaplama	194
8.2.7. Fibonacci Sayıları	195
8.3. Fonksiyonları Tekrar Kullanılabilir Yapma	195

## 9. NESNELER

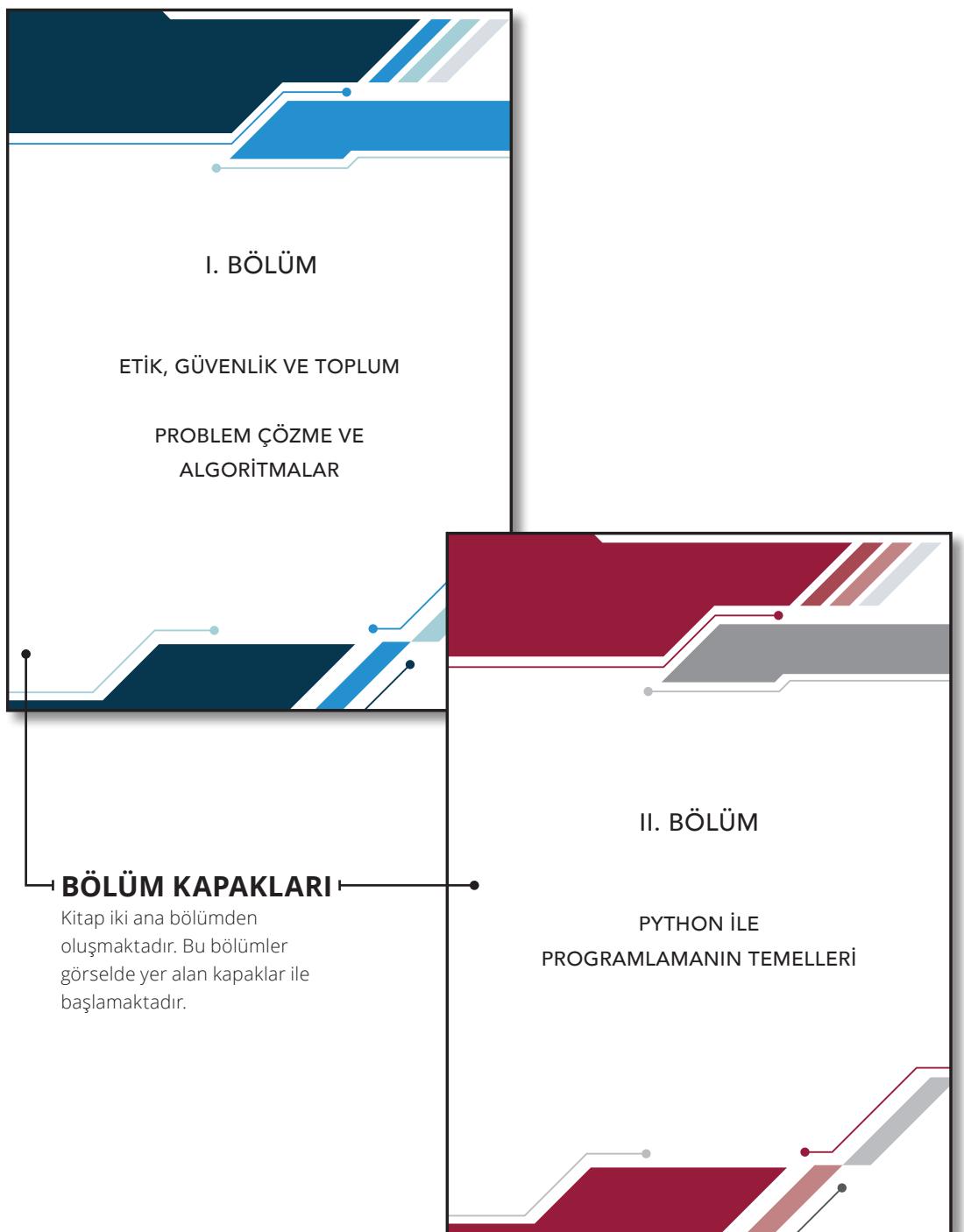
9.1. Nesne Kavramı	198
9.2. Nesneleri Kullanmak	198
9.3. Dizi Nesneleri	198
9.3.1 str Nesnesi İçin Yöntemler	200
9.3.2 getitem Kullanımı	201
9.4. Dosya Nesneleri	202
9.4.1. Dosya Okuma ve Yazma İşlemleri	204
9.4.2. Dosya Okuma ve Yazma İşlemlerinde with/as kullanımı	204
9.4.3. TextIOWrapper Yöntemleri	206

9.5. Turtle Grafik Nesneleri	207
9.5.1. Turtle Grafik & tkinter Nesneleri	208
9.5.2. Buton Test Etme	208
9.6. Nesne Değişkenliği ve Örtüşme	211

## 10. LİSTEler

10.1. Liste Kavramı	215
10.1.1. Listeleri Kullanmak	216
10.1.2. Listelerin Farklı Kullanımları	218
10.1.3. Değerler Arası Gezinme	218
10.2. Liste Oluşturmak	219
10.3. List Fonksiyonu ile Tam Sayı Liste Yapmak	221
10.4. * Operatörü ile Liste Oluşturma	222
10.5. Liste Üyeliği	224
10.6. Listeye Değer Atama ve Eşitleme	225
10.7. Listenin Sınırları	229
10.8. Dilimleme	229
10.8.1. Dilimleme İçin Farklı Kullanımlar	230
10.8.2. Dilimleme Örnekleri	230
10.9. Listeden Eleman Çıkarma	232
10.10. Listeler ve Fonksiyonlar	233
10.11. Listede Kullanılan Yöntemler	234
10.12. Çok Boyutlu Listeler	234
10.13. Çok Boyutlu Diziler	235
10.14. _ Sembolü	237
10.15. Liste Oluşturma Tekniklerinin Özeti	238
SÖZLÜK	239
KAYNAKÇA	240

# ORGANİZASYON ŞEMASI





## ÜNİTE ADI ve NUMARASI

Ünitein adı ve numarasını gösterir.

## ÜNİTE KAPAĞI

Ünitein içeriğini ifade eden bir görsel ile her üniteye özgü renk şablonu yer almaktadır.

## EDİNİMLER

Ünite sonunda elde edilecek bilgi ve becerilerin listesi yer almaktadır.

Bu bölümde:

- ✓ Bilişim teknolojilerini kullanırken dikkat edilmesi gereken etik ilkeleri kavrayacak,
- ✓ Internet ortamını etik ilkelerde uygun nasıl kullanabileceğinizi anlayacak,
- ✓ Etik ilkelerin ilahi sonucunda oluşabilecek durumlara örnek verebilecek,
- ✓ Bilişim teknolojileri ve Internet'in kullanıldığında etik ilkelerin gerekliliğini sorgulayabileceksiniz.

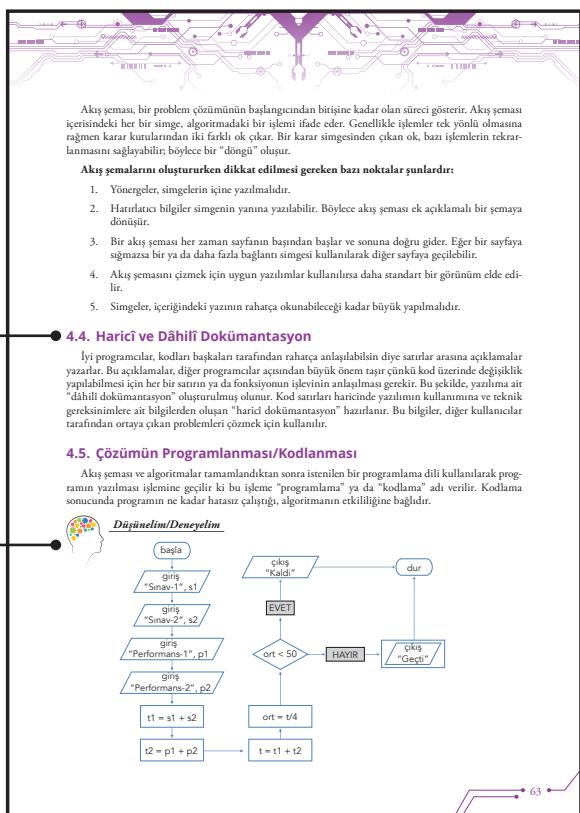
16

## KONU BAŞLIKLARI

İçindekiler bölümündeki sırayla listelenen konuların başlıklarını yer almaktadır.

## DÜŞÜNELİM/DENEYELİM

Konuya ilişkin örnek sorular yer almaktadır.



## I. BÖLÜM

ETİK, GÜVENLİK VE TOPLUM

PROBLEM ÇÖZME VE  
ALGORİTMALAR

# 1. ETİK, GÜVENLİK VE TOPLUM



Bu bölümde;

- ✓ Bilişim teknolojilerini kullanırken dikkat edilmesi gereken etik ilkeleri kavrayacak,
- ✓ İnternet ortamını etik ilkelere uygun nasıl kullanabileceğinizi anlayacak,
- ✓ Etik ilkelerin ihlali sonucunda oluşabilecek durumlara örnek verebilecek,
- ✓ Bilişim teknolojileri ve İnternet'i kullanırken etik ilkelerin gerekliliğini sorgulayabileceksiniz.



## 1.1. Etik Değerler

**Etik**; bireylerin ahlaklı ve erdemli bir hayat yaşayabilmesi için hangi davranışlarının doğru, hangilerinin yanlış olduğunu araştıran bir felsefe dalıdır. Temelinde barındırdığı güzel ahlaklı, adaletli ve iyi insan olma özellikleri değişimse de zamana, bilimsel gelişmelere ve toplumun gereklerine göre etik kavramına yüklenen anlam değişebilmektedir. Bir konuya ya da belirli bir meslek dalına özgü etik davranışların tamamı **etik değerler** olarak tanımlanabilir. Bir alanın evrensel etik değerlerinin belirlenmesi için o alanın toplum tarafından kabul gören ve görmeyen davranışlarının tanımlanması ve bireylerin bunlara uygun davranışlarının sağlanması gerekmektedir. Etik dışı eylemlere ilişkin yaptırımlar, çoğu zaman toplum tarafından belirlenmekte ve bu yaptırımlar gerekirse yasal düzenlemeler için belirleyici olmaktadır.



Günlük hayatımızın vazgeçilmez parçası hâline gelen bilişim teknolojileri; eğitim, sağlık, medya, iletişim, ticaret ve bankacılık gibi sektörler başta olmak üzere pek çok alanda yaygın bir şekilde kullanılmaktadır. Bilişim teknolojilerinde yaşanan bu hızlı değişim ve yaygınlık, istenen bilgiye her zaman ve her yerde erişebilme imkânı gibi faydalar sunmasının yanı sıra bu teknolojilerin tam olarak anlaşılmadan kullanımına yol açmakta ve bu durum da beraberinde pek çok sorun ortaya çıkarmaktadır. Bu anlamda yaşanan sorunlardan birisi de zaman ve mekân sınırı olmaksızın erişilen bilginin doğruluğunu ve kaynağının tespitidir. Gelişmekte olan ülkelerde toplumsal ve bireysel düzeyde artan rekabet ortamı, maddi kazanç sağlama ve bilginin kaynağından çok sonuca odaklanan yaklaşımlar, etiğin geri plana itilmesine yol açmaktadır. Gelişmiş toplumun önemli göstergelerinden birisi de gerek üretilen bilginin gerekse bu bilgiyi kullanan bireylerin etik kurallara uyup uymadıklarıdır. Geleceğin bireylerinin şekillenmesinde bilginin üretilmesi kadar bu bilgilerin etik kurallara göre üretilip paylaşılmasını da sağlamak önem kazanmaktadır. Bu da etiğin ne kadar önemli olduğunu göstermektedir.

## 1.2. Bilişim Teknolojileri ve İnternet Kullanımında Dikkat Edilmesi Gereken Etik İlkeler

Bilişim teknolojilerinin ve İnternet'in kullanımı sırasında uyulması gereken kuralları tanımlayan ilkelere **bilişim etiği** denir. Bu ilkelerin temel amacı, bilişim teknolojileri ve İnternet'i kullanan bireylerin yanlış bir davranış sergilemesine engel olarak onları güvence altına almaktır. Buna göre bilişim etiği, bilişim teknolojilerinin kullanımı esnasında toplum tarafından kabul gören uyulması gereken kurallar bütünüdür. Bilişim teknolojilerinin kullanımında yaşanan etik sorunların dört temel başlıkta (fikri mülkiyet, erişim, gizlilik ve doğruluk) ele alındığı görülmektedir. Aşağıda bu başlıklara kısaca değinilmiştir.

### 1.2.1. Fikrî Mülkiyet

Bilişim teknolojileri alanında geliştirilen ürünler özellikle yazılım alanında ise arsa, ev, bilgisayar kasası gibi maddi bir varlığın dışında somut olmayan bir kavramın sahipliği söz konusu olmaktadır. Bu durumda bu sahiplığın ispatı çeşitli sıkıntılar doğurmaktadır. Aslında günümüzde bu sorunlar müzik, edebiyat alanları için de söz konusudur. Hatta sanat alanındaki bu etik sorun, doğrudan bilişim teknolojilerinin gelişmesi ile çığ gibi büyümüştür. Bu tür eserlerin günümüz teknolojisi ile kopyalanıp dağıtımasının oldukça kolay olması, asıl sahibine dair bilginin korunmasında önemli bir engel olarak

karşımıza çıkmaktadır. "Eserlerin (bilişim alanı için geliştirilen yazılımlar) sahibi kimdir ve kimlerin kullanımına izin verilmiştir?" sorularının cevabı fikrî mülkiyet başlığının altında irdelenmektedir.

Fikrî mülkiyet; kişinin kendi zihni tarafından ürettiği her türlü ürün olarak tanımlanmaktadır. Türk Dil Kurumu ise Bilim ve Sanat Terimleri Sözlüğü'nde fikrî mülkiyet kavramını "düşünü çalışması sonunda ortaya konulan yazın ve bilim ürünlerleri üzerindeki iyelik" olarak tanımlamıştır.

Fikrî mülkiyet denince karşımıza hukuki ve etik boyutlar çıkmaktadır. Kimi sorunlar yasal olup etik olmazken kimi de etik olup yasal olmayabilmekte ya da iki boyut birden temelsiz kalabilmektedir. Bu nedenle fikrî mülkiyete ilişkin yasalar, günümüz koşullarına uygun olarak güncellenmeye muhtaç olmaktadır. Telif hakkı, patent, şifreleme gibi kavamlar da bu gereksinim sonucunda ortaya çıkmıştır.

"Fikrî ve kültürel eserlerden bazıları Creative Commons (CC) organizasyonuna dâhildir. Creative Commons, telif hakları konusunda esneklik sağlamayı amaçlayan, eser sahibinin haklarını koruyarak, eserlerin paylaşımını kolaylaştırıcı modeller sunan, kâr amacı gütmeyen bir organizasyondur. Bu organizasyona dâhil olan eserler, kaynağı belirtmek ön şartıyla belirli kısıtlamalar göz önünde bulundurularak kullanılabilir."

## Koşullar



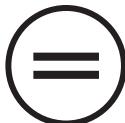
**Attribution - Atıf:** Eserin ilk sahibinin belirtilmesi koşulu. Bu koşulu barındıran lisansa sahip eserlerde, eseri yaratan ilk kişinin mutlaka belirtilmesi gerekiyor.



**Share Alike - Aynı Lisansla Paylaş:** Lisans modelinin korunması koşulu. Bu koşula sahip eserlerin türetilmesi veya yeniden yayınlanması ancak onu barındıran yeni eserin de aynı lisansa sahip olması şartıyla gerçekleşebilir.



**Non-Commercial - Ticari Olmayan:** Eserin ticari amaçlı kullanılmasının engellanması koşulu. Bu koşulu şart koşan eserlerin türevlerinin veya orijinalerinin sadece ticari olmayan ürünlerde kullanılması mümkün (Ticari amaçlı kullanmak için eser sahibine başvurmak mümkün.).



**No Derivative Works - Türetilemez:** Eserin türevinin yaratılmaması koşulu. Bu koşulu içeren lisanslı eserlerin türevlerinin yapılmasına izin verilmemektedir eğer isteniyorsa sadece olduğu gibi kullanılması gereklidir.

CC lisanslı eserler bu kısıtlamaların yalnızca birine sahip olabileceği gibi birden fazlasına aynı anda sahip olabilir. Bu eserlerin kısıtlamaları, eserin bulunduğu sayfanın alt kısmında görülebilir.

Bilişim dünyasında yazılımları lisanslarına göre, özgür yazılımlar ve ticari yazılımlar olmak üzere ikiye ayırilabiliriz. Özgür yazılım dünyasına ait GPL'ye (General Public Licence - Genel Kamu Lisansı) sahip yazılımlar ücretsiz olarak (ya da özelleştirilmiş versyonları düşük ücretlerle) kullanılabilirken, ticari faaliyet gösteren firmaların ürettiği yazılımların lisanslarıysa çoğulukla yüksek bedeller karşılığında alınabilmektedir. Kişi ya da kuruluşlar yazılım seçimi yaparken ihtiyaçlarını doğru şekilde belirledikten sonra tercih yapmalıdır. Böylece yüksek mal yet ödemekten ve bekłentileri karşılamayan program edinmiş olmaktan kaçınmış olurlar.



Lisanssız yazılım kullanmanın etik uygunsuzluk yanında teknik sakıncaları da vardır. Firmalarca sunulan yazılımlar, zaman zaman güvenlik açılarını kapatmak ya da ek özelliklerle donanmak amacıyla güncelleme alır. Lisanssız kullanılan yazılımlar, bu güncellemeleri alamaz ve bilgi güvenliği açısından bilgisayarları savunmasız kılar.

### 1.2.2. Erişim

Bu başlık bilgiye erişimi anlatmaktadır. Sıradan bir vatandaş için herhangi bir bilişim teknolojisi ürününden bilgiye erişim olarak düşünülebilir. Örneğin herhangi bir arama sitesini kullanarak, istedigimiz bilgiye hızla erişebiliriz. Ancak bilgi daha özel bir formatta sunulmuş olabilir. Örneğin bir veri tabanında saklanıyor olabilir. Bu durumda karşımıza üç sorun çıkmaktadır:

1. Bilgiye erişebilecek düzeyde bilişim bilgisi,
2. Bilginin yararlılığını test edecek düzeyde bilgi okuryazarlığı,
3. Bilgiye erişmenin varsa maddi karşılığı olan ekonomik güç.

Günümüz insanı birinci sorunu aşmakta oldukça başarılı gibi görünürken, ikinci sorunun aşılmasında hâlâ güçlükler söz konusudur. Çünkü bilgi yığınları artmakta ve bu bilginin doğruluğunu test etmek güçleşmekte ayrıca son kullanıcı dedigimiz vatandaşın bunu test etme bilincinin eğitilmesi gerekmektedir. Üçüncü sorun olan ekonomik boyut için kütüphane veri tabanları bir çözüm olarak karşımıza çıkmaktadır. Bu durumda bilginin ücretsiz olması “Herkesin eşit derecede bilgiden yararlanması sağlanır.” çözüm önerisi, fikri mülkiyet ile çelişecektir.

### 1.2.3. Gizlilik

Bir önceki başlıkta bahsettiğimiz gibi, bir arama sitesi kullanarak bilgiye hızla erişim, herhangi bir kişi için sıradan bir davranış hâline gelmiştir. Bugün herkes aklına gelen her şeyi özgürce Google ya da Yandex gibi arama motorlarında aramaktadır. Oysa ki her arattığımız şey ile birlikte hatta bilişim ortamında yaptığımız her eylem ile aramızda “ekmek kırtıtı” olarak tabir edilen izler bırakıyoruz. Eğer birilerinin bizim bu bıraktığımız ekmek kırtıtlarını takip ettiği hissine kapılırsak ne kadar rahatsız olacağımızı bir düşünün. Örneğin Google’da arama yaparken karşınıza çıkan reklamların, sizin daha önce ziyaret ettiğiniz siteler ve bunların içeriklerinden elde edilen verilerle tespit edilen ilgi alanlarınıza yönelik olduğunu görmüşsunuzdur. Sadece tarama yaparken değil, birçok kurum ve kuruluşla üye olurken dijital teknolojilerden yararlanıyoruz. Örneğin hastane kayıtları. Çoğu hasta hastane kayıtlarının başka kişilerle paylaşılmasını istemez. İşte gizlilik dedigimiz kavram kişiye ait her türlü bilgiyi (ki bu bilgi sadece ad ve soyadı değil, kişinin duygusu, düşünce, siyasi eğilim, dini inancı, planı, fantezi dünyası ve korku gibi bilgilerini de içerir) saklama becerisidir. Ancak bilginin saklanması dışında bu bilginin doğru kişilerle doğru zaman diliminde de paylaşılması gizlilik başlığını ilgilendirir. Örneğin hasta, bilgilerini doktoru ile paylaşmak zorundadır.

İzlenmekten kaçınmak için açık kaynak dünyasından alternatifler kullanılabilir.

Örnek olarak <https://duckduckgo.com> sitesini inceleyiniz.

### 1.2.4. Doğruluk

Tahmin edilebileceği gibi bilişim alanında şahsimiz ait bilgiler bizim dışımızdaki kişiler tarafından da kayıt altına alınabilmektedir. Ancak bu bilgilerin doğruluğu kimin sorumluluğundadır? Biz kendimize ait bilgileri kontrol etme hakkına sahip olmalıyız ve kendimize ait bilgileri kendimiz kodlayacağımızda bunun sorumluluğunu da üstlenmek zorundayız. Ayrıca anonim bilgilerin doğruluğunun sorumluluğu kimde olmalıdır? Örneğin, içeriğini kullananların oluşturduğu bilgi paylaşım siteleri (wiki

ortamları) açık sistemlerdir. Bu sistemlerdeki verilerin doğruluğunun garantisini kimdedir gibi sorular bu başlık altında ele alınmaktadır.

Uluslararası Bilgisayar Etik Enstitüsüne göre bilişim teknolojilerinin doğru bir şekilde kullanılabilmesi için aşağıda belirtilen 10 kurala uyulması gerekmektedir.

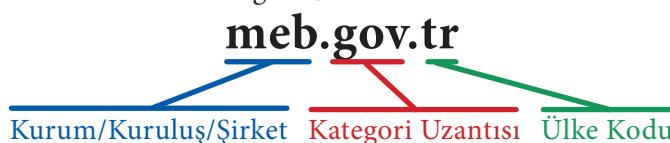
1. Bilişim teknolojilerini başkalarına zarar vermek için kullanmamalısınız.
2. Başkalarının bilişim teknolojisi aracılığı ile oluşturduğu çalışmaları karıştırmamalısınız.
3. Başkasına ait olan verileri incelememelisiniz.
4. Bilişim teknolojilerini hırsızlık yapmak için kullanmamalısınız.
5. Bilişim teknolojilerini yalancı şahitlik yapmak için kullanmamalısınız.
6. Lisanssız ya da kırılmış/kopyalanmış yazılımları kullanmamalısınız.
7. Başkalarının bilişim teknolojilerini izinsiz kullanmamalısınız.
8. Başkalarının bilişim teknolojileri aracılığı ile elde ettiği çalışmalarını kendinize mal etmemelisiniz.
9. Yazdığınız programların ya da tasarladığınız sistemlerin sonuçlarını göz önünde bulundurmamalısınız.
10. Bilişim teknolojilerini her zaman saygı kuralları çerçevesinde kullanmalı ve diğer insanlara saygı duymalısınız.

Günümüzde İnternet kullanıcıları, bilgiye kolay ulaşabilirken amaçları bu olmadığı zamanlarda da sıklıkla bilgi akışına maruz kalmaktadırlar. Bu bilgi akışı her zaman doğru ve iyi niyetli olmayabilir. Bu sebeple elde edilen bilgiler kullanılmadan önce bir dizi tedbir almak önemlidir. Bu tedbirler:

- Kullanıcıya bilgi aktaran kanal (İnternet sitesi, sosyal medya hesabı), kaynak belirtmelidir. Kaynağı belirtilmemiş bilgiye şüphelenme yaklaşımmalıdır.
- Elde edilen bilgiler üç farklı kaynaktan teyit edilmelidir.
- Bilgiyi aktaran İnternet sitesinin adresi kontrol edilmelidir. Alan adı uzantıları birçok İnternet sitesi için fikir verebilir. Örneğin;
  - o **.com** ya da **.net** alan adı uzantısına sahip İnternet siteleri ticari amaçlı sitelerdir.
  - o **.gov**: Devlet kurumlarının resmi sitelerinin uzantısıdır.
  - o **.org**: Ticari amacı olmayan vakıf, dernek ve organizasyonların kullandığı uzantıdır.
  - o **.edu**: Üniversite ve akademik kuruluşların siteleri için kullanılır.
  - o **.k12**: Okul öncesi, ilkokul, ortaokul ve lise gibi eğitim kurumlarına ait uzantıdır.

Bilgi edinilen İnternet siteleri, uzantılarına göre değerlendirilerek kaynak güvenilirliği konusunda bir kanya varılabilir. Türkiye Cumhuriyeti'nin İnternet ülke kodu **.tr**'dır. Bu uzantıya sahip sitelere yönelik ülke içinde ayrı bir kontrol gerçekleştirildiği için bu sitelerin güvenilirliklerinin daha yüksek olduğu söylenebilir. Örneğin; Millî Eğitim Bakanlığının İnternet site adresi **meb.gov.tr**, Türkiye Erozyonla Mücadele ve Ağaçlandırma Vakfının adresi de **tema.org.tr**'dır.

Adresler incelendiğinde,



Bu adresin Türkiye Cumhuriyeti'ne (.tr) ait bir devlet/hükûmet (.gov) sitesi olduğu görülebilir.



Bu adresin de Türkiye Cumhuriyeti'nde (.tr) faaliyet gösteren bir vakıf ya da derneği (.org) ait olduğu anlaşılabılır.



## Düşünelim/Deneyelim

Bir arama sitesine “e-okul” ifadesini yazıp listelenen arama sonuçlarını inceleyerek hangisinin e-okul uygulamasının resmi sitesi olduğunu bulunuz.

Internet sitelerinin adreslerini tanımak, yalnızca doğru bilgiye ulaşmak için gerekli değildir. Aynı zamanda karşılaşabilecek sahtecilik ve bilgi hırsızlığından korunmak için de çok önemlidir. Bir önceki etkinlikte e-okul başlığıyla listelenen birbirinden farklı adresler görmüş olmalısınız. e-okul gibi hizmetlere ya da bankaların İnternet sitelerine giriş yaparken bazı özel bilgiler girmeniz gereklidir. Özel bilgilerinizi girdiğiniz sitenin doğru site olduğundan emin olmalısınız.

### Altay Spor Kulübü Resmi Web Sitesi

[www.altay.org.tr/](http://www.altay.org.tr/)

Altay SDD; Altay Vakfı; Tesisler; Takımlar. A Takım (teknik ... BÜYÜK ALTAY 6-1 BEYLERBEYİ. 100. YIL FOTOĞRAF ... İletişim. Haberler. Müzemiz. Spor Okulları.

Kuruluş Öyküsü · İletişim · Ana Yönetim · Başkanlar

Görselde bir arama sonucunu görmektesiniz. Arama sonuçlarının en üstündeki mavi renkli kısım başlıktır. Arama sonuçlarının başlıklarını link/bağlantı niteliğindedir. Yani tıklandığında bir İnternet adresine yönlendirilirsiniz. Hangi adrese yönlendirildiğini arama başlığından değil, başlığın altındaki -yeşil renkli- adres bilgisinden anlayabilirsiniz. Görseldeki arama sonucunun başlığında tıklandığında İnternet tarayıcınız [www.altay.org.tr](http://www.altay.org.tr) sayfasını görüntüleyecektir. Arama sonuçlarında görüntülenen diğer kısım olan siyah renkli metindeyse siteye ilişkin tanıtıci bilgi ve açıklamalar görülebilir.

İnternet ortamında karşılaşılan bilgilerin doğruluğunu teyit etmek ve ayrıca sahteciliğe maruz kalmamak için İnternet sitelerinin adreslerini tanımanın önemini görmüş olduk. Bundan ayrı olarak özellikle sosyal medyada ya da bazen İnternet sitelerinde çeşitli görseller manipüle edilerek ya da olduğundan çok farklılaşmış gibi anlatılarak yanlış bilgilendirme, hatta kıskırtma yapılmaktadır. Böyle durumlarda da görsele dayalı doğrulama yapmak mümkündür.

Bir paylaşım, insanları kıskırtıp şiddet olayları çıkarmak için yapılmış olabilir. Bu tip paylaşımıları doğru kabul etmeden önce kontrol edilmek, istenmeyen olayların önüne geçecektir. Bunu anlamak için haberde kullanılan görselin üzerine sağ tıklayıp fotoğrafı bilgisayarınıza indirebilir ya da yine sağ tıklama seçeneklerindeki “bağlantı adresini kopyala” komutunu kullanabilirsiniz.



Bir arama sitesinin görsel arama sayfasını açıp işaretli yere tıkladığınızda sizden görsel yüklemenizi isteyecektir. Burada, bilgisayarınızda kayıtlı bir görseli yükleyebileceğiniz gibi görselin adresini ilgili alanına girebilirsiniz.



Bu iki yoldan biriyle görselimizi yükledikten sonra arama motoru benzer görselleri listeleyecektir. Listelenen görseller arasında, aradığınız görselin aynısının daha önce paylaştığını görüporsanız ilgili görseli tıklayabilir ve görselin bulunduğu Internet sitesini açabilirsiniz. Böylece, paylaşılan bilgiyi inanılar kılmak için kullanılan görselin Internet'te farklı bir ortamdan alınan bambaşka bir görsel olduğunu görebilir ve paylaşılan bilginin doğru olmadığını belirleyebilirsiniz.

### 1.2.5. Internet Etiği

Internet kullanımı ile ilgili olarak dikkat edilmesi gereken etik ilkeler; kişilik hakları, özel yaşamın gizliliği ve veri güvenliği gibi başlıklar altında incelenebilir. Internet ortamında uyulması gereken etik kurallar aşağıda verilmiştir:

- Bize yapılmasından hoşlanmadığımız davranışları başkalarına yapmaktan kaçınmalıyız.
- Bir durum karşısında Internet'te nasıl davranışımız gerektiği konusunda kararsız kaldığımız zaman gerçek hayatı böyle bir durum karşısında nasıl davranıyorsak öyle davranmalıyız.
- Internet'te karşılaştığımız ancak yüzünü görmedigimiz, sesini duymadığımız kişilere saygı kuralları çerçevesinde davranmalıyız.
- Internet sadece belirli bir ırkın, topluluğun ya da ülkenin tekelinde değildir. Tüm dünyadan pek çok farklı kültür ve inanca sahip insan Internet ortamında varlık göstermektedir. Internet'i kullanırken her kültüre ve inanca saygılı olmak, yanlış anlaşabilecek davranışlardan kaçınmak gerektiği unutulmamalıdır.
- Internet'i yeni kullanmaya başlayan kişilerin yapacağı yanlış davranışlara karşı onlara anlayış gösterip yardımcı olmaya çalışmak ve yol göstermek gerektiği de unutulmamalıdır.
- Özellikle sosyal medya, sohbet ve forum alanlarındaki kişiler ile ağız dalaşı yapmaktan kaçınmalı, başka insanları rahatsız etmeden yazışmaya özen göstermeliyiz. Ayrıca, sürekli olarak büyük harfler ile yazışmanın Internet ortamında bağırmak anlamına geldiği unutulmamalıdır.
- İnsanların özel hayatına karşı saygı göstererek kişilerin sırlarının Internet ortamında paylaşılmasına dikkat edilmesi gerektiği unutulmamalıdır.
- Internet'te kaba ve küfürlü bir dil kullanımından kaçınarak gerçek hayatı karşımızdaki insanla söylemeyeceğimiz ya da yazamayacağımız bir dil kullanmamalıyız.
- Internet'i başkalarına zarar vermek ya da yasa dışı amaçlar için kullanmamalı ve başkalarının da bu amaçla kullanmasına izin vermemeliyiz.
- Internet ortamında insanların kişilik haklarına özen göstererek onların paylaştığı bilginin izinsiz kullanımından kaçınmamız gerektiği de unutulmamalıdır.

Internet ortamında nasıl davranışması gerektiğini öğrenmiş oldunuz. Sizin doğru davranışın oluşturunuz, herkesin de size doğru davranışmasını sağlayamayabilir. Internet ortamında başkalarından kaynak-

lanan kötü davranışlara maruz kalabilirsiniz. İnternet etiğine uymayan bu davranışlara **siber (dijital) zorbalık** denir.

Siber zorbalığa maruz kalmanız durumunda yapmanız gerekenleri söyle sıralayabiliriz:

- Zorbalık yapan hesaplara cevap vermeyiniz, onlarla tartışmaya girmeyiniz. İlk yapmanız gereken, zorbalık yapan hesabı engellemektir.
- Bu hesapları, bulunduğu sosyal medya platformundaki “Bildir/Shikâyet Et” bağlantısını kullanarak şikayet ediniz. Böylece bu kişilerin size yaptığı etik dışı davranışları başkalarına da yapmasını engellemiş olursunuz.
- Size yönelik etik dışı davranışlar artarak ve ağırlaşarak devam ederse bunların ekran görüntülerini ve mesajları kaydediniz. Bu kanıtlarla birlikte ailenizin ya da rehber öğretmeninizin gözetiminde hukuki yollara başvurunuz.
- Siber zorbalığa maruz kalan başka kişiler de olabilir. Böyle durumlarda bu kişilere ne yapmaları gerekiği konusunda yardımcı olabilir, kötü kullanım bildirimini siz de yapabilirsiniz. Zorba bir hesap için kötü kullanım bildirimi sayısı fazla olursa o hesabın site yönetimi tarafından incelenmesi ve kapatılması daha çabuk olacaktır.

### 1.3. Bilgi Güvenliği

Bu bölümde;

- ✓ Bilgi güvenliğinin önemini açıklayacak,
- ✓ Bilgi güvenliğine yönelik tehditleri kavrayacak,
- ✓ Sayısal dünyada kimlik yönetimi konusunda güvenlik açısından yapılması gerekenleri listeleyecek,
- ✓ Kişisel bilgisayar ve ağ ortamında bilgi güvenliğini sağlamaya yönelik işlemleri gerçekleştireceksiniz.



Bilim ve teknolojide yaşanan hızlı ilerleme, içinde bulduğumuz 21. yüzyılın bilgi çağının isimlendirilmesine yol açmıştır. Günümüzde bilişim teknolojilerinin yaygın kullanımı ile birlikte bilginin üretimi ve kullanılması büyük önem kazanmış ve bu teknolojiler aracılığı ile üretilen veri miktarında da büyük bir artış olmuştur. Bilgisayarlar ve akıllı cihazlar aracılığı ile başta İnternet olmak üzere bilgiye erişimin farklı yollarının ortaya çıkması da bilginin depolanması, iletilmesi ve korunması ile ilgili pek çok problemi de beraberinde getirmiştir. Bu problemlerden biri de kişisel ya da kurumsal bilgiyi erişilmez kılmaya, ele geçirmeye ya da değiştirmeye yönelik olanlardır. Kişisel ya da kurumsal düzeyde bizim için büyük önem teşkil eden her tür bilgiye izin alınmadan ya da yetki verilmeden erişilmesi, bilginin ifşa edilmesi, kullanımı, değiştirilmesi, yok edilmesi gibi tehditlere karşı alınan tüm tedbirlere **bilgi güvenliği** denir. Bilgi güvenliği, “gizlilik”, “büyünlük” ve “erişilebilirlik” olarak isimlendirilen üç temel ögeden meydana gelmektedir. Bu üç temel güvenlik unsurundan birinin zarar görmesi durumunda güvenlik zaafiyeti oluşabilir.

Bilgi güvenliğini oluşturan unsurlardan **gizlilik**, bilginin yetkisiz kişilerin eline geçmemesi için korunmasıdır. Başka bir deyişle gizlilik, bilginin yetkisiz kişilerce görülmesinin engellenmesidir. e-posta hesap bilgisinin bir saldırgan tarafından ele geçirilmesi buna örnek verilebilir. **Bütünlük**, bilginin yetkisiz kişiler tarafından değiştirilmesi ya da silinmesi gibi tehditlere karşı korunması ya da bozulma-

masıdır. Bir web sayfasında yer alan bilgilerin saldırgan tarafından değiştirilmesi, bütünlük ilkesinin bozulmasına örnek verilebilir. **Erişilebilirlik** ise bilginin yetkili kişilerce ihtiyaç duyulduğunda ulaşılabilir ve kullanıma hazır durumda olmasıdır. Bir web sitesine erişimin saldırısı sonucunda engellenmesi erişilebilirlik ilkesinin ihlal edilmesine örnek olarak verilebilir.

### 1.3.1. Bilgi Güvenliğine Yönelik Tehditler

Bilgi ve bilişim teknolojileri güvenliğinde başlıca tehdit, korsan ya da saldırgan olarak adlandırılan kötü niyetli kişiler ve bu kişilerin yaptıkları saldırılardır. Bir bilişim teknolojisi sistemine sızmak, sistemi zaafiyete uğratmak, sistemlerin işleyişini bozmak ve durdurmak gibi kötü niyetli davranışlar; **siber saldırı** veya **atak** olarak adlandırılmaktadır. Günümüzde siber dendögünde ilk akla gelen “Sanal Dünya (Internet)” olsa da bir cihazın siber kavramı içinde yer alması için İnternet bağlantısına sahip olması gerekmektedir. **Siber** ya da **siber uzay**; temeli bilişim teknolojilerine dayanan, tüm cihaz ve sistemleri kapsayan yapıya verilen genel addır. Fiziki sınırları ve kuralları olmayan bu siber dünya içinde yaşanan saldırı, suç, terör, savaş gibi kötü niyetli hareketler daha çok elle tutulur, gözle görülür varlıklarımız için oluşturulmuş kurallar ve yasalar ile engellenemez/korunamaz. Korsanlar ya da saldırganlar, istediklerini elde edebilmek için çok farklı teknikler kullanabilirler. Bu tür saldırı türlerinin tanınması, doğru şekilde analiz edilmesi ve gereken tedbirlerin alınabilmesi siber güvenlik için çok önemlidir. Siber güvenlik; siber ortamda yaşanabilecek suç, saldırı, terörizm, savaş, gibi tüm kötü niyetli hareketlere karşı alınacak tedbirler bütündür. Siber ortamda yaşanabilecek kötü niyetli hareketler aşağıda tanımlanmıştır:



**Siber Suç:** Bilişim teknolojileri kullanılarak gerçekleştirilen her tür yasa dışı işlemidir.

**Siber Saldırı:** Hedef seçilen şahıs, şirket, kurum, örgüt gibi yapıların bilgi sistemlerine veya iletişim altyapılarına yapılan planlı ve koordineli saldırıdır.

**Siber Savaş:** Farklı bir ülkenin bilgi sistemlerine veya iletişim altyapılarına yapılan planlı ve koordineli saldırılardır.

**Siber Terörizm:** Bilişim teknolojilerinin belirli bir politik ve sosyal amaca ulaşabilmek için hükümetleri, toplumu, bireyleri, kurum ve kuruluşları yıldırmaya, baskı altında tutma ya da zarar verme amacıyla kullanılmasıdır.

**Siber Zorbalık:** Bilgi ve iletişim teknolojilerini kullanarak bir birey ya da gruba, özel ya da tüzel bir kişiliğe karşı yapılan teknik ya da ilişkisel tarzda zarar verme davranışlarının tümüdür.

Yukarıda bahsedilen saldırılar bireysel, kurumsal ve toplumsal hedeflere yönelik olabilmektedir. Bireysel saldırırlarda ana hedefi kişisel bilgilerin ele geçirilmesi, değiştirilmesi ya da yok edilmesi olşturken kurumsal ve toplumsal saldırırlarda ise çoğunlukla kurumlar ve devletin zarara uğratılması hedeflenmektedir. Daha çok devletler arası düzeyde gerçekleşen siber savaşlarda ise ana hedef; sağlık, enerji, ulaşım, haberleşme gibi kritik altyapılardır.



### 1.3.2. Sayısal Dünyada Kimlik ve Parola Yönetimi

Her gün sıkça kullandığımız şifre ve parola kavramlarını inceleyecek olursak “**parola**” bir hizmete erişebilmek için gerekli olan, kullanıcıya özel karakter dizisidir. “**Sifre**” ise sanal ortamdaki verilerin gizliliğini sağlamak için veriyi belirli bir algoritma kullanarak dönüştüren yapıdır.

Bir bilişim sistemine erişimin belli kurallar çerçevesinde yapılması amacıyla o sistemi kullanmak isteyen kişilerin kullanıcı adı ve/veya parola ile erişim yetkisine sahip olduklarını ispatlamaları gerekmektedir. Buradaki kullanıcı adı ve parola, bilgiye erişim yetkisinin kanıtı olarak kullanılmaktadır.

Kullanıcı adı, her kullanıcının sadece bir tane olduğunu garantilemek amacıyla benzersiz bir bilgi olarak oluşturulur. Bu bilgi, bilişim sistemi tarafından otomatik olarak verilebileceği gibi kullanıcılarından kimlik No., öğrenci No. ya da e-posta gibi bilgiler istenerek de oluşturulabilir. Parola ise içinde büyük ya da küçük harfler, rakamlar ve özel karakterler barındıran bir karakter dizisidir.

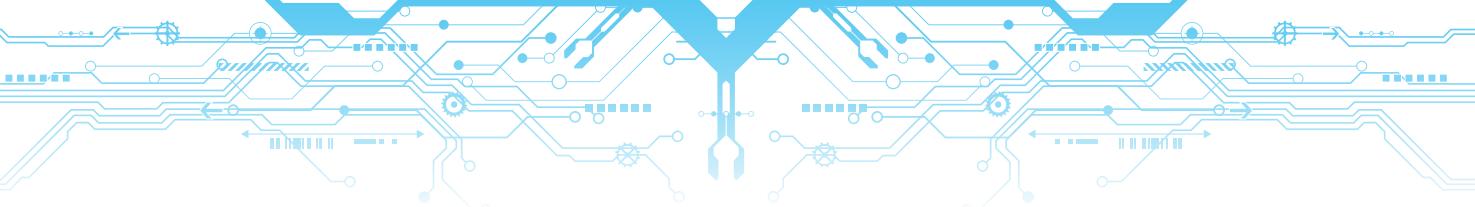


Parola, bilgi güvenliğinin en önemli öğesidir. Parolanın da ele geçirilmesi durumunda oluşacak zarar, bir evin anahtarını ele geçiren hırsızın sebep olduğu zarardan çok daha fazla olabilir. Parolanın kötü niyetli kişiler tarafından ele geçmesi durumunda,

- Elde edilen bilgiler yetkisiz kişiler ile paylaşılabilir ya da şantaj amacıyla kullanabilir.
- Parolası ele geçirilen sistem başka bir bilişim sisteme saldırı amacıyla kullanılabilir.
- Parola sahibinin saygınlığının zarar görmesine yol açabilecek eylemlerde bulunulabilir.
- Ele geçirilen parola ile ekonomik kayba uğrayabilecek işlemler yapılabilir.
- Parola sahibinin yasal yaptırımla karşı karşıya kalmasına yol açabilir.

Bilişim sistemlerinde parolanın ele geçirildiğinin ispatlanması ya da bu işi yapan kişilerin belirlenmesi çok zor olduğundan parolası ele geçirilen sistem üzerinde yapılacak kötü niyetli eylemler parola sahibinin ciddi yaptırımlar ile karşı karşıya kalmasına neden olabilir. Bir bilişim sistemine erişim için kanıt olarak kullanılan parolanın dikkatle belirlenmesi ve titizlikle korunması gereklidir. Sadece rakamlardan oluşan 6 haneli bir parolanın özel programlar yardımı ile dakikalar içinde kırılması mümkün değildir. Güçlü ve kırılması zor bir parolanın oluşturulması için olabildiğince sayı, büyük/küçük harfler ile özel karakterler içermesine dikkat edilmesi son derece önemlidir. Başkalarının da kolaylıkla tahmin edebileceği qwerty, 123456 gibi ardışık harfler ve sayıların kullanılması, parolanın doğum yılı ya da mezuniyet tarihi gibi kişisel bilgi içermesi de zayıf parolalar için örnek gösterilebilir. Günümüzde saldırganların parolarları ele geçirmek ya da tahmin edebilmek için sosyal medyadan faydalandıkları da unutulmamalıdır. Sosyal medya aracılığı ile ulaşılabilen aile fertlerinin adı, doğum tarihi gibi bilgiler de parola belirlemek amacıyla kullanılmamalıdır. Saldırganlar, sosyal medya ortamlarını kendi çıkarları için kullanarak sosyal mühendislik adı verilen ikna ve kandırma teknikleri ile bu bilgileri elde edebilirler. Güçlü bir parolanın belirlenmesi için aşağıdaki kurallar uygulanmalıdır.

- Parola, büyük/küçük harfler ile noktalama işaretleri ve özel karakterler içermelidir.
- Parola, -aksi belirtildiğinde- en az sekiz karakter uzunluğunda olmalıdır.
- Parola, başkaları tarafından tahmin edilebilecek ardışık harfler ya da sayılar içermemelidir.
- Her parola için bir kullanım ömrü belirleyerek belirli aralıklar ile yeni parola oluşturulması gereklidir.



Parolanın güvenliği açısından, aşağıdaki kurallara dikkat edilmelidir:

- Parolanın başkalarıyla paylaşılmaması son derece önemlidir.
- Parolalar, basılı ya da elektronik olarak hiçbir yerde saklanmamalıdır.
- Başta e-posta adresinin parolası olmak üzere farklı bilişim sistemleri ve hizmetler için aynı parolanın kullanılmaması gereklidir.

Bu durumda, bir kullanıcının onlarca parola üretmesi gerekebilir. Bu da parolaların unutulması sorununu beraberinde getirir. Böyle bir sorun yaşamamak için kullanıcılar kendilerine özgü kalıplardan yararlanmalıdır.

Örnek olarak;

- Bir anahtar kelime belirlenerek kelime, parola kriterlerine uygun hâle getirilebilir. "Alsancak" kelimesi, parola oluşturma kriterleri göz önüne alınarak "A1s@nc@k" şeklinde düzenlenebilir (8 karakter, büyük harf, küçük harf, sayı ve özel karakter içeriyor.). Bu anahtar kelimenin başına, ortasına ya da sonuna kullanılan platformun kısa ismi eklenecek o hizmete özgü parola oluşturulmuş olur. Twitter için A1s@nc@kTW, Facebook için A1s@nc@kFB gibi.
- Bir anahtar cümle belirlenerek bu cümlenin bazı harfleri kullanılabilir. Örneğin "Muhtaç olduğun kudret, damalarındaki asıl kanda mevcuttur." cümlesindeki kelimelerin baş harfleri kullanılarak 7 karakter elde edilir. Bu yedi karakterin yanına, kullanılacak platformun kodu da eklendiğinde 9 karakterli bir parola elde edilebilir.

e-posta hesabı için: M0kd@kM@il, Instagram için: M0kd@kMig gibi...

\* Anahtar kelime oluştururken;

G yerine 6,

g yerine 9,

Ş yerine \$

a yerine @

i, l yerine 1 gibi karakterler kullanılabilir.



### Düşünelim/Deneyelim

Hayalî bir kişinin üç farklı sosyal medya hesabı için güvenli parolalar oluşturunuz.

### **1.3.3. Kişisel Bilgisayarlarda ve Ağ Ortamında Bilgi Güvenliği**

Bilgisayar ve İnternet teknolojisindeki hızlı ilerleme sonucunda üretilen veri, hiç durmadan artmaktadır. Özellikle İnternet kullanımının oluşturduğu büyük miktardaki bilgi, her gün milyonlarca insan tarafından kullanılmaktadır. Bu veriyi üreten, kullanan ve paylaşan insanların çok küçük bir kısmı ise İnternet'in tehlikeleri ve bilgi güvenliği konusunda bilgi sahibidir. Bilişim teknolojisinin kullanımında temel amaç bilgiye erişmektir. Ancak, teknolojinin hızlı ilerleyışı ile birlikte gelen güvenlik riskleri ve insanların bu konudaki yetersiz farkındalıkları bilgisayar ve İnternet kullanımı sırasında pek çok tehlikenin ortayamasına neden olmaktadır.

Bilişim sistemlerinin çalışmasını bozan veya sistem içinden bilgi çalmayı amaçlayan Virüs, Solucan, Truva Atı ya da Casus yazılım gibi kötü niyetlerle hazırlanmış yazılım veya kod parçaları zararlı programlar olarak adlandırılır.



Bu zararlı programlar,

- İşletim sisteminin ya da diğer programların çalışmasına engel olabilir.
- Sistemdeki dosyaları silebilir, değiştirebilir ya da yeni dosyalar ekleyebilir.
- Bilişim sisteminde bulunan verilerin ele geçirilmesine neden olabilir.
- Güvenlik açıkları oluşturabilir.
- Başka bilişim sistemlerine saldırı amacıyla kullanılabilir.
- Bilişim sisteminin, sahibinin izni dışında kullanımına neden olabilir.
- Sistem kaynaklarının izinsiz kullanımına neden olabilir.

**Virüsler**, bulaşıkları bilgisayar sisteminde çalışarak sisteme ya da programlara zarar vermek amacıyla oluşturur. Virüsler bilgisayara e-posta, bellekler, İnternet üzerinden bulaşabilir. Bilgisayarın yavaşlaması, programların çalışmaması, dosyaların silinmesi, bozulması ya da yeni dosyaların eklenmesi virüs belirtisi olabilir.

**Bilgisayar Solucanları**; kendi kendine çoğalan ve çalışabilen, bulaşmak için ağ bağlantılarını kullanan kötü niyetli programlardır. Sistem için gerekli olan dosyaları bozarak bilgisayarı büyük ölçüde yavaşlatırabilir ya da programların çökmesine yol açabilir. Ayrıca sistem üzerinde arka kapı olarak adlandırılan ve saldırganların sisteme istedikleri zaman erişmelerini sağlayan güvenlik açıkları oluşturabilir.

**Truva Atları**, kötü niyetli programların çalışması için kullanıcının izin vermesi ya da kendi isteği ile kurması gereği için bunlara Truva Atı denmektedir. Truva Atları saldırganların bilişim sistemi üzerinde tam yetki ile istediklerini yapmalarına izin verir. Sisteme bulaşan bir Truva Atı ilk olarak güvenlik yazılımlarını devre dışı bırakarak saldırganların bilişim sisteminin tüm kaynaklarına, programlarına ve dosyalarına erişmesine olanak sağlar. Güvensiz sitelerden indirilen dosyalar, tanınmayan kişilerden gelen e-postalar ya da taşınabilir bellekler aracılığı ile yayılabilir.

**Casus Yazılımlar**, İnternet'ten indirilerek bilgisayara bulaşan ve gerçekte başka bir amaç ile kullanılısa bile arka planda kullanıcıya ait bilgileri de elde etmeye çalışan programlardır. Bunlar, sürekli reklam amaçlı pencerelerin açılması ya da İnternet tarayıcıya yeni araçların eklenmesine neden olabilir.

### Zararlı Programlara Karşı Alınacak Tedbirler

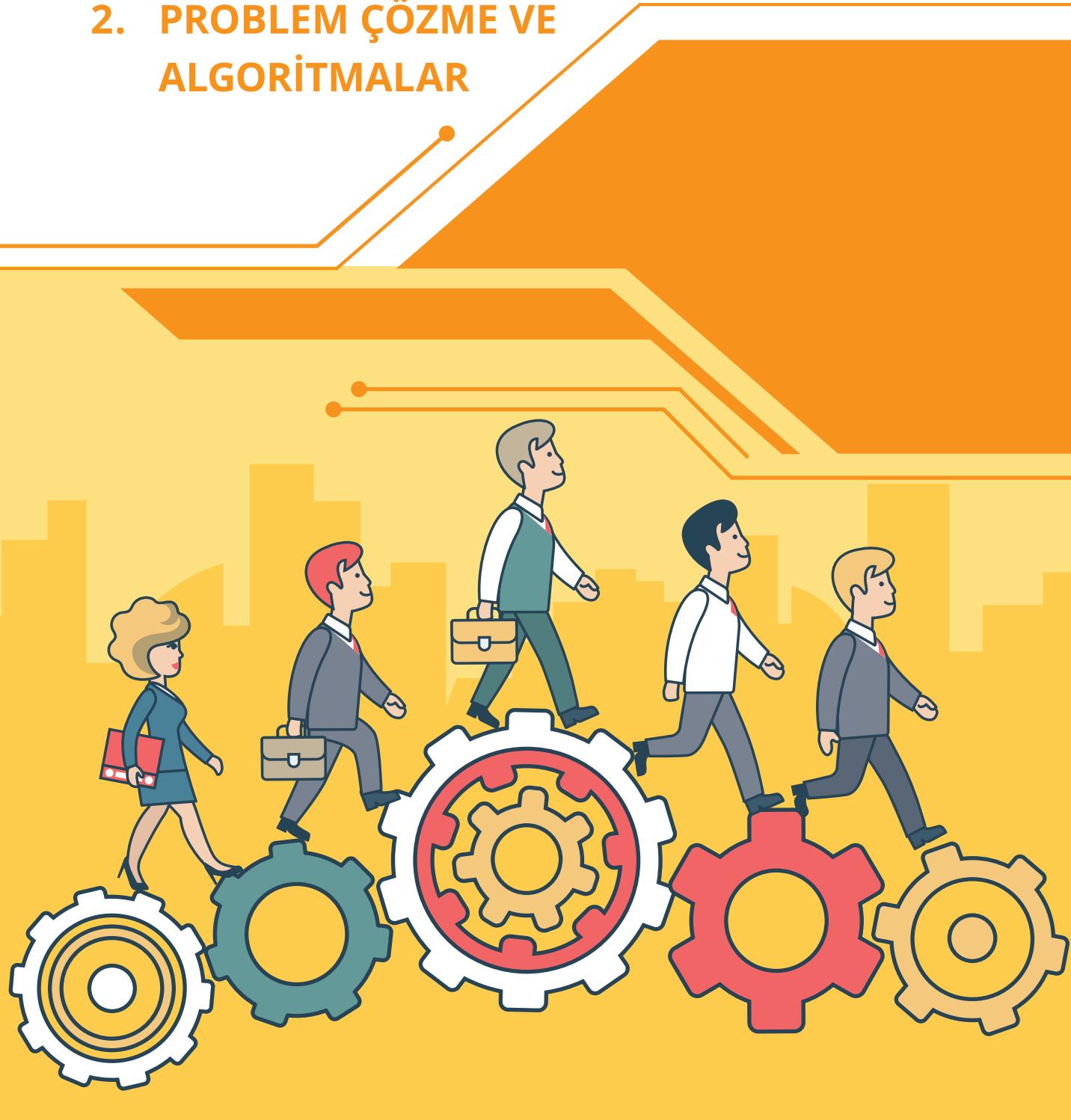
- Bilgisayara antivirüs ve İnternet güvenlik programları kurularak bu programların sürekli güncel tutulması sağlanmalıdır.
- Tanınmayan/güvenilmeyen e-postalar ve ekleri kesinlikle açılmamalıdır.
- Ekinde şüpheli bir dosya olan e-postalar açılmamalıdır. Örneğin *resim.jpg.exe* isimli dosya bir resim dosyası gibi görünse de uzantısı *.exe* olduğu için uygulama dosyasıdır.
- Zararlı içerik barındıran ya da tanınmayan web sitelerinden uzak durulmalıdır.
- Lisanssız ya da kırlımış programlar kullanılmamalıdır.
- Güvenilmeyen İnternet kaynaklarından dosya indirilmemelidir.



### Düşünelim/Deneyelim

Zararlı yazılımları ve bu yazılımların ne tür zararlar verebileceğini inceleyiniz.

## 2. PROBLEM ÇÖZME VE ALGORİTMALAR



Bu bölümde;

- ✓ Genel problem çözme kavramlarını,
- ✓ Klasik bulmacaları ve çözümlerini detaylı biçimde öğreneceksiniz.

## 2.1. Problem Çözme Kavramları ve Yaklaşımlar

### 2.1.1. Programlama Nedir?

Bir bilgisayar bilimcisi gibi düşünmek ve programlama ne demektir? Bu düşünme şekli matematiğin, mühendisliğin ve doğa bilimlerinin bazı özelliklerini birleştirmektedir. Bilgisayar bilimcileri genel olarak matematiksel semboller, işlemleri ve formülleri kullanır, mühendisler gibi tasarım yaparak farklı sistemler oluşturur ve bilim insanları gibi deney yaparak teknoloji desteği ile çözüm üretir.

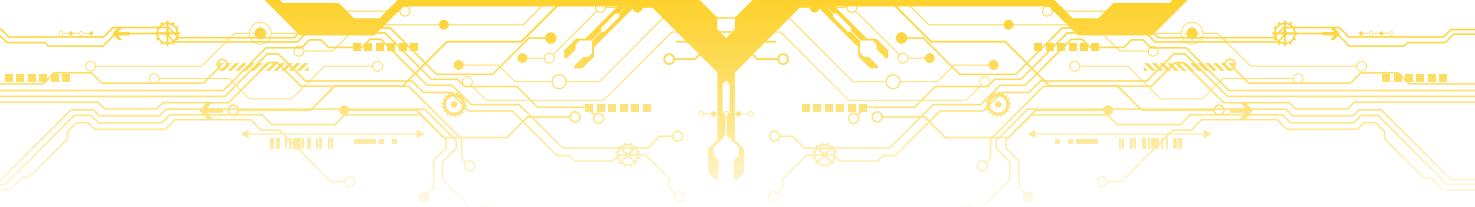
Bir bilgisayar bilimcisi için en önemli beceri problem çözme becerisidir. Problem çözme; problemleri formüle edebilme, farklı ve yaratıcı çözüm yolları önerme, çözümü kesin ve doğru biçimde ifade edebilme becerisidir. Programlamayı öğrenme sürecinde yalnızca problem çözme becerisi yeterli değildir çünkü programlama aynı zamanda bir düşünme biçimidir. Bir insan makine değildir, o yüzden o şekilde düşünmeye zorlanamaz. Ancak, bilgiyi işleme süreçleri vardır ve verileri yorumlama, dönüştürme ve sunma gibi farklı süreçlerin yaratıcı düşünme ile desteklenmesi çok önemlidir. Programlama, hem problem çözme becerisi hem de bilgi işlemel düşünme becerisine sahip olmayı gerektirir.



**Bilgi işlemel düşünme;** bilgisayar biliminin kavramlarından yararlanarak problem çözme, sistem tasarılama ve insan davranışlarını anlama olarak tanımlanabilir. Ayrıca Bilgisayar Bilimi Öğretmenleri Birliği (Computer Science Teachers Association - CSTA) ve Uluslararası Eğitimde Teknoloji Topluluğu (International Society for Teachnology in Education - ISTE) tarafından tanımlandığı şekliyle bilgi işlemel düşünme aşağıdaki özellikleri barındıran bir problem çözme sürecidir.

- Problemleri bilgisayar veya başka araçlar yardımı ile çözebilir hâle getirme
- Mantıklı bir şekilde verileri düzenlemeye ve çözümleme
- Model ve benzetim desteği ile verileri sunma
- Algoritmik düşünme çerçevesinde çözümleri otomatikleştirme
- Kaynakları verimli bir şekilde kullanarak uygun çözümleri tanımlama, çözümleme ve uygulama
- Bulunan çözümü farklı problemlere transfer etme ve genelleştirme

Bilgi işlemel düşünme becerisi; problem çözümleme, veri sunma ve modelleme gibi bazı benzer kavramlar ile ilişkili görülmekte ve sadece bilgisayar bilimcileri için değil, herkes için gerekli temel bir beceri olarak tanımlanmaktadır. Bilgi işlemel düşünme sayesinde siz öğrenciler bilgisayarlar ile çözümlerini otomatik hâle getirip problemleri daha etkili çözebilecek ve düşünmenin sınırlarını genişletebileceksiniz. Dahası, bilgisayar biliminin kavramlarını ve ilkelerini öğrendiğiniz zaman gittikçe değişen teknolojik hayatı ve iş yaşamına daha iyi hazırlanabileceksiniz. Bilgisayarın bilgi işleme süreci ile benzerlik gösteren düşünme yaklaşımıyla, değişen araçlar ve uygulamalardan etkilenmeden, yaşam boyu öğrenen bireyler olabileceksiniz.



Farklı bir bakış açısı ile programlama; bilgisayarın donanıma nasıl davranacağını anlatan, bilgisayara yön veren komutlar ve işlemler bütünüdür. Kısaca yazılım geliştirme, test etme ve bakımını yapma sürecidir. Bir programlama sisteminin iki bileşeni vardır:

1. Bilgisayara kurulmuş olan bileşen – programlama ortamı
2. Programcı tarafından oluşturulan algoritma ve program kodları



Kullandığımız programlama ortamı ile programcı tarafından kullanılacak kelime ve komutları oluşturur, program akışını ve mevcut durumu kontrol edebilir, adım adım işlemleri takip edebilir, oluşturduğumuz işlemleri genelleştirerek soyutlaştıracaktır. Kullandığımız programlama dili ile yapmak istediğimiz işlemleri bilgisayarın anlayacağı biçimde ifade edebilir, işlemleri parçalara bölebilir, parçalardan farklı ve anlamlı bütünlükler oluşturabiliriz.

### 2.1.2. Program Nedir?

**Program**, yapılacak bir işlemi ya da hesaplamayı gerçekleştirmek için birbirini izleyen komut ya da yönergelerden oluşan yapıdır. İşlemler matematiksel ya da mantıksak olabilir. Örneğin bir formülün sonucunun hesaplanması ya da bir doküman içerisinde belirli bir metnin aranması gibi. Ayrıntılar programlama dillerine göre farklılaşsa bile belirli komutlar her dilde yer alır.

**Girdi:** Klavyeden, dosyadan veya başka bir aygıtta veri almazdır.

**Çıktı:** Ekranda veriyi görüntüleme veya veriyi dosyaya veya başka bir aygıta göndermedir.

**Matematik:** Toplama, çarpma gibi bazı temel matematiksel işlemleri gerçekleştirmezdir.

**Koşullu yürütme:** Belirli durumları sınamak ve komutları uygun bir sırada göre çalıştırmaktadır.

**Tekrarlama:** Bazı eylemleri genellikle ufak tefek değişikliklerle yineleme işlemidir.



Programların çoğu, ne kadar basit ya da karmaşık olursa olsun temel olarak bu işlemlere dayalı olarak çalışır. Bu nedenle programlama, büyük ve karmaşık bir görevi bu temel komutlarla gerçekleştirebilecek kadar basit biçimde küçük alt görevlere bölme olarak tanımlanabilir.



### 2.1.3. Hata Ayıklama Nedir?

Programlama, karmaşık bir süreçtir ve programcılar programlamada hata (bug) yapabilirler. Programlama hatalarını bulma ve düzeltme işlemine **hata ayıklama (debugging)** denilir. Bir programda üç tür hata oluşabilir: söz dizimsel hatalar, çalışma zamanı hataları ve anlam bilimsel hatalar.

#### 1. Söz dizimsel hatalar

Söz dizimi, programın yapısı ve bu yapılarındaki kurallar demektir. Örneğin Türkçede bir cümle büyük harfle başlamalı ve uygun bir noktalama işaretiyile sona ermeli. Bu kurallara uymayan cümlelere "Söz dizimi hatası içermektedir." diyebiliriz. Programlama dilleri için söz dizimi, yorumla açık olmayacak şekilde kesin ve net ifadeler içermelidir. Aksi takdirde program, söz dizimi hatası verir ve programın doğru çalışmasını bekleyemeyiz.

#### 2. Çalışma zamanı hataları

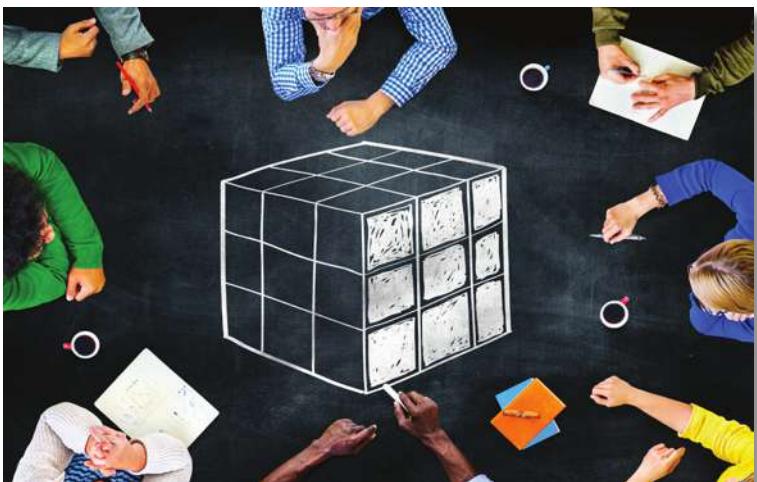
Bu hatalar ancak program çalıştırıldıktan sonra ortaya çıkar. Hesaplanması mümkün olmayan işlemler (sıfır bölünme) ya da hiç gerçekleşmeyecek koşulların ( $5 < 3$ ) yürütülmesi gibi durumlarda ortaya çıkar.

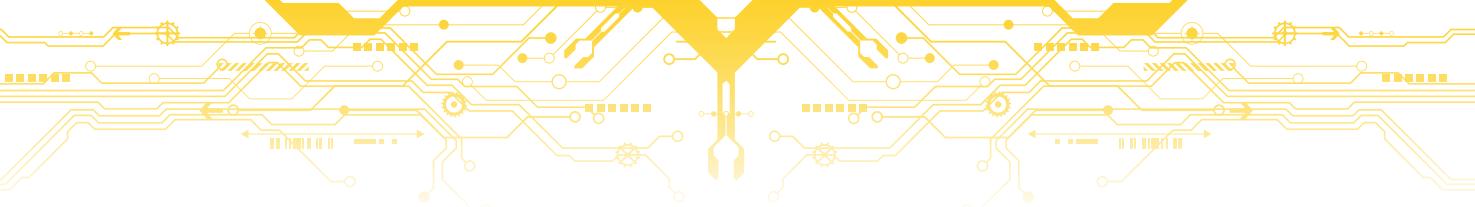
#### 3. Anlam bilimsel hatalar

Bu durumda program, genellikle hata vermeden çalışır ancak çoğu zaman beklenen sonucu üretmez. Bu yüzden programı satır satır çalıştırarak, farklı adımlardaki çıktıları gözlemleyerek nerede mantık hatası yapıldığını bularak program doğru biçimde çalışana kadar bu hataları ayıklamak gereklidir. Programlamayı öğrenirken kazanılacak önemli becerilerden biri de hata ayıklamadır. Yorucumasına rağmen, programlamada bilişsel yoğunluk gerektiren ilginç bir süreçtir. Hata ayıklama deneyimsel bir yaklaşımındır. Neyin hatalı gittiğine dair bir fikir olduğunda programı değiştirerek tekrar çalıştırırız. Böylece programlamaya yeni bir boyut daha kazandırmış oluruz. Programlama, program doğru biçimde çalışana kadar aşamalı olarak hata ayıklama sürecidir.

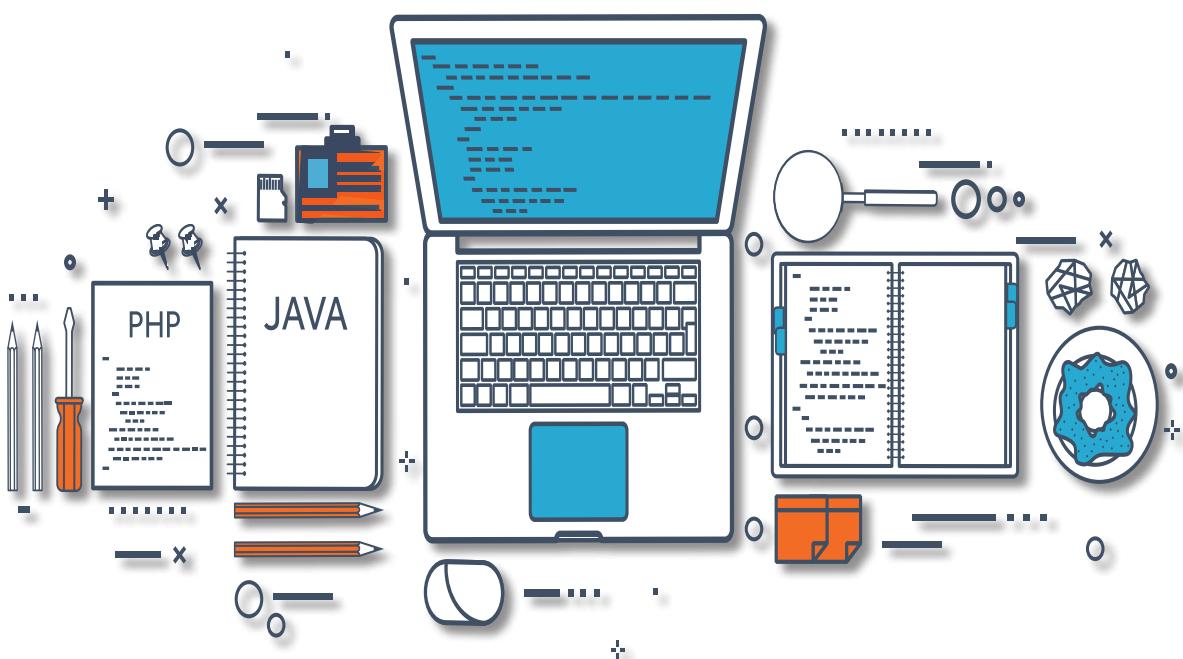
### 2.1.4. Günlük Hayatta Problem Çözme

Günlük hayatımızda problemlerimizi çözmek için yaşıntımızı etkileyen pek çok karar veririz. Bu kararlar yalnızca yaşıntımızı etkilemekle kalmaz, bazen yaşam kalitemizi ve geleceğimizi bile etkileyebilir. Örneğin karşılaştığımız problemler, televizyonda hangi kanalı seyretsem gibi basit de olabilir, hangi mesleği seçmeliyim gibi çok önemli de olabilir. Yanlış bir karar verilirse zaman ve kaynaklar boş gidebilir, bu nedenle nasıl doğru karar verildiğini öğrenmek önemlidir. En iyi kararı vermek aslında problem çözmektir. İnsan hayatı aslında bir problem çözme sürecidir. Genellikle bir problemin birden fazla çözümü vardır, her bir çözüm bir alternatif olarak düşünülebilir. Problem çözme, amaca ulaşabilmek için alternatifler arasından en uygun yolu belirlemektir. Alternatifler, farklı koşul ve beklenilere göre şekillenir. En uygun çözüm ise farklı koşul ve durumlar için değişiklik gösterebilir. Bu nedenle farklı kişiler ve problemler için çözüm önerileri de farklılık gösterebilir.





Problemler çözülmeye çalışılırken dikkate alınması gereken sınırlılıklar ve koşullar ile uyulması gereken kurallar vardır. Tüm bu veriler dikkate alınmaz ise doğru çözüme ulaşılamaz ya da problem geçici olarak göz ardı edilmiş olabilir. Bir problemi yazılım geliştirerek çözerken de çeşitli sınırlılıklar vardır: kullandığınız programlama dili, çalıştığınız ortam (kişisel bilgisayar, tablet vb.) ve performans (kullanığınız işlemci, hafıza, disk vb.). Bu nedenle programcılar için problem çözme, “bir dizi işlemi, belirtilen sınırlılıklara uygun biçimde gerçekleştirebilen programın yazılması” anlamına gelir. Programlamaya yeni başlayanlar işlemleri gerçekleştirdiğinde hedefe ulaşma konusunda daha fazla istekli olduklarından genellikle belirtilen sınırlılıklara uyma konusunda zorluk yaşayabilirler. Bu nedenle böyle programlar, istenilen işlemleri yalnızca “belli şartlarda (sadece 2 basamaklı sayılar için, tek bir döngü için vb.)” yerine getirebilen programlara dönüşebilir. Bu nedenle bu tür uygulama hatalarından kaçınarak tüm detayları dikkatli bir şekilde işe koşarak programlama yapmamız son derece önemlidir.



## 2.1.5. Problem Çözme Süreci

Problem çözme farklı biçimlerde düşünmeyi gerektiren bir eylemdir. Öncelikle klasikleşmiş bazı klasik bulmacaları ve bu bulmacaların çözümlerini inceleyelim.

### 2.1.5.1. Tilki, Kaz ve Mısır Çuvalı

Konuşacağımız ilk klasik problem, beraberindeki nesneleri nehrin karşısına taşıması gereken bir çiftçiyile ilgili bir bulmaca. Bu çiftçinin bir tilkiyi, bir kazı ve bir mısır çuvalını nehrin karşısına geçirmesi gerekmektedir. Çiftçinin bu işlemi gerçekleştirmek için küçük bir tekneleri var ancak bu teknede çiftçi ile birlikte en fazla bir nesneye daha yer var. Ne yazık ki tilki ve kaz açtır. Bu yüzden tilki kaz ile yalnız kalamaz çünkü tilki kazı yiyebilir. Aynı şekilde kaz ve mısır çuvalı yalnız bırakılamaz çünkü kaz mısırı yiyebilir. Bu koşullarda çiftçi nehrin karşısına tilki, kaz ve mısırı sorunsuz bir şekilde nasıl geçirebilir?

Bu soru, Şekil 1.1'de görülmektedir. Bu soruna daha önce hiç rastlamadıysanız burada durun ve çözüm için birkaç dakika harcayınc. Bu bilmeciyi daha önce duydusunuz ve çözüdüyseniz nasıl çözüm bulduğunuzu hatırlamaya çalışın.



*Şekil 1.1: Tilki, kaz ve mısır çuvalı*

Tekne ile aynı anda en fazla iki nesne taşınabildiğini biliyoruz. Tilki ve kaz aynı kıyıda yalnız bırakılamayacağı gibi kaz da mısır çuvalıyla aynı kıyıda yalnız bırakılamaz.

Bu bilmeceyi bir ipucu olmadan çözmekte zorlanabiliriz. İşte mantık yürütme biçimi. Çiftçi her seferinde nesnelerden birini alabilecekinden çiftçinin her şeyi kıyıya götürmek için birden fazla gidip gelmesi gerekecektir. İlk gidiş sırasında çiftçi tilkiyi alırsa kaz, mısır çuvalıyla yalnız kalır ve kaz mısıri yiyebilir. Aynı şekilde çiftçi ilk gidiş sırasında mısır çuvalını alacak olursa tilki kaz ile yalnız kalacak ve tilki kazı yiyecektir. Bu nedenle çiftçi, Şekil 1.2'de gösterilen çözümde sunulduğu gibi ilk turda kazı almalıdır. Bununla birlikte bu adımdan sonraki tüm adımlar başarısızlığa uğramış gibi görünüyor.



*Şekil 1.2: Tilki, kaz ve mısır çuvalı, probleminin çözümü için gerekli olan ilk adım*

Kazın uzakta olması son derece iyi bir çözümdür. Ancak ikinci turda, çiftçi tilki ya da mısıri almalarıdır. Bununla birlikte çiftçi neyi alıyor olursa olsun geri kalanlar için yakın kıyıya dönerken kaz uzak kıyıda kalmalıdır. Bu, tilkinin ve kazın birlikte bırakılacağı ya da kaz ve mısır çuvalını birlikte bırakılacağı anlamına gelir. Bu durumlardan hiçbirini kabul edilemez çünkü bu durumda problem çözülemez.

Daha önce bu sorunu gördüğseniz muhtemelen çözümün kilit unsuru hatırlarsınız. Çiftçi daha önce açıklandığı gibi ilk turda kazı almak zorundadır. İkinci turda, çiftçinin tilkiyi aldığı varsayılmı. Bununla birlikte tilkiyi kaz ile bırakmak yerine çiftçi kazı yanına alarak yakın kıyıya geri götürür. Tilki uzak kıyıda yalnız kalır. Sonra çiftçi dördüncü turda tilkiyi yakın kıyıda yalnız bırakarak mısır çuvalını uzak kıyıya götürür. Çözüm süreci Şekil 1.3'te gösterilmektedir.

Pek çok kişi yakın veya uzak kıyıdan birini geri almayı düşünmez çünkü bu bulmaca zordur. Bazı insanlar, sorunun haksız olduğunu önererek “Geri alabileceğimi söylememiştiniz.” şeklinde tepki verebilir. Oysaki problem ifadesinde aksi yönde bir açıklama bulunmamaktadır.



1



2



3

Çözüm Adımı



4



5



6

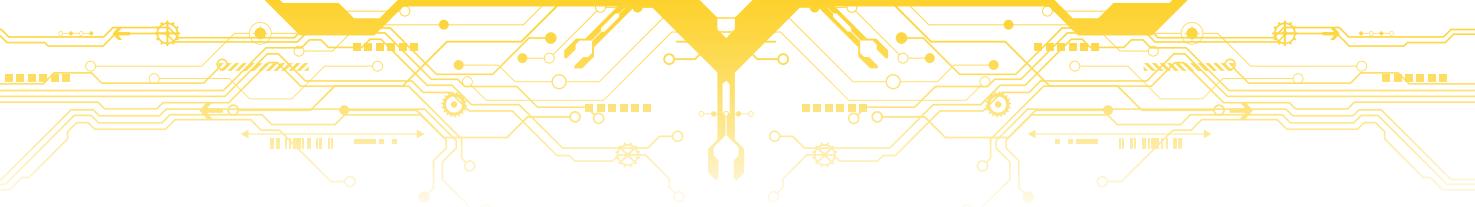


7



8

*Şekil 1.3: Tilki, kaz ve misir çuvalı bulmacasının adım adım çözümü*



Ögelerden birini yakın kıyıya geri götürme imkânı açıkça belirtilseydi bulmacanın ne kadar kolay çözüleceğini düşünün. O zaman problemin çözümüne ulaşmak için düşünmemize gerek bile kalma-yacaktır. Bu yüzden farklı biçimde düşününen ve farklı çözüm yolları üretelebilen kişiler bu problemi çözecektir. Bu, problem çözmede önemli bir ilkeyi göstermektedir: **Yapabileceğiniz olası tüm hareketleri öngöremezseniz sorunu çözemezsınız**. Diğer yandan olası tüm işlemleri numaralandırarak, çalışan birleşimini bulana kadar farklı işlemleri deneyerek birçok problemi çözebilirsiniz. Genel olarak bir sorunu daha biçimsel adımlarla yeniden düzenleyerek elinizden kaçırığınız çözümleri ortaya çıkarabilirsiniz.

Artık çözümü bildiğimize göre problemi daha biçimsel ifade etmeyi deneyebiliriz. Öncelikle kısıtlamaları listeleyeceğiz. Buradaki anahtar kısıtlamalar şunlardır:

1. Çiftçi tekne içerisinde tek seferde kendisi dışında yalnızca bir nesne daha alabilir.
2. Tilki ve kaz aynı kıyıda yalnız bırakılamaz.
3. Kaz ve misir aynı kıyıda yalnız bırakılamaz.

Bu problem, kısıtlamaların önemini ortaya koyan iyi bir örnektir. Bu kısıtlamalardan herhangi birini kaldırırsak bulmaca kolay hâle gelir. İlk kısıtlamayı kaldırırsak üç öğeyi tek bir turda alabiliyoruz. Tekneye sadece iki öğe alsak bile tilki ve misir çuvalını alıp kaz için geri dönebiliriz. İkinci kısıtlamayı kaldırırsak (ancak diğer kısıtlamaları yerinde bırakırsak) önce kaz, sonra tilki ve sonunda misiri alabiliyoruz. Kaz ve misirda dikkatli olmamız gereklidir. Bu nedenle kısıtlamaların herhangi birini unutursak veya yok sayarsak problemi doğru bir şekilde çözemeyiz.

Sonra, işlemleri listeleyelim. Bu bulmacanın işlemlerini belirlemenin çeşitli yolları vardır. Yapabileceğimizi düşündüğümüz eylemlerin belirli bir listesini hazırlayabiliriz:

1. **İşlem:** Tilkiyi nehrin karşı tarafına taşıyın.
2. **İşlem:** Kazı nehrin karşı tarafına taşıyın.
3. **İşlem:** Misir çuvalını nehrin karşı tarafına taşıyın.

Ancak, sorunu biçimsel olarak yeniden ifade etme amacının çözüm için bir fikir edinmek olduğunu unutmayın. Eğer problemi çözemediyseniz ve diğer gizli işlemleri keşfedemişseniz eylem listesi oluşturanken de problemi çözemeyiz demektir. Bunun yerine işlemleri kapsamlı (biçimsel) yapmalı ve koşullu hâle getirmeliyiz.

1. **İşlem:** Tekneyi bir kıyıdan diğerine götürün.
2. **İşlem:** Teknede yalnızca çiftçi varsa (sadece bir nesne) tekneye kıyıdan bir nesne yükleyin.
3. **İşlem:** Teknede iki nesne varsa (çiftçi ve bir nesne daha) nesneyi kıyıya bırakın.

Bu ikinci işlem listesi; problem hakkında en biçimsel anlamda düşünmemizi, koşullara uygun biçimde problemi çözmemizi, kaz ile yakın kıyıya geri yolculuğu düşünmemizi sağlayacaktır. Olası tüm hareket dizilerini üretirsek her bir diziyi kısıtlamalardan birini ihlal ederek veya daha önce gördüğümüz bir yapılandırmaya ulaşarak bitirirsek sonunda Şekil 1.3'teki diziye ulaşıp bulmacayı çözeriz. Bulmacanın doğasında olan zorluk, kısıtlamaların ve işlemlerin düzgün olarak tekrar konması yoluyla engellenecektir. Böylece bu problemi 7 adımda çözdüğümüzü görürüz.

1. **İşlem:** Kazı nehrin karşı tarafına taşıyın.
2. **İşlem:** Tekneyi bir kıyıdan diğerine götürün.
3. **İşlem:** Tilkiyi nehrin karşı tarafına taşıyın.
4. **İşlem:** Kazı nehrin karşı tarafına taşıyın.
5. **İşlem:** Misir çuvalını nehrin karşı tarafına taşıyın.



**6. İşlem:** Tekneyi bir kıyıdan diğerine götürün.

**7. İşlem:** Kazı nehrin karşı tarafına taşıyın.

### Bu Problemden Neler Öğrendik?

Sorunu daha biçimsel bir şekilde yeniden ifade etmek, bir problemi anlamak için mükemmel bir tekniktir. Birçok programcı, diğer programcılar bir sorunu tartışmak için arar; sadece diğer programcılar yanıt olabileceğini düşünür fakat aynı zamanda problemi yüksek sesle ifade etmek genellikle yeni ve yararlı düşünceleri tetikler. Bir sorunun tekrar okunması, bu tartışmayı başka bir programcıya yaptırmak gibidir ancak her iki noktadan da destek alırsınız.

Daha kapsamlı ders ise sorunun düşünülmesi, çözümü düşünmek kadar üretken olabileceğim gibi bazı durumlarda daha üretken olabileceğidir. Çoğu durumda, çözüme doğru yaklaşım biçimini olabilir.

### 2.1.5.2. Sudoku

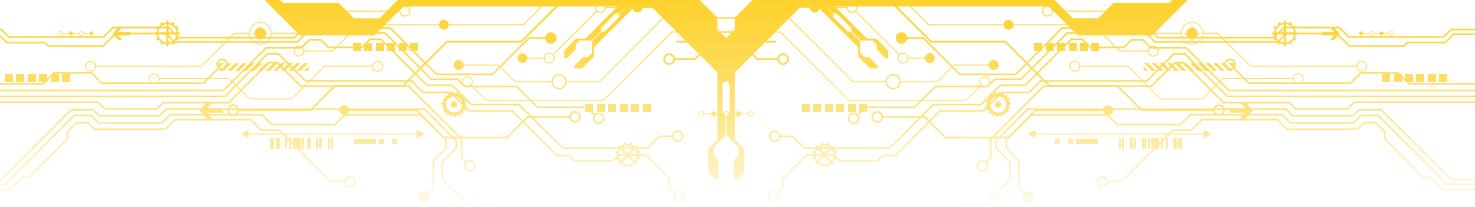
Sudoku oyunu gazete ve dergilerde yer almasıyla ayrıca web ve telefon tabanlı bir oyun olarak sunulmasıyla son derece popüler hâle gelmiştir. Farklı varyasyonlar olmakla birlikte burada kısa olan geleneksel sürümü tartışılacaktır.

9x9 boyutlu bir tablo kısmen tek basamaklı (1-9 arası) sayı ile doldurulur ve oyuncu belirli kısıtlamalara göre hareket ederken yalnızca boş kareleri doldurmmalıdır: Her bir satır ve sütunda, her rakam tam olarak bir kez yazılmalıdır ve her doldurulmuş 3x3 alanda her bir rakam tam olarak bir kez yer almalıdır. O zaman, verilen bir sudoku yapısındaki boşlukları 1-9 arasındaki her bir sayıyı; bulunduğu satır, sütun ve kare içinde yalnızca bir kez kullanılacak biçimde nasıl doldururuz?

Daha önce bu oyunu oynadıysanız muhtemelen en az zamanda bir kareyi tamamlamak için bir dizi stratejiniz vardır. Şekil 1.4'te gösterilen örnek kareye bakarak anahtar başlangıç stratejisine odaklanalım.

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4		6	8		
1	4		8	2	5			
			9			5		7
6	7		1	5				
		5		4		6	9	

*Şekil 1.4: Kolay bir sudoku kare bulmaca*



Sudoku bulmacanın zorluk derecesi doldurulması gereken karelerin sayısı ile belirlenir. Bu örnek çok kolay bir bulmacadır. 36 kare zaten doldurulmuş olduğundan, bulmacayı tamamlamak için doldurulması gereken sadece 45 kare bulunmaktadır. Soru şudur: Hangi kareleri ilk olarak doldurmaya çalışalım?

Bulmacanın kısıtlamalarını hatırlayın. Dokuz ana karenin her bir satır ve sütunda, her rakam tam olarak bir kez yazılmalıdır ve her doldurulmuş  $3 \times 3$  alanda her bir rakam tam olarak bir kez yer almazıdır. Bu kurallar, çabalarımıza nereden başlamamız gerektiğini belirtir. Bulmacanın ortasındaki  $3 \times 3$  alan, dokuz kareden sekizinde zaten sayıya sahip. Bu nedenle ortadaki kare, yalnızca  $3 \times 3$  alanında başka bir karedede temsil edilmeyen tek bir olası değeri içerebilir. İşte bulmacayı buradan çözmeye başlamalıyız. Bu alandaki eksik rakam 7'dir, bu nedenle orta kareye 7 rakamını yerlestireceğiz.

Bu değeri yerinde yazdığımızda, bu sütunun artık dokuz kareden yedisinde değerlere sahip olduğunu ve bunların her bir sütunda bulunmayan bir değeri olması gerektiğini ve yalnızca iki kareyi boş bıraktığını unutmayın: İki boş karenin değeri 3 ve 9'dur. Bu sütundaki kısıtlama her iki numarayı da her iki yere yazmamıza izin verir ancak 3'ün üçüncü satırda, 9'un zaten yedinci satırda bulunduğuuna dikkat edin. Bu nedenle satır kısıtlamaları, 9'un orta sütunun üçüncü satırına, 3'ün orta sütunun yedinci satırına yazılmasını gerektirir. Bu adımlar, Şekil 1.5'te özetlenmiştir.

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4		6	8		
1	4		8	2	5			
		9			5		7	
6	7		1	5				
		5		4	6	9		

1

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4	7	6	8		
1	4		8	2	5			
		9			5		7	
6	7		1	5				
		5		4	6	9		

2

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4	7	6	8		
1	4		8	2	5			
		9			5		7	
6	7		1	5				
		5		4	6	9		

3

*Şekil 1.5: Örnek sudoku bulmacanın çözümünde ilk adımlar*

Tüm bulmacayı burada çözmeyeceğiz ancak bu ilk adımlar, ideal olarak mümkün olan en düşük sayıdaki kareler için aradığımız sadece bir tane önemli noktayı görmemizi sağlar.

### Bu problemden neler öğrendik?

Sudoku'nun temel dersi, problemin en kısıtlı bölümüne baktırmamız gerektidir. Kısıtlamalar, çoğu zaman bir problemi zorlaştıran şeyken (Tilkiyi, kazı ve mısır cuvalını unutmuyın.) aynı zamanda çözüm hakkındaki düşüncemizi basitleştirebilir çünkü seçenekleri ortadan kaldırır.

Yapay zekâyı bu kitapta özel olarak tartışmayacak olsak da, yapay zekâda “en katı değişken” olarak adlandırılan belirli türdeki problemleri çözmek için bir kural vardır. Kısıtlamaları karşılamak için farklı değişkenlere farklı değerler atamaya çalıştığınız bir problemde, en fazla kısıtlamaları olan değişkenle başlamak ya da olası değerlerin en düşük sayısına sahip değişkeni başka bir şekilde koymak gerekir.

İşte bu tür düşünmeye bir örnek; bir grup iş arkadaşınızın sizinle beraber öğle yemeğine gitmek istedigini ve herkesin begeneceği bir restoran bulmanızı rica ettiğini varsayıyalım. Problem şu: Meslektaşlarınız her biri grup kararında birtakım kısıtlamalar getiriyor; Ayşe vejetaryen, Can deniz ürünlerini sevmiyor, Serhan'ın çoklu gıda alerjisi var vb. Hedefiniz bir restoran bulmak için gereken süreyi en aza indirmek



ise en sıkı kısıtlamalara sahip olan iş arkadaşınızla konuşarak başlayın. Örneğin Serhan'ın bir dizi geniş gıda alerjisi varsa Can'dan başlamak yerine, Serhan'ın yiyeceği yemeklerin olduğu bir restoran listesi bularak başlamak daha mantıklı olacaktır. Deniz ürünlerinden hoşlanmaması daha kolay çözülebilir.

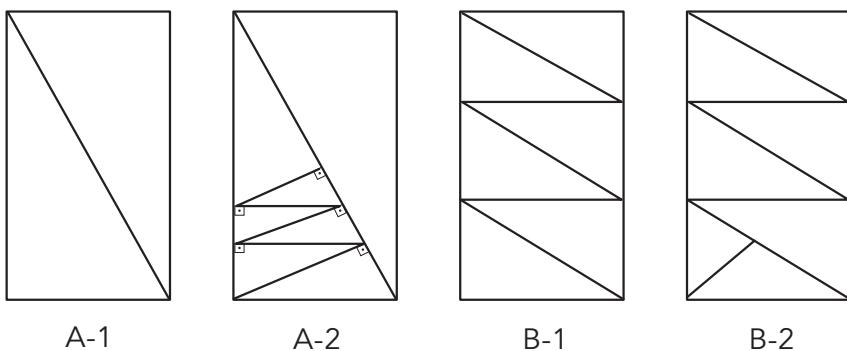
Aynı teknik, genellikle programlama problemlerine uygulanabilir. Sorunun bir kısmı aşırı derecede kısıtlıysa bu, başlamak için harika bir yer çünkü daha sonradan çözüleceğe iş yerinde boş vakit geçirginizi düşünmeden ilerleme kaydedebilirsiniz. Bununla ilgili bir sonuç, belirgin olan kısmı ile başlanması gerektidir. Sorunun bir bölümünü çözebilerseniz devam edin ve mümkün olanı yapın. Kendi kodunuza görmek, gerisini çözmek hayal gücünüzü artıracak ve bundan bir şeyle öğrenebileceksiniz.

### 2.1.5.3. Dikdörtgeni Parçalara Ayırma

Bir dikdörtgenden nasıl dik üçgenler oluşturabiliriz?. Buna göre, bir dikdörtgeni bölgerek, oluşturabilecek dik üçgenleri şekil çizerek gösteriniz.

Bir dikdörtgenin kısa kenarı 1 cm olduğunda uzun kenarının 1 cm'den büyük olması gerekmektedir. Uzun kenar 2 cm olduğunda A-1 şeklinde gösterilen dik üçgenler elde edilir. Oluşan dik üçgenlerden her birini de A-2 şeklinde dik üçgenlere ayırbiliriz.

İkinci çözüm yolu ise B-1 şeklindeki gibi dikdörtgen içerisinde birbirine paralel çizgiler oluşturmaktır. Oluşan küçük üçgenleri de B-2 şeklinde gösterilen biçimde dik üçgen olacak şekilde bölebiliriz.

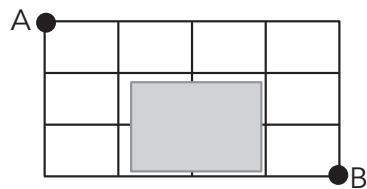


### Bu problemden neler öğrendik?

İlk çözümde yukarıda adım adım yapılan bir strateji kullanılmıştır. İkinci çözümde ise dikdörtgen farklı üçgenlere çevrilmiş ve elde edilen üçgenlerden tekrar dik üçgen olacak şekilde edilmişdir.

### 2.1.5.4. Engelli Yollar

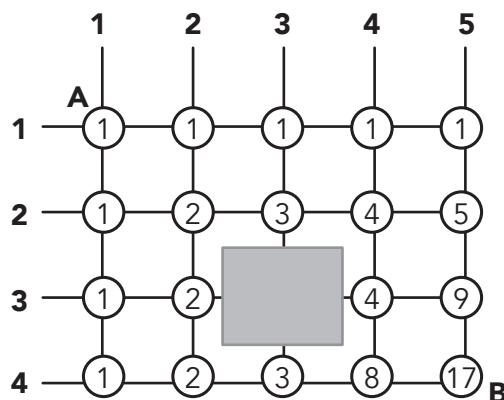
Şekil 1.6'da gösterilen A noktasından B noktasına gidebilmek için gri ile gösterilen alandan geçiş bulunmamaktadır. Buna göre A'dan B'ye gidebilmek için kaç farklı yol kullanılabilir?



Sekil 1.6



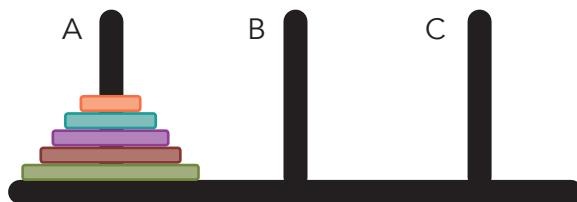
Doğru cevap 17'dir. Sorunun çözümünde satır ve sütunların kesişim noktalarından yararlanılmıştır. Şekilde görüldüğü gibi 1. satır 1. sütundan başlanarak satır satır işlem yapılır ve her kesişim noktasında soldaki ve üstteki değerler toplanır. Kesişim noktasının üstünde ya da solunda değer yoksa aynı sayı tekrar yazılır. Örneğin 1. satır 1. sütundaki değer 1'dir. 1. satır 2. sütuna geçildiğinde sadece sol tarafta 1 değeri olduğu için 1 yazılır. 2. satır 2. sütunda ise kesişim noktasının solunda ve üstünde 1 değeri vardır. Bu nedenle bu kesişim noktasının değeri  $1+1=2$  olur. En son 5. sütun 4. satıra gelindiğinde ise  $9+8=17$  olur.



### Bu problemden neler öğrendik?

Yol hesaplama, çok iyi bilinen dinamik programlama uygulamalarından biridir.

#### 2.1.5.5. Hanoi Kulesi

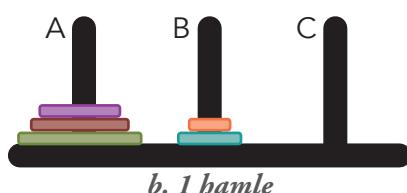
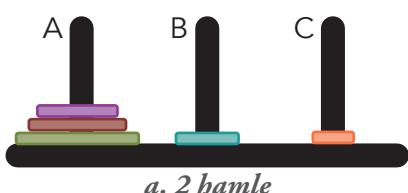


Yukarıdaki şekilde çeşitli renklerle gösterilen tahta parçaları, A çivisinden C çivisine aşağıdaki kurallara göre geçirilmek istenmektedir.

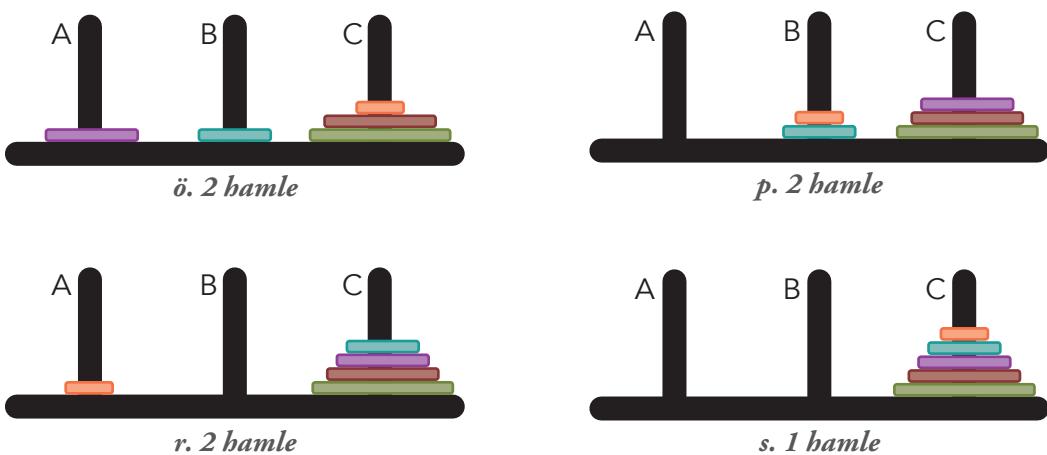
- Küçük tahtaların üzerine büyük tahtalar yerleştirilemez.
- Aynı anda sadece bir disk oynatılabilir.

Buna göre tahta parçalarını A'dan C'ye taşıyınız.

Bu soruda 5 tahta parçasının C çivisine taşınması gerekmektedir. Sorunun çözümü aşağıdaki gibidir.







### Bu problemden neler öğrendik?

Bu soru, sıralama için iyi bir örnektir.



#### Düşünelim/Deneyelim

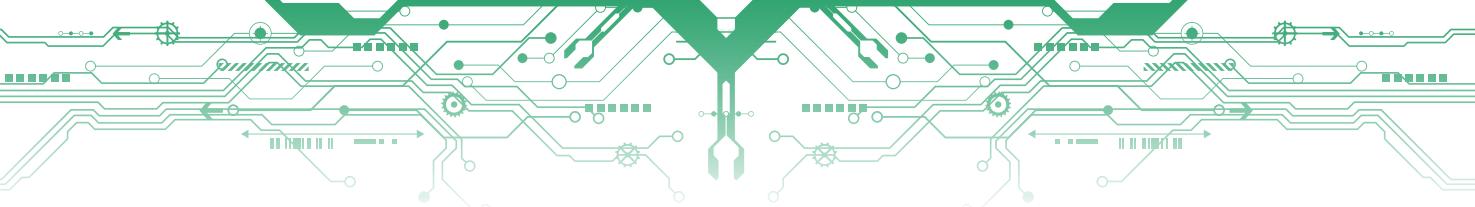
1. Siz de farklı çözümler ve bakış açıları gerektiren problemler bularak arkadaşlarınızla birlikte çözünüz.
2. Aşağıda İnternet adresi verilen Bilge Kunduz Etkinliği'ndeki soruları inceleyiniz.  
(<http://www.bilgekunduz.org/2016-gecmis-gorevler/>)

### 3. PROBLEM ÇÖZME SÜRECİ



Bu bölümde;

- ✓ Problemlerin keşifsel ve algoritmik çözümleri arasındaki farkları belirtebilecek,
- ✓ Algoritmik bir çözümü olan problemleri çözmek için gereken 6 problem çözme adımını açıklayıp listeleyebilecek,
- ✓ Bir problemi çözebilmek için 6 problem çözme adımını kullanacak,
- ✓ Değişken ve sabit arasındaki farkı algılayacak,
- ✓ Karakter, sayısal ve mantıksal veri türlerini anlayacak,
- ✓ Operatör, işlemci ve sonucu ayırt edecek,
- ✓ Fonksiyon tanımlayıp kullanacak,
- ✓ İlişkisel ve mantıksal operatörleri kullanacak,
- ✓ İşlem önceliğini kavrayacak,
- ✓ İfade ve eşitlikleri kullanarak işlem yapacaksınız.



## 3.1. Problem Çözme Teknikleri

İncelediğimiz problemleri çözerken çözüm sürecine destek olan bazı yaklaşımları kullandığımızı fark etmişinizdir. Programlama sürecinde de problemin çözümüne yönelik yol ve yaklaşımları belirlemek gerekir ama öncelikle genel kural ve teknikleri bilmek yararlıdır. Bazı genel kurallar neredeyse tüm problemler için kullanılabilir. Bu nedenle, bu kuralları içselleştirir ve düşünme sürecinizin bir parçası hâline getirebilirseniz herhangi bir problemi çözmeye çalışırken mutlaka bir fikriniz olur.

### 3.1.1. Her Zaman Bir Planınız Olsun

Belirsiz bir durumu yaşamak yerine her zaman bir planınız olmalıdır. Bu, en önemli kuraldır. Belki oluşturduğunuz çözüm planı ilk denemelerde sonuç vermeyecek ama her seferinde sizi çözüme biraz daha yaklaştıracak ipuçları elde etmenizi sağlayacaktır. Denediğiniz yaklaşım ne olursa olsun (doğru ya da yanlış), fikir üretmemekten ve deneme yapmamaktan her zaman daha iyidir. Planlama yapmak aynı zamanda hedef belirlemek ve bu hedefe ulaşmak anlamına gelir. Planınız olmazsa tek bir hedefiniz olur; o da problemi çözmektir. Bu durumda problemi çözene kadar herhangi bir başarı kaydetmiş olmazsınız. Dahası sadece nihai hedefe ulaşmaya çalışmak, her başarısızlıkta moralin bozulmasına neden olabilir. Ama küçük hedeflerden bile oluşan bir plan yaparsanız çözüme yönelik adımlar attığınızı ve zamanı etkili biçimde kullandığınızı göreceksiniz. Her bir adımda kendinize güveniniz artacak ve çözüme bir adım daha yaklaşmış olacaksınız.

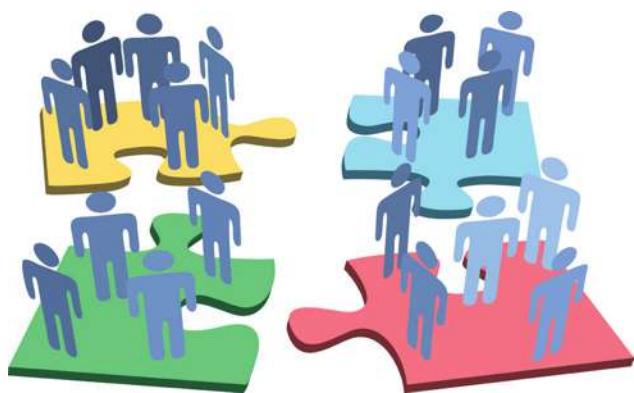


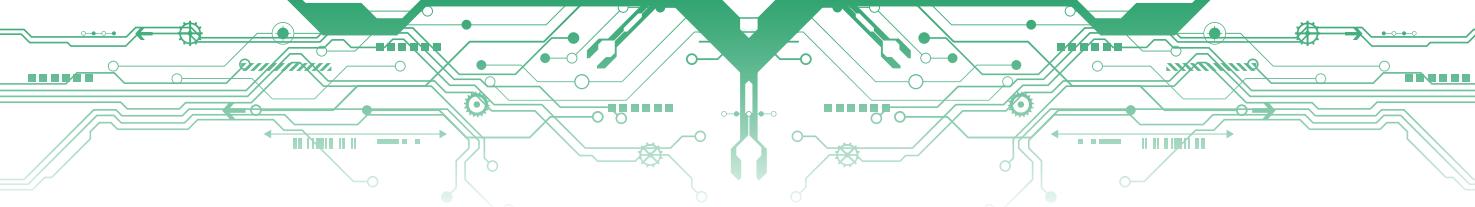
### 3.1.2. Problemi Tekrar İfade Edin

Önceki problemlerde de gördüğümüz üzere bazen problemi tekrar ifade etmek, göremedigimiz bir ayrıntıyı görmemizi ya da problemi daha kolay çözmek adına bir ipucu yakalamamızı sağlayabilir. Hatta bazen probleme ilişkin bir yanlış anlamın ortaya çıkmasına ya da hedefin daha iyi anlaşılmasına neden olur. Problemi farklı biçimlerde sunmak çözüm sürecine ışık tutmasa bile bazen yalnızca problemi doğru anlayıp anlamadığımızı bile teyit etmek açısından önemlidir. Ayrıca tekrar ifade ederek problemi küçük alt parçalara ayırmak gibi yaygın işlemleri de kolaylaştırmış olabiliriz.

### 3.1.3. Problemi Küçük Parçalara Ayırın

Verilen problemi adımlara ya da bölgelere ayırmak, çözümü kolaylaştırır. Bir problemi iki bölüme ayırdığımız düşünüldüğünde, her bir parçanın çözümünün tümünü çözmeye göre yarı yarıya kolaylaştığını düşünebiliriz. Bu durum için sıralama örneğini ele alalım. Elinizde 100 kişisel dosya olduğunu ve bu dosyaların alfabetik sıraya göre dizilmesi gerektiğini düşünelim. Önce bir dosya alıp sonra her aldiğinizi doğru yere yerleştirerek 100 adımda bu işlemi bitirebilirsiniz. Peki ya biri





sizin için 100 dosyayı A-F, G-M, N-Ş ve T-Z olacak biçimde 25 adetten oluşan 4 gruba ayırsayı işlemi daha kolay yapmaz mıydık? Her seferinde tek tek harf sırasını kontrol etme işlemi, dosya sayısı çoğalınca zorlaştığından sayıca küçük bir grupla çalışmak işlemi hızlandıracaktır. Bu nedenle bir problemi çözülebilir küçük parçalara bölmek, çözüm işlemini kolaylaştırır ve hızlandırır.

### 3.1.4. Önce Bildiklerinizden Yola Çıkın

Programlama yaparken öncelikle bildiklerimiz ile başlamalı ve sonra yeni çözümler arayışına girmeliyiz. Problemi küçük parçalara bölgerek çözübildiğiniz parçadan başlayınız. Bu parçaları çözerken diğer parçalarla ilgili olarak aklınıza yeni fikirler geldiğini ve aynı zamanda kendinize olan güvenin arttığını göreceksiniz. Programlama süreci boyunca çoğu zaman çok iyi olduğunuz konular, zorlandığınız konular ve henüz öğrenmediğiniz konular olacaktır. Bir problemin mevcut becerilerinizle çözülmüş çözülemeyeceğine karar vermek için problemi çok iyi incelemeniz gereklidir. Böylece bilgi dağarcığınızda olmayan ama çözüm için gerekli olan işlemlerin farkına varabilirsiniz.

### 3.1.5. Problemi Basitleştirin

Çözmekte zorlandığınız bir probleme karşılaşırsanız problemin kapsamını daraltmayı deneyebilirsiniz. Bunun için koşulları azaltmayı ya da çözüibileceğiniz biçimde dönüştürmeyi, değişkenleri azaltmayı ya da problemin kapsama alanını küçültmeyi düşünebilirsiniz. Temel amacınız problemi basitçe ifade etmeye çalışmak olmalıdır. Çözüm için denediğiniz yaklaşımlar, size gerçek çözüm için yol gösterecektir. Problemi basitleştirmek size aslında problemdeki zorluğun neden kaynaklandığını da gösterecektir.



### 3.1.6. Benzerlikleri Arayın

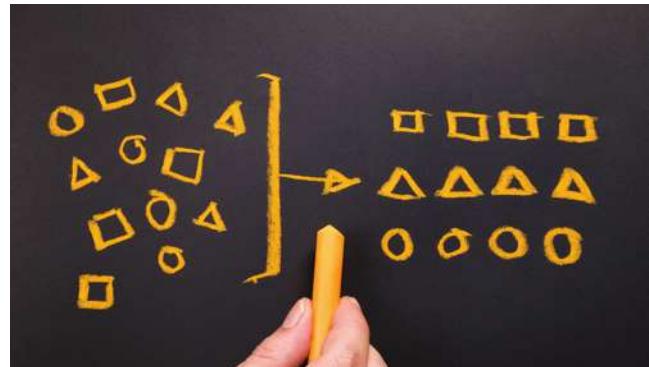
Burada ele aldığımız benzerlik kavramı, çözülmesi istenen probleme önceden çözülen problem arasındaki olası örtüşme ya da yeni çözüme ilham verme olarak tanımlanabilir. Benzerlik, farklı biçimlerde karşımıza çıkabilir. Bazen problemler aynı, değişkenler ya da veriler farklıdır. Bazen problemin belirli bir bölümü başka bir probleme benzerlik gösterebilir. Problem çözme sürecinde hızınızı ve becerinizi artıracak en önemli yaklaşım, benzerliklerin farkına varmaktır ancak bu, aynı zamanda kazanılması en zor beceridir. Buradaki zorluk, benzerliklerin farkına varacak kadar problem çözmek ve deneyim kazanmak gereğinden ortaya çıkmaktadır. Bu nedenle programlamaya yeni başlayanlar için bir problemi çözerken hazır yazılmış bir kodu bulmak ve onu güncelleyerek problemi çözmeye çalışmak son derece



yanlıştır. Bir çözümü kendiniz üretmezseniz tamamen anlayamaz ve içselleştiremezsiniz. Ayrıca çalışan her bir program, problemin çözümü olduğu kadar, daha sonraki problemleri çözmek için kazandığınız deneyimlerdir. Şu anda ne kadar yazılmış kodlar kullanarak problemleri çözerseniz gelecekte de sürekli bu yaklaşımı kullanacaksınız demektir.

### 3.1.7. Deneme Yapın

Bazen bir problemi çözmenin en kolay yolu denemek ve sonuçlarını gözlemlemektir. Bu, tahmin etmekten çok farklıdır. Bir çözümü tahminen öngörmek ile kodu yazarak denemek ve sonuçlarını incelemek çok farklı sonuçlar verir. Böylece problemi çözebilmek için gereken ipuçlarını elde edebilirsiniz. Denemek, ara yüz tasarlarken, çizim yaparken ya da kütüphaneleri kullanırken yararlı bir yaklaşımdır. Diğer yandan hata ayıklama süreci de bir tür deneysel yaklaşımındır. Bir problemin çıktılarına bakarak sorunun nereden kaynaklandığını anlayabilir ve problemi çözebiliriz. Bu nedenle programda önemli noktalara gözlemek amacıyla değer verebiliriz. Değerlerin değişimini gözleyerek çözüme daha hızlı ulaşabiliyoruz.



### 3.1.8. Asla Vazgeçmeyin

Asla vazgeçmemek, kişisel bir özelliklektir. Kararlılık, güven ve istek kaybolduğu zaman açık düşünemezsiniz, işlemler olması gerekiğinden uzun süreler ve gittikçe zorlaşır. Hatta öfke ve kızgınlığa bile dönenüsebilir. Ekrandaki program kodu çalışmada olduğu zaman programcı koda değil, kendisine ve aslında problemin kaynağına, yani kendi aklına kızmaktadır. Bu noktada moralimizin bozulmasına izin vererek aslında başarısız olmak için bir bahane üretmiş oluruz. Bu duygudan kurtulmak, programcı tarafından verilecek bir karardır. Böyle durumlarda en etkili çözüm ara vermektedir. Problemden tamamen uzaklaşarak geçirilecek vakit sonrasında çok daha verimli çalışmak olasıdır.



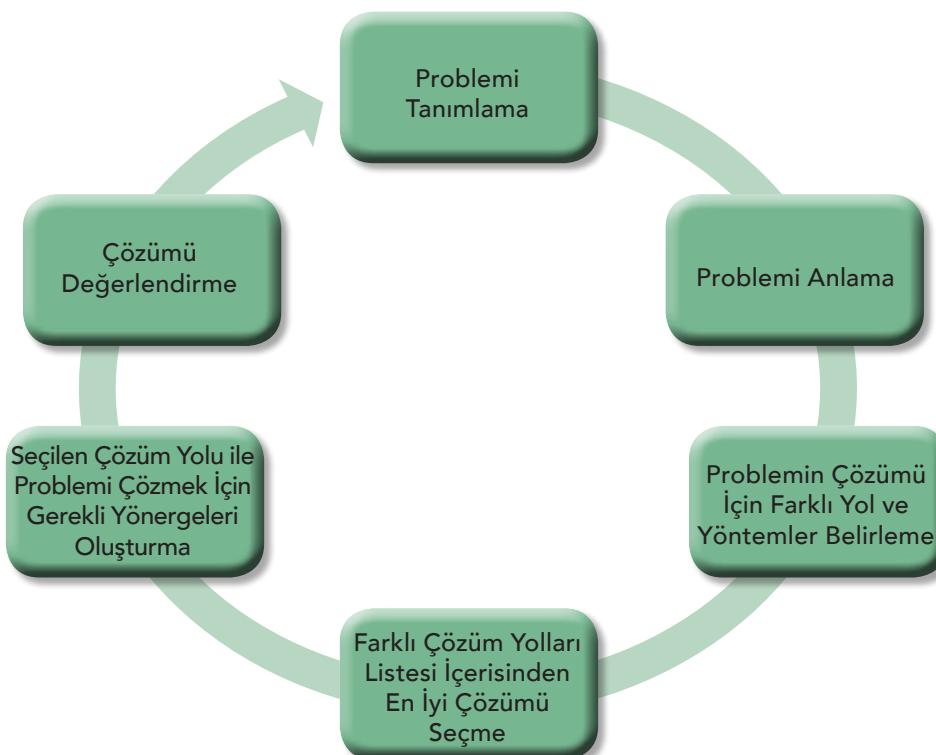
## 3.2. Problem Çözme Adımları

Problem çözme sürecinde en iyi kararı verebilmek için izlenmesi gereken 6 adım vardır (Şekil 1.7):

- 1. Problemi Tanımlama:** Problemi çözmeye başlamadan önce problemin açık, anlaşılır ve çok doğru bir şekilde tanımlanmış olması gereklidir. Problemin ne olduğunu bilemeyecekseniz onu çözemezsiniz.
- 2. Problemi Anlama:** Çözüme doğru yol almadan önce problemi çok iyi anladığınızdan emin olmanız gereklidir. Problemin neler içerdigini ve kapsamını doğru anlamalısınız. Ayrıca problemi

çözmeniz gereken insan ya da sistemin bilgi tabanında neler olduğunu da çok iyi anlamalısınız. Mevcut bilgi tabanında olmayan herhangi bir kavram ya da yönergeyi problemin çözüm sürecinde kullanamazsınız. Bu konuda klasik ve önemli bir söz vardır: “**Problemi anlamak, problemi yarı yarıya çözmek demektir.**”

3. **Problemin Çözümü İçin Farklı Yol ve Yöntemler Belirleme:** Problemin çözümü için olabildiğince farklı yol ve yöntem belirlemeli ve bu listenin, tüm olasılıkları içerdiginden emin olmalısınız. Bunun için konu hakkında farklı kişilerin görüşlerini alabilirsiniz. Farklı çözümler kabul edilebilir olmalıdır. Problem çözmek için tek bir yol yoktur; pek çok yol vardır.
4. **Farklı Çözüm Yolları Listesi İçerisinden En İyi Çözümü Seçme:** Bu adımda her bir çözümün olumlu ve olumsuz yönlerini ortaya koymalısınız. Bu nedenle değerlendirme yapabilmek için ölçütler oluşturmalısınız. Bu ölçütler her bir çözüm yolunu değerlendirmek için size rehber olacaktır. Problem çözmek için tek bir yol yoktur; en iyi yol vardır.
5. **Seçilen Çözüm Yolu ile Problemi Çözmek İçin Gerekli Yönergeleri Oluşturma:** Bu adımda numaralandırılmış ve adım adım yönergeler oluşturmanız gereklidir. Bu yönergelerin ikinci adımda belirtilen bilgi tabanı kapsamında olmasına dikkat ediniz. Bu durum, özellikle bilgisayarlar ile çalışırken son derece kısıtlı davranışımıza neden olabilir.
6. **Çözümü Değerlendirme:** Çözümü test etmek ya da değerlendirmek, sonucun doğruluğunu kontrol etmek anlamına gelir. Sonucun doğru olması ve problemi olan bireyin bekłentilerini karşılama düzeyi önemlidir. Sonuç yanlış çıkmış ya da bireyin bekłentilerini karşılamamış ise problem çözmeye sürecine baştan başlamak gereklidir.



*Sekil 1.7: Problem çözme adımları*

Problem çözme sürecinde bu 6 adım tam olarak uygulanmaz ise sonuç beklenmediği gibi olmayıabilir.



Bireyler olarak evde, işte ya da farklı ortamlarda sürekli problem çözeriz. Evde karşılaşılan problemler; akşam yemeği için ne pişirileceği, yemekten sonraki boş vaktin nasıl değerlendirileceği, hangi kitabın okunulacağı ve marketten nelerin alınacağı gibi konulardır. İş yerinde ise problemler; yönetim, iş yeri kuralları ve takımların verimli çalışması gibi konuları içerebilir. Ne kadar doğru kararlar alırsanız o kadar mutlu ve değerli yaştılar geçirirsınız. Çoğu kişi bu süreçlerde problem çözügünün bile farkına varmaz.

### 3.3. Problem Türleri

Problemlerin her zaman sıradan çözümleri olmaz. Kek yapmak ya da araba kullanmak gibi problemleri çözmek için bir dizi eylem gereklidir. Adım adım yönergelere dayalı olan bu çözümlere “**algoritmik çözümler**” denir. En iyi yolu seçtikten sonra sonuca, ilgili adımları izleyerek ulaşılır. Bu adımlardan oluşan yapıya “**algoritma**” denir. En lezzetli ekmeği seçmek ya da işleri büyütmek için yatırım yapmak gibi problemlerin ise açık ve net ifade edilen yanıtları yoktur. Bu çözümler bilgi ve deneyim gerektirir, bir dizi deneme ve yanlışlık sürecinden oluşur. Doğrudan işlem adımları ile ulaşamayan sonuçlara “**keşfe dayalı çözümler**” denir.

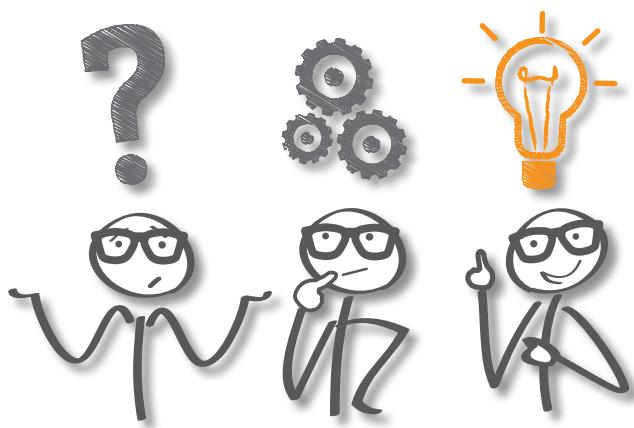
Problemi çözen kişi her iki türdeki problem için problem çözmenin 6 adımıını kullanabilir. Ancak keşifsel çözümler için son adım çok doğru sonuç vermeyecek. Bazı problemler ise her iki türdeki çözüm de kullanılmasını gerektirir.

### 3.4. Bilgisayarlar ile Problem Çözme

Bu ders kapsamında “çözüm” demek problem çözme sürecinin 5. adımda yer alan işlem adımları ya da yönergeler anlamına gelmektedir. “Sonuç” demek, çıktı ya da tamamlanmış bilgisayar destekli yanıt demektir. “Program” ise herhangi bir bilgisayar dilinde kodlanmış, çözümü oluşturan işlem adımlarının tamamını ifade etmektedir.

Bilgisayarlar, zor ve zaman alıcı olabilen algoritmik çözümler ile ilgilenmek üzere tasarlanmıştır. İnsanlar, keşifsel çözümleri bulma konusunda daha iyidirler ancak bilgisayarların çözebildiği ileri düzey hesaplama ve karmaşık problemleri çözme konusunda bilgisayarların hızlarına ulaşamazlar. Bilgisayarlar, üst düzey matematik problemlerini kolayca çözebilir ancak konuşmak ya da top atmak gibi davranışları yapamaz. Bu işlemlerin zorluğu programlama sürecindedir. Bu tür işlemleri bilgisayarların anlayacağı bir dizi adım şeklinde nasıl dönüştürebiliriz?

Keşifsel problem türleri ile ilgilenen bilgisayar dalına “yapay zekâ” adı verilmektedir. Yapay zekâ uygulamaları, bilgisayarlarla mevcut bilgileri kullanarak yeni bilgiler inşa etmesini sağlamaktadır. Böylece bilgisayarın problem çözme becerileri insanların yeteneklerine daha çok benzemektedir. Yapay zekâ özellikle robotik uygulamaları ile son yıllarda popüler bir çalışma alanı olmuştur. Bilgisayarlar insanlar gibi düşünmeye başlayana kadar daha çok algoritmik problemlerin çözüm süreçlerinde kullanılacaktır. Bu nedenle bu derste ağırlıklı olarak algoritmik çözümler üzerinde durulacaktır.



### 3.5. Problem Çözme Kavramları

Günlük hayatta karşılaştığımız problemler çok çeşitli olmasına rağmen bilgisayar ile çözebildiğimiz yalnızca 3 tür vardır:

1. Hesaplamlı–matematiksel işlem ve süreçler içeren problemler,
2. Mantıksal–ilişkisel süreçler içeren problemler,
3. Tekrarlayan–matematiksel ya da mantıksal bir dizi işlemin yinelenme sürecini içeren problemler.

Bu bölümde, bilgisayara ilişkin temel kavramlar ve belirtilen türdeki problemleri çözmek için kullanılan ifade ve eşitlikler anlatılmaktadır. “Sabit” ve “değişken” önemli iki kavramdır. Programcı işlenmemiş hâlde veriyi alır, işlenmiş hâle yani bilgiye dönüştürür. Bunlar eşitlik ve ifadelerin yapı taşlarıdır. Programcı, problemi çözebilmek için gereken sabit ve değişkenleri, uygun “veri türü”nde, örneğin sayısal olarak tanımlar.

Diğer önemli kavramlar ise operatör ve fonksiyonlardır. “Operatör”, sabit ve değişkenler arasındaki ilişkileri gösteren, eşitlik ve ifadelerde kullanılan işaret ve sembollerini ifade eder. Operatörlerin belirli bir hiyerarsık yapı içerisinde kullanılması gereklidir. Operatörler sabit ve değişkenlerle birlikte kullanıldığında “eşitlik” ve “ifade” olarak adlandırılan yapılar oluşur. Eşitlik ve ifadeler ise çözüm sürecinin yapı taşları olan işlemlerdir. “Fonksiyonlar” bir dizi işlem seti olarak tanımlanabilir. Tüm bu kavramları anlamadan bilgisayarlar ile problem çözmek olası değildir.

### 3.6. Veri Türleri

Çevremizdeki kavram ve nesneleri farklı şekillerde anlamlandırmak için farklı veri türleri kullanırız. Çözümler üretebilmek için bilgisayarlar “veri”ye gerekşim duyar. Ham veriler, bilgisayar tarafından “girdi” olarak algılanır ve program aracılığı ile işlenir. Kullanıcıya geri dönen değer, işlenmiş veridir; “çıktı” ya da

“bilgi” olarak adlandırılır. Bilgisayara hangi veri türüyle çalışıyor olduğu mutlaka belirtilmelidir. Bir programda farklı veri türleriyle işlem yapılabilir. Örneğin tam sayılar, kesirli sayılar, karakterler, simgeler, metinler ve mantıksal değerler, veri türlerini oluşturur.



#### 3.6.1. Sayısal Veri

Sayısal veriler tüm sayı tiplerini içerir. Sayısal veri, hesaplama işlemlerinde kullanılabilen tek veri türüdür. Pozitif ya da negatif tam sayılar ve reel sayılar kullanılabilir. Sayısal veriler; açılar, uzaklık, nüfus, ücret, yarıçap gibi hesaplama sürecinde gerekli değerler için tanımlanır. Banka hesap numarası ya da posta kodu gibi sayısal ama hesaplama için kullanılmayan veriler de vardır. Bu tür veriler sayısal olarak tanımlanmaz. Her bir veri türünün bir veri seti vardır. Sayısal veri için tanımlanmış veri seti 0-9 arasındaki sayılar ve “+” ile “-” işaretlerini kapsar. Örneğin 66578 ve -2356 tam sayı örnekleridir. Reel sayılar, tüm reel ve ondalık sayıları kapsar. Örneğin -56.23, 8695.235 ya da 0.005 reel sayı için örneklerdir. Sayıların alabileceği en küçük ve en büyük değerler kullanılan bilgisayar ve programlama diline göre değişimlidir.

Veri Türü	Veri Seti	Örnek
<b>Sayısal: Tam sayı</b>	Tüm sayılar	66578 -2356
<b>Sayısal: Reel sayı</b>	Tüm reel sayılar ve ondalık sayılar	-56.23 8695.235 0.005

### 3.6.2. Alfanümerik/Karakter Veri

Karakter veri seti; tüm tek haneli sayılar ("0".."9"), harfler ("a".."z", "A".."Z") ve özel karakterleri ("#", "&", "\*", ..) kapsar. Bu veri setinden oluşturulan değer, tırnak içinde belirtilir. Büyük ve küçük harf duyarlıdır yani "a" ile "A" farklı algılanır. ASCII (American Standard Code for Information Interchange) olarak adlandırılan karakter seti 256 karakterden oluşur. Karakterler sadece sayıdan oluşsa bile hesaplama işlemlerinde kullanılamaz. Birden fazla karakter bir araya getirilirse bilgisayar, bu yapıyı "dizi" olarak adlandırır. Karakter ve dizi verileri karşılaştırılabilir ve alfabetik sıraya göre sıralanabilir. Bilgisayar her karaktere bir numara verir ve işlemi bu şekilde gerçekleştirir çünkü bilgisayarlar sayısal işlem yapabilen cihazlardır. Veriler birbirleri ile karşılaştırılır ve azalan ya da artan şekilde sıralanır. Örneğin Muz ile Elma karşılaşıldığında M harfi E harfinden daha büyük bir sayıya sahip olduğu için Muz dizisinin değeri daha büyüktür. Elif ile Esra karşılaşıldığında ise Esra daha büyük değer alır çünkü s harfi l harfinden daha sonra gelir. Büyük harflerin küçük harflerden daha düşük sayısal değerleri vardır.

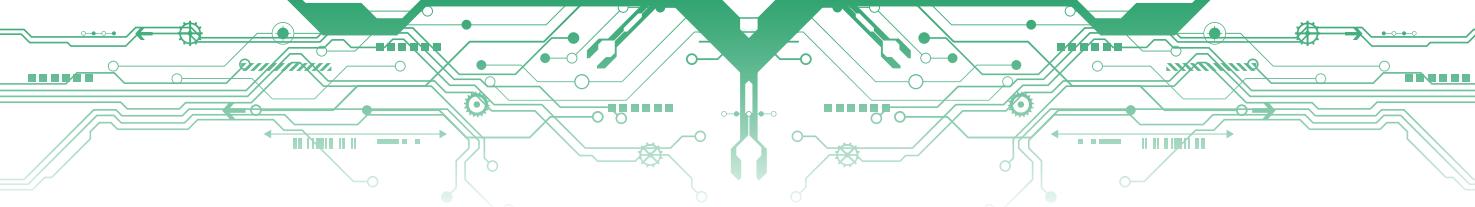
Veri Türü	Veri Seti	Örnek
Karakter	Tüm rakamlar, harfler ve özel semboller	"A", "Y", "k", "ı", "6", "0", "+", "%"
Dizi	Birden fazla karakterden oluşan kombinasyon	"Bilgisayar", "532-5556633"

Karakterler ve diziler + operatörü kullanarak birbirine bağlanabilir. Birleştirme olarak adlandırılan bu işlem, iki karakter parçasını yan yana getirir. Örneğin "6"+"6" = "66" olur. Genellikle matematiksel işlem gerektirmeyen verilerin, dizi şeklinde tanımlanması önerilir.

### 3.6.3. Mantıksal Veri

Mantıksal veri, veri setinde yalnızca iki kelime barındırır: doğru ve yanlış. Bu veri evet ya da hayır şeklindeki karar verme süreçlerinde kullanılır. Örneğin elde edilen değer, beklenen değer mi, evli mi, arabası var mı, öğrenci lise mezunu mu gibi sonucu kesin doğru ya da yanlış olan durumlarda mantıksal veri tanımlaması yapılır. Bu kelimeler ayrılmış özel kelimelerdir ve dizi olarak algılanmaz.

Veri Türü	Veri Seti	Örnek
Mantıksal	Doğru Yanlış True False	Doğru Yanlış True False



## Veri Türleri İçin Kurallar

1. Tanımladığınız veri genellikle sayısal, karakter, dizi ya da mantıksal olmalıdır.
2. Programcı programlama sürecinde verinin adını ve türünü belirtir. Bilgisayar çalışmaya başladığında verinin adı ile türünü eşleştirir.
3. Veri türleri karışık kullanılamaz. Örneğin sayısal olarak tanımlanmış bir veri, dizi olarak algılanamaz. Bu durumda bilgisayar, beklediği veri türü ile karşılaşamaz ve hata verir.
4. Her bir veri türü kendisi için tanımlı veri setini kullanır.
5. Matematiksel işlemlerde kullanılacak tüm veriler sayısal olarak, diğerleri karakter ya da dizi olarak tanımlanmalıdır.
6. Programcılar kendi tanımladıkları veri türlerini de oluşturabilirler. Kullanıcı-tanımlı olarak adlandırılan bu veri türleri, bugünün tarihi, hedef, varılacak süre gibi hem dizi hem de sayısal veriler içeren yapılar oluşturulabilir.

## Örnekler

Veri	Veri Türü	Açıklama
Ürün satış bedeli: 49.99, 101.50	Sayısal: Reel	Bir ürünün satış bedeli hesaplama işlemlerinde kullanılır.
T.C. Kimlik No.: 10654876542	Karakter dizisi	Kimlik bilgileri hesaplama amaçlı kullanılmaz.
Ağırlık: 67	Sayısal: Tam sayı	Kilo cinsinden tam sayı olabilir ve hesaplamalarda kullanılır.
Şirket İsmi: ABC Firması	Karakter dizisi	Tamamen karakterlerden oluşur.
Kredi Onayı: Var, Yok	Mantıksal	Bu durumda onay ya vardır “Doğru” ya da yoktur “Yanlış”.
Posta Kodu: 06110, 34217	Karakter dizisi	Posta kodları işlem yapmak için kullanılmaz.
Tarih: 21042017	Karakter dizisi, Sayısal Tam sayı	İşlem yapmak için tam sayı biçiminde tanımlanabilir; aksi takdirde dizi olarak tanımlanması daha uygundur.
IBAN: TR0600006543000012	Karakter dizisi	Para transferi için bankaya verilen kodlar hesaplama amaçlı kullanılmaz.



### Düşünelim/Deneyelim

Aşağıda verilen değişkenlerin veri türlerini belirleyiniz.

- İsim
- Yaş
- Cinsiyet
- Vergi Numarası
- Okul Numarası
- Geçti/Kaldı

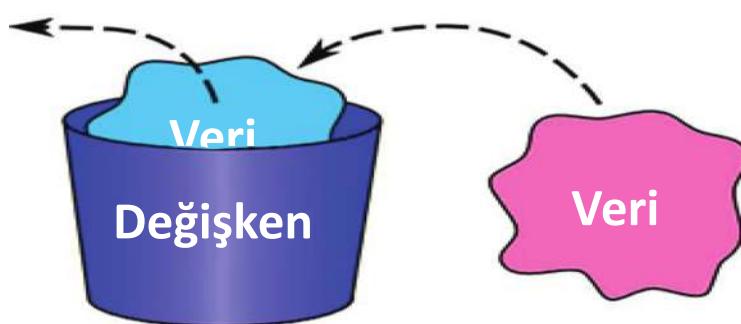


## 3.7. Bilgisayar Veriyi Nasıl Saklar?

Bilgisayar veriyi hafızada saklar. Her bir değişken için hafızada belirli bir alan ayrıılır ve bu alan her seferinde tek bir değer saklayabilir. Kullanıcı, var olan değer yerine yeni bir değer atadığında eski değer silinir. Hafızada bu konumlar geçicidir. Programın çalışması bittiğinde ya da bilgisayar kapatıldığında bu veriler silinir. Verilerin daha sonra tekrar kullanılması gerekiyorsa sabit disk gibi kalıcı bir konuma kaydedilmeleri gereklidir. Bu şekilde kaydedilen verilere "dosya" adı verilir. Temel anlamda program dosyaları ve veri dosyaları olmak üzere iki dosya türü vardır. Program dosyaları, bilgisayarın yapması istenen komutları ve işlemleri içerir. Veri dosyaları ise programlar çalışırken gereken verileri kapsar.

## 3.8. Sabit ve Değişkenler

Bilgisayarlar problemleri çözmek için süreç boyunca sabit ve değişken olarak adlandırılan verileri kullanır. "Sabit" olarak tanımlanan veriler problemin çözüm süreci boyunca asla değişimyen değerlerdir. Sabit değerler sayısal, karakter ya da özel semboller olabilir. Bu durumda bu değere bilgisayarın hafızasında bir yer ayrıılır ve bir isim verilir. Program çalıştığı sürece bu değer kendisine verilen isim ile çağrılrı ve değeri asla değiştirilemez. Örneğin, pi değeri değişimyen bir değer olacağı için sabit olarak tanımlanmalıdır.



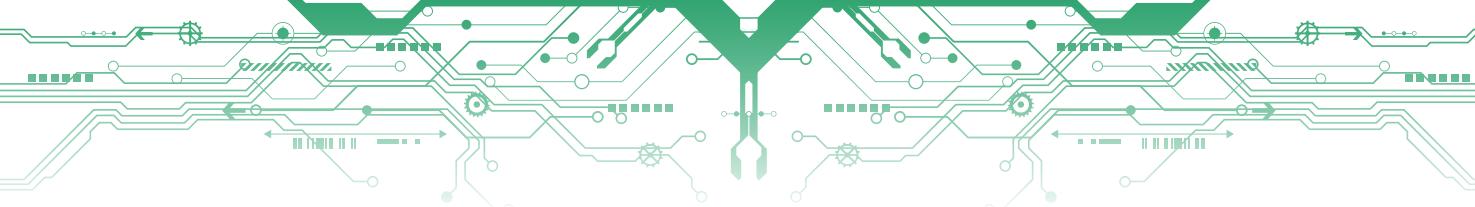
*Şekil 1.8: Değişken - Veri İlişkisi*

Bu durumun tam tersi şekilde bir "değişken" tanımlandığında değeri, program çalıştığı sürece değişimdir (Şekil 1.8). Değişkenlere taşıdığı değerleri ifade eden isimler verilir, bu şekilde belirleyici özellikleri de oluşur. Programcılar çözüm sürecinde ihtiyaç duyulan her bir değişkene ayrı bir isim vermelidir. Böylece bilgisayar bu ismi, ilgili değeri hafızada bulmak için kullanır. Değişken, farklı veri türlerinde olabilir ancak ismi, içerdığı değer ile tutarlı olmalıdır. Örneğin fiyat isimli bir değişkenin içerisinde 50 değeri atanmış olabilir, program çalıştığı süre içerisinde bu değer değişimdir ancak değişkenin ismi hiçbir zaman değişmez.

Değişkenlere isim verirken ve bunları kullanırken dikkat edilmesi gereken kurallar şunlardır:

1. Değişkene içerdiği değer ile tutarlı isimler veriniz.
2. Değişkenlere isim verirken boşluk kullanmayın.
3. Değişkenlere isim verirken bir karakter ile başlayınız.
4. Matematiksel semboller kullanmamaya dikkat ediniz.

Aşağıda "Doğru" ve "Yanlış" olarak kullanılmış değişken isimleri yer almaktadır:



Yanlış	Doğru
1 sayı	sayı1
Okul No.	okulNo
Soru?	soru

Değişken isimleri konusunda aşağıdaki noktalara dikkat edilmelidir.

- Bazı platformlar desteklemediği için Türkçe karakter kullanımı tavsiye edilmez.
- Programlama dillerinde kullanılan komut isimleri değişken olarak kullanılamaz. Çok bilinenleri; if, for, while, else, do, int, vb.
- Değişken isimlendirmelerinde boşluk karakteri yerine alt çizgi ( \_ ) karakteri kullanılabilir ancak değişken isimlendirmede genellikle küçük harfle başlanır ve ikinci bir kelime yazılacaksa ilk kelimenin hemen ardından büyük harfle devam edilir. Buna “Camel Karakter” kullanımı denir. Örnek: tcKimlikNo
- Özel karakterler değişken isimlerinde kullanılamaz (\*, /, -, +, #, %, &, (=, ?, \$, [, { gibi...}).



### Düşünelim/Deneyelim

1. Günlük hayattan Hesaplamalı, Mantıksal ve Tekrarlayan süreçlere örnekler veriniz.
2. Aşağıdaki problemlerin çözümü için hangi değişkenlere ihtiyaç duyulur?
  - a) Bir memurun maaşını hesaplayan programı yazmak,
  - b) Bir sinema salonu için bilet satış programı yazmak,
  - c) Bir yolculuk için rezervasyon programı yazmak.
3. Siz de doğru ve yanlış değişken isimlerine örnekler veriniz.

## 3.9. Fonksiyonlar

Fonksiyonlar, belirli işlemleri yürüten ve sonuçları döndüren bir işlem kümesidir. Genellikle bilgisayar dilinde oluşturulur. Fonksiyonlar, bir çözüm sürecinin belirli parçaları olarak kullanılır. Problem çözme sürecinde tekrarlanan işlemler için kullanılır ve böylece programcının, hem problemi daha hızlı çözmesci hem de programın daha anlaşıllır olmasını sağlar. Her programlama dili, içerisinde kendine özgü fonksiyonlar barındırır. Bu fonksiyonlar kütüphanesi, programlama dili bilgisayara göre değişiklik gösterir. Ayrıca pek çok programlama dili, programcının kendi fonksiyonlarını yazmalarına da olanak verir. Fonksiyon kütüphaneleri, pek çok program diline eklenebilir.

Fonksiyonlar, kendilerine verilen isim ve ayrıca içerisinde gönderilen veri ile tanımlanır.

### Fonksiyon İsmi (Veri)

Fonksiyon kapsamında elde edilen sonuç, fonksiyonun ismi ile döndürülür. Fonksiyonlara veri gönderilir. Fonksiyona gönderilen verilere **“parametre”** denir. Fonksiyonlar parametreleri değiştirmez ama işlemlerde kullanır. Örneğin karekök fonksiyonunu ele alalım. Sqrt(N), gönderilen N değeri için karekök değeri hesaplamaktadır. Sqrt fonksiyonun ismi, N işlem yapılacak veri yani parametredir. Parametrelere yanıt olarak içinde yazılır. Programcı olarak kullandığınız dilin kütüphanesinde hangi fonksiyonların olduğunu araştırmanız işlerinizi kolaylaştıracaktır. Fonksiyonlar gruplara ayrılır:

- 1. Matematiksel Fonksiyonlar:** Matematiksel işlemler için kullanılır.
- 2. Dizi Fonksiyonlar:** Dizi ve karakterlerle ilgili işlemleri gerçekleştirmek için kullanılır.
- 3. Dönüştürme Fonksiyonları:** Veriyi bir türden diğerine dönüştürmek için kullanılır.



**4. İstatistiksel Fonksiyonlar:** Maksimum değer, ortalama gibi değerleri hesaplamak için kullanılır.

**5. Yardımcı Fonksiyonlar:** Program dışındaki verilere erişerek işlem yapmak için kullanılır.

Bu fonksiyonlara bazı örnekler Tablo 1'de görülmektedir.

*Tablo 1: Fonksiyon türleri ve örnekler*

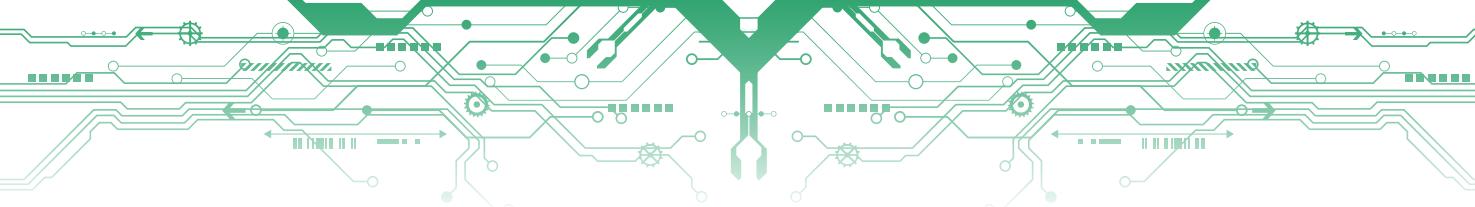
Fonksiyon	Tanım	Örnek	Sonuç
<b>Matematiksel Fonksiyonlar</b>			
Sqrt (N)	N değerinin karekökünü döndürür.	Sqrt(16)	4
Abs (N)	N değerinin mutlak değerini döndürür.	Abs(-6)	6
Integer (N)	N değerine en yakın ya da eşit tam sayıyı döndürür.	Integer(6.7689)	6
Random	0 ile 1 arasında rastgele bir sayı döndürür.	Random	0.6783456
<b>Dizi Fonksiyonları</b>			
Mid (S, n1, n2)	Dizinin n1 pozisyonundan başlayan n2 kadar karakteri döndürür.	Mid(S, 3, 3) S= "Yasemin"	"sem"
Left (S, n)	Dizinin sol tarafındaki n kadar karakteri döndürür.	Left(S, 3) S= "Yasemin"	"yas"
Right (S, n)	Dizinin sağ tarafındaki n kadar karakteri döndürür.	Right(S, 4) S= "Yasemin"	"emin"
Length (S)	Dizideki karakter sayısını döndürür.	Length(S) S= "Yasemin"	7
<b>Dönüştürme Fonksiyonları</b>			
Value (S)	Dizi olarak tanımlanan değişkeni sayısal değere çevirir.	Value("65.21")	+65.21
String (N)	Sayısal değeri dizi değerine çevirir.	String(+65.21)	"65.21"
<b>İstatistiksel Fonksiyonlar</b>			
Average (list)	Birkaç sayı için ortalama değeri döndürür.	Average(12, 24, 6)	14
Sum (list)	Birkaç sayının toplam değerini döndürür.	Sum(3, 5, 8)	16
<b>Yardımcı Fonksiyonlar</b>			
Date	Sistemin andaki tarih değerini döndürür.	Date	04/23/2017
Time	Sistemin şu andaki zaman değerini döndürür.	Time	20.57.36



### Düşünelim/Deneyelim

Aşağıdaki fonksiyon komutlarının çıktılarını yazınız.

Değişken	Fonksiyon	Çıktı
s=16	Sqrt (s)	
s= -64	Sqrt (Abs(s))	
bolum="Bilgisayar"	Left (bolum,5)	
bolum= "Bilgisayar"	Right (Mid(bolum,6,3),2)	
s=25 a=15 y=s+10	Sum (s,a,y)	



### 3.10. Operatörler

Bilgisayara, verileri nasıl işleyeceğini belirtmek gerekir. Bu işlem için operatörler kullanılır. "Operatörler" verileri, ifade ve eşitlikler ile birleştirir. Bu yazım, aynı zamanda operatörler bilgisayara ne tür bir işlem (matematiksel, mantıksal vb.) olduğuna dair bilgi verir. "İşlemci" ve "sonuç", operatörlere ilişkin iki kavramdır. İşlemci, verileri bağlayan ve işleme alan yapı; sonuç ise yapılan işlemin yanıdır. Örnek vermek gerekirse  $6 + 5$  ifadesinde yer alan "+" operatör, 6 ve 5 işlemci, 11 ise sonuçtur. İşlemciler sabit ya da değişken olabilir.

Operatörler; matematiksel, mantıksal ve ilişkisel operatörler olarak sınıflandırılabilir. Operatör türlerine ilişkin örnekler Tablo 2'de yer almaktadır.

Tablo 2: Operatör türleri ve örnekler

Operatör	Bilgisayar Sembolu	İşlem	Sonuç
<b>Matematiksel</b>			
Toplama	+	$6.7 + 2.1$	8.8
Çıkarma	-	$5.6 - 3.4$	2.2
Çarpma	*	$3.0 * 4.0$	12.0
Bölme	/	$40.0 / 8.0$	5
Modül Alma	MOD	9 MOD 3	3
<b>İlişkisel**</b>			
Eşit	==	$6 == 8$	False
Küçüktür	<	$6 < 8$	True
Büyükür	>	$6 > 8$	False
Küçük ya da eşittir	<=	$6 <= 8$	True
Büyük ya da eşittir	>=	$6 >= 8$	False
Eşit değildir	<>	$6 <> 8$	True
<b>Mantıksal</b>			
Değil	NOT	NOT True	False
Ve	AND	True AND True	True
Veya	OR	True OR False	True

\*\*İlişkisel operatörlerle yapılan işlemlerin sonucunda ortaya mantıksal değer olarak Doğru (True) ya da Yanlış (False) çıkar.

### 3.11. İşlem Önceliği

Matematiksel, mantıksal ve ilişkisel operatörlerin bir hiyerarşisi yani öncelikleri vardır. İşlemler, bu sıralamaya göre yapılmaz ise sonuç, bekleniği gibi çıkmayabilir. En içteki ayrıcaً en dıştakine doğru işlem yapılmalı, ayrıca içerisinde ise işlem önceliklerine dikkat edilmelidir. İşlem öncelikleri Tablo 3'te görülmektedir.

*Tablo 3: İşlem önceliği*

İşlem Sırası	Veri Türü	Sonuç Değeri Türü
() hiyerarşiyi sıralar, ayrıç içerisindeki işlemler en içten en dışa doğru yapılmalıdır.		
<b>Fonkiyonlar</b>		
<b>Matematiksel Operatörler</b>		
Kuvveti (Üs)	Sayısal	Sayısal
\, MOD	Sayısal	Sayısal
*, /	Sayısal	Sayısal
+, -	Sayısal	Sayısal
<b>İlişkisel Operatörler</b>		
=, <, >, <=, >=, <>	Sayısal, dizi ya da karakter	Mantıksal
<b>İlişkisel Operatörler</b>		
NOT	Mantıksal	Mantıksal
AND	Mantıksal	Mantıksal
OR	Mantıksal	Mantıksal

Örnek değişkenlere ilişkin işlem sonuçları için aşağıdaki tabloyu inceleyiniz.

Değişken	İşlem	Çıktı
$x=10$ $y=15$ $z=20$	$x+y-z$	5
	$x-y^*z$	-290
	$z \bmod y$	5
	$(x-y)^*5$	-25
	$x < y$	True
	$x < y \text{ AND } x < z$	True
	NOT ( $x > 0$ )	False
	$x < > y \text{ OR } z >= 20$	True



### Düşünelim/Deneyelim

Puanları 68, 80, 40 olan öğrencinin puan ortalamasını hesaplayınız.

## 3.12. İfade ve Eşitlikler

Şu ana kadar gördüğümüz tüm bileşenler, ifade ya da eşitlik biçiminde kullanılmadığı sürece bir anlam ifade etmez. Çözülmeye çalışılan problem vergi ya da maaş hesaplama, değerleri sıralama, en büyük değeri bulma gibi farklı işlemlerden oluşabilir. Bir “ifade” operatörleri kullanarak veriyi işler.

Uzunluk \* Genişlik

“Eşitlik” ise ifadenin sonucunu saklar.

Alan=Uzunluk \* Genişlik

Bu durumda uzunluk ve genişlik değerlerinin çarpım sonucu hafızada “alan” olarak ayrılan yerde korunur. İfadelerde eşit operatörü kullanılmaz. İfadeler eşitlik ve yönergelerin yalnızca bir bölümünü oluşturur. Bu yüzden sonuçlar o an kullanılır ancak korunmaz. Oysaki eşitlik ifadelerinde mutlaka sonuç korunur. Bu yüzden eşitliklere “**atama ifadeleri(ifadeler)**” de denir. Örnek ifade ve eşitlikler Tablo 4’te incelenebilir.

*Tablo 4: İfade ve eşitlikler*

İfadeler	Eşitlikler
<b>A + B</b> A ve B sayısal veridir. Sonuç sayısalıdır ve hafızada korunmaz.	<b>C = A + B</b> A, B ve C sayısal veridir. Sonuç sayısalıdır ve C değişkenine atanarak korunur.
<b>A &lt; B</b> A ve B sayısal, karakter ya da dizi olabilir. Sonuç mantıksal değerdir ve hafızada korunmaz.	<b>C = A &lt; B</b> A, B ve C sayısal, karakter ya da dizi olabilir. Sonuç mantıksal değerdir ve C değişkenine atanarak korunur.
<b>A OR B</b> A ve B mantıksal veridir. Sonuç mantıksaldır ve hafızada korunmaz.	<b>C = A OR B</b> A, B ve C mantıksal veridir. Sonuç mantıksaldır ve C değişkenine atanarak korunur.

## 4. PROBLEM ÇÖZME YAKLAŞIMLARI



Bu bölümde;

- ✓ Bir problem için çözüm üretirken destek olabilecek temel araçları listeleyebilecek,
- ✓ Problem analiz çizelgesini kullanarak problem verisini birleştirebilecek,
- ✓ Problemin çözümünü oluşturabilecek modüllerin belirlemek için etkileşim çizelgesini kullanabilecek,
- ✓ Bir problem için girdi, süreç, modül numarası ve çıktı belirlemek için GSC çizelgesini kullanabilecek,
- ✓ Problemin çözümü için gerekli algoritma ve akış şemalarını kullanabilecek,
- ✓ Dâhilî ve harici dokümantasyonun önemini açıklayabileceksiniz.

## 4.1. Bilgisayar ile Nasıl İletişim Kurulur?

Bilgisayarlar ancak donanımları, yazılımları ve onları kullanan kişiler kadar iyidir. Yapılacak işlem için mevcut donanımın yeterli olduğunu varsayırsak bir bilgisayarın etkililiği onu kullanan programcının yetkinliğine bağlıdır. Bilgisayara bir dizi işlem aracı ile ne yapması gerektiği söylemenmelidir. Bu işlemler bir programlama dili aracı ile kodlandığında bir program haline gelir.

Bilgisayarların çok iyi çözdüğü problemler, algoritmik yapıda olanlardır yani adım adım işlemlerden oluşan yapılardır. Bu işlemler bilgisayarın anlayabileceği biçimde ifade edilir ve bilgisayar bu satırları sıra ile çalıştırır.

**Bilgisayarlar bizim konuştuğumuz dili bilemediğinden onlarla anlaşmamız için bizim onların konuştuğu dili öğrenmemiz gereklidir. Bilgisayarın işletim sistemi, dili ve uygulamalarına ilişkin kurallara “*söz dizimi*” denir. Bir hata oluşursa buna “*yazılım hatası*”; hatayı bulup düzeneleme işlemine ise “*bata ayıklama*” denir.** Yazılım hataları bazen söz dizimi hatalarından bazen de mantık hatalarından kaynaklanabilir. Bu hatalar problem çözme sürecinde bulunarak düzelttilir. Programın hatasız çalışması ve doğru sonucu üretmesi için tüm hataların düzeltilmiş olması gereklidir.



## 4.2. Çözümün Düzenlenmesi

Problem çözme sürecini destekleyen bazı düzeneleme araçları vardır. “Bunlar;

1. Problem Analiz Çizelgesi,
2. Etkileşim Çizelgesi,
3. GSÇ (Girdi Süreç Çıktı) Çizelgesi,
4. Algoritmalar,
5. Akış Şemaları”dır.

Bu araçları kullanmak; çözüme daha hızlı ulaşmak, en etkili programı yazmak, anlaşılır olmak ve süreci kolaylaştırmak için önemlidir.

**Problem Analiz  
Çizelgesi**

**Etkileşim  
Çizelgesi**

**GSÇ Çizelgesi**

**Algoritmalar**

**Akış Şemaları**

#### 4.2.1. Problemin Analiz Çizelgesi

Çözümü düzenlemek için önce programın bekleyicilerini analiz etmek gerekir. Bunun için en iyi yol, problemi dört aşamada ele almaktır:

1. Eldeki veri
2. Beklenen sonuç
3. Problemin çözüm süreci
4. Çözüm seçenekleri

*Tablo 5: Problem Analiz Çizelgesi*

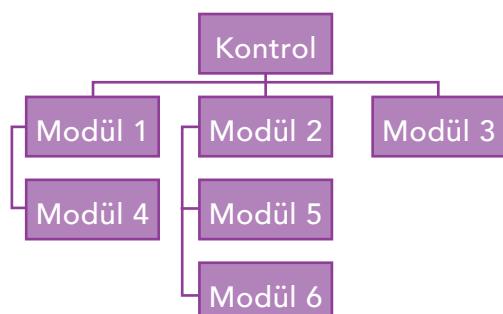
Eldeki Veri	Beklenen Sonuç
Problemde verilen ya da kullanıcı tarafından sağlanan veri	Sonuç için bekleyiciler, hangi bilginin nasıl biçimde sunulacağı
Problemin Çözüm Süreci	Çözüm Seçenekleri
İfade ve eşitlikler listesi, sıralama, arama, hesaplama vb.	Problemi çözebilmek için olası fikirler

Bir örnek problem için problem analiz çizelgesinin nasıl olduğuna bir göz atalım: sınav ve performans puanlarına göre ortalama hesaplama ve geçme kalma durumunun kontrolü:

Eldeki Veri	Beklenen Sonuç
2 Yazılı ve 2 Performans Puanı	Geçme/Kalma Durumu
Problemin Çözüm Süreci	Çözüm Seçenekleri
-Ortalama = $(\text{Yazılı 1} + \text{Yazılı 2} + \text{Performans 1} + \text{Performans 2})/4$ -Geçme/Kalma Durumu= Eğer ortalama 50'den küçükse "Kaldı", değilse "Geçti"	Yazılı ve performans puanlarını girilecek değerler olarak tanımlama

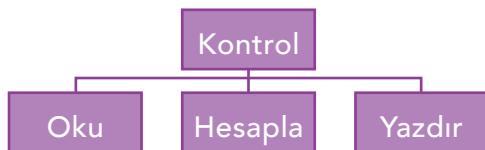
#### 4.2.2. Etkileşim Çizelgesi Geliştirme

Çözüme ulaşma yolunda ikinci adım, çözüm sürecini modüllere ayırmak ve süreçteki modüllerin birbiri ile etkileşimiğini görmek için modülleri birleştirmektir. Yönetsel etkileşim çizelgesi hazırlanırken yukarıdan aşağıya yaklaşım kullanılır. Tüm modüllerin kontrol eden bir ana kontrol mekanizması dâhilinde süreç yukarıdan aşağıya doğru işler.



*Şekil 1.9: Etkileşim çizelgesi*

Çoğu programda kontrol modülünden sonra ilk değerler ataması, sonra okuma, hesaplama, ekrana yazdırma, çıktı alma vb. gibi işlemler gerçekleştirilir. Kontrol modülü bu süreci kontrol eder. Örneğin brüt maaş hesaplama problemi için etkileşim çizelgesi aşağıdaki gibi olabilir.



### 4.2.3. GSÇ Çizelgesi

GSÇ (girdi-süreç-çıktı) çizelgesi problem analiz çizelgesindeki bilgiyi detaylandırır ve düzenler. GSÇ çizelgesi dört bölümden oluşur: girdi, süreç, modül referansı ve çıktı (Tablo 6).

*Tablo 6: GSÇ Çizelgesi*

Girdi	Süreç	Modül Referansı	Çıktı
Program için gerekli tüm veriler	Adım adım işlemler (Problem Analiz Çizelgesindeki 3 ve 4. Adımlar)	Etkileşim çizelgesindeki modüller	Tüm çıktı beklentileri (Problem Analiz Çizelgesindeki 1 ve 2. Adımlar)

Geçme/Durumunu tespit etmek için GSÇ çizelgesi aşağıdaki gibidir.

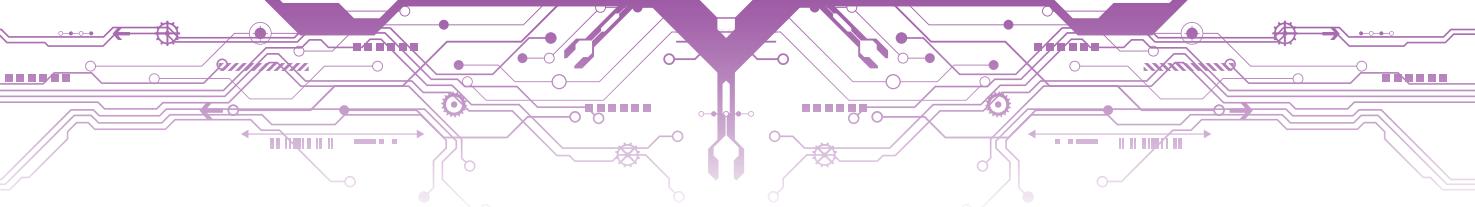
Girdi	Süreç	Modül Referansı	Çıktı
Sınav ve Performans Puanları	Sınav puanlarını gir. Performans puanlarını gir. Puan ortalamasını hesapla. Puan ortalamasının 50'den küçük olup olmadığını kontrol et. Geçme kalma durumunu ekrana yazdır. Bitir.	Oku Oku Hesapla Karar Yazdır Kontrol	Geçti/Kaldı

### 4.2.4. Algoritmalar

Bu çizelgeleri geliştirdikten sonraki adım, yapılacak işlemleri bilgisayarın anladığı dilde yazabilmektir. Bu yönergeler “algoritma” olarak adlandırılır. “Sözde kod” algoritmaya çok benzer bir dildir ve bazen algoritma yerine kullanılabilir. Algoritmayı oluşturmak, bilgisayarda problem çözme sürecinin en zor bölümüdür. Modüller etkileşim çizelgesinden ve süreç GSÇ çizelgesinden alınır. Algoritmadaki işlem sayısı, programcinin problemi çözme yoluna bağlıdır.

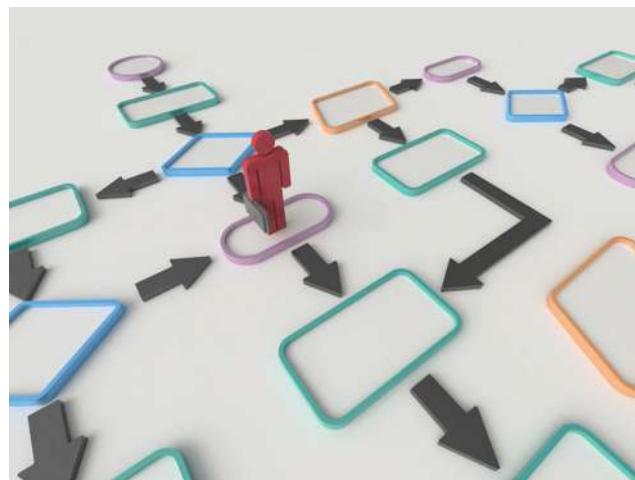
### 4.2.5. Akış Şemaları

Problem çözme sürecimiz, bilgisayarın iletişim kurma yöntemi ile şekillenir. Algoritma, bilgisayara hangi işlemi hangi sırada yapması gerektiğini söyleyen yönergeler bütünüdür. Akış şeması ise algoritmanın görsel gösterimidir. Programcı, oluşturulan algoritmdan grafiksel gösterimler oluşturur. Akış şeması, program geliştirmeye başlamadan önceki son adımdır. Akış şemasında hatalar rahatlıkla görü-



lüp düzeltilebilir. Akış şemalarını oluşturmak için kullanılan evrensel simgeler ve bu her bir simgenin anlamı vardır (Şekil 1.10).

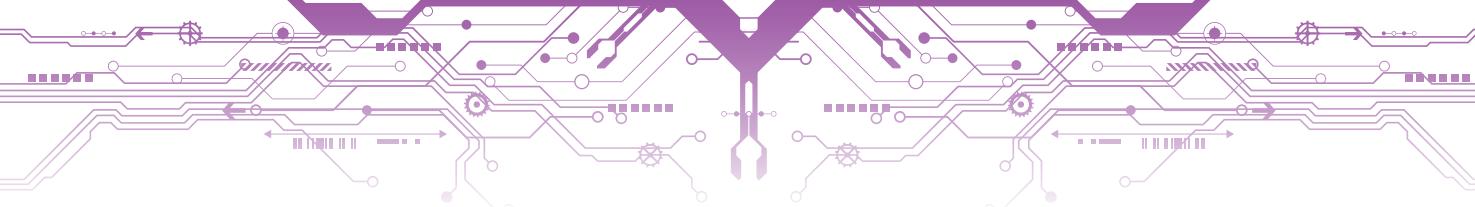
### 4.3. Algoritma Yönergeleri ve Akış Şeması Sembollerı



Simge	İşlev
	Başla/Bitir
	Giriş
	Atama/İşlem
	Denetim (Karar)
	Çıkış
	Döngü
	Akış Yönü
	Bağlaç
	Önceden Tanımlı İşlem/Fonksiyon

Şekil 1.10: Akış şeması sembollerı

Akış şeması, bir problem çözümünün başlangıcından bitişine kadar olan süreci gösterir. Akış şeması içerisindeki her bir simge, algoritmada bir işlemi ifade eder. Genellikle işlemler tek yönlü olmasına rağmen karar kutularından iki farklı ok çıkar. Bir karar simgesinden çıkan ok, bazı işlemlerin tekrarlanmasını sağlayabilir; böylece bir “döngü” oluşur.



Akiş şemalarını oluştururken dikkat edilmesi gereken bazı noktalar şunlardır:

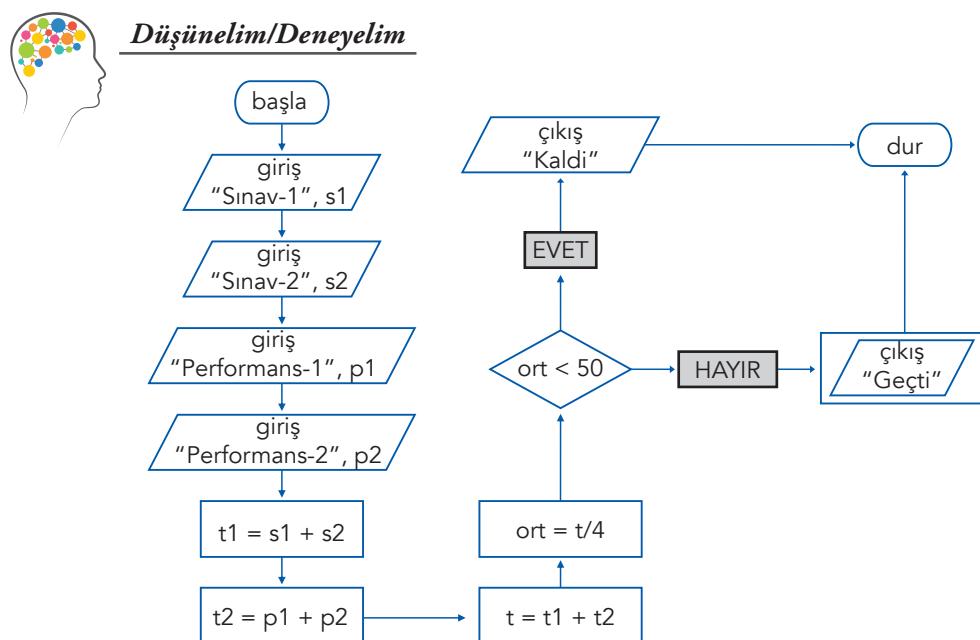
1. Yönergeler, simgelerin içine yazılmalıdır.
2. Hatırlatıcı bilgiler simgenin yanına yazılabılır. Böylece akış şeması ek açıklamalı bir şemaya dönüştür.
3. Bir akış şeması her zaman sayfanın başından başlar ve sonuna doğru gider. Eğer bir sayfaya sığmazsa bir ya da daha fazla bağlantı simgesi kullanılarak diğer sayfaya geçilebilir.
4. Akış şemasını çizmek için uygun yazılımlar kullanılırsa daha standart bir görünüm elde edilir.
5. Simgeler, içeriğindeki yazının rahatça okunabileceği kadar büyük yapılmalıdır.

#### 4.4. Haricî ve Dâhilî Dokümantasyon

İyi programcılar, kodları başkaları tarafından rahatça anlaşılabilse diye satırlar arasına açıklamalar yazarlar. Bu açıklamalar, diğer programcılar açısından büyük önem taşır çünkü kod üzerinde değişiklik yapılmaması için her bir satırın ya da fonksiyonun işlevinin anlaşılması gereklidir. Bu şekilde, yazılıma ait “dâhilî dokümantasyon” oluşturulmuş olunur. Kod satırları haricinde yazılımın kullanımına ve teknik gereksinimlere ait bilgilerden oluşan “haricî dokümantasyon” hazırlanır. Bu bilgiler, diğer kullanıcılar tarafından ortaya çıkan problemleri çözmek için kullanılır.

#### 4.5. Çözümün Programlanması/Kodlanması

Akiş şeması ve algoritmalar tamamlandıktan sonra istenilen bir programlama dili kullanılarak programın yapılması işlemine geçilir ki bu işleme “programlama” ya da “kodlama” adı verilir. Kodlama sonucunda programın ne kadar hatalız çalıştığı, algoritmanın etkililiğine bağlıdır.



Yukarıda iki yazılı ve iki performans puanı almış bir öğrencinin puan ortalamasını hesaplayarak, dersten geçip geçmediğini belirleyen akış şeması yer almaktadır. Siz de benzer bir problemi çözüme kavuşturacak basamakları akış şemasiyla oluşturmayı deneyiniz.

## 5. PROGRAMLAMA YAPISI

```
(function repeat() {  
    eat();  
    sleep();  
    code();  
    repeat();  
})();
```

Bu bölümde;

- ✓ Yapılandırılmış problemin önemini kavrayacak,
- ✓ Bağlantı için modül ve fonksiyonların nasıl tasarılanması gerektiğini açıklayabilecek,
- ✓ Değişken türlerini ve aralarındaki farkı açıklayabilecek,
- ✓ Parametrelerin nasıl kullanıldığını anlatabilecek,
- ✓ Dört mantık yapısının (doğrusal, döngüsel, karar ve durumsal) neler olduğunu söyleyeceksiniz.

## 5.1. Programlama Yapısına Giriş

Bir önceki bölümde anlatılan yaklaşımlar, problem çözümlerinin organize edilmesi için yardımcı olarak kullanılan araçlardır. Bu bölümden itibaren çözümleri bilgisayarın daha iyi anlayıp işleyebilmesi için kullanılan teknikler anlatılacaktır. Diğer bir ifade ile bu teknikler, algoritmayı oluşturan yönereleri farklı biçimlerde yazmanıza olanak sağlayacaktır.

### 5.1.1. Göstergeler

Bilgisayarlar; problemleri çözmek, işlerimizi kolaylaştmak, daha hızlı ve etkili çözümler üretmek için kullanılır. Gerçekten yeterli çözümler üretmek için aşağıdaki göstergeleri önemsemek gereklidir.

1. Bütünü, her biri anlamlı işlemler içeren parçalara bölünüz, modüllerini kullanınız.
2. Farklı satırlar arasında bağlantı kurmak yerine mantıksal yapıları kullanınız.
  - a) Doğrusal yapı, işlemleri sıra ile çalıştırır. Aşağıda, klavyeden girilen iki sınav puanının aritmetik ortalamasını hesaplayan yapı görülmektedir:

Algoritma	Akış Şeması	Sözde Kod
<ol style="list-style-type: none"><li>1. Başla.</li><li>2. Notları Oku.</li><li>3. Ortalamayı Hesapla.</li><li>4. Ortalamayı Yaz.</li><li>5. Bitir.</li></ol>	<pre>graph TD     Start([Başla]) --&gt; Decision{not1, not2}     Decision -- True --&gt; Calc[ort = (not1 + not2)/2]     Calc --&gt; Data[ort]     Data --&gt; End([Bitir])     Decision -- False --&gt; End</pre>	<ol style="list-style-type: none"><li>1. Başla.</li><li>2. Oku <i>not1, not2</i></li><li>3. <i>ort</i> = (<i>not1</i> + <i>not2</i>) / 2</li><li>4. Yaz <i>ort</i></li><li>5. Bitir.</li></ol>

- b) Karar yapısı, iki olasılıktan birini seçmek ve ona göre devam etmek için kullanılır. Aşağıda, klavyeden girilen iki sınav puanının ortalamasını bularak öğrencinin dersten geçip geçmediğini kontrol eden yapı görülmektedir.

Algoritma	Akış Şeması	Sözde Kod
<ol style="list-style-type: none"> <li>1. Başla.</li> <li>2. Notları oku.</li> <li>3. Ortalamayı hesapla.</li> <li>4. Eğer ortalama <math>\geq 50</math> ise “Geçti” yaz. Değilse “Kaldı” yaz.</li> <li>5. Bitir.</li> </ol>	<pre> graph TD     Start([Başla]) --&gt; Read[/not1, not2/]     Read --&gt; Calc[ort = (not1 + not2)/2]     Calc --&gt; Decision{ort &gt;= 50}     Decision -- Evet --&gt; Passed[Geçti]     Passed --&gt; End([Bitir])     Decision -- Hayır --&gt; Failed[Kaldı]     Failed --&gt; End   </pre>	<ol style="list-style-type: none"> <li>1. Başla.</li> <li>2. Oku <i>not1, not2</i></li> <li>3. <math>ort = (not1 + not2)/2</math></li> <li>4. if (<math>ort \geq 50</math>) then     Yaz “Geçti” else Yaz “Kaldı”</li> <li>5. Bitir.</li> </ol>

- c) Döngüsel yapı, bir dizi işlemi tekrarlamak için kullanılır. Aşağıda, 20 öğrencilik bir sınıfının bir dersten aldığı iki not üzerine sınıf ortalamasını hesaplayan yapı görülmektedir.

Algoritma	Akış Şeması	Sözde Kod
<p>1. Başla.</p> <p>2. Sınıf ortalamasını 0'a eşitle</p> <p>3. Döngü <math>i \leq 20</math> olana kadar dön, <math>i &gt; 20</math> olunca 7. adıma git.</p> <p>4. Notları oku.</p> <p>5. Ortalamayı hesapla.</p> <p>6. Öğrenci ortalamasını toplama aktar.</p> <p>7. <math>i</math>'yi 1 arttır.</p> <p>8. Sınıf ortalamasını hesapla.</p> <p>9. Sınıf ortalamasını yaz.</p> <p>10. Bitir.</p>	<pre> graph TD     Start([Başla]) --&gt; TopOrtInit[/topOrt = 0/]     TopOrtInit --&gt; Loop{<i>i=1; i&lt;=20; i++</i>}     Loop --&gt; NotInput[/not1, not2/]     NotInput --&gt; OrtCalc[ort = (not1 + not2)/2]     OrtCalc --&gt; TopOrtAdd[topOrt = topOrt + ort]     TopOrtAdd --&gt; i((i))     i --&gt; Loop     Loop --&gt; ClassAvg[sınıfOrt = topOrt/20]     ClassAvg --&gt; SınıfOrt[/SınıfOrt/]     SınıfOrt --&gt; Bitir([Bitir])   </pre>	<p>1. Başla.</p> <p>2. <math>topOrt = 0</math></p> <p>3. <math>\text{for } (i=1; i \leq 20; i++)</math></p> <p>4. Oku <math>not1, not2</math></p> <p>5. <math>ort = (not1 + not2)/2</math></p> <p>6. <math>topOrt = topOrt + ort</math></p> <p>7. <math>i++</math></p> <p>8. <math>sınıfOrt = topOrt/20</math></p> <p>9. Yaz <math>sınıfOrt</math></p> <p>10. Bitir.</p>

- c) Durumsal yapı ise belirli bir duruma göre farklı işlemlerin yapılmasına olanak sağlar. Aşağıda, klavyeden girilen sayıya göre haftanın gününü yazan yapı görülmektedir.

Algoritma	Akış Şeması	Sözde Kod
<ol style="list-style-type: none"> <li>1. Başla</li> <li>2. 1 ile 7 arasında bir sayı gir.</li> <li>3. Sayı 1 ise "Pazartesi" yaz.</li> <li>4. Sayı 2 ise "Salı" yaz.</li> <li>5. Sayı 3 ise "Çarşamba" yaz.</li> <li>6. Sayı 4 ise "Perşembe" yaz.</li> <li>7. Sayı 5 ise "Cuma" yaz.</li> <li>8. Sayı 6 ise "Cumartesi" yaz.</li> <li>9. Sayı 7 ise "Pazar" yaz.</li> <li>10. Sayı 1 ile 7 arasında değilse "Girdığınız sayı 1 ile 7 arasında olmalıdır." yaz.</li> <li>11. Bitir.</li> </ol>	<pre> graph TD     Start([Başla]) --&gt; Input[/sayı/]     Input --&gt; Decision1{sayı = 1}     Decision1 -- D --&gt; Pazartesi[Pazartesi]     Pazartesi -- Break --&gt; End([Bitir])     Decision1 -- Y --&gt; Decision2{sayı = 2}     Decision2 -- D --&gt; Salı[Salı]     Salı -- Break --&gt; End     Decision2 -- Y --&gt; Decision3{sayı = 3}     Decision3 -- D --&gt; Çarşamba[Çarşamba]     Çarşamba -- Break --&gt; End     Decision3 -- Y --&gt; Decision4{sayı = 4}     Decision4 -- D --&gt; Perşembe[Perşembe]     Perşembe -- Break --&gt; End     Decision4 -- Y --&gt; Decision5{sayı = 5}     Decision5 -- D --&gt; Cuma[Cuma]     Cuma -- Break --&gt; End     Decision5 -- Y --&gt; Decision6{sayı = 6}     Decision6 -- D --&gt; Cumartesi[Cumartesi]     Cumartesi -- Break --&gt; End     Decision6 -- Y --&gt; Decision7{sayı = 7}     Decision7 -- D --&gt; Pazar[Pazar]     Pazar -- Break --&gt; End     Decision7 -- Y --&gt; Default[Girdığınız sayı 1 ile 7 arasında olmalıdır.]     Default -- Break --&gt; End   </pre>	<ol style="list-style-type: none"> <li>1. Başla</li> <li>2. Oku <i>sayı</i></li> <li>3. <i>Switch</i> (<i>sayı</i>)</li> <li>4. Case 1:     "**Pazartesi**";     <i>Break</i>;</li> <li>5. Case 2: "Salı";     <i>Break</i>;</li> <li>6. Case 3:     "Çarşamba";     <i>Break</i>;</li> <li>7. Case 4:     "Perşembe";     <i>Break</i>;</li> <li>8. Case 5: "Cuma";     <i>Break</i>;</li> <li>9. Case 6:     "Cumartesi";     <i>Break</i>;</li> <li>10. Case 7: "Pazar";     <i>Break</i>;</li> <li>11. Default:     "Girdığınız sayı     1 ile 7 arasında     olmalıdır.";     <i>Break</i>;</li> <li>12. Bitir.</li> </ol>

3. Tekrarlayan işlemlerin tekrar tekrar yazılmasını önlemek için modüler yapı kullanınız.
4. Okunabilirliği ve anlaşılabilirliği artırmak için anlamlı değişken isimleri seçiniz ve çok iyi dokümanasyon hazırlayınız.



## *Düşünelim/Deneyelim*

1. Klavyeden ismini giren kişiyi “Merhaba ‘İsim’” şeklinde selamlayan algoritma ve akış şemasını oluşturunuz.
  2. Bir marketteki bir paket çayın fiyatı 10 TL'dir. Müşteri 10 ile 49 arasında paket aldığında %5, 50 - 99 adet arası paket aldığında %10; 100 üzeri paket aldığında %15 indirim uygulanmaktadır. Faturaya uygulanacak KDV %8'dir. Müşterinin aldığı paket sayısına göre ödeyeceği ücreti hesaplayan algoritma ve akış şemasını yazınız.
  3. Klavyeden, uzunlukları girilen üç doğru parçasının bir üçgen oluşturup oluşturamayacağını hesaplayan algoritma ve akış şemasını oluşturunuz.  
*(Bir üçgenin iki kenarının uzunluğu toplamı, üçüncü kenardan büyük olmalıdır.)*
  4. Klavyeden, X karakteri girilene kadar girilen isimlere kaçinci kişi olduğunu yayan algoritma ve akış şemasını oluşturunuz.
  5. Klavyeden, girilen sıcaklık derecesine göre suyun katı/sıvı/gaz hâllerinden hangisinde olduğunu yayan algoritma ve akış şemasını oluşturunuz.
  6. Klavyeden, 0 girilene kadar kaç adet negatif; kaç adet pozitif sayı girildiğini hesaplayan algoritma ve akış şemasını oluşturunuz.
  7. Klavyeden, sayıya kadar olan sayıların çift olanlarının toplamını hesaplayan ve yazdırın algoritma ve akış şemasını yazınız.
  8. Girilen iki değer arasındaki sayıların toplamını bulan ve ekrana yazdırın algoritma ve akış şemasını oluşturunuz.
  9. Bir öğrencinin ortalama ve devamsızlık bilgisine göre geçme/kalma durumunu kontrol ediniz ve ekrana yazdırın algoritma akış şemasını oluşturunuz.
  10. Aşağıdaki şekilleri, ekrana yazdırın algoritma akış şemasını oluşturunuz.

(Bir üçgenin iki kenarının uzunluğu toplamı, üçüncü kenardan büyük olmalıdır.)

- Klavyeden, X karakteri girilene kadar girilen isimlere kaçinci kişi olduğunu yayan algoritma ve akış şemasını oluşturunuz.
  - Klavyeden, girilen sıcaklık derecesine göre suyun katı/sıvı/gaz hâllerinden hangisinde olduğunu yayan algoritma ve akış şemasını oluşturunuz.
  - Klavyeden, 0 girilene kadar kaç adet negatif; kaç adet pozitif sayı girildiğini hesaplayan algoritma ve akış şemasını oluşturunuz.
  - Klavyeden, girilen sayıya kadar olan sayıların çift olanlarının toplamını hesaplayan ve yazdırın algoritma ve akış şemasını yazınız.
  - Girilen iki değer arasındaki sayıların toplamını bulan ve ekrana yazdırın algoritma ve akış şemasını oluşturunuz.
  - Bir öğrencinin ortalama ve devamsızlık bilgisine göre geçme/kalma durumunu kontrol ediniz ve ekrana yazdırın algoritma akış şemasını oluşturunuz.
  - Aşağıdaki şekilleri, ekrana yazdırın algoritma akış şemasını oluşturunuz.

\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*

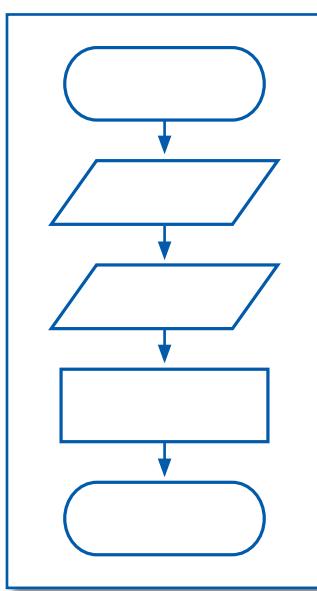
\*  
\*\*  
\*\*\*  
\*\*\*\*

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*



11. Klavyeden, girilen ay bilgisine göre kuzey yarımkürede hangi mevsimin yaşandığını ekrana yazdırın algoritma akış şemasını oluşturunuz.
12. Klavyeden, bir üçgene ait kenar uzunluğu ve o kenarın yükseklik bilgisinin girişi yapıldıktan sonra ekrana üçgenin alanını yazan programın akış diyagramının doğru olşabilmesi için şeklärin üzerine satır numaralarını yazınız.

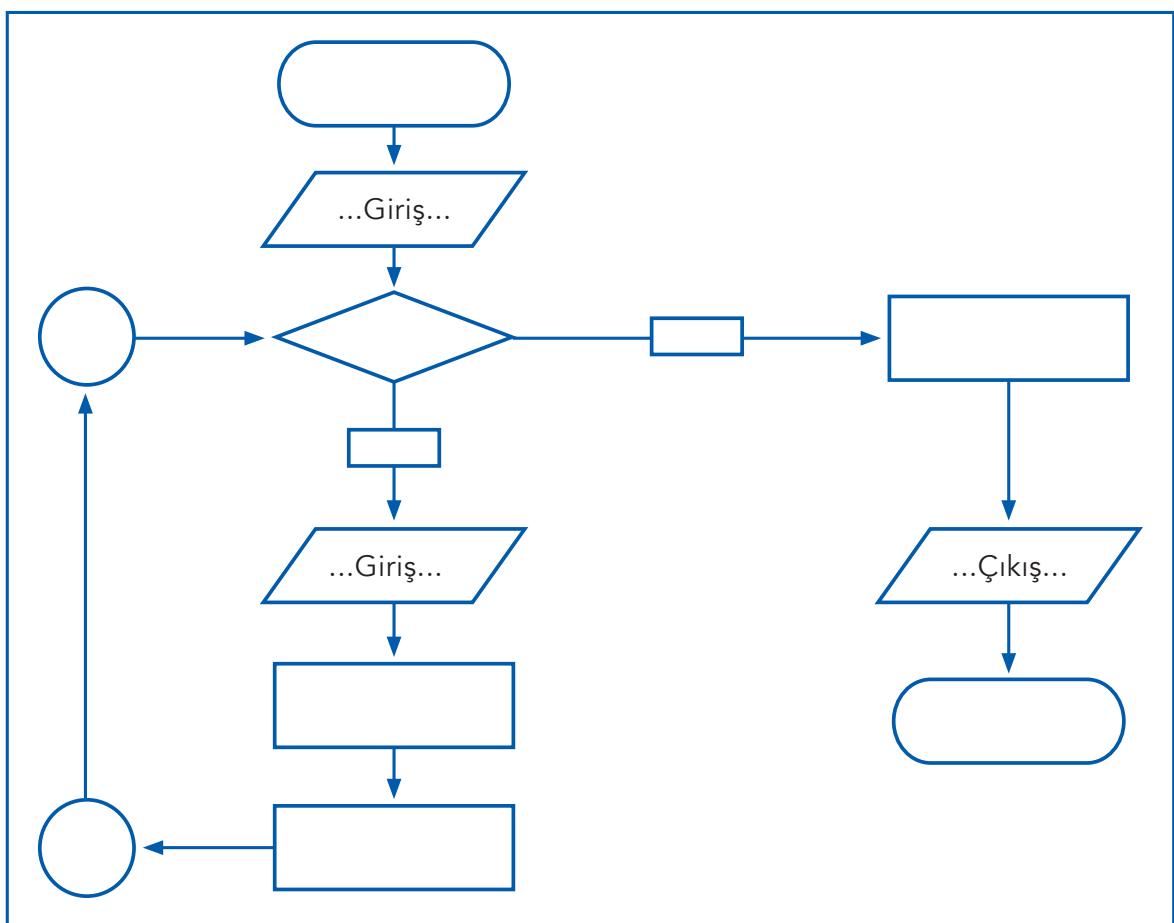
1. a
2. alan = a\*h/2
3. h
4. Başla
5. Bitir
6. Alan



13. Aşağıdaki problemde, ortalaması girilen bir öğrencinin dersten başarılı ya da başarısız olduğunu ekrana yazabilmesi için boşlukları doldurunuz.
  1. Başla
  2. Gir .....
  3. Eğer ort>=50 ise yaz .....
  4. Değilse .....
  5. .....
14. Klavyeden, girilen sıcaklık bilgisine göre aşağıdaki durumların kontrolünün yapılması istenmektedir. Gerekli algoritmanın olşabilmesi için boşlukları doldurunuz.
  - Sıcaklık 0 °C'nin altındaysa “Çok Soğuk”
  - Sıcaklık 0 °C – 20 °C arasındaysa “Serin”
  - Sıcaklık 20 °C üzerindeyse “Sıcak” olmalı.
  1. .....
  2. Oku .....
  3. Eğer ..... Yaz “Çok Soğuk”
  4. ..... “Serin”
  5. Değilse Yaz .....
  6. Bitir.

15. Aşağıdaki akış diyagramında öğrenci sayısı klavyeden girildikten sonra öğrenci yaşları okunarak, sınıf yaş ortalaması bulunup ekrana yazdırılmaktadır. Bu amaçla, aşağıda verilen adımları madde numaralarını, akış diyagramında uygun yerlere yerleştiriniz.

1. ortalama
2. ogrSayisi
3. sayac = sayac + 1
4. ortalama = yasToplam/ogrSayisi
5. yasToplam = yasToplam + ogrYas
6. Başla
7. ogrSayisi > sayac
8. Evet
9. Hayır
10. Bitir
11. ogrYas





### 5.1.2. Modüller ve İşlevleri

Bir yazarın, kitabını yazmaya başlamadan önce konuyu ve bölümleri düşünmesi, bir aşçının menüyü hazırlamaya başlamadan önce yemek türlerini, malzemeleri ve miktarları düşünmesi gibi bir programcı da programı yazmaya başlamadan önce detaylı bir biçimde problemi irdelemeli ve işlemleri gruplandırmalıdır. Ne zaman modüller etkileşim çizelgesinde doğru sıralanmış ise programcı her bir modül için kodu yazmaya başlayabilir. İyi bir programcı algoritmayı her bir modül için test eder, sorunlar varsa hemen çözüm üretir. Büyük bir program yerine küçük parçaları kontrol etmek daha kolaydır ve bu, zamanдан kazanç sağlar. Modülleri oluştururken aşağıdaki noktalara dikkat edilmesi önerilir.

1. Her bir modül başlar, işlemleri yapar ve biter. Süreç içerisinde modüller arasında dallanma olmaz.
2. Her bir modülün tek bir işlevi vardır: yazdırma, karekök bulma, büyük harfe çevirme vb.
3. Her modül rahat anlaşılabilecek ve kolayca güncellenebilecek kadar kısa olmalıdır.
4. Modülün uzunluğu işlevine ve yönerge sayısına göre değişebilir.
5. Modüller süreç akışlarını kontrol etmek için oluşturulur.

Çok sık kullanılan modül türleri şu şekilde sıralanabilir:

- Kontrol modülü programın genel akışını gösterir.
- Başlama modülü program ilk başladığında yalnız bir kez yapılması gereken işlemleri gerçekleştir (ilk değerlerin atanması).
- Süreç modülleri bir ya da birden fazla belirli bir işlemi yapmak için kullanılır (hesaplama, veri okuma, yazdırma vb.).
- Bitiş modülü ise program bitmeden önce yapılacak son işlemleri içerir.

Farklı modüller bir problemin çözümü için bir araya gelebilir. Program yönetimini kolaylaştırmak için bir modül birden fazla modülden de oluşabilir.

### 5.1.3. Bağlılık ve Birleşim

Problem çözme sürecindeki en zor adım, çözümü parçalara ayırmaktır. Her bir modülün hangi işlemlerden oluşması gerektiği önemli bir karardır. Modüller hem farklı işlemleri yürütecek kadar birbirinden bağımsız olmalı hem de aynı veriler ile çalışacak kadar birleşik olmalıdır. Birbirine zıt bu iki kavram bağlılık ve birleşim olarak adlandırılır. “Bağlılık” bir modülün diğer modüllerden bağımsız çalışabilme yeteneğidir. Her bir modül, bağımsız olarak tanımlanmış işlem setini çalıştırır ve sonucu gönderir. Ancak modüller çalışırken verileri almaları ve sonucu göndermeleri sürecinde bilgi paylaşır. Bu veri paylaşımı sürecinde “birleşim” yaşanır.

Birleşim, modüller arası iletişim olmasını sağlar.

“Bağlılık”, bir modül içerisindeki fonksiyonların birbiri ile ne derece ilişkili olduğunu ölçümüdür. Eğer fonksiyonlar birbiri ile yakın ilişki içerisinde ise modülün yüksek bağlılık düzeyi olduğu söylenir. Benzer durumda birleşim düzeyi düşük olacaktır. Yüksek bağlılık ve düşük birleşim genellikle iyi yapılandırılmış bir tasarım demektir. Böyle tasarlanmış bir program daha anlaşırlı, güvenilir ve tekrar kullanılabilir olacaktır. “Birleşim” ise modüllerin birbirine





bağımlı olma düzeyidir. Düşük düzey birleşim her zaman tercih edilir çünkü bu durum modülün yönetimini ve okunabilirliğini arttırmır.

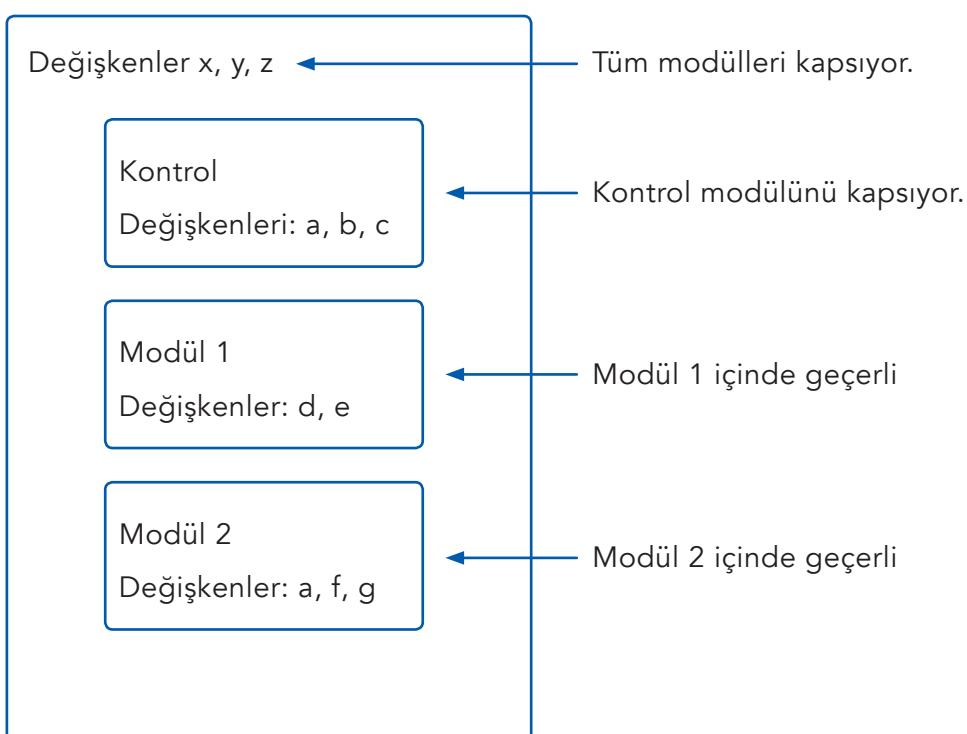
Yüksek bağılalık düzeyi programın daha anlaşılır, güvenilir ve tekrar kullanılabilir olması açısından çok önemlidir. "Birleşim" ise modüllerin birbirine bağımlı olma düzeyidir. Düşük düzey birleşim her zaman tercih edilir çünkü bu durum modülün yönetimini ve okunabilirliğini arttırmır.

#### 5.1.4. Yerel ve Global Değişkenler

Yerel ve global değişken kavramı tüm programlama dilleri için çok önemli kavumlardır. Programcılar, yerel ve global değişkenleri bağılilik ve yapışkanlık oluşturmak amacıyla kullanırlar. Bir modül içinde tanımlanmış değişkenler "yerel"; modüler dışında program genelinde kullanılmak üzere tanımlanmış değişkenler ise "global" değişkenler olarak adlandırılır. Aralarındaki en önemli fark, kapsamlarıdır. Bu kapsam, değişkenin ne zamanda kullanılabileceğini belirler.

Yerel değişkenler, yalnızca tanımlandıkları modül içerisinde kullanılabilir. Diğer modüllerin bu değişkenlere ilişkin hiçbir bilgisi bulunmaz. Böylece değişken isimlerinin çakışması gibi sorunlar yaşanmaz. Bu nedenle yerel bir değişkenin diğer modüller tarafından kullanılması gereklse bu değişkenin parametre ya da dönen değerler ile eşleştirilmesi gereklidir.

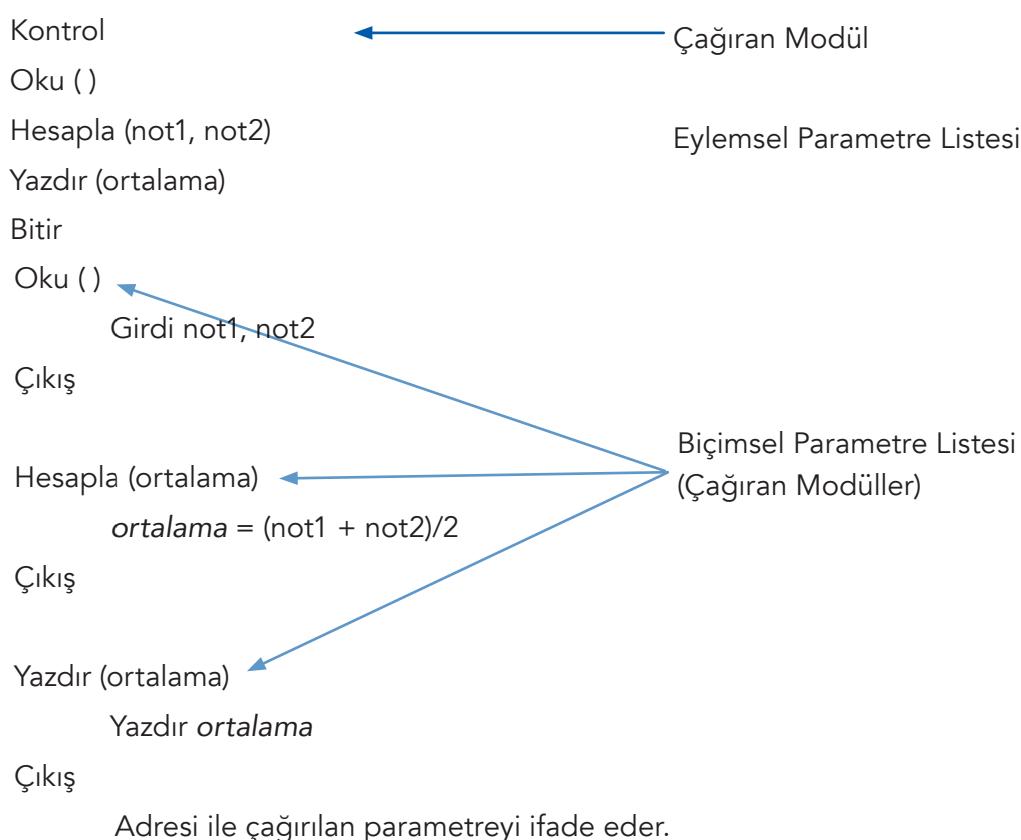
Global olarak tanımlanan değişkenler ise bütün modüller tarafından tanınır. Program çalıştığı sürece hafızada tutulan ve işlem yapılabilen değişkenlerdir. Program akışında bağılilik oluşturmak için gereklidir. Programın çalışma sürecinde değişkenler için kullanılan bir hiyerarşi vardır. Buna göre önce yerel değişken, sonra parametre ve daha sonra da global değişken kullanılır. Bu yüzden programcılar aynı değişkeni hem yerel hem de global olarak tanımlarsa sorun yaşanır. Global değişkenler modüller arasında veri geçişini sağladığı için bu tür durumlarda parametre ve dönen değerlerin kullanılmasına gerek olmaz. Aşağıda yerel ve global değişkenlere ilişkin bir örnek görülmektedir.





### 5.1.5. Parametreler

Programlama sürecinde anlaşılması zor ama çok önemli konulardan biri parametrelerdir. Parametrelerin kullanımı benzer değişken isimlerinin çakışmasını önleyerek sürecin daha hızlı ilerlemesini sağlar. Modüller ayrı ayrı çalışır ve veriler parametreler aracılığı ile iletilir. "Parametreler" bir modülden diğerine geçen yerel değişkenlerdir. Modüller arasındaki iletişimi sağlar. Modül adından sonra ayraç içerisinde belirtilek kullanıllar: Oku (a, b, c) gibi. Eylemsel parametreler listesi, kontrol modülü tarafından çağrılan parametrelerin listesidir. Biçimsel parametre listesi ise ilgili modülü takip eden parametre listesidir.



Parametreleri kullanarak bir modülden diğerine veri göndermenin iki yolu vardır: Parametreleri değeri ya da adresi ile çağrıbiliriz. Önünde “\*\*” simgesi olmayan parametreler, değeri ile çağrılan parametrelerdir. Parametre değeri ile ilgili modüle iletildiğinde, o modül o değerle ilk kez karşılaştığı için hafızada yeni bir yer oluşturur. Bu durumda ilgili değer değiştiğinde programın geneli etkilenmez; yalnızca modül aktif olduğu sürece geçerli olur çünkü çağrılan modülde bu değer için hafızada ayrılmış farklı bir yer vardır. Böylece değeri ile çağrılan değişken için hafızada iki farklı yer ayrılmıştır ve bunlar, farklı modüller tarafından değiştirilebilir.

Parametre, değeri ile değil, adresi ile çağrıldığında, ilgili modüle değer yerine hafızadaki yer bilgisi gönderilir. Çağırılan modül, hafızada ilgili yerdeki değer ile işlem yapar. Bir değişiklik olduğunda hem çağrıran modül hem de çalışan modül bu değişikliği fark eder yani değişiklik program genelinde etkili olur. Adresi ile çağrılan parametrede hafızada aynı yer paylaşılırken, değeri ile çağrıldığında hafızada aynı değişken iki farklı yer ayrılr. Parametreler modüller arasında yapışkanlık yaratmak için en iyi yöntemdir.

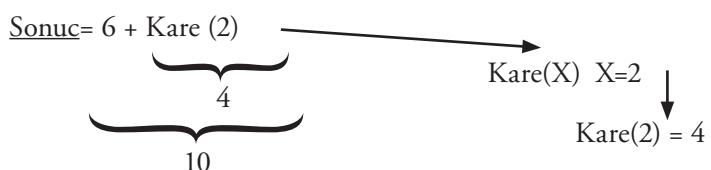


### 5.1.6. Dönen Değerler

Modüller arasında yapışkanlık yaratmanın üç yolu vardır.

1. Yerel değişkenler
2. Parametreler
3. Dönen değerler

Bir fonksiyonu çağrıp belirli işlemleri gerçekleştirdiğimiz zaman fonksiyon bize bir ya da daha fazla işlem sonucunu döndürür. Dönen değer, fonksiyon sonucudur. Bu işlem, fonksiyonu adı ile çağrıarak gerçekleşir. İşlem sonucundaki değer, geçici olarak ilgili değişkene atanır. Fonksiyon, çalışmasını bitirdiğinde artık o isme atanmış bir değer bulunmaz çünkü bu değer, çağrıyan modüle geri dönmüştür. Aşağıdaki şekilde bu akış görülmektedir.



Algoritma	Akış Şeması	Sözde Kod
<ol style="list-style-type: none"><li>1. Başla.</li><li>2. Kare(2) işleminden dönen değer ile 6'yı topla.</li><li>3. Sonucu yaz.</li><li>4. Bitir.</li></ol> <ol style="list-style-type: none"><li>1. Kare modülünü oluştur.</li><li>2. Başla.</li><li>3. Klavyeden girilen değeri kendisiyle çarp.</li><li>4. Sonucu döndür.</li><li>5. Bitir.</li></ol>	<pre>graph TD; subgraph TopModule [Top Module]; Start1([Başla]) --&gt; Process1[sonuc = 6 + kare(2)]; Process1 --&gt; Output1[sonuc]; Output1 --&gt; Input2[kare(x)]; end; subgraph BottomModule [Bottom Module]; Start2([Başla]); Process2[s = x * x]; Process2 --&gt; Return2[Return s]; end; Input2 --&gt; Process2;</pre>	<ol style="list-style-type: none"><li>1. Başla</li><li>2. <code>sonuc = 6 + kare(2);</code></li><li>3. Yaz sonuc;</li><li>4. Bitir</li></ol> <ol style="list-style-type: none"><li>1. Modul kare(x);</li><li>2. Başla</li><li>3. <code>s=x*x;</code></li><li>4. <code>return s;</code></li></ol>



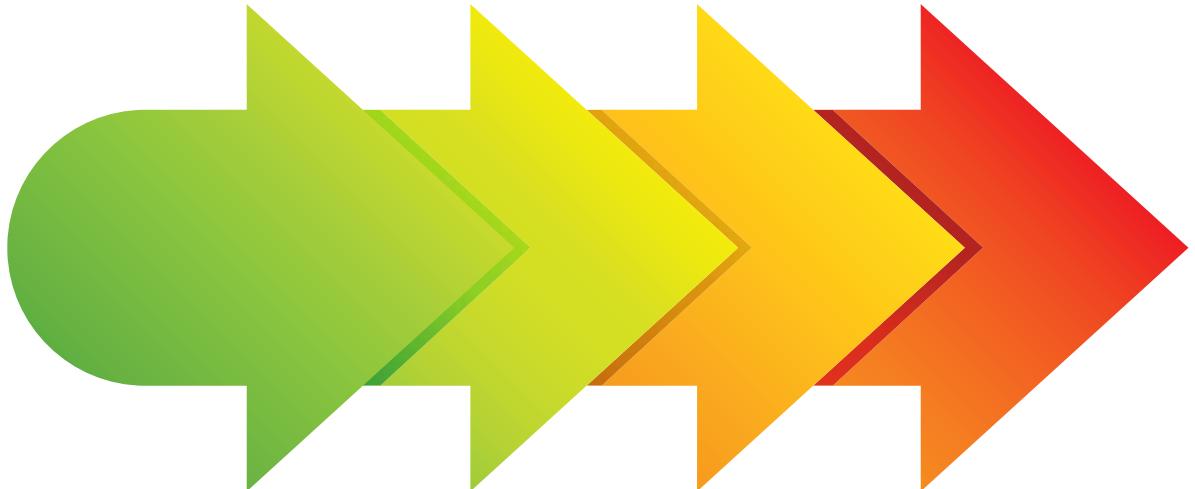
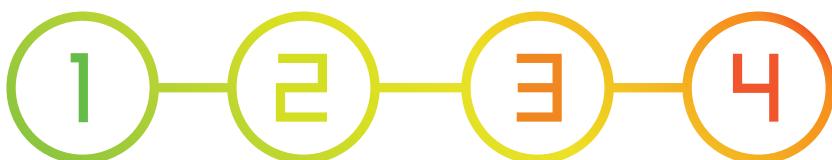
### Düşünelim/Deneyelim

- Klavyeden girilen iki sayının toplamını alan algoritmayı geriye değer döndüren parametreli fonksiyon kullanarak çözünüz.
- Ekrandan girilen iki sayının önce pozitif ya da negatif olduğunu; sonra en küçük ortak bölenini bulan algoritmayı yazınız. Bu iki işlem için ayrı fonksiyonlar kullanın. Sayı değerlerini global; diğer sonuçları yerel olarak tanımlayınız. Çıktıları öncelikle fonksiyonların içinde yazdırıp sonra çıktıları ana programda yazdıracak biçimde düzenleyiniz.
- Girilen bir sayıya ilişkin olarak aşağıdaki özellikleri verilen programı yazınız.
  - a) Sayı kaç basamaktan oluşmaktadır?
  - b) Sayı 3'ün ve 10'un katlarına bölünebiliyor mu?
  - c) Sayı asal sayı mı?

Bu problemleri önce parametre kullanmadan sonra da parametre kullanarak çözünüz.

- Bir dikdörtgen prizmanın;
  - a) Yüzey alanını,
  - b) Hacmini bulunuz.
- Klavyeden üç sayı giriliyor.
  - a) Bu sayılarla bir üçgen çizilip çizilemeyeceğini,
  - b) Çiziliyorsa türünü (eşkenar, ikizkenar, çeşitkenar),
  - c) Üçgenin çevresini,
  - ç) Üçgenin alanını hesaplayan algoritmayı yazınız.

## 6. DOĞRUSAL MANTIK YAPISI İLE PROBLEM ÇÖZME



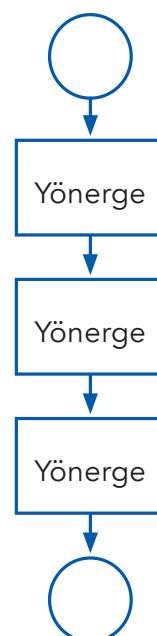
Bu bölümde;

- ✓ Bir probleme çözüm üretEBilmek için doğrusal mantık yapısını kullanabilecek,
- ✓ Bir algoritma ve akış şeması oluştururken uygun yönergeleri kullanabilecek,
- ✓ Probleme çözüm üretirken yedi problem çözme aracını kullanabileceksiniz.

## 6.1. Doğrusal Mantık Yapısı

Problem çözme sürecinde çok basit ve çok sık kullanılan yaklaşımlardan biri doğrusal mantık yapısıdır. Bilgisayara birbiri ardına algoritmanın başından sonuna kadar sırası ile işlemesi gereken komutları veren bir programcı, bu yaklaşımı kullanıyor demektir. Algoritma ve akış şeması genel olarak şu şekilde:

Modül Adı  
1. Yönerge  
2. Yönerge  
3. Yönerge  
4. ....  
....  
XX. Bitir, Çıkış ya da Döndür(değer)



Matematiksel bir formülün hesaplanması, kullanıcının girdiği veriye dayalı basit işlemler yaparak sonucu döndürmek gibi durumlarda, doğrusal mantık yapısı kullanılır. Bir dairenin alanını hesaplama, 3 kişinin yaş ortalamasını bulma, küçük harf olarak girilen metni büyük harfe çevirme gibi işlemler buna örnek olarak verilebilir.

## 6.2. Çözüm Üretilmesi

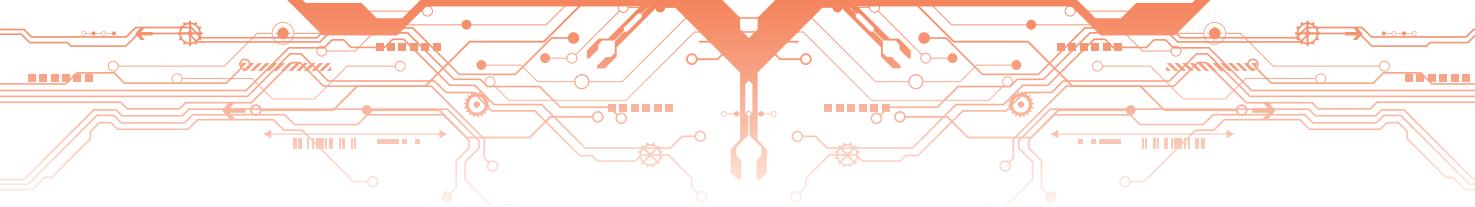
Bir önceki bölümde çözüm üretirken kullanılacak bazı yaklaşımları incelemiştik.

1. Problem Analiz Çizelgesi
2. Etkileşim Çizelgesi
3. GSC Çizelgesi
4. Birleşim Çizelgesi ve Veri Sözlüğü
5. Algoritmalar ve Akış Şemaları
6. Çözümün Test Edilmesi

Bu adımları izlemenin, çözümü ne kadar kolaylaştırdığını ve özellikle karmaşık problemlerin çözümünde ne kadar önemli olduğunu unutmayalım. Şimdi bir örneği ele alalım.

**Problem:** Sevgi Merit, önumüzdeki 5 yıl boyunca kendisine en iyi faizi verecek bankayı aramaktadır. Sevgi, ilgili bankaya 20.000 TL değerinde yatırım yapacaktır. Faiz hesaplaması için kullanılan standart formül şu şekildedir:

$$\text{Miktar} = A * (1 + F/S)^{(Y * S)}$$



A = Anapara (Bu örnek için 20.000)

F = Faiz Yüzdesi (Bankanın müsteriye ödeyeceği kârın anaparaya oranı)

Y = Yıl (Anaparanın değerlendirilme süresi)

S = Bileşik Faiz İçin Süre

Örneğin çözümünü aşağıdaki başlıklar altında inceleyelim:

### 6.2.1. Problem Analiz Çizelgesi

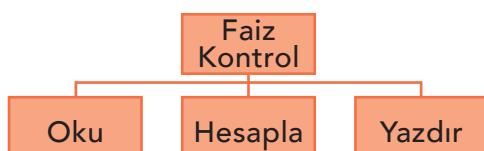
Öncelikle yapılması gereken, problem analizidir yani elde ne olduğu ve bizden ne istendiğini açık ve net biçimde analiz etmektir.

1. Eldeki veri	2. Beklenen sonuç
Anapara Faiz Yüzdesi Yıl Süre	Süre sonunda elde edilen toplam gelir: anapara + faiz
3. Problemin çözüm süreci	4. Çözüm seçenekleri
Miktar = A * (1 + F/S)^{Y * S}	Bütün verileri değişken olarak tanımla ve girdi olarak al. Anapara ve faiz yüzdesini sabit olarak tanımla. Her bir seferde bir banka için hesaplama yap. Tek seferde bütün bankalar için hesaplama yap.

Bu çizelge hazırlandıktan sonra olası çözüm seçenekleri içerisinde en etkili ve esnek olan çözümün seçilmesi gereklidir. Yukarıdaki çizelgede 1 ve 3 numaralı çözümler en uygun çözüm olarak seçilebilir. Faiz 2 numaralı seçenekte olduğu gibi sabit olarak tanımlarsak bunu, her banka için değiştirmemiz gereklidir ki bu da programın esnek olmasını öner. En son seçenekteki tüm bankalar için programın çalışması ise çok fazla değişken tanımlanması gerektirdiğinden karışıklığa neden olabilir. Bu nedenle bütün verileri değişken olarak tanımlamak ve her seferinde tek bir banka için işlem yapmak, esnek ve uygun bir çözüm olarak ortaya çıkmaktadır.

### 6.2.2. Etkileşim Çizelgesi

Bu noktada etkileşim çizelgesini oluşturabilirmiz. Bu amaçla: "Bu problemi yönetilebilir olacak biçimde ne kadar küçük parçalara bölebilirim?" sorusuna yanıt vermemiz gereklidir.

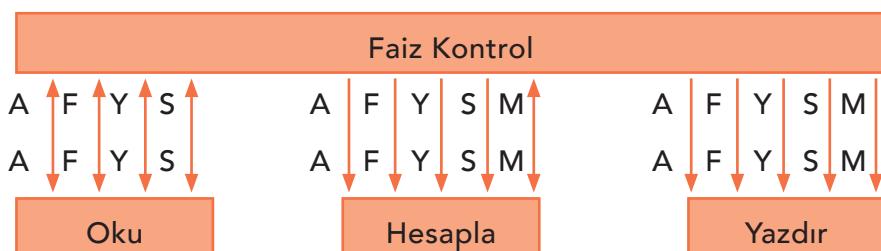


### 6.2.3. GSÇ Çizelgesi

Girdi	Süreç	Modül Referansı	Cıktı
1. Anapara	5. Verileri gir.	Oku.	9. Elde edilen gelir ve ana para
2. Faiz Yüzdesi	6. Faizi ile birlikte süreç sonundaki ana parayı hesapla.	Hesapla.	10. Tüm girilen değerler
3. Yıl	7. Sonucu ekrana yazdır.	Yazdır.	
4. Süre (bileşik faiz için yıllık tekrarlama sayısı)	8. Bitir.		

### 6.2.4. Birleşim Çizelgesi ve Veri Sözlüğü

Birleşim çizelgesi hangi değişkenlerin bir modülden diğerine geçtiğini gösterir. Veri sözlüğü ise her bir modülün hangi değişkenleri kullandığını ve kapsamını belirtir. İlgili problem için birleşim çizelgesi aşağıda görülmektedir.



Yine ilgili problem için oluşturulan veri sözlüğü ise şu şekildedir:

Veri	Değişken İsmi	Veri Türü	Modül(ler)	Kapsam	Hata Kontrolü
Miktar	miktar	Sayısal, reel	Faiz Kontrol, Oku, Hesapla, Yazdır.	Yerel parametre	yok
Anapara	anapara	Sayısal, reel	Faiz Kontrol, Oku, Hesapla, Yazdır.	Yerel parametre	yok
Faiz Yüzdesi	faiz	Sayısal, reel	Faiz Kontrol, Oku, Hesapla, Yazdır.	Yerel parametre	yok
Yıl	yıl	Sayısal, reel	Faiz Kontrol, Oku, Hesapla, Yazdır.	Yerel parametre	yok
Süre	süre	Sayısal, reel	Faiz Kontrol, Hesapla, Yazdır.	Yerel parametre	yok

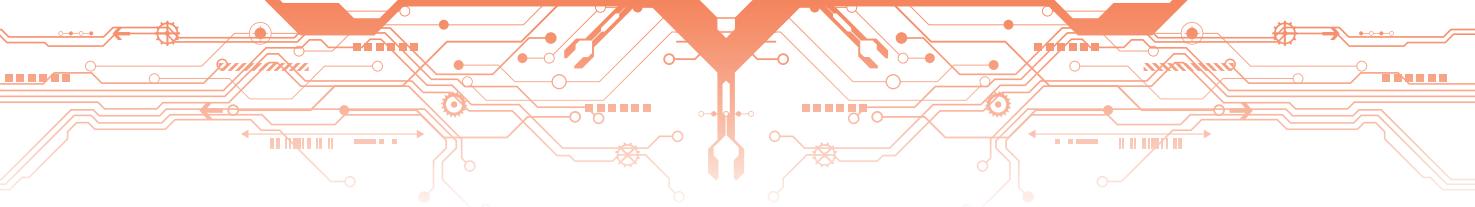
### 6.2.5. Algoritma ve Akış Şemaları

Bu çizelgeleri oluşturuktan sonra artık algoritma ve akış şemalarını hazırlamaya gelebiliriz.

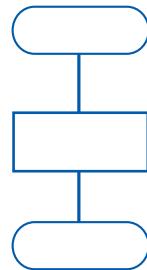
Algoritma	Akış Şeması	Bilgi Notu	Test
<b>Faiz Kontrol</b> <ol style="list-style-type: none"> <li>1. Süreç Oku. (*anapara, *faiz, *yıl, *sure)</li> <li>2. Süreç Hesapla. (*miktar, anapara, faiz, yıl, süre)</li> <li>3. Süreç Yazdır. (miktar, anapara, faiz, yıl, süre)</li> <li>4. Bitir.</li> </ol>	<pre> graph TD     Start([ ]) --&gt; Input[/]     Input --&gt; Calculation1[/]     Calculation1 --&gt; Calculation2[/]     Calculation2 --&gt; Output([ ])     Output --&gt; End([ ]) </pre>	<p>Bütün veriler klavyeden girilir.</p> <p>Miktar hesaplanır.</p> <p>Veriler ve sonuç yazdırılır.</p>	<ol style="list-style-type: none"> <li>1. Başla.</li> <li>2. Okumaya transfer et.</li> <li>3. Hesaplama transfer et.</li> <li>4. Yazdırma transfer et.</li> </ol>
<b>Dâhilî Dokümantasyon</b>		<b>Harici Dokümantasyon</b>	
<ol style="list-style-type: none"> <li>1. Program başında hangi değişkenleri alarak neyin hesapladığı konusunda bilgi verilir.</li> <li>2. İlgili satırlarda işlemler ve nedenleri eklenir.</li> </ol>		Dâhilî dokümantasyon ile benzerdir.	

Faiz kontrolü modülü için gerekli algoritma ve akış şemاسını hazırladıktan sonra sırası ile diğer modülleri oluşturalım. Oku modülüne bakalım.

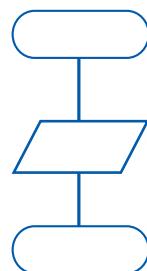
Algoritma	Akış Şeması	Bilgi Notu	Test
<ol style="list-style-type: none"> <li>1. Oku. (*anapara, *faiz, *yıl, *sure)  Veri gir. anapara, faiz, yıl, süre</li> <li>2. faiz = faiz/100</li> <li>3. Çıkış</li> </ol>	<pre> graph TD     Start([ ]) --&gt; Input[/]     Input --&gt; Calculation[/]     Calculation --&gt; Output([ ]) </pre>	<ol style="list-style-type: none"> <li>1. Faiz yüzde olarak alınır.</li> <li>2. Süre yıllık bileşik faiz için sıklık değeridir.</li> </ol>	Anapara = 20.000 Faiz = %5 Yıl = 5 Süre = 2
<b>Dâhilî Dokümantasyon</b>		<b>Harici Dokümantasyon</b>	
Girilen veri ve faizin yüzdelik değere dönüştürülmesi anlatılır.		Girilecek verinin açıklanması önerilir.	



Şimdi ise hesapla modülü için ilgili işlemleri yapalım.

Algoritma	Akış Şeması	Bilgi Notu	Test
<b>Hesapla</b> (*miktar, anapara, faiz, yıl, süre) 1. miktar = anapara * (1 + faiz/süre)^(yıl * süre) 2. Çıkış		yok	Miktar = $20.000 * (1 + .05/2)^{(5 * 2)} = 25600$
<b>Dâhilî Dokümantasyon</b>		<b>Harici Dokümantasyon</b>	
Girilmesi gereken veri belirtilir.		Formül belirtilir.	

Son olarak yazdır modülüne göz atalım.

Algoritma	Akış Şeması	Bilgi Notu	Test
<b>Yazdır</b> (miktar, anapara, faiz, yıl, süre) Yazdır miktar, anapara, faiz, yıl, süre Çıkış		Her bir değişkeni tanımı ile birlikte ayrı bir satırda yazdır.	Sonuç yazdırılır.
<b>Dâhilî Dokümantasyon</b>		<b>Harici Dokümantasyon</b>	
Girilmesi gereken veriler belirtilir.		Beklenen çıktı ve biçimini açıklanır.	

### 6.2.6. Çözümün Test Edilmesi:

Çözüm süreci sonucunda ortaya çıkan ürün, algoritma oluşturma programı yardımıyla çalıştırılarak her adımın doğru çalışıp çalışmadığı test edilmelidir. Hata tespiti durumunda sürecin ilgili basamakları gözden geçirilerek hata(lar) giderilip çözüme ulaşılmalıdır.

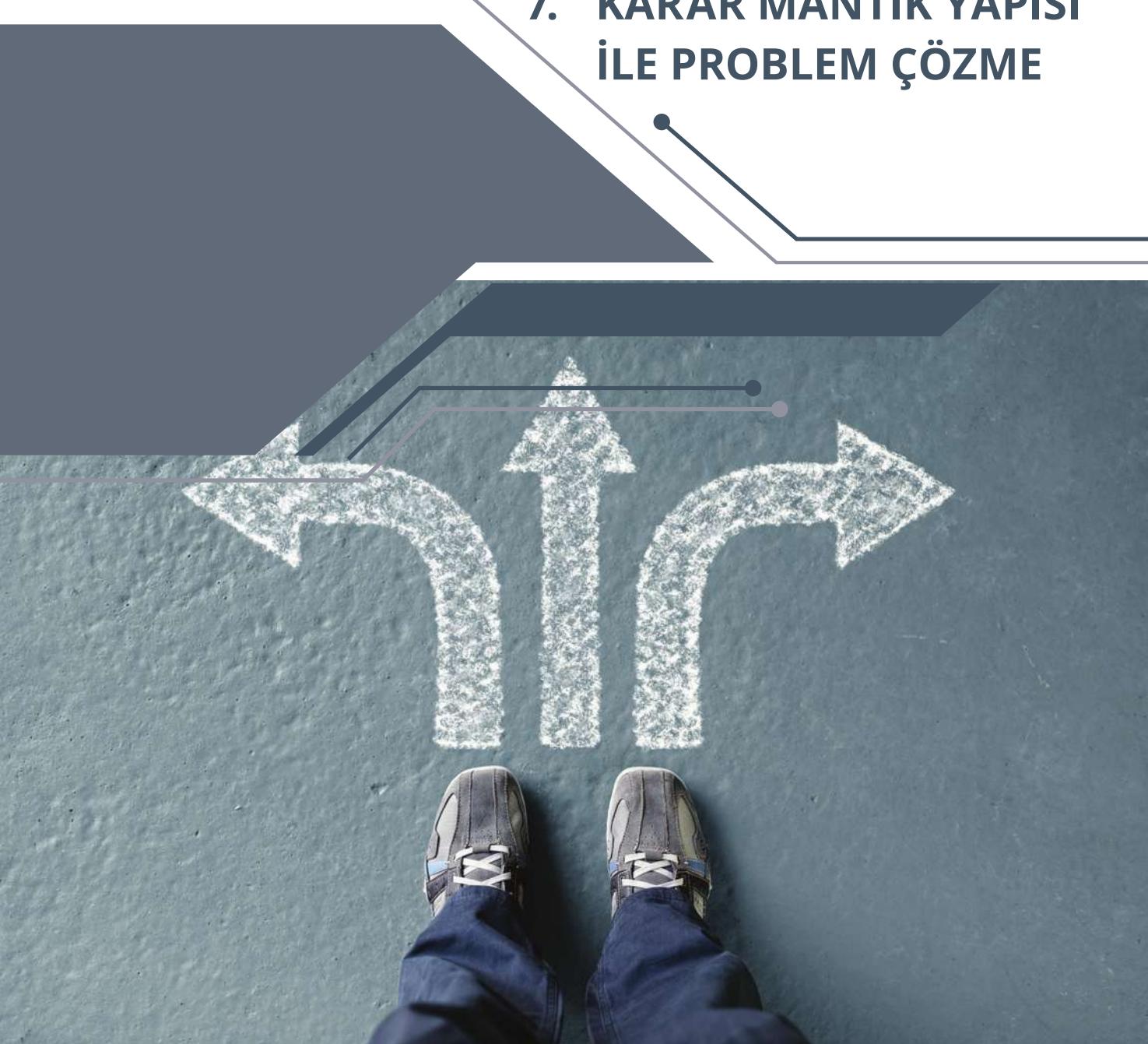
## 6.3 Özet

Bir problem için çözüm süreci yedi adımdan oluşur:

1. Problem Çözme Çizelgesi
2. Etkileşim Çizelgesi
3. GSÇ Çizelgesi
4. Birleşim Çizelgesi ve Veri Sözlüğü
5. Algoritmalar ve Akış Şemaları
6. Çözümün Test Edilmesi

Tüm bu adımlar en hızlı ve doğru sonuca ulaşabilmek için vazgeçilmezdir.

## 7. KARAR MANTIK YAPISI İLE PROBLEM ÇÖZME



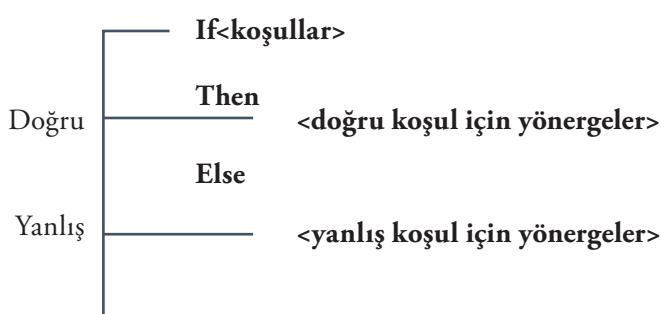
Bu bölümde;

- ✓ Bir probleme çözüm üretEBilmek için karar mantık yapısını kullanabilecek,
- ✓ Problem çözme araçlarını kullanarak çözüm üretEBilecek,
- ✓ İç içe karar yönergeleri oluşturabilecek,
- ✓ Pozitif ve negatif yapıları iç içe kullanabilecek,
- ✓ Mantıksal dönüşümü kavrayabilecek,
- ✓ Doğru karar yapısını oluşturabilecek,
- ✓ Verilen bir dizi kuraldan karar tablosu hazırlayabilecek,
- ✓ Bir karar tablosundan karar mantık yapısı oluşturabileceksiniz.

## 7.1. Karar Mantık Yapısı

Karar yapıları, bilgisayara iki ya da daha fazla seçenek arasından seçim yapmak hakkı tanıyan önemli ve güçlü bir mantık yapısıdır. Eğer karar yapıları olmasaydı bilgisayarlar hızlı bir hesap makinesi olmanın ötesine gidemezdi. Karar yapıları, insanın düşünme tarzına çok uygun olduğu için anlaşılması son derece kolaydır. Karmaşık durumlarda karar vermek zorlaştığı için programcının kararların arkasında yatan nedenleri çok iyi anlayarak tasarım yapması gereklidir. Ayrıca bir karar durumunun çok farklı şekilde ifade edilebilmesi de bu karmaşıklığa neden olabilmektedir.

Karar mantık yapısı, if-then-else (eğer-koşul sağlanırsa-x, değilse y) yönergusonini kullanır. Bu durumda, eğer bir koşul doğru ise belli yönnergeler; değilse farklı yönnergeler çalıştırılabilir. "else" kısmı kullanılmak zorunda değildir; bazen bu durumlarda hiçbir yönerge olmayabilir. Aşağıdaki yapıyı ve satır başlarındaki boşlukları inceleyelim.



**Bu kodlamada yer alan koşul;**

1. Mantıksal bir ifade (AND (VE), OR (YA DA) veya NOT (DEĞİL))
2. İlişkisel operatörleri kullanan bir ifade (<, >, <=, >=, =),
3. Sonucu doğru ya da yanlış çıkan mantıksal bir değişken,
4. Bu üç seçeneğin birleşiminden oluşan bir ifade olabilir.

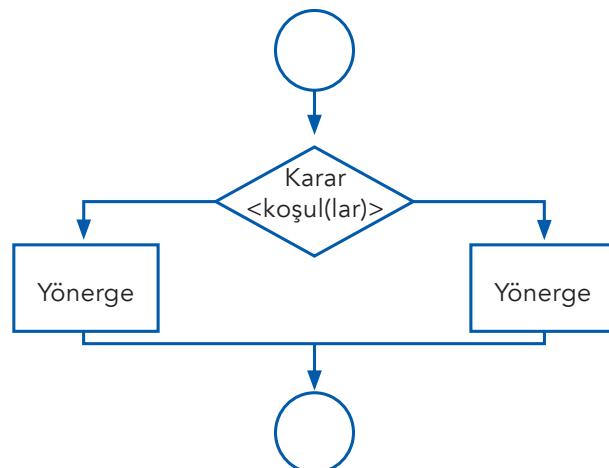
**Koşullara ilişkin açıklamalar aşağıdaki gibi olabilir:**

1.  $A < B$  ( $A$  ve  $B$  sayısal, karakter ya da dizi gibi aynı veri türündedir.)
2.  $X + 5 = Z$  ( $X$  ve  $Z$  sayısal veridir.)
3.  $E < 5 \text{ OR } F > 12$  ( $E$  ve  $F$  sayısal veridir.)
4.  $(A < B) \text{ AND } (X = 10 \text{ OR } Y > 15)$  ( $A$  ve  $B$  sayısal, karakter ya da dizi gibi aynı veri türündedir ve  $X$  ve  $Y$  sayısal veridir.)

Mantıksal operatörler bir ya da daha fazla durumu bağlamak için kullanılır. Örneğin sürücü belgesi alabilmek için 18 yaşını doldurmuş ve bir sürücü kursunu başarı ile tamamlamış olma şartı vardır. Bu örnekteki kontrol, bu iki durumu AND operatörü ile bağlayarak kontrol etmeyi gerektirir.

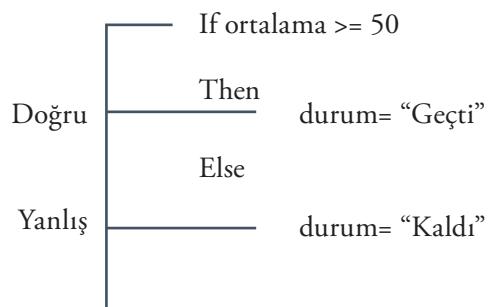
### 7.1.1. Tek Koşullu Yapılar

Tek bir koşulun sorgulandığı döngü yapısı için akış şeması aşağıdaki gibidir:

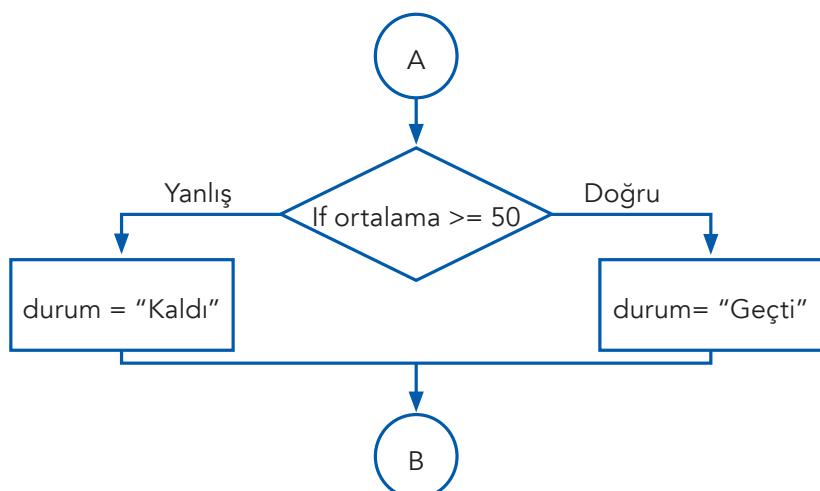


Koşulun sağlanıp sağlanmaması durumuna göre programın akışı değişir ve program, karara uygun yönergelerle çalışmaya devam eder. Hiçbir zaman üçüncü bir seçenek olamaz çünkü karar symbolünden yalnızca iki olasılık çıkabilir. Diğer bir ifade ile belirtilen durum ya doğrudur ya da yanlıştır. Örneğin bir öğrencinin puan ortalamasına bakarak Geçme/Kalma durumunu belirleyen bir program yazalım.

Bu programın algoritması aşağıdaki gibi olacaktır:



Bu programa ait akış şeması ise şu şekildedir:





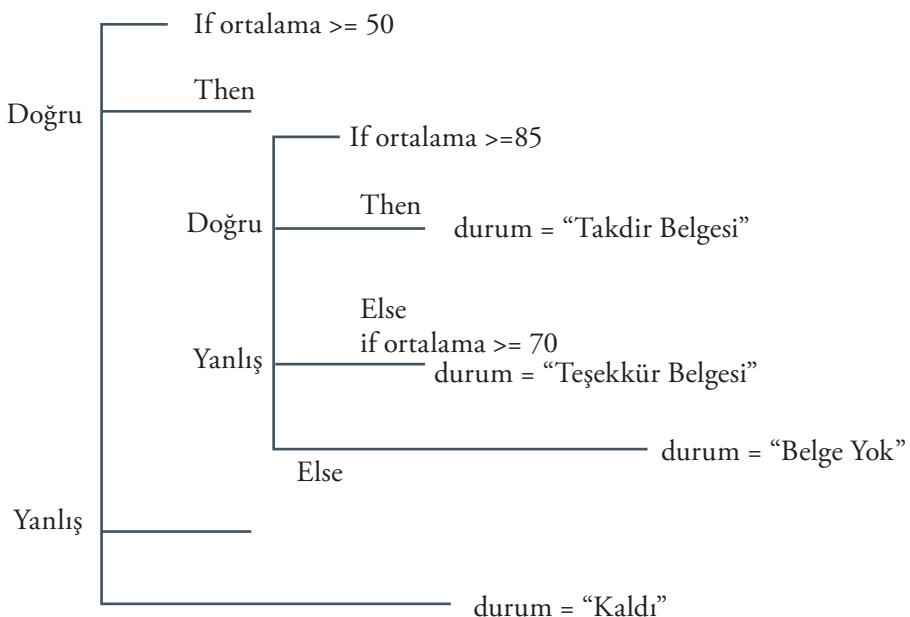
Birden fazla koşulun olduğu durumlar biraz daha karmaşıktır. Bu tür kararlarda durumları birleştirmek için mantık operatörlerinden yararlanılır. Durumlar arttıkça karar yapısı da karmaşık hâle gelir ve "Doğru" ya da "Yanlış" için atılacak adım sayısı da artar.

### 7.1.2. Çok Koşullu Karar Yapıları

Birden fazla karar içeren algoritmaları yazmak için kullanılacak üç tür karar yapısı vardır: Düz mantık, Pozitif Mantık ve Negatif Mantık. Düz mantık bütün koşulların doğrusal olarak işlenmesi anlamına gelir. Bu durumda else ile ifade edilen diğer seçenekleri bulunmaz. Koşul yanlış olduğunda program doğrudan bir sonraki koşula geçer, koşul doğru ise gerekli işlemler yapıldıktan sonra bir sonraki koşula geçilir. Program akışında bütün koşullar sıra ile gözden geçirilir. Diğer yandan, pozitif mantık ile bütün yönergeler işlenmez. Eğer koşul doğru ise bu kararların yönergeleri yerine program akışı modül içinde devam eder. Koşul doğru olduğu sürece akış bu şekilde ancak koşulun yanlış olma durumunda diğer koşula geçilir ve doğru olana kadar devam edilir. Negatif mantık da pozitif mantığa benzemekle beraber burada program akışı karar yanlış olduğu sürece devam eder. Bazı karar durumları bu türlerin bir ya da birkaçını kullanmayı gerektirebilir.

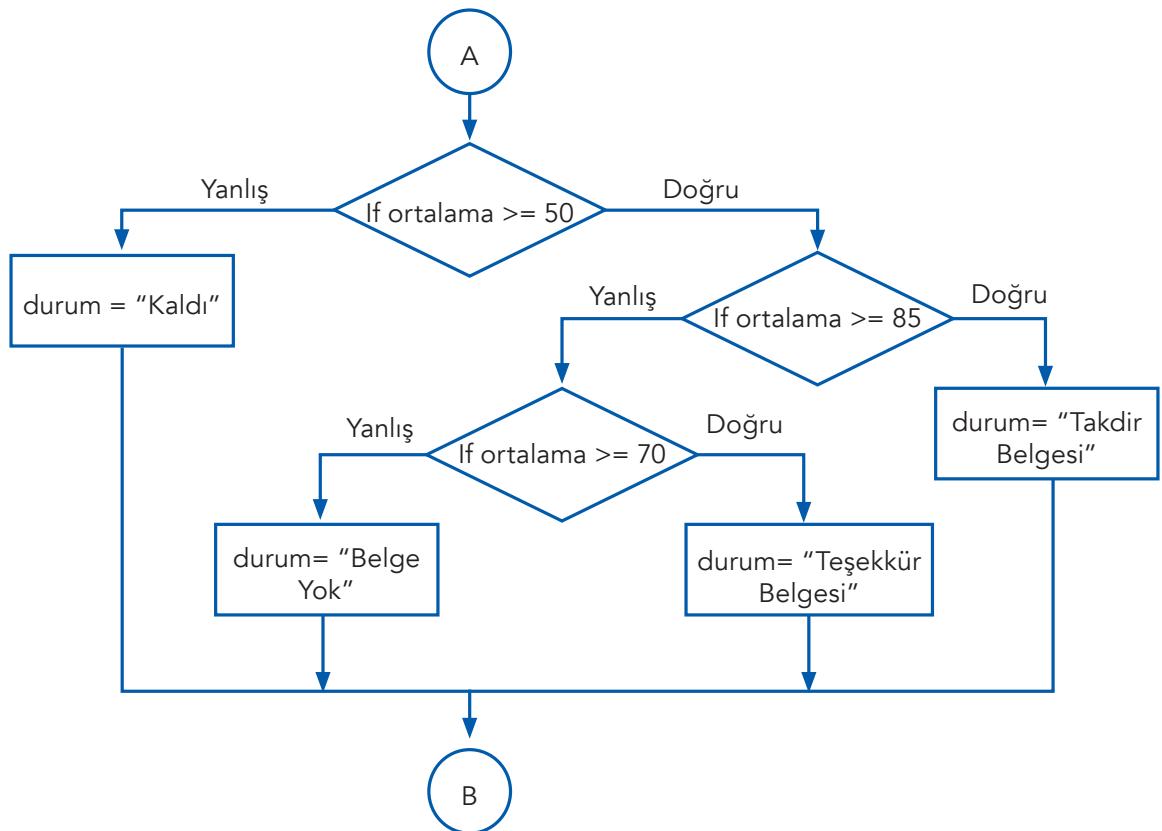
### 7.1.3. İç İçe Karar Yapıları

Çoklu karar yapıları içeren algoritmalarda eğer koşullarını iç içe yazmamız gerekebilir. Bu durumda pozitif ve negatif mantık yapıları kullanılabilir; düz mantık yapısı kullanılmaz. Aşağıdaki algoritma, öğrencinin puan ortalamasına göre Geçme/Kalma durumunu kontrol ettikten sonra, geçiyorsa öğrencinin belge alma durumunu belirlemektedir. Bu durumda bu örneğin algoritmasını aşağıdaki biçimde düzenleyebiliriz.





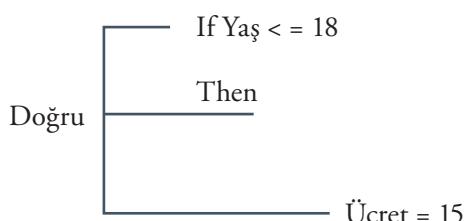
Akiş şeması ise şu şekilde olacaktır:



## 7.2. Düz Mantık Kullanımı

Düz mantık ile çalışan kararlarla bütün koşullar test edilir. Bir koşulun test edilmesi, "Doğru" ya da "Yanlış" sonuç elde etmek için durumun işlenmesidir. Düz mantık çözümlerin içinde en yetersizi, çözüm olarak nitelendirilebilir çünkü bütün koşulların test edilmesi, programın çalışmasını da uzatır. Bazen birbiri ile ilişkisiz durumlar olduğunda ya da tüm durumların kontrolü gerekiğinde bunu kullanmak mecburi olmaktadır. Düz mantık yapısı, genellikle diğer karar yapıları ile bir arada kullanılır.

Şimdi düz mantık ile çözülebilen bir örneği inceleyelim. Tiyatro biletleri alırken bilet fiyatı yaşa göre değişmektedir. Yaşı 18'den küçük olanlar için bilet ücreti 15 TL; yaşı 18'den büyük ve 65'ten küçük olanlar için 20 TL ve yaşı 65'ten büyük olanlar için 10 TL olarak belirlenmiştir. Bu durumda tiyatro seyircilerinin yaşlarına göre bilet almalarına olanak sağlayan bir çözüm geliştirmemiz gereklidir. Bu problemin çözümü için algoritma şu şekildedir:

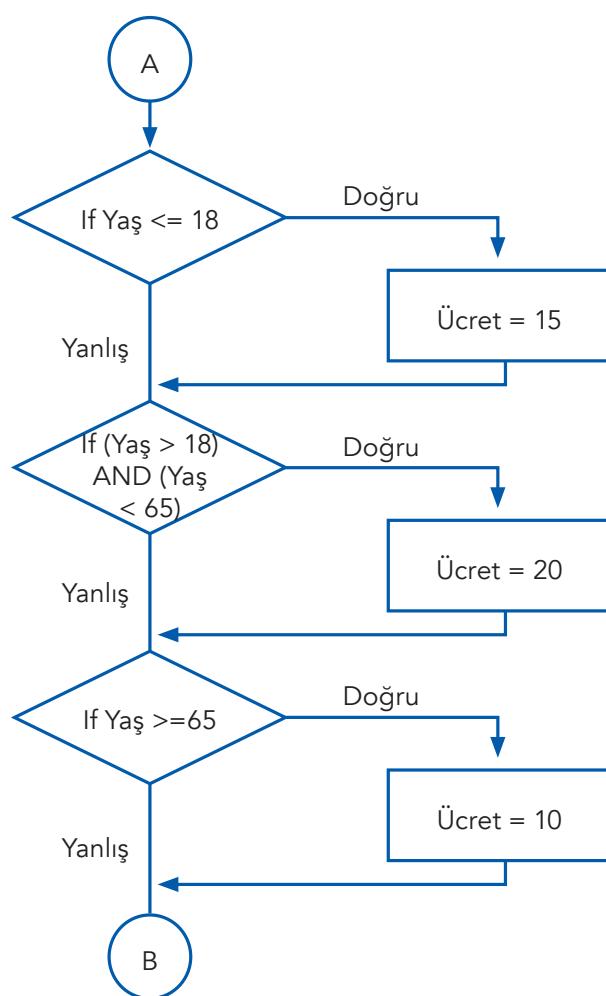




If (Yaş > 18) AND (Yaş < 65)  
Then  
Doğru  
Ücret = 20

If Yaş >= 65  
Then  
Doğru  
Ücret = 10

Bu problemin akış şeması ise aşağıdaki gibidir.

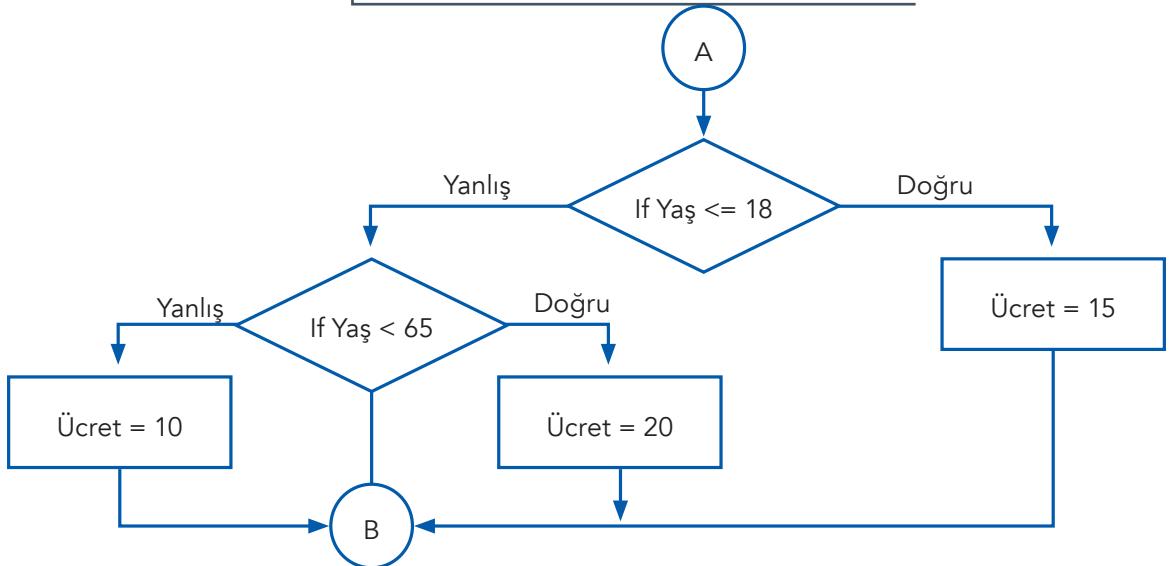
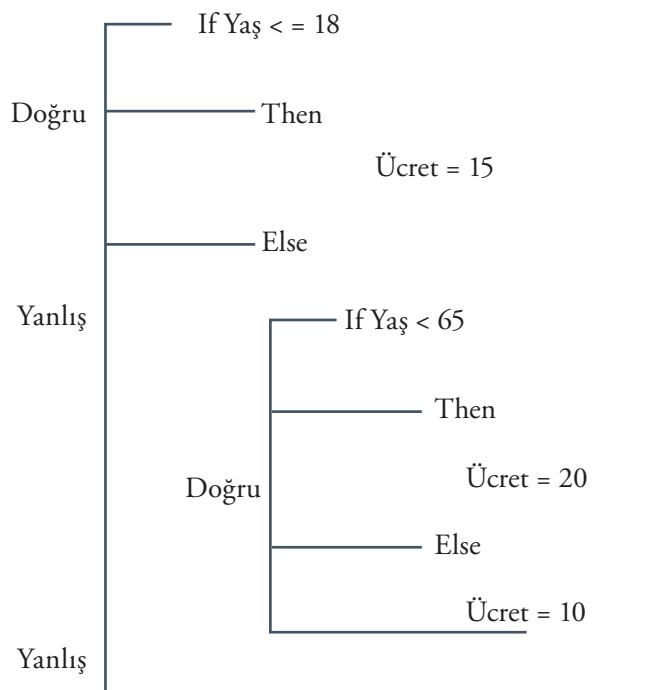




Algoritma ve akış şeması içerisinde “else” kullanılmadığına dikkat ediniz. Kontrol edilen koşul yanlış ise doğrudan bir sonraki koşul kontrol edilmektedir ki bu yüzden “değilse” durumuna gerek yoktur. Kontrol edilen durum doğru olsa ve işlem yapılsa bile bütün koşullar yine de sıra ile kontrol edilmektedir. Program çoğu durumda gereksiz kontroller yapmaktadır.

### 7.3. Pozitif Mantık Kullanımı

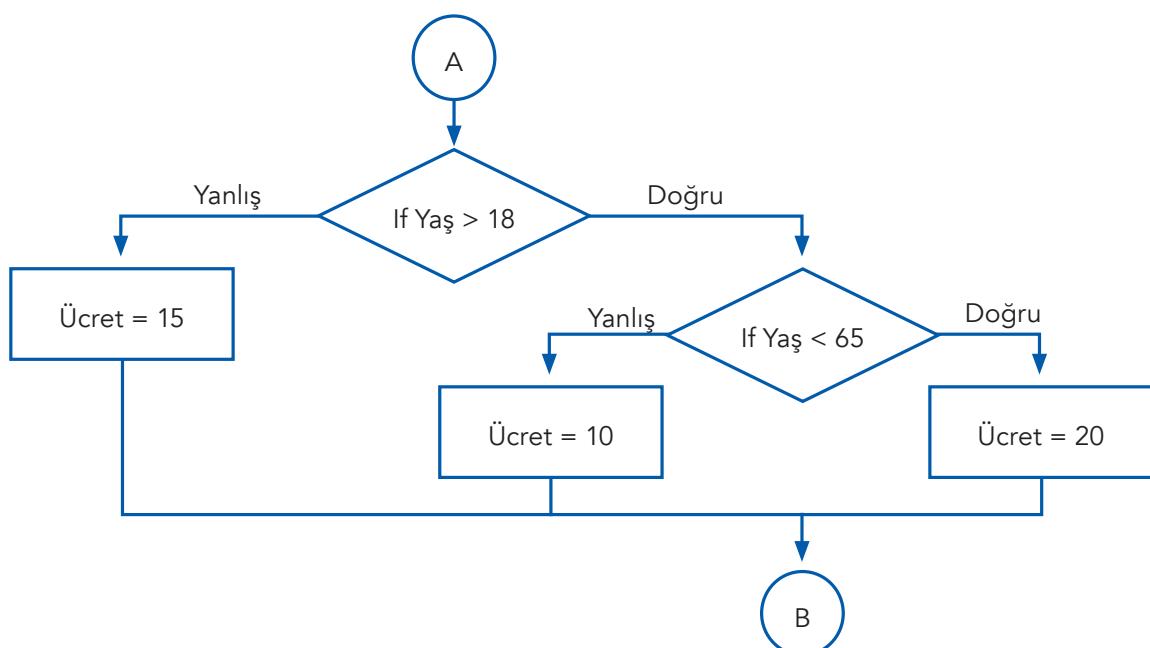
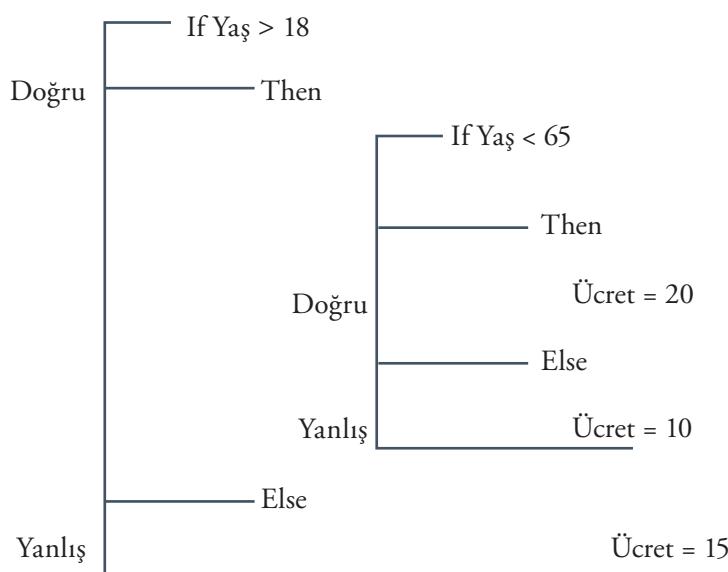
Düşünme biçimimize en çok benzeyen yapı olması nedeni ile pozitif mantık kullanımı en kolay yapıdır. Pozitif mantık her zaman iç içe If/Then/Else yapısını kullanır. Bu yapı; kullanıldığından genellikle bilgisayardan, koşulun doğru olması durumunda işlem yapması, yanlış olması durumunda farklı bir karar vermesi beklenir. Böylece daha az adımda karar verilebilir. Bir önceki problemin bu yaklaşım ile çözüm algoritmasını ve akış şemasını inceleyelim.





## 7.4. Negatif Mantık Kullanımı

Genellikle tersten düşünmediğimiz için negatif mantık yapısı, kurgusu, programcılara en zor gelen yapıdır. Negatif mantık kullanıldığında bilgisayardan, koşulun doğru olması durumunda farklı yönereleri takip etmesi beklenir. Negatif mantık kullanmak, kontrol edilecek koşul sayısını azalttılarından programı daha anlaşıllır kılarak geliştirir.



## 7.5. Mantık Dönüşümleri

Bazen programın yeterliğini ve okunabilirliğini artırmak için karar mantık yapılarını, pozitiften negatife ya da tam tersine dönüştürmek gerekebilir. İşlenmesi gereken yönerelerin doğası ve sayısı gereği ya da işlenecek yönerge olmadığı için bazen çözümün yapısı, kullandığımız mantık yapısına uymaz.

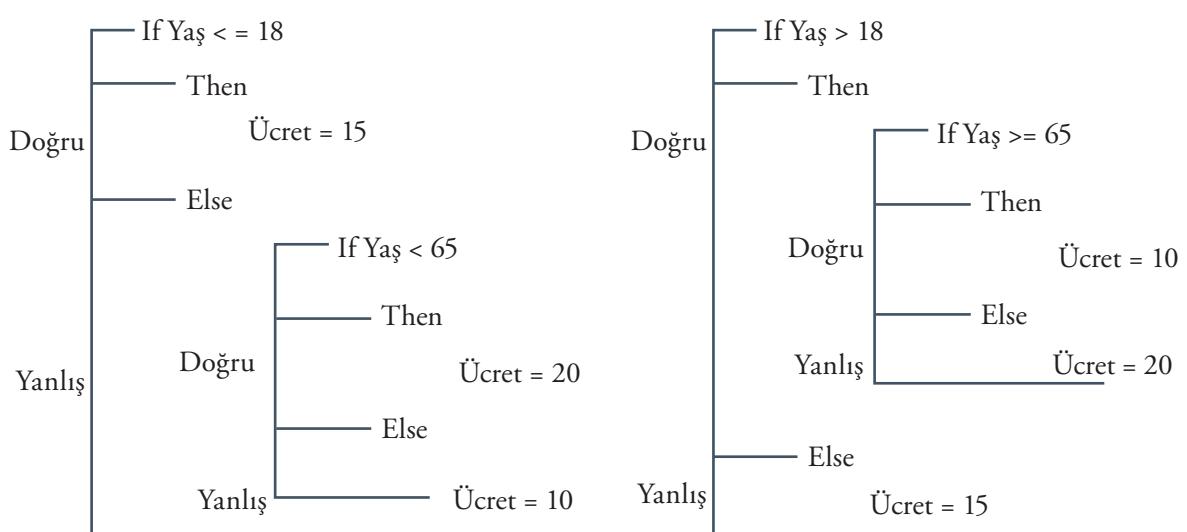


Karar verilirken koşulun yanlış olduğunda uygulanacak yönergeler olmasa bile her zaman doğru olduğu durum için uygulanacak yönergeler olmalıdır. Eğer doğru olması koşuluna uygun yönerge yoksa mantık yapısını dönüştürmek yerinde olacaktır.

Pozitif mantıktan diğerine ya da tam tersi biçimde dönüşüm yapmak için aşağıdaki kuralları uygulayınız.

1. Tüm  $<$  koşullarını  $\geq$  ile değiştirin.
2. Tüm  $<$  koşullarını  $>$  ile değiştirin.
3. Tüm  $>$  koşullarını  $\leq$  ile değiştirin.
4. Tüm  $\geq$  koşullarını  $>$  ile değiştirin.
5. Tüm  $=$  koşullarını  $\neq$  ile değiştirin.
6. Tüm  $\neq$  koşullarını  $=$  ile değiştirin.
7. Then kapsamındaki tüm yönergeleri else kapsamı ile karşılıklı olarak değiştirin.

Bu dönüşüm daha önce incelediğimiz problemin çözümü için şu biçimde yapılmaktadır:

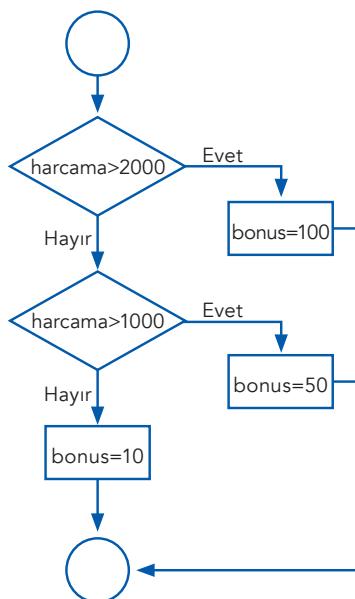


## 7.6. Hangi Mantık Yapısı?

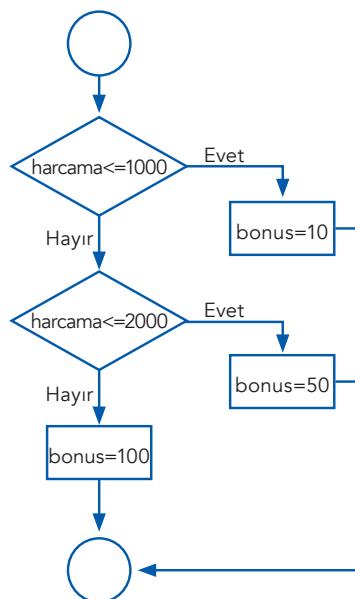
Bir problemi çözmek için hangi karar yapısını seçeceğimize nasıl karar vereceğiz? Bunun en kolay yolu her 3 yapı için çözümü yazmak ve bu çözümler içinden en hızlı, kolay algılanan ve en az koşulla çalışanı seçmektir. Her zaman aynı yapıyı kullanmak ya da problemden istenildiği sıradaki yönergeleri kullanarak çözüm üretmek, sıkça başvurulan yollardır ancak bu yaklaşım her zaman en etkili çözüm ile sonuçlanmamayabilir.

Şimdi bir problemi çözmenin 4 farklı yolunu inceleyelim. Harcanan para miktarına göre belli sayıda “bonus” verilmesi için farklı mantık yapılarında akış şemaları şu şekilde dir:

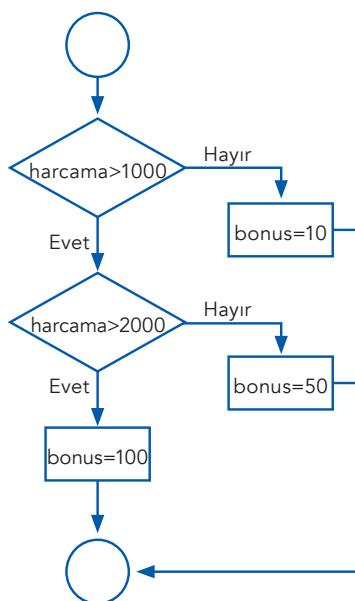
**Pozitif Mantık 1**



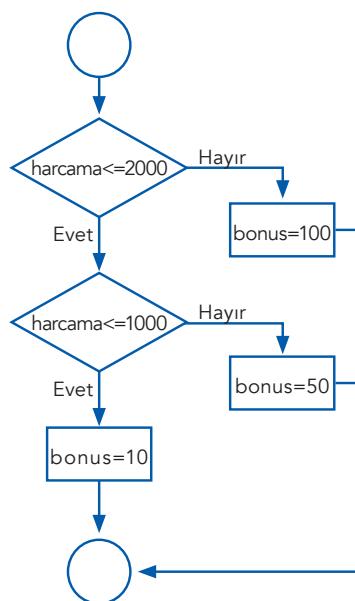
**Pozitif Mantık 2**



**Negatif Mantık 1**



**Negatif Mantık 2**



**Mantık yapısına ilişkin seçim konusunda düşünürken aşağıdaki sorulara yanıt aramak önemlidir.**

1. Hangi mantık yapısı, programı daha anlaşılır kılmaktadır?
2. Hangi mantık yapısını yönetmek ve değiştirmek daha kolaydır?
3. Veriye ilişkin fazla bilgi yokken en az sayıda koşul kontrolü yapan, hangi mantık yapısıdır?
4. Veri sunulduğunda en az koşul kontrolünü yapan mantık yapısı hangisidir?

Karar süreci sonunda iki yapının eşit sonuç verdiği görürseniz herhangi birini seçebilirsiniz.

## 7.7. Karar Tabloları

Birden fazla koşul ve çoklu eylem içeren bir probleminiz var. Bu durumda farklı koşullar için bütün eylemleri keşfetmek ve hatta koşul birlikte olmasına göre olası eylemleri belirlemek zor ve karmaşık bir süreç olur. Bu süreci basitleştirmenin yolu karar tablosu kullanmaktır. Karar tabloları, problemi birlikte çözdüğünüz kişi ile iletişim sağlamak ve süreci anlaşılır kılmak için çok kullanışlı bir araçtır. Karar tablosu, problem çözme mantığını tablo biçiminde gösteren bir araçtır. Akış şemalarının alternatifi de olabilir.

**Bir karar tablosu 4 bölümden oluşur:**

1. Koşullar: Tablonun sol üst bölümüne olası tüm koşullar yazılır.
2. Eylemler: Sol alt bölümde, şartların birleşimleri sonucu yapılan tüm olası eylemler listelenir.
3. "Doğru" ve "Yanlış" koşulların birleşimleri: Kurallar, sağ üst bölümde oluşturulur. Şartın durumuna bağlı olarak şartın karşılığında olasılıkları ifade eden gösterimler kullanılır. Örneğin Evet, Hayır'ı temsil için E ve H harfleri kullanılır.
4. Durumların birleşimlerine ilişkin gerçekleşecək eylemler: Sağ alt bölümde ise verilen bir kural için geçerli olan eylemlerin gösteriminde "X" simgesi kullanılır.

"Doğru" ve "Yanlış" koşulların olası bileşkelerini gösteren, her biri küçük bölgelere ayrılmış, temelde dört bölüme ayrılmış bir dikdörtgen düşünebilirsiniz. Olası bileşkelerin sayısını  $2^{\text{koşul sayısı}}$  şeklinde hesaplayın. Örneğin 2 koşulunuz varsa  $2^2=4$  olası bileşkeniz; 4 koşulunuz varsa  $2^4=16$  olası bileşkeniz var demektir. Bu karar tablosuna bakarak daha sonra akış şeması oluşturulabilir.

Aşağıda örnek bir karar tablosu görüyorsunuz.

*İndirim Hesaplama Süreci İçin Karar Tablosu*

İndirim Hesapla	KURALLAR																																		
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2			
ŞARTLAR																																			
Satin Alma <100\$	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H			
Özel Teklif	E	E	E	E	E	E	E	H	H	H	H	H	H	H	H	E	E	E	E	E	E	E	H	H	H	H	H	H	H	H	H	H			
İndirim <2\$	E	E	E	E	H	H	H	H	E	E	E	H	H	H	H	E	E	E	H	H	H	H	E	E	E	H	H	H	H	H	H	H			
İndirim Sonrası >45\$	E	E	H	H	E	E	H	H	E	E	H	H	E	E	H	H	E	E	H	H	E	E	H	H	E	E	E	H	H	E	E	H			
7 gün içinde ödeme	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H	E	H			
FAALİYETLER																																			
2\$ indirim	X	X	X	X																															
%5 indirim									X	X	X	X	X	X	X	X																			
%7,5 indirim						X	X	X	X																										
%8 indirim																			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Ekstra %1 indirim						X			X			X						X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	

<http://slideplayer.biz.tr/slide/2872575/>



### Düşünelim/Deneyelim

1. Aşağıdaki karar tablosu için koşulları, eylemleri, doğru ve yanlış koşulların birleşimlerini ve kuralları belirleyiniz.

*Karar Tablosu*

Maaş Hesapla	KURALLAR					
	1	2	3	4	5	6
ŞARTLAR						
Çalışan tipi	M	S	M	S	M	S
Çalışılan saatler	<40	<40	40	40	>40	>40
FAALİYETLER						
Taban ücreti öde	X		X		X	
Saatlik ücreti hesapla		X		X		X
Fazla mesai ücreti hesapla					X	X
Devamsızlık raporu üretme		X				

2. Bir öğrencinin harf notunu hesaplayan program için 4 farklı akış şeması ve algoritma oluşturunuz (Düz mantık kullanmayınız).

90 -100 = A

80-89 = B

70-79 = C

60-69 = D

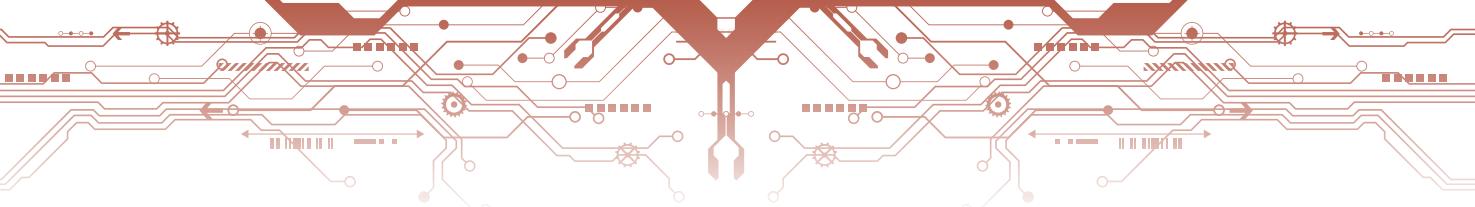
0-59 = F

## 8. DÖNGÜ YAPISI İLE PROBLEM ÇÖZME



Bu bölümde;

- ✓ Bir probleme çözüm üretEBilmek için döngü yapısını kullanabilecek,
- ✓ Döngü yapısını karar yapıları ve doğrusal mantık ile birleştirebilecek,
- ✓ Problem çözme araçlarını kullanarak çözüm üretEBilecek,
- ✓ Sayaç ve birikeçleri problem çözümünde kullanabilecek,
- ✓ Problemin çözümü için iç içe döngüler oluşturabilecek,
- ✓ Döngü için kullanılabilecek 3 farklı türü ayırt edebilecek,
- ✓ Öz yinelemeyi kullanabileceksiniz.



## 8.1. Döngü Mantık Yapısı

Problem çözüm sürecinde kullanılacak üçüncü bir karar yapısı, döngü yapısıdır. Bu yapı, tekrarlayan bir yapıdır. Çoğu problem, aynı işlemi farklı verileri kullanarak tekrar etmemi gerektirir. Bu yüzden bu yapı son derece önemlidir. Bu yapı kullanılarak isim listesinin alfabetik sıraya dizilmesi ya da ödeme kayıtlarının araştırılması gibi tekrarlayan işlemler yapılabilir. Bu yapı, karar yapısından daha zor anlaşılan bir yapıdır ama bu yapının kullanımını daha kolaydır. Bu yapıdaki zorluk, hangi yönergelerin tekrarladığını belirleme sürecidir.

Tekrarlayan işlemleri yürütmenin yanı sıra döngü yapısı işlemi, çözüm basamakları içerisinde önceki basamaklara yönlendirmek amacıyla da kullanılır. Bu durumda GoTo (Git) komutu kullanılabilir. Bu komut numarası verilen satırı yönlendirme işlemi yapar. GoTo komutu yerine döngü kullanmak, programın okunabilirliğini de artırmaktadır. Her biri If/Then/Else ya da GoTo komutlarını kullanan farklı yönerge setleri ile oluşturabilen 3 farklı döngü yapısı vardır. Ancak If/Then/Else karar mantık yapısıdır ve bir döngüyü yönetmek amacıyla kullanılmamalıdır.

Döngü türlerinden biri, bir koşul doğru olduğu sürece işlemleri tekrarlayan While/WhileEnd döngü yapısıdır. Bu yapı, koşul yanlış olduğunda döngü işlemleri tekrarlamayı durdurur. Bir diğeri bir koşul yanlış olduğu sürece ya da doğru olana kadar işlemleri tekrarlayan Repeat/Until döngü yapısıdır. Üçüncü bir tür ise otomatik sayaç döngüsüdür. Bu yapıda bir değişkene ilk değer atanır, belirlenen sabit bir biçimde sürekli arttırılarak hedef sayıdan büyük ya da eşit olana kadar işlemlerin tekrarlanması sağlanır. Problemin çözüm sürecine göre bu 3 türden herhangi biri tercih edilebilir. Her üç tür için oluşturulacak algoritma ve akış şeması birbirinden farklıdır. Programın okunabilirliğini artırmak için girinti kullanmak son derece önemlidir.

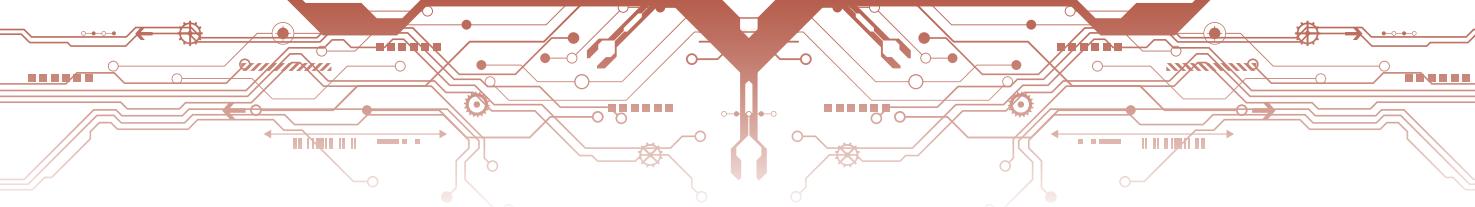
Döngü yapısı kapsamında birkaç standart işlem türü vardır. Bunlardan ikisi; sayaç kullanarak artırma ya da azaltma yapmak, diğeri ise birikeç kullanarak toplam ya da işlem sonucunu hesaplamaktır. Her iki durumda değişkene atanın değer değişir ve yine aynı değişkende tutulur. Sayaç ve birikeç arasındaki temel fark, eklenen ya da çıkarılan değerdir. Sayarken değer sabittir ancak biriktirirken değişebilir. Her iki durumda da işlemlere başlamadan önce değişkene 0 (sıfır) değeri atanır ki bu işleme “**ilk değer atama**” denir.

## 8.2. Arttırma

Sayma ya da artırma işlemi bir değişkene sürekli sabit bir değer ekleyerek gerçekleştirilir. Bu işlemi gerçekleştirmek için bir atama satırı gereklidir. Örneğin

$$\text{sayaç} = \text{sayaç} + 1 \text{ ya da } s = s + 1$$

değişkeni her işlem adımında değişken değerine bir ekleyerek değerini artırır. Atama işlemi “= (eşit)” işaretinin sağ tarafındaki işlem sonucunun sol taraftaki değişkene yüklenmesini sağlar. Arttırma işlemi bir atama yönergesidir. Bu yönerge programa, maddelerin ve insanların derece ve benzeri bir dizi olgu sayısını problemin çözümünün bir parçası olarak hesaba katmasını sağlar. Arttırma işlemi yapılmırken kullanılan atama yönergesini inceleyelim. Kullandığımız değişken, atama ifadesinin her iki tarafında da yer almaktadır. Değişken + işaretinden sonra gelen değer kadar artırılır. Bu yönerge, sayacın değerini alır ve bir ekleyerek değiştirir. Böylece değişkenin yeni bir değeri olur. Artırmak için kullanılan değer 1, 2, 3 gibi pozitif sayılar arasından seçilen farklı bir değer olabilir hatta artırmak yerine azaltmak istenirse negatif değerler de kullanılabilir. Bu örnekte sayaç değişkenine, döngüye girmeden önce “0” değeri verilerek ilk değer ataması yapılmıştır.



### 8.3. Biriktirme

Programın diğer bir görevi ise bir grup sayıyı toplayıp biriktirerek toplamı ya da sonucu bulmaktadır. Biriktirme işlemi arttırma işlemi ile çok benzerdir ancak her seferinde toplama ya da sonuca eklenen değer sabit olmayabilir. Biriktirme işlemi için kullanılan yönerge şu şekildedir:

$$\text{toplam} = \text{toplam} + \text{değişken} \text{ ya da } t = t + d$$

Örneğin toplam satışı bulmak için kullanılacak yönerge şöyledir:

$$\text{toplam satış} = \text{toplam satış} + \text{satış}$$

Arttırma örneğinde olduğu gibi kullanılan değişken atama yönergesinde her iki tarafta da yer alır ancak eklenen değer (bu örnekte satış değişkeni) her defasında değişebilir. Diğer bir ifade ile, birikeç ile toplam ya da sonuç değeri içeren bir değişkene yeni bir değer eklenmektedir.

Bu örneklerde döngüye girmeden önce toplam satış ya da toplam değerine ilk değer olarak "0" atanmalıdır. Bir dizi sayının çarpımını hesaplamak, iki durum hariç, bu sayıların toplamını bulmaya benzemektedir: (1) "+" simbolü yerine "\*" kullanılır, (2) sonuç değişkenine atanan ilk değer "0" değil "1" olmalıdır.

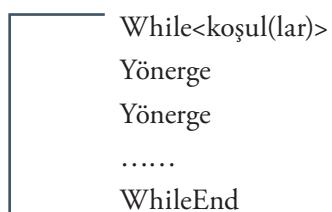
$$\text{sonuç} = 1$$

$$\text{sonuç} = \text{sonuç} * \text{sayı}$$

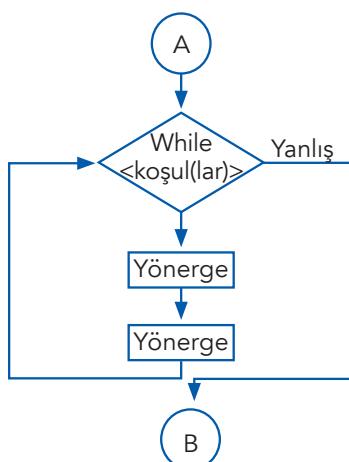
Bir değişken tekrarlı yapı içerisinde yeniden kullanılabilir.

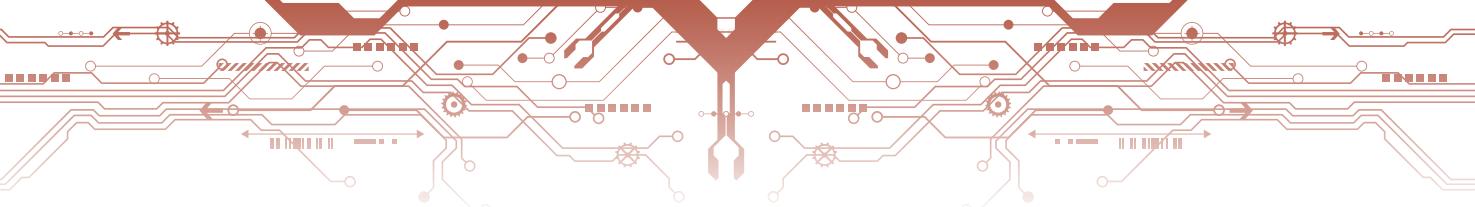
### 8.4. While/While End Döngüsü

Ele alacağımız ilk yapı, While/While End döngü yapısıdır. Bu döngü yapısı bilgisayara, koşul doğru olduğu sürece işlemleri tekrarlanması gerektiğini belirtir. Algoritma yapısı şu şekildedir:



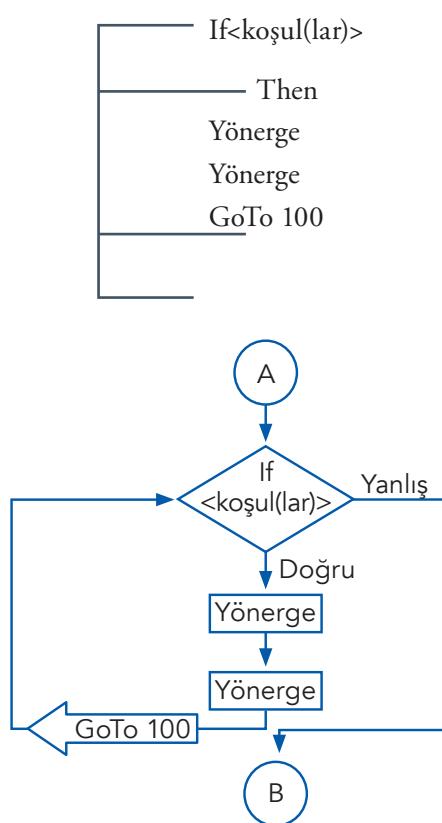
While/While End döngü yapısı tasarladığınız zaman, algoritmayı daha anlaşılır kılmak için girinti ve köşeli ayraçlardan yaralanabilirisiniz. Bu yapıya ilişkin akış şeması ise şöyledir:





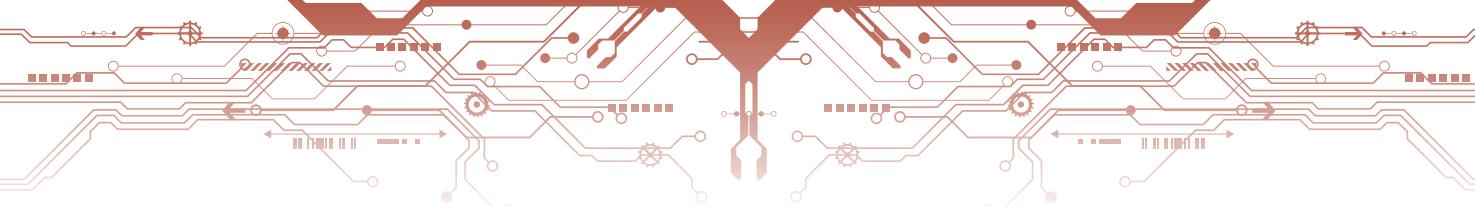
Döngünün başlangıcında, döngü içerisinde yer alan yönergelerin işlenip işlenmeyeceğine ilişkin karar vermek için koşul kontrol edilir. Eğer koşul yanlış olursa döngü içerisindeki hiçbir yönerge işleme alınmaz. Eğer koşul doğru ise döngü içerisindeki tüm yönergeler çalıştırılır ve tekrar döngünün başına dönülür. Döngü, koşul durumu yanlış olana kadar devam eder.

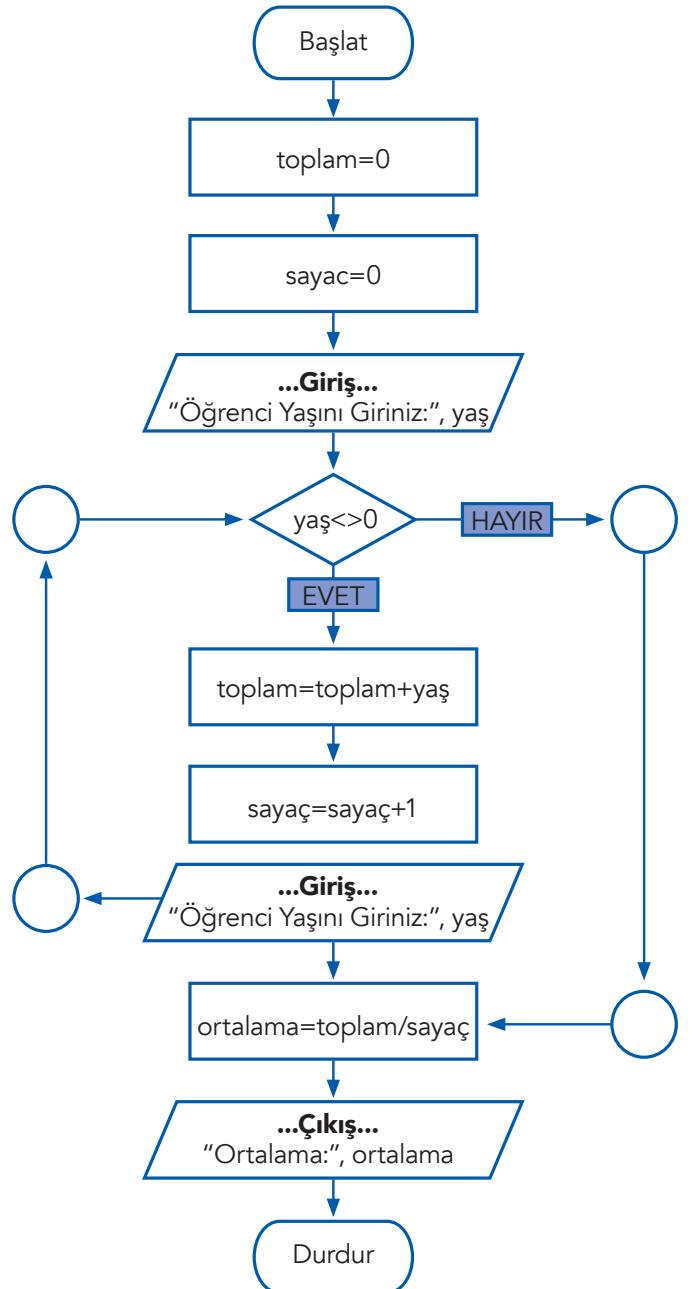
Aşağıda aynı döngünün If/Then/Else karar yapısına eş değer algoritması ve akış şeması görülmektedir.



Yönergelerin kaç kez tekrarlanacağıının belli olmadığı zaman ya da döngüdeki yönergelerin işleme alınmayacağı durumlar olduğunda, While/WhileEnd döngü yapısı kullanılır. Bu durumlarda döngüye girerken kontrol edilen durum yanlışır.

Örneğin bir sınıfındaki öğrencilerin yaş ortalamasını hesaplayan algoritma ve akış şeması nasıl olmalıdır?



Algoritma	Akış Şeması
<ol style="list-style-type: none"> <li>1. Başla.</li> <li>2. <math>toplam=0</math></li> <li>3. <math>sayaç=0</math></li> <li>4. Yaşı Gir.</li> <li>5. Yaş 0'a eşit değilse dön  <math>toplam=toplam+yaş</math>  <math>sayaç=sayaç+1</math>  Yaşı Gir.  Döngüyü Bitir.</li> <li>6. <math>ortalama=toplam/sayaç</math></li> <li>7. Ortalamayı yazdır.</li> <li>8. Bitir.</li> </ol>	 <pre> graph TD     Start([Başlat]) --&gt; ToplamInit[toplam=0]     ToplamInit --&gt; SayacInit[sayaç=0]     SayacInit --&gt; Giris1{...Giriş... "Öğrenci Yaşını Giriniz:", yaş}     Giris1 --&gt; Decision{yaş&lt;&gt;0}     Decision -- HAYIR --&gt; End([Durdur])     Decision -- EVET --&gt; ToplamAdd[toplam=toplam+yaş]     ToplamAdd --&gt; SayacAdd[sayaç=sayaç+1]     SayacAdd --&gt; Giris2{...Giriş... "Öğrenci Yaşını Giriniz:", yaş}     Giris2 --&gt; OrtalamaCalc[ortalama=toplam/sayaç]     OrtalamaCalc --&gt; Cikis1{...Çıkış... "Ortalama:", ortalama}     Cikis1 --&gt; End     </pre>

## 8.5. Repeat/Until Döngüsü

Bir diğer döngü türü Repeat/Until yapısıdır. Bu yapı bilgisayara, yönergeleri Repeat ve Until komutları arasında bir koşul doğru olana kadar yapmasını iletir. Bu yapı ile While yapısı arasında iki temel fark vardır:

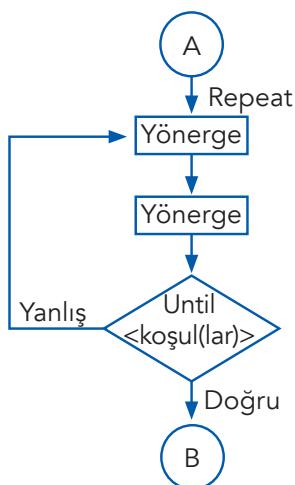
- (1) While döngü yapısında döngü koşul doğru olduğu sürece çalışır, oysa Repeat yapısında döngü koşul doğru olduğunda durur,
- (2) While döngüsünde koşul sürecin başında; Repeat yapısında ise döngünün sonunda kontrol edilir.

Böylece döngü içerisindeki yönergeleri, koşul kontrol edilmeden önce en az bir kere çalıştırılmış olur. Bu yüzden koşulu kontrol etmeksizin en az bir kez yapılması gereken işlemler varsa Repat/Until döngü yapısının tercih edilmesi gereklidir. Repeat/Until yapısının algoritması ve akış şeması şu şekildedir:

```

Repeat
Yönerge
Yönerge
...
Until<koşul(lar)>

```

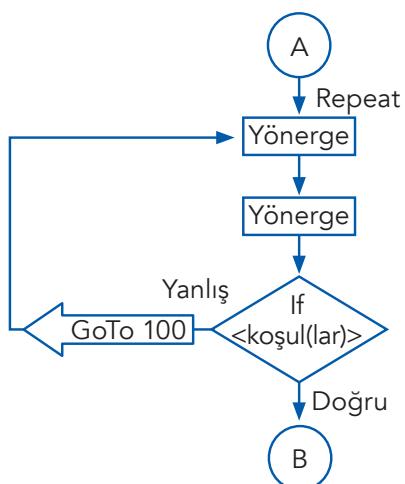


Şimdi, bu yapıya eş değer If/Then/Else yapısını inceleyelim.

```

Yönerge
Yönerge
If<koşul(lar)>
Then
Continue
GoTo 100
Doğru
Yanlış

```



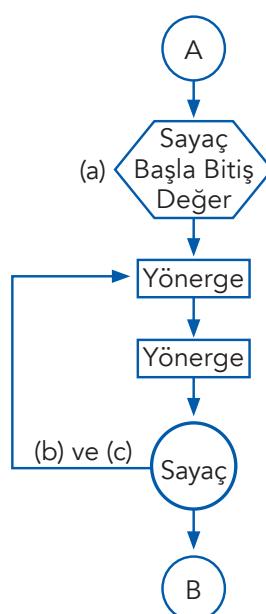
Algoritma	Akış Şeması
<ol style="list-style-type: none"> <li>1. Başla.</li> <li>2. <math>toplam=0</math></li> <li>3. <math>sayaç=0</math></li> <li>4. Yaşı Gir.</li> <li><b>5. Tekrarla.</b>  <math>toplam=toplam+yaş</math>  <math>sayaç=sayaç+1</math>  Yaşı Gir.  Yaş 0 Olana Kadar </li> <li>6. <math>ortalama=toplam/sayaç</math></li> <li>7. Ortalamayı yazdır.</li> <li>8. Bitir.</li> </ol>	<pre> graph TD     Baslat([Başlat]) --&gt; Toplam0[toplam=0]     Toplam0 --&gt; Sayac0[sayaç=0]     Sayac0 --&gt; Giris1{...Giriş... "Öğrenci Yaşını Giriniz:", yaş}     Giris1 --&gt; Toplamplusyaş1[toplam=toplam+yaş]     Toplamplusyaş1 --&gt; Sayacplus1[sayaç=sayaç+1]     Sayacplus1 --&gt; Giris2{...Giriş... "Öğrenci Yaşını Giriniz:", yaş}     Giris2 --&gt; Yaş0{yaş=0}     Yaş0 -- HAYIR --&gt; Ortalama[ortalama=toplam/sayaç]     Ortalama --&gt; Çikis2{...Çıkış... "Ortalama:", ortalama}     Çikis2 --&gt; Durdur([Durdur])     Yaş0 -- EVET --&gt; Toplamplusyaş1   </pre>

## 8.6. Otomatik Sayaç Döngüsü

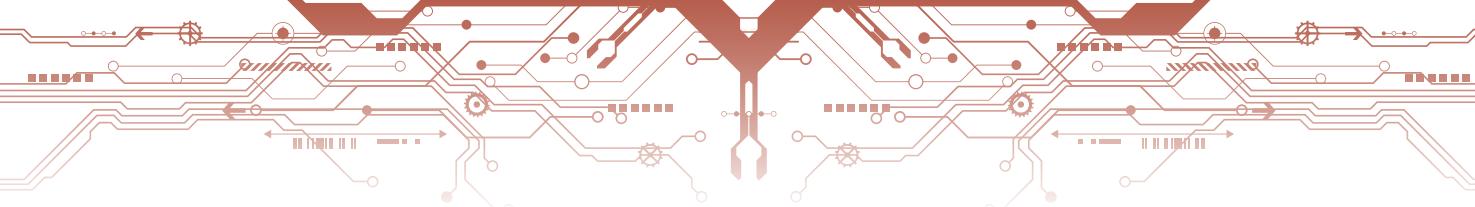
Döngü yapısının üçüncü türü otomatik sayıç döngüsüdür. Bu yapıda döngünün her tekrarında bir değişkenin değeri arttırılır ya da azaltılır. Arttırma değeri yönerge tarafından belirtilir. Bu döngüyü tasarlayan programcı, döngü tekrar sayısını kontrol etmek için değişken olarak tanımlanan bir sayıç tutar. Döngü, bu sayıç bitirme sayısından büyük olana kadar tekrar eder. Başlama ve bitiş değerleri ile artırma değeri sabit; değişken ya da hesaplamalı olarak değişen bir değer olabilir. Döngü tamamlandıktan sonra tamamen değişimdir. Döngüye ilişkin koşul, döngünün başlangıcında ya da bitişinde olabilir.

Otomatik sayaç döngüsüne ilişkin algoritma ve akış şeması şu şekildedir.

```
Loop: sayaç = başlangıçtan bitişe kadar artacak değer  
Yönerge  
Yönerge  
...  
Loop-End: sayaç
```



Koşulun, döngünün başında ya da sonunda kontrol edilmesine göre akış şeması da değişir.



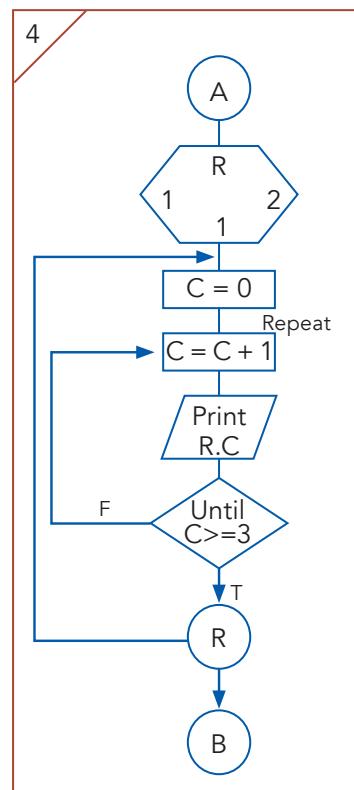
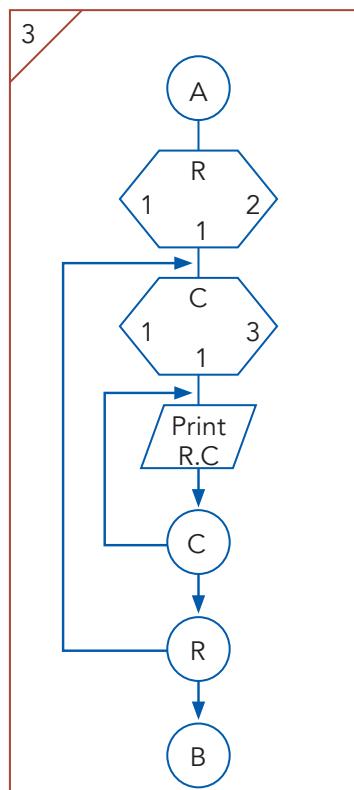
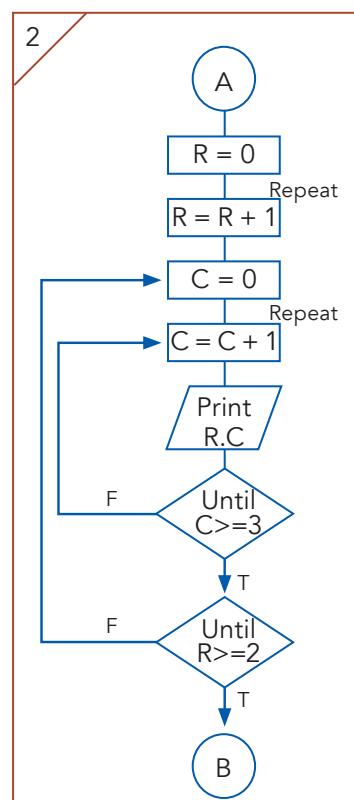
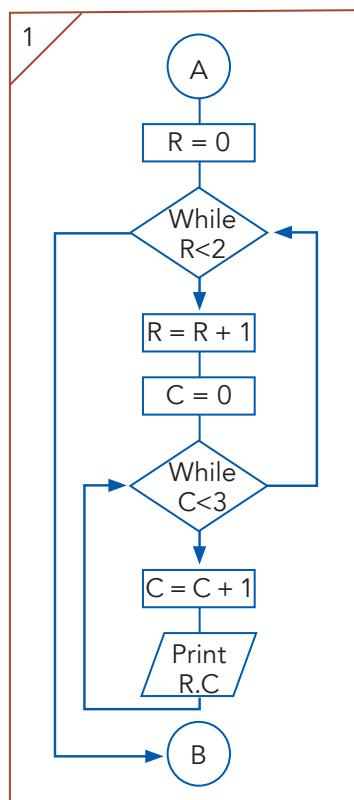
Sınıfın yaş ortalamasını bulmak için bu otomatik sayaç döngü yapısına ait algoritma ve akış şemasını inceleyelim.

Algoritma	Akış Şeması
<ol style="list-style-type: none"><li>1. Başla.</li><li>2. toplam=0</li><li>3. j 1'den 12 olana kadar dön Yaşı Gir. <math>\text{toplam} = \text{toplam} + \text{yaş}</math> Döngüyü bitir.</li><li>4. ortalama= toplam/j</li><li>5. Ortalamayı yazdır, ortalama</li><li>6. Bitir.</li></ol>	<pre>graph TD; Start([Başlat]) --&gt; Init[toplam=0]; Init --&gt; Decision{J}; Decision -- 1 --&gt; Input[/Yaşı Giriniz/]; Input --&gt; Add[toplam=toplam+yaş]; Add --&gt; Decision; Decision -- 12 --&gt; Average[ortalama=toplam/j]; Average --&gt; Output[/Ortalamayı yaz/]; Output --&gt; End([Bitir]);</pre>

## 8.7. İç İçe Döngüler

Karar yapılarında olduğu gibi döngüler de iç içe kullanılabilir. İç içe döngüler, tekrarlayan bir dizi işlem yine tekrar edilmek istendiğinde kullanılır. İçteki döngülerin dıştaki döngülerle aynı döngü yapısında olması gerekmek. Dıştaki döngü Repeat/Until (For) yapısında iken içteki döngü While/While-End yapısında ya da tam tersi biçimde olabilir. Ancak içteki ve dıştaki döngüyü kontrol eden değişkenin farklı olması gereklidir.

Örneğin aşağıda görülen 4 farklı yapı da aynı çıktıyı üretmektedir. Görüldüğü gibi dışta ve içte yer alan döngüler için farklı yapılar kullanılmıştır.



## 8.8. Göstergeler

Göstergeler, programının bir akışı durdurmak ya da döngüyü sonlandırmak için kontrol amaçlı kullandığı değişkenlerdir. Göstergeler mantıksal bir veri ya da değer olabilir ve indikatör olarak da ifade edilir. Bu değerler, akışı kontrol etmek ve akışa müdahale etmek için kullanılır. Kullanıcılar bu değişkenler hakkında bilgi sahibi değildirler. Hata göstergesi, girdi ya da çıktı da bir hata olduğunu belirtir. Veri sonu göstergesi, girilecek başka veri olmadığı anlamına gelir.

Bu algoritmada döngü, gösterge olarak kullanılan “i” ifadesinin değerine göre sonlandırılmaktadır.

1. Başla.
2.  $i = 0$ ,
3. Yaşı Oku; (yaş)
4. Eğer  $yaş < 19$ ; “Merhaba Genç” yaz;  
değilse “Yaşınız 18’den büyük.” yaz;
5.  $i = i+1$ ;
6. Eğer  $i=25$ ; 7. adıma git;  
değilse 3. adıma git
7. Bitir.

## 8.9. Öz Yineleme

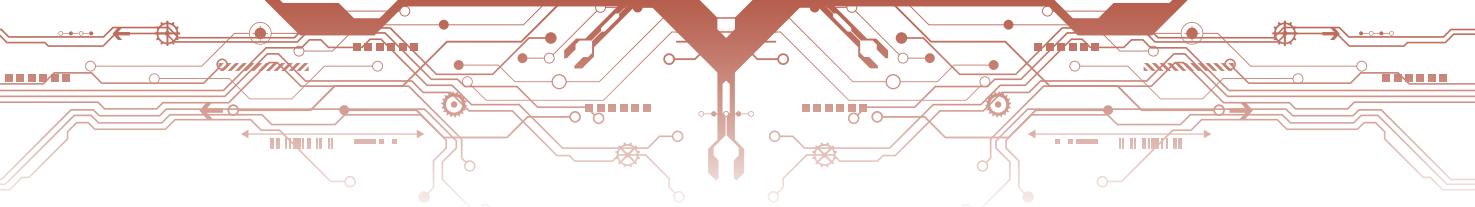
Öz yineleme, farklı bir döngü yapısıdır. Bu durum, bir fonksiyon ya da modül kendi kendini çağrıdağında oluşur. Bu durum için aşağıda verilen faktör hesaplama örneği incelenebilir.

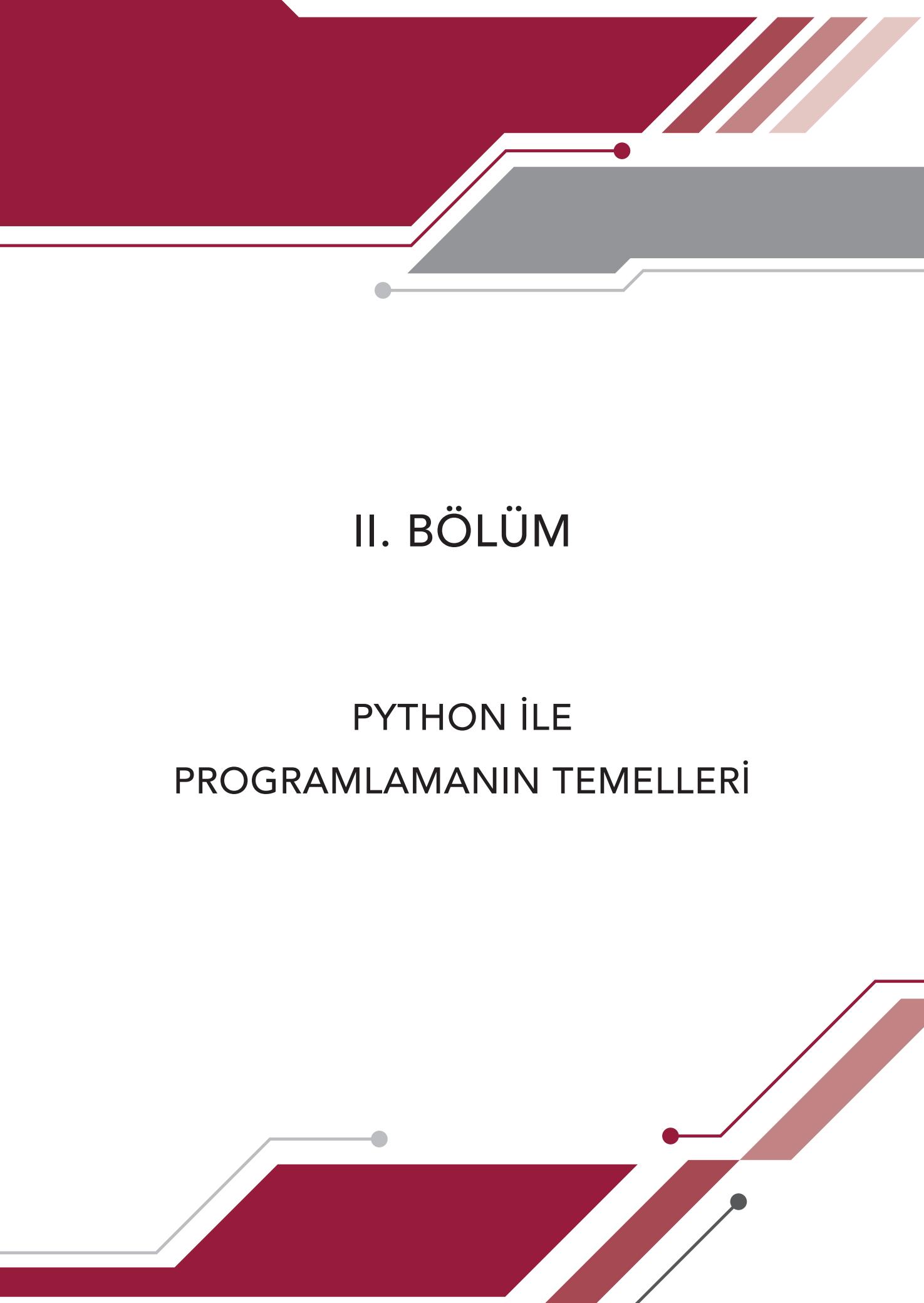
1. Başla
2. Sayı Gir; n
3. Faktoriyel (n);  
Eğer  $n > 1$  ise  
 $Faktoriyel = n * Faktoriyel(n-1)$ ;  
değilse Faktoriyel = 1;
4. Bitir.



### Düşünelim/Deneyelim

1. Bir sınıfta 20 öğrenci vardır. Bu öğrencilerin sınavdan aldıkları notu okuyup sınıf ortalamasını hesaplayan programın algoritmasını yazınız.
2. Kullanıcının girdiği 10 adet sayı için;
  - a. 2 ile bölünebilen sayıların adedi ve toplamını,
  - b. 3 ile bölünebilen sayıların adedi ve toplamını,
  - c. 5 ile bölünebilen sayıların adedi ve toplamını bulan programın algoritmasını yazınız.

- 
- 3.** Ekrandan girilen bir şifre için;
- En az 8 karakter olup olmadığını,
  - En az bir simge (\*,+,&) içerip içermediğini,
  - En az bir sayı içerip içermediğini bulan ve tüm sonuçlara ilişkin mesaj veren programın algoritmasını yazınız.
- ç.** Yukarıdaki kriterlerden üçünü de içeren şifreler içi “Güçlü Şifre”, ikisini içerenler için “Kabul Edilebilir”, birini içeren içinse “Zayıf” ve hiç birini içermeyorsa “Asla Kullanılamaz” mesajı yazan programın algoritmasını oluşturunuz.
- 4.** Girilen bir cümle içinde belirli bir kelimenin geçip geçmediğini arayan programı yazınız. Örnek “Bu gün hava çok güzel!” cümlesindeki “hava” kelimesini bulmak gibi...
- 5.** Öğretmen, tiyatro oyunu için sınıfından bir kişi seçecektir. Sınıfta 10 öğrenci vardır. Öğretmen seçimi yaparken farklı bir yol izlemek istemektedir. Buna göre; öğrenciler rastgele sıraya dizilecek ve baştan saymak şartıyla her üçüncü öğrenci elenecektir. Bu durumda en sona hangi öğrenci kalır?
- A B C Ç D E F G H I
- 6.** Sabit maaşla çalışan öğretmenlere performanslarına ve öğrenci sayısına göre ek maaş verecektir. Buna göre;
- Sınıf mevcudu 20 ile 25 arasındaysa %5 artış,
  - 25 ile 30 arasındaysa %10 artış,
  - 30'dan fazlaysa %15 artış.
- Haftalık ders saati;
- 15 ile 20 arasındaysa %5 artış,
  - 20 ile 25 arasındaysa %10 artış,
  - 25'ten fazlaysa %15 artış.
- Bu durumda girilen ders saati ve öğrenci sayısına göre hak edilen maaşı hesaplayan algoritmayı yazınız.



## II. BÖLÜM

# PYTHON İLE PROGRAMLAMANIN TEMELLERİ

# 1. PYTHON İLE PROGRAMLAMANIN TEMELLERİ



Bu bölümde;

- ✓ Yazılım geliştirme süreci konusunda bilgi sahibi olacak,
- ✓ Yazılım geliştirme sürecinde gerekli olan araçları tanıyacak,
- ✓ Python dilinde program geliştirme ortamlarını inceleyebileceksiniz.



## 1.1. Yazılım Geliştirme Süreci

Bir bilgisayar programı, bilgisayar sistemindeki elektrik sinyallerinin akışını yöneten bir dizi yönergedir. Bu sinyaller, bilgisayarın hafızasını etkileyerek ekran, klavye, fare ve hatta diğer bilgisayarlar ile etkileşim sağlar. Bu şekilde insanlar da karmaşık problemleri çözmek, oyun oynamak gibi değişik işlemleri gerçekleştirebilir. Bir program hesap makinesi rolü üstlenirken bir diğeri makineyi satranç tahtasına dönüştürebilir. Buradaki iki örnek;

- Alt düzeyde, elektrik sinyalleri somut biçimde bilgisayarın mevcut durumunu değiştirirken,
- Üst düzeyde, soyut bir biçimde kullanıcıların eğlenme ya da işleri gereği farklı işlemleri gerçekleştirmelerini sağlar.

Günümüzde, kullanıcı arayüzüne aktarılan üst düzey işlemleri ile alt düzey işlemleri kontrol edebilmek, son derece kolaylaşmıştır ve bu nedenle pek çok programcı bu dilleri kullanarak daha soyut bir biçimde kod yazmaktadır. Programlama seçenekleri programlamayı sürpriz bir biçimde basit hâle getirmiştir. Programlama kavramları, temelinde mantıksal ve matematsel olarak sınıflandırılabilir. Bilgisayar programları bir bilgisayar kullanmadan da yazılabılır. Programcılar, bir programın çalışabilirliğini ve doğruluğunu, gerçek yaşamda yeri olmayan soyut semboller kullanarak tartışabilirler. Bu yaklaşım çok değerli olmakla beraber, programcılar çoğu zaman makinelerden uzak kalamazlar. Yazılımlar, gerçek bilgisayar sistemlerinde kullanılmak üzere yazılır. Yazılım mühendisleri belirli sistemler üzerinde çalışacak şekilde programları üretirler. Bu sistemler, donanım alt yapıları ve kullandıkları işletim sistemi ile tanımlanır. Yazılım geliştirenler; derleyici (compiler), hata ayıklayıcı (debugger) ve yanaylaç (profilers) gibi somut araçları kullanırlar.

Yazılım geliştirme süreci şu şekilde işler;

- Programcı programlama dili kullanarak kodları oluşturur,
- Yazılan kod bütünü, hata ayıklayıcı (debugger) kullanılarak hatalara karşı denetlenir,
- Hataları giderilmiş kodlar, derleyici (compiler) kullanılarak bilgisayarın yorumlayabileceği elektriksel sinyallere dönüştürülür.

Bu süreç sonunda bilgisayar, elektriksel sinyalleri yorumlayarak komutların gereğini yapar. Ayrıca yazılım geliştirme sürecinde yanaylaç (profilers) kullanımı, yazılımcının, programın daha hızlı çalışmasını sağlayacak şekilde kodları revize etmesine olanak sağlar.

### 1.1.1 Yazılım

**Yazılım**, bilgisayarın donanımını anlamlı hale getiren, bilgisayarları kullanıcıların amaçları doğrultusunda kullanmasını sağlayan kod, komut ve programlardır.

Bilgisayar yazılımlarına örnek olarak bilgisayar programları verilebilir. Bir program, bir yazılım parçası olarak nitelendirilebilir. Yazılımlar daha soyut, programlar ise biraz daha somuttur. Yazılımlar CD, DVD, sabit disk, taşınabilir bellek gibi farklı araçlar üzerinde saklanabilir. Yazılımların kullanabilmesi için bilgisayarın hafızasında kayıtlı olması gereklidir. Bilgisayar programları hafızaya belirtilen araçlardan yüklenir. Bilgisayarın sabit diskine yüklenmiş olan program elektromanyetik bir örüntü oluşturur. Bu elektronik sembollerden oluşan örüntünün, program çalışmadan önce hafızaya transfer edilmesi gereklidir. Program bir CD ya da İnternet üzerinden yüklenebilir. Bu elektronik semboller ikilik sayı sistemini kullanan, sıfır ve bir değerlerinden oluşan bir dizidir.

Örnek: Bir ikilik program şu şekilde görünür:

10001011011000010001000001001110

İşlemci olarak adlandırılan bilgisayar donanımı, bu sinyalleri çözümleyerek yapılmak istenen işlemi (grafik ara birimine veri göndererek ekranın bir bölümünün mavi görünmesini sağlamak gibi) gerçekleştirir.

## 1.1.2. Yazılım Geliştirme Ortamları

Yazılımlar, ikilik dizileri daha anlaşılabilir kılan kelime ve sembollerini kullanır. Böylece bilgisayarların dilini öğrenmek ve karmaşık problemleri çözen programlar yazmak kolaylaşır. Bu amaçla, komutları üst düzey yazılımlardan alt düzey makine diline çevirebilen araçlar da kullanılabilir. Python gibi üst düzey diller, programcıların İngilizce konuşma diline çok yakın bir şekilde program kodlarını yazabilmelerine olanak sağlar. Geçtiğimiz yaklaşık 60 yıllık süreçten günümüze kadar FORTRAN, COBOL, Lisp, Haskell, C, Perl, C++, Java ve C# gibi pek çok üst düzey dil geliştirilmiştir. Bu tür programlama dilleri ile uygulama geliştiren programcılar, donanım ya da makine dili gibi konulardaki detaylar ile ilgilenmeden çok etkili yazılımlar geliştirebilir. Böyle bir dönüştürme aracının, kendi dilimizi anlayıp işleme dökmesini bekleyebiliriz ancak günlük konuşma dilleri programlama dillerine göre son derece karmaşık olduğundan bu işlem olası değildir. Derleyici olarak kullandığımız, bir programlama dilini diğerine çeviren programlar yaklaşık 60 yıldır hayatımızdadır ancak konuşma dilinin işlenerek programa dönüştürülmesi hâlâ yapay zekâ araştırma konuları arasındadır. Günlük konuşma dilini belirli standartlar çerçevesinde anlaşılır kılmak, bugünkü yazılımların kapasitesinin çok üstünde bir beklentidir. Programlama dilleri oldukça basit bir yapı ve kesin kurallarlığında bilgisayar tarafından çözülebilecek problemler için çözüm üretmektedir.

Geleneksel olarak yeni bir dilde yazılan ilk program “Merhaba, Dünya!” adı verilen programdır. Python’da aşağıdaki şekilde yazılmaktadır:

Python dili ile yazılmış ilk örneğe bakalım:

```
print "Merhaba, Dünya!"
```

Bu print komutunun bir örneği olup ekranda Merhaba, Dünya! yazar. Tırnak işaretleri programda bir değerin başlangıcını ve sonucunu gösterir ve ekranda gözükmeyecektir.

Python dili ile yazılmış aşağıdaki örneğe bakalım:

```
Toplam = 0  
DersSaati = 3  
Hafta = 14  
Toplam = DersSaati * Hafta
```

Bu satırlar bir Python programındaki bazı satırlar olabilir. Bu satırlar bazı hesaplama işlemleri (= ve \*) ile benzerlik göstermektedir. DersSaati, hafta ve toplam olarak ifade edilen kelimeler, değişken olarak adlandırılmaktadır. Bu değişkenler verileri bilgisayarın hafızasında korumak için kullanılır. Bu satırlar Python dili ile yazıldığından herhangi bir makine dili tarafından anlaşılabilir değildir. Kullanıcı programı çalıştırduğunda, yorumlayıcı programlar, Python kodunu makine koduna çevirir. Üst düzey program kodu kaynak kod (source code) olarak adlandırılır. Bu koda karşılık gelen makine diline ise hedef kod (target code) adı verilir. Yorumlayıcı, kaynak kodu hedef koda dönüştürür. Üst düzey programların güzelliği, kodlamanın donanımdan bağımsız olarak yapılabilmesidir. Üstünde çalışılan platform ne olursa olsun, Python yorumlayıcısı kurulu ise tüm programlar tüm platformlarda çalıştırılabilir. Programcılar yazılım geliştirme sürecini destekleyen pek çok araç vardır. Bunlardan bazıları aşağıda listelenmiştir.

## 1.1.3. Editörler

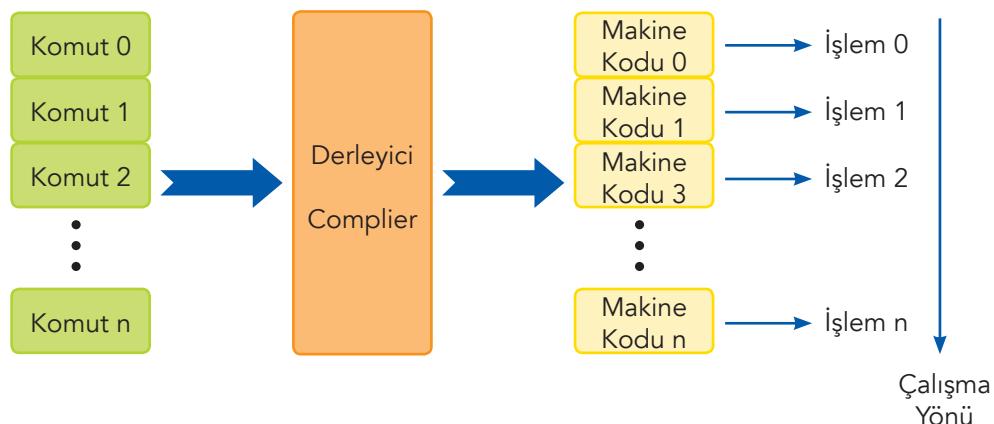
Bir editör, programcının kaynak kodu yazmasını ve dosyaya kaydetmesini sağlar. Çoğu editör, renklenme desteği sunarak dilin özelliklerini ortaya çıkarır ve programcının üretkenliğinin artmasını destekler. Dili oluşturan parçaların kurallara uygun bir şekilde düzenlenmesi söz dizimi (syntax) olarak ifade edilir. Geliştirme araçlarının yazılanları tam olarak doğru anlaması için, kullanılan kelime ve sembollerin kurallara uygun biçimde dizilmesi önemlidir. Yalnızca doğru biçimde ifade edilen programlar



makine koduna dönüştürülerek kabul edilir. Bu nedenle bazı editörler yazım yanlışları konusunda renkleri ya da farklı vurgulamaları kullanarak yazım hataları konusunda programcayı uyarır.

#### 1.1.4. Derleyiciler

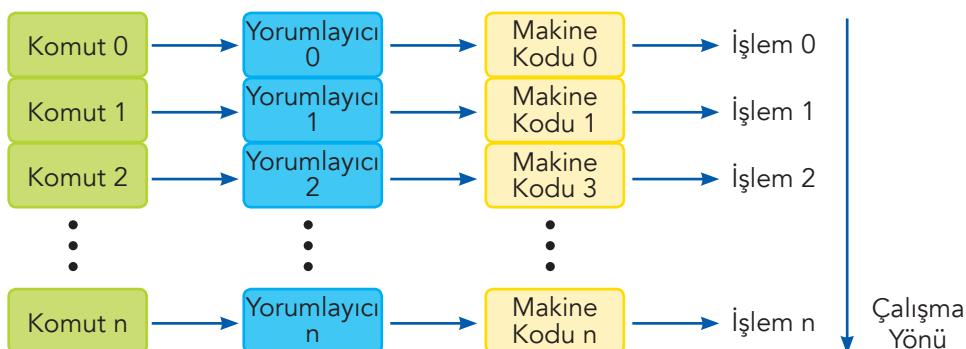
Derleyiciler, kaynak kodları hedef koda dönüştürür. Hedef kod, belirli bir platform ya da gömülü bir araç için makine dili olabilir (Şekil 2.1). Hedef kod diğer bir üst düzey kaynak dil de olabilir. Derleyiciler, kaynak kod içini dönüştürerek hedef kod içeren bir dosya oluşturur. Derlenerek oluşturulan popüler dillere örnek olarak C, C++, Java ve C# verilebilir.



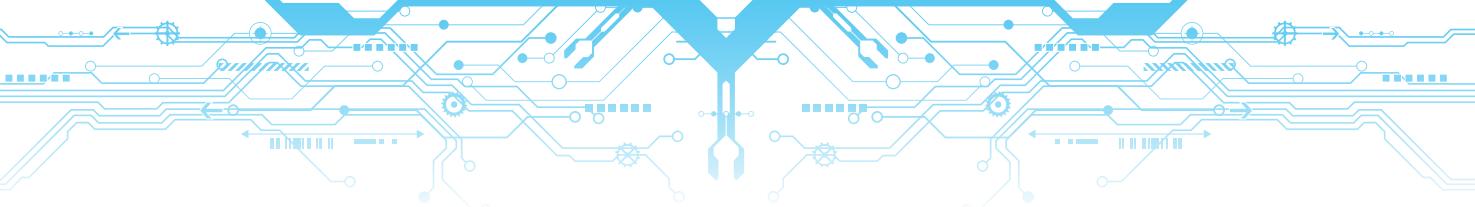
Şekil 2.1: Derleyici yapısının grafiksel gösterimi

#### 1.1.5. Yorumlayıcılar

Yorumlayıcılar da derleyiciler gibi üst düzey kaynak kodu hedef koda (genellikle makine kodu) çevirir ancak derleyicilerden farklı olarak çalışır (Şekil 2.2). Derleyiciler herhangi farklı bir dönüşüm gerekmeden defalarca çalışılabilir bir program kodu üretirken yorumlayıcılar kullanıcı kaynak kodu her çalıştırıldığında satır satır makine diline çevirir. Derlenmiş bir program, değişiklik yapılmadığı sürece tekrar derlenmez ancak yorumlayıcı ile çalışan program için yorumlama işlemi değişiklik yapılmamış olsa bile tekrarlanmalıdır. Bu nedenle yorumlanan diller daha çok senaryo dili (scripting language) olarak ifade edilir. Yorumlayıcı, programın kaynak kodu olan senaryoyu okur. Genellikle derlenen programlar yorumlanan programlara göre daha hızlı çalışır çünkü derleme işlemi yalnızca bir kez yapılır. Diğer yandan yorumlanan programlar, herhangi bir platformda tekrar derlenmelerine gerek kalmadan uygun bir yorumlayıcı ile hemen çalıştırılabilir. Python, yorumlanan bir dil olmakla birlikte, bunun derleyicileri de vardır. Popüler senaryo dillerine örnek olarak Python, Ruby, Perl ve web ortamı için Javascript verilebilir.



Şekil 2.2: Yorumlayıcı yapısının grafiksel gösterimi



### 1.1.6. Hata Ayıklayıcılar

Hata ayıklayıcılar, programcının bir programdaki olası hataları bulmasına ve düzeltmesine olanak sağlayarak programın doğru çalışması için yardımcı olur. Hata ayıklayıcı programları ile programın hangi satırlarında hata olduğu belirlenir. Programcı, değişkenlerin değerlerine bakarak neyin yanlış gittiğini anlayabilir.

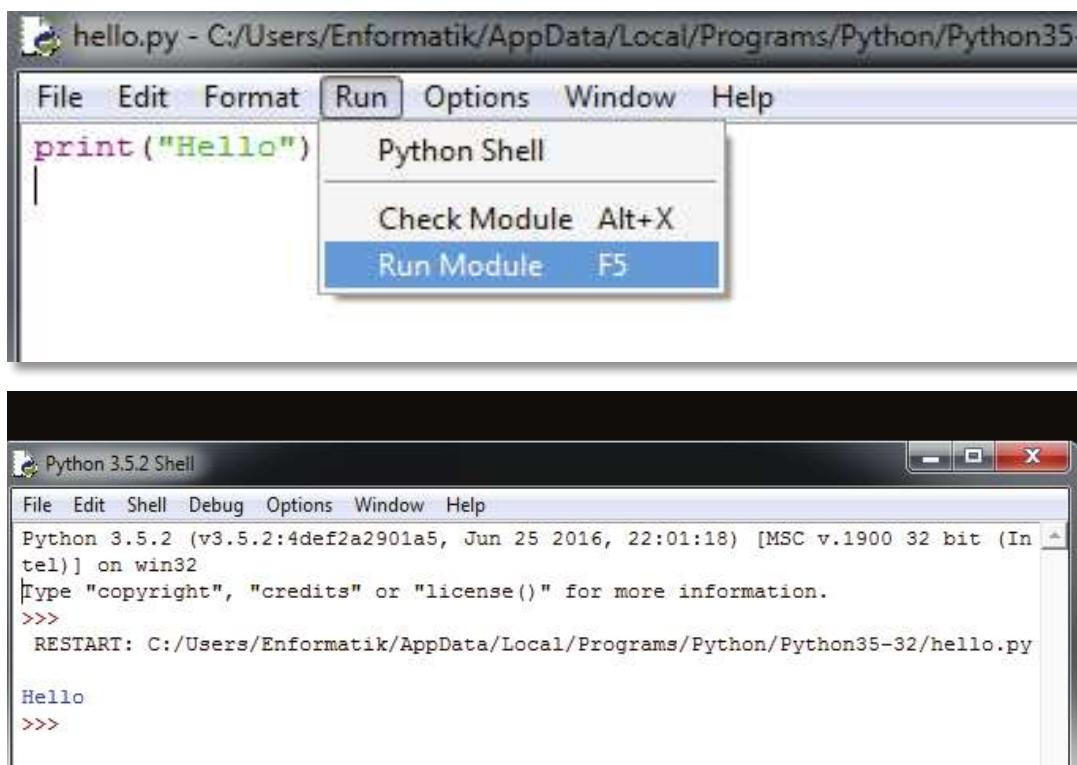
### 1.1.7. Yanaylaçlar

Yanaylaçlar, bir programın çalışmasına ilişkin istatistik veri toplar. Böylece programcılar, genel olarak performansını artırmaya yönelik önlemler alabilir ve programın belirli bölümlerini yeniden yapılandırabilir. Yanaylaç, program her çalıştırıldığında program parçalarının kaç kere çalıştırıldığını ve bu işlemin ne kadar süredüğünü ortaya çıkarır. Bu işlem, programın gerçekten tüm parçalarının kullanılıp kullanılmadığını belirlemek için de kullanılabilir. Buna kaplam (coverage) denilir. Genel olarak programın belirlenen parçaları iyileştirilerek programın daha hızlı çalışması sağlanır.

### 1.1.8. Bütünleştirilmiş Geliştirme Ortamları

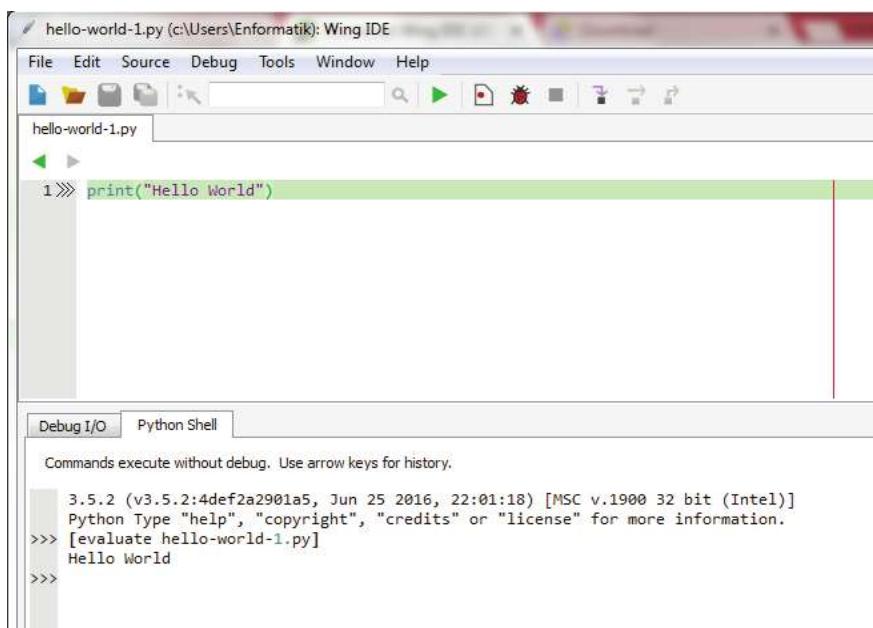
Çoğu yazılım, geliştirici bütünleştirilmiş geliştirme ortamlarını (integrated development environment-IDE) kullanır. Bu ortamlar, editörleri, hata ayıklayıcıları ve diğer programlama yardımcılarını kapsar. Aşağıda en sık kullanılan Python IDE ortamları listelenmiştir.

- IDLE: Python.org web sitesinde yer alan ücretsiz program geliştirme ortamıdır (Şekil 2.3). Bu site içerisinde downloads menü başlığı altında Python için son sürümleri bulmak mümkündür. Ayrıca, birçok işletim sistemi için gerekli dosyalara erişim bağlantıları da bu sayfada yer almaktadır.



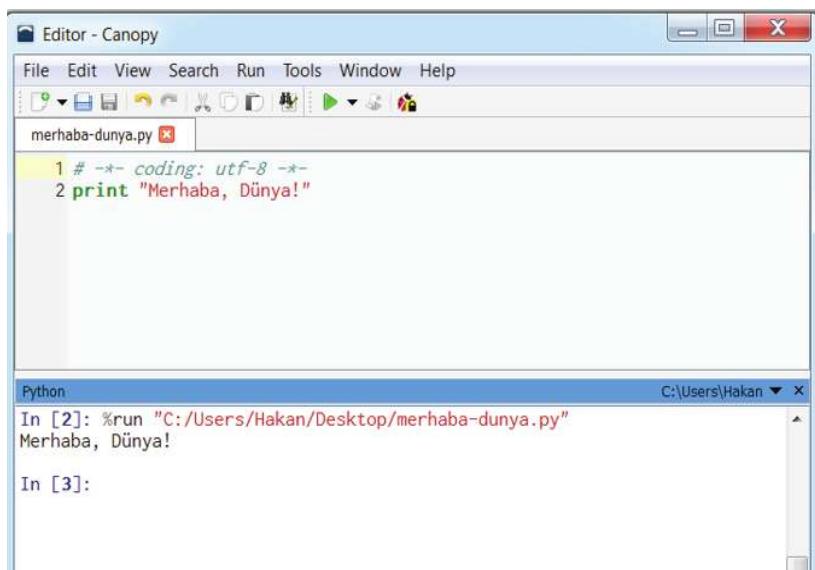
Sekil 2.3: IDLE geliştirme ortamı

- Wing IDE: Wingware Python'da program geliştiriciler için üç farklı türde IDE ortamı sunmaktadır: Wing IDE Professional, Wing IDE Personal ve Wing IDE 101. Bunlardan Wing IDE 101 ücretsiz olup başlangıç seviyesindeki Python kullanıcıları için basit bir IDE ortamı sunmaktadır (Şekil 2.4). <https://wingware.com/downloads/wingide-101> adresinden indirilebilir. Bu ortamda kod yazmak, hata bulmak ve çalışma alanını kişiselleştirmek mümkündür.

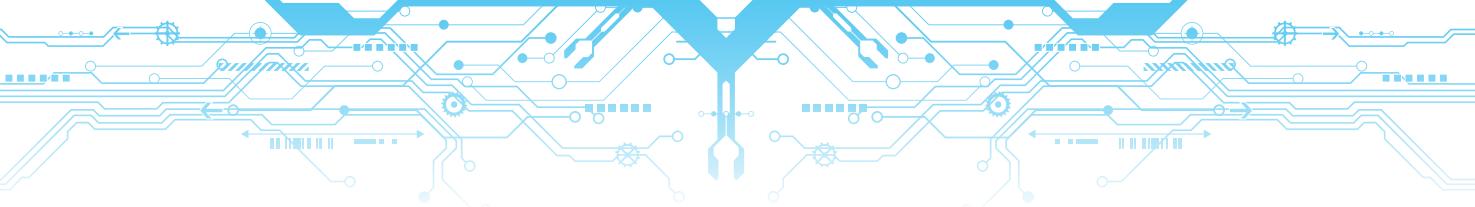


*Sekil 2.4: Wing IDE 101 geliştirme ortamı*

- Canopy: Enthought çatısı altında yer alan Canopy, içerisinde birçok farklı programlama dilinde kod yazmayı ve derlemeyi kolaylaştırmaktadır. <https://store.enthought.com/downloads/#default> adresinden Canopy program geliştirme ortamı ücretsiz indirilebilir (Şekil 2.5).



*Sekil 2.5: Canopy program geliştirme ortamı*



### Düşünelim/Deneyelim

Farklı özellikleri olan ve farklı işletim sistemlerinde çalışan IDE'ler için <https://wiki.python.org/moin/PythonEditors> adresini ziyaret ediniz.

## 1.2. Neden Python?

Python, öğrenmesi kolay, tamamen özgür ve ücretsiz bir programlama dilidir. Nesnelere dayalı bir dil olup okunabilirliği yüksektir. Python'un dili başka programlama dilleri ile kıyaslandığında, bunun daha az kod ile işlemleri yapmasının mümkün olduğu görülecektir. Python, bütün işletim sistemleri ile uyum içerisinde çalışmaktadır.

Programlama yapısı içerisinde birçok kütüphaneyi barındırmaktadır. Bu kaynaklarla daha az kod yazmak mümkündür. Python ile masaüstünde çalışan uygulamalar geliştirilebileceği gibi, web üzerinde çalışan uygulamalar geliştirmek hatta Raspberry-Pi gibi donanımları da programlamak mümkündür.

## 1.3. Python Sürümleri

Python programlama dilinin 2016 yılı için en güncel sürümü Python 3.5.2'dir. Bu kitapta yer alan örnekler 3.X sürümlerinde çalışan uygulamalardan oluşmaktadır. Ancak başka kaynaklarda yer alan birçok örneğin daha önceki sürümlerde yazıldığını görmeniz mümkün olabilir. Sürümler arasındaki farklılıklar, özellikle ileride görülecek olan fonksiyonların, print komutu gibi bazı komutların farklı olarak ifade edilmesini kapsamaktadır. Farklı kaynaklardan bulduğunuz örnekler yeni sürümlerde çalışmamayabilir. Bu nedenle bu kitapta yer alan söz dizimi kurallarına göre komutların değiştirilmesi gerekmektedir.

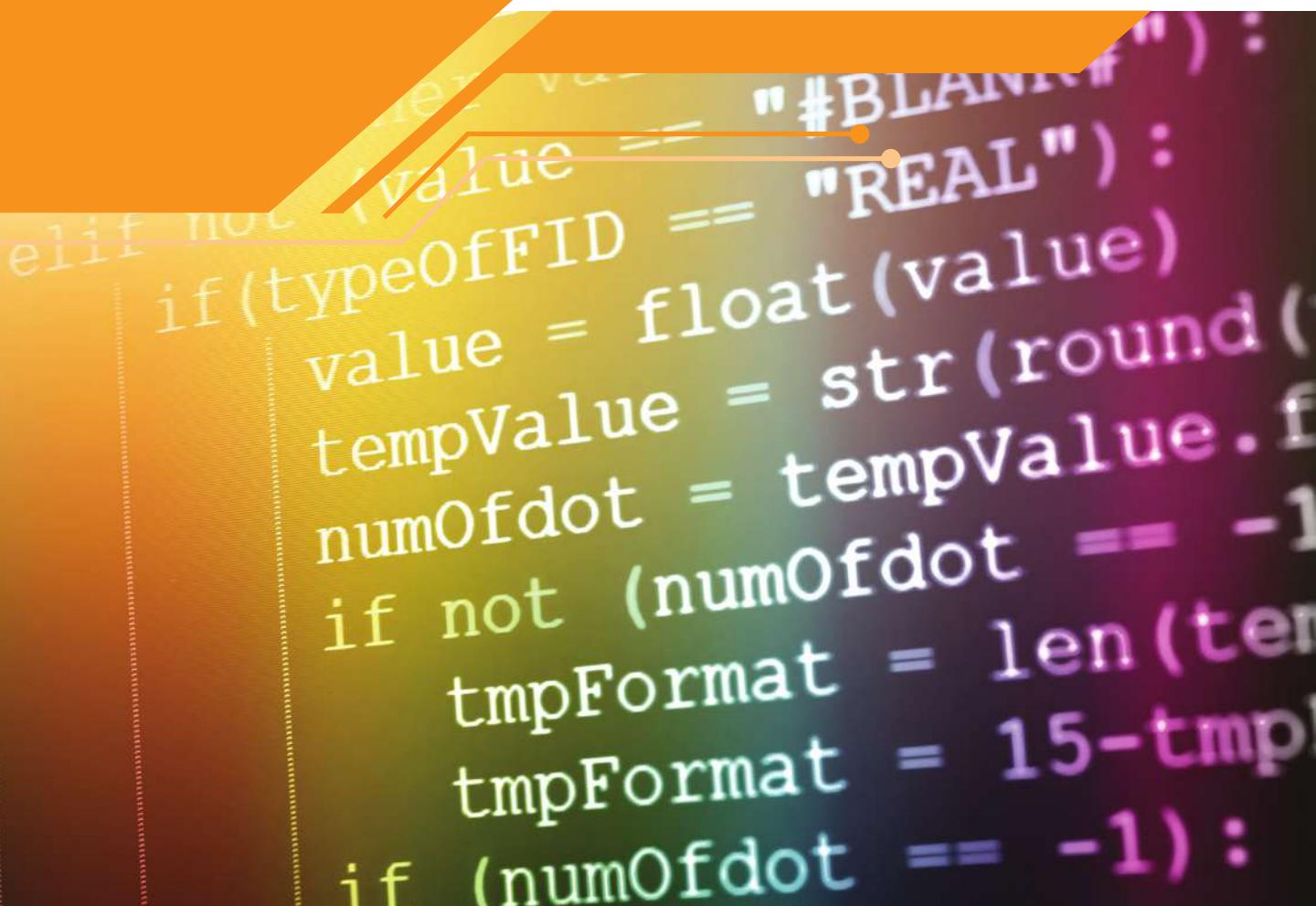


### Düşünelim/Deneyelim

1. Derleyici ve yorumlayıcı nedir? Benzerlik ve farklılıklarını nelerdir?
2. Derlenen ve yorumlanan kodun, kaynak koddan farkı nedir?
3. Python programlama dili nasıl bir dildir ve programı çalıştırmak için ne gereklidir?
4. Üst düzey programlama dili kullanmanın yararları nelerdir?
5. Bütünleştirilmiş geliştirme ortamlarının programcıya sağladığı yararlar nelerdir?
6. Programlama dillerinde seviye kavramı neyi ifade etmektedir? Bu konuda bir araştırma yapınız ve bulduklarınızı raporlaştırınız.



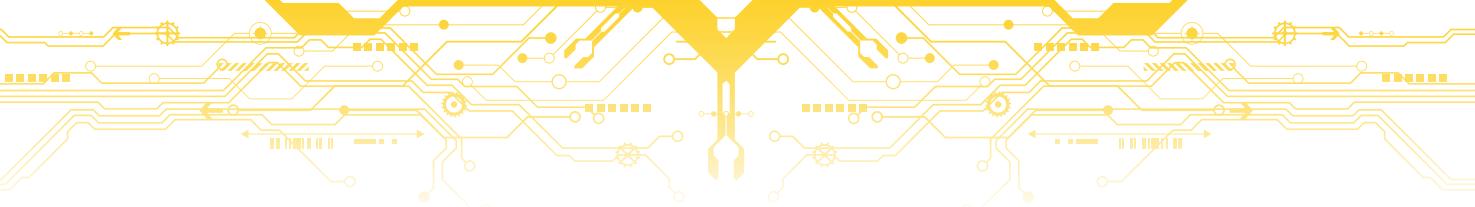
## 2. DEĞERLER VE DEĞİŞKENLER



```
elif not (value == "#BLANK"):  
    if (typeofID == "REAL"):  
        value = float(value)  
        tempValue = str(round(  
            numOfdot = tempValue.  
            if not (numOfdot == -1):  
                tmpFormat = len(tem  
                tmpFormat = 15-tmp  
                if (numOfdot == -1):
```

Bu bölümde;

- ✓ Python programla dilinde değerler ve değişkenler konusunda bilgi sahibi olacak,
- ✓ Veri türlerini tanıယacak,
- ✓ Farklı değişkenleri kullanan program yazabileceksiniz.



## 2.1. Tam Sayı ve Diziler

Herhangi bir sayı, sayısal değerdir. Örneğin matematikte 4 sayısı tam sayı olarak ifade edilir. Tam sayılar pozitif, negatif ya da sıfır değeri alabilir. Kesirli değerleri içermez. Örneğin 99, 105, 0, -56 ve 7896 birer tam sayıdır. Ancak 4.5 bir tam sayı değildir. Python programlama dili, sayısal ve sözel ifade kullanımını destekler. Python programları, tam sayıları kullanarak işlem yapabilir. Örneğin;

```
print(4)
```

İfadeleri 4 değerini ekrana yazdırır. İfadede tırnak ("") işaretini kullanılmadığına dikkat ediniz. Python tam sayılar dışındaki diğer veri türlerini de desteklemektedir. Bu ifade Python satırlarının temel yapı taşıdır. Tek başına 4 sayısı bir anlam ifade etmez. Ancak yorumlayıcılar, bir anlatım olursa Python ifadelerini değerlendirdir. Python etkileşimli yorumlayıcısı, hem ifadeleri hem de anlatımları değerlendirir. Örneğin;

```
>>> 4
```

Komutu yazıldığında, yorumlayıcı satırı okur, işler ve sonucu 4 olarak ekrana yazdırır. x = 10 girilirse bu ifadenin sonuca yönelik bir anlamı olmadığı için etkileşimli yorumlayıcı ekrana hiçbir şey yazdırır. Eğer kullanıcı bu ifadeden sonra yalnızca "x" girerse yorumlayıcı bunu değerlendirir ve ekrana x değeri olan 10 olmasını yazar. Şayet kullanıcı bundan sonra ekrana "y" yazarsa, bu değer daha önce tanımlanmadığı için yorumlayıcı hata verir.

Python, toplama işlemi için toplama (+) simbolünü kullanır. Böylece yorumlayıcı bir hesap makinesi gibi işlem yapar.

```
>>> 5 + 4  
9  
>>> 1 + 6 + 4 + 10  
21  
>>> print(1 + 6 + 4 + 10)  
21
```

Son satırda toplama işleminin, doğrudan print ifadesi içerisinde yer alabildiği de görülmektedir. Şimdi de aşağıdaki işleme bakalım:

```
>>> 16  
16  
>>> "16"  
"16"  
>>> '16'  
'16'
```

İlk satırda tam sayı olan 16 değerini tek ya da çift tırnak içine alındığında sayı olarak değil metin (alfasayısal kelime yani dizi) olarak işlem görmektedir. Bu ifadeler karakter dizisi ya da dizi olarak anılır. Python, dizileri ayırt etmek ve sınırları belirlemek için hem tek (') hem de çift tırnak (") kullanımına izin vermektedir. Sınırları belirlemek demek, başlangıç ve bitiş noktalarını işaret etmek demektir. Kullanılan ilk ve soldaki simbol (') dizinin başlangıcını, sonra kullanılan ve sağda bulunan simbol (') ise dizinin bitişini ifade eder. Benzer biçimde çift tırnak sembollerini de (") başlangıç ve bitiş için iki kez kullanılmalıdır.



İki simbol tek tırnak ('') ve çift tırnak (""), birbiri yerine kullanılamaz. Bu nedenle;

```
>>> "ABC"
```

```
"ABC"
```

```
>>> "ABC"
```

```
"ABC"
```

İfadeleri doğru çalışırken;

```
>>> "ABC"
```

```
File "<stdin>", line 1
```

```
"ABC"
```

```
^
```

```
SyntaxError: EOL while scanning string literal
```

```
>>> "ABC"
```

```
File "<stdin>", line 1
```

```
"ABC"
```

İfadeleri hataya neden olur.

İfade olarak 4 ve '4' yazımının farklı anlamlar taşıdığınıza dikkat ediniz. İlk değer bir tam sayı iken, ikinci değer karakter olarak algılandığı için bir dizidir. Python içindeki tüm ifadelerin bir türü vardır. İfadenin türü anlatımının da türünü ifade eder. Bazen ifadelerin türü onların sınıfı olarak belirtilir.

Şu ana kadar yalnızca tam sayı ve dizileri inceledik. Gömülü bir fonksiyon, Python ifadelerinin türünü belirtmek için kullanılabilir.

```
>>> type(4)
```

```
<class "int">
```

```
>>> type("4")
```

```
<class "str">
```

Gömülü str fonksiyonu tam sayı olarak görülen bir ifadeden dizi oluşturur.

```
>>> str(4)
```

```
"4"
```

```
>>> "5"
```

```
"5"
```

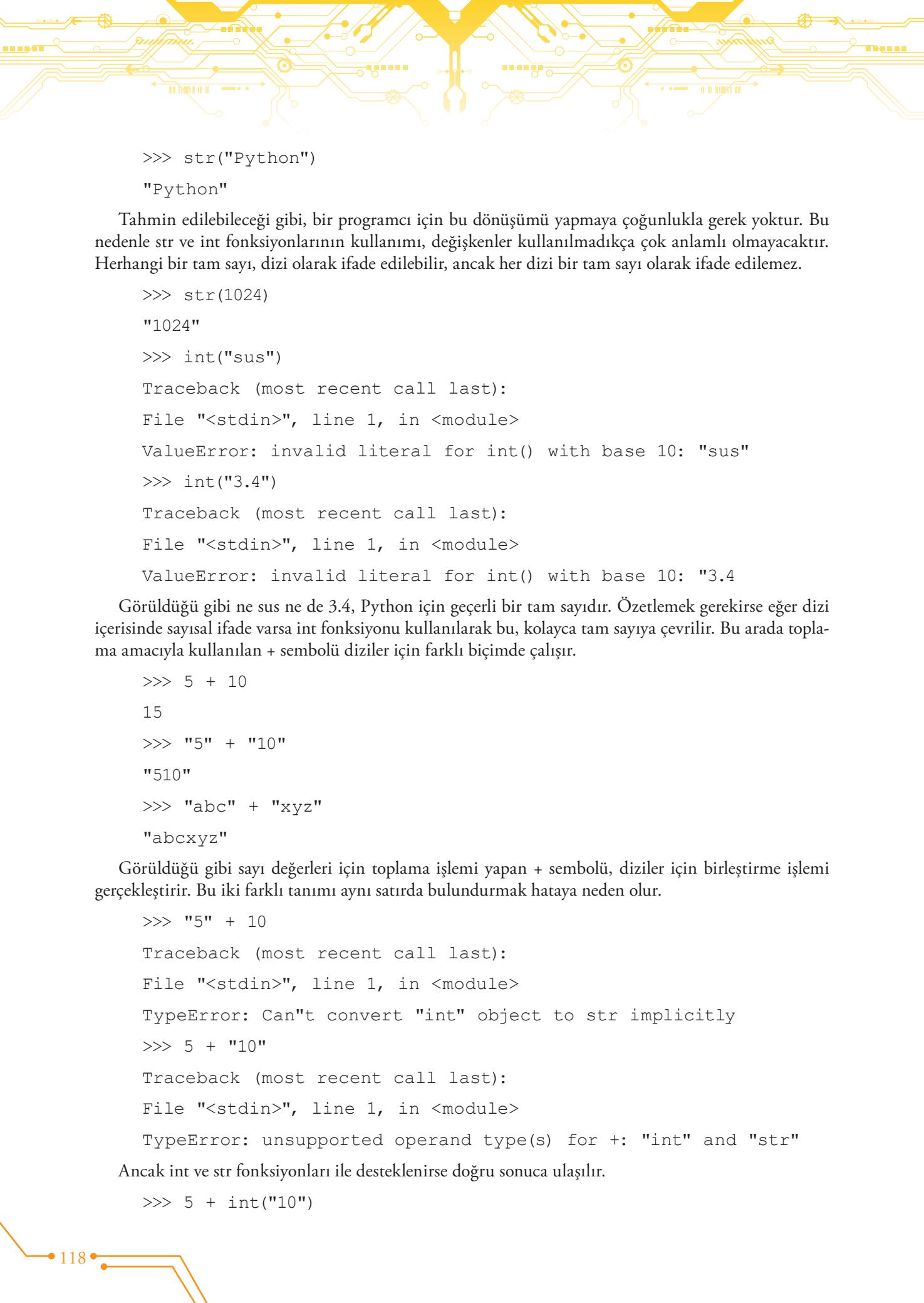
```
>>> int("5")
```

```
5
```

str(4) ifadesi 4 değerini karakter olarak değerlendirir, int('5') ifadesi ise bu karakter değeri tam sayıya dönüştürür.

```
>>> int(4)
```

```
4
```



```
>>> str("Python")
"Python"
```

Tahmin edilebileceği gibi, bir programcı için bu dönüşümü yapmaya çoğunlukla gerek yoktur. Bu nedenle str ve int fonksiyonlarının kullanımı, değişkenler kullanılmadıkça çok anlamlı olmayacağından emin olabiliriz. Herhangi bir tam sayı, dizi olarak ifade edilebilir, ancak her dizi bir tam sayı olarak ifade edilemez.

```
>>> str(1024)
"1024"
>>> int("sus")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: "sus"
>>> int("3.4")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: "3.4"
```

Göründüğü gibi ne sus ne de 3.4, Python için geçerli bir tam sayıdır. Özetlemek gerekirse eğer dizi içerisinde sayısal ifade varsa int fonksiyonu kullanılarak bu, kolayca tam sayıya çevrilir. Bu arada toplama amacıyla kullanılan + simbolü diziler için farklı biçimde çalışır.

```
>>> 5 + 10
15
>>> "5" + "10"
"510"
>>> "abc" + "xyz"
"abcxyz"
```

Göründüğü gibi sayı değerleri için toplama işlemi yapan + simbolü, diziler için birleştirme işlemi gerçekleştirir. Bu iki farklı tanımı aynı satırda bulundurmak hataya neden olur.

```
>>> "5" + 10
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert "int" object to str implicitly
>>> 5 + "10"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: "int" and "str"
```

Ancak int ve str fonksiyonları ile desteklenirse doğru sonuca ulaşılır.

```
>>> 5 + int("10")
```



15

```
>>> "5" + str(10)  
"510"
```

Python için type fonksiyonu karmaşık ifadeler için kullanılabilir.

```
>>> type(4)  
<class "int">  
>>> type("4")  
<class "str">  
>>> type(4 + 7)  
<class "int">  
>>> type("4" + "7")  
<class "str">  
>>> type(int("3") + int(4))  
<class "int">
```

Python kapsamında sayıları ifade ederken “,” ya da “.” gibi ayırma sembollerini kullanılmaz. Dört bin altı yüz eksi sekiz, 4658 olarak yazılmalıdır.

## 2.2. Değişkenler ve Atama

Değişkenler, değerleri korumak için kullanılır. Bu değerler sayı, dizi gibi farklı biçimlerde olabilir. Örneğin;

```
x = 10
```

ifadesi bir atama satırıdır. Atama işlemi bir değeri bir değişken ile eşleştirir. Bu ifadedeki en önemli ayrıntı, atama (=) simbolüdür. Bu ifade ile 10 değeri, x değişkenine atanmaktadır. Bu noktada, x değişkeninin türü tam sayı olur çünkü atanın değer bir tam sayı değerdir. Bir değişkene birden fazla kez atama yapılabilir. Eğer bu sırada öncekinden farklı türdeki bir değer ataması yapılrsa değişkenin türü de değişir. Burada atama (=) simbolünün anlamı matematikte kullanıldığı şeklinde daha farklıdır. Matematikte bu simbol, eşitlik sağlar, bu yüzden bu simbolün sağ ve sol tarafında yer alan ifadelerin birbirine eşit olduğu anlamına gelir. Python dilinde ise atama (=) simbolünün sol tarafında yer alan ifade, sağ tarafındaki ifadeyi üstlenir. Bu yüzden bu ifadeyi “5 değeri x değişkenine atandı.” ya da x’e 5 atandı.” şeklinde yorumlamak doğru olacaktır. Buradaki kullanım, matematikteki kullanımından farklı olduğu için önemlidir. Matematikte eşitlik ve eşitliğin her iki yanında simetri söz konusudur yani matematiksel açıdan  $x = 5$  ve  $5 = x$  olacak biçimde her iki ifadede doğru iken Python kapsamında bu, simetri olmadığı için  $5 = x$  ifadesi hatalı olacaktır. Çünkü tam sayı bir değişken olmadığı için atama yapma davranışını da yanlışır.

Bir değişkene defalarca farklı değerler atayabiliriz.

```
x = 10  
print(x)  
x = 20  
print(x)  
x = 30  
print(x)
```



Göründüğü gibi print fonksiyonu her satırda aynı olmasına rağmen her seferinde farklı bir değer yazdırılmaktadır.

```
10  
20  
30
```

Bu program bize, değişkenlere bağlı durumlardaki davranışları her zaman öngörmemişimizi göstermektedir. Bazı fonksiyonlar bir ya da daha fazla değişkene bağımlı hareket edebilir.

```
x = 10  
print("x = " + str(x))  
x = 20  
print("x = " + str(x))  
x = 30  
print("x = " + str(x))
```

Yukarıdaki program aşağıdaki çıktıyı oluşturur.

```
x = 10  
x = 20  
x = 30
```

Burada print fonksiyonu içerisinde kullanılan toplama + işlemi dizileri birleştirmek amacıyla kullanılmaktadır. Bu nedenle “x =” + x ifadesi çıktıda görülen sonucu üretmektedir. Bu örnekte print fonksiyonu, iki parametre kabul etmektedir.

```
print("x =", x)
```

İlk parametre “x =” dizisi ve ikinci parametre ise x değişkeni ile eşleşmiş değerdir. Print fonksiyonu her biri virgül ile ayrılmış çoklu parametre kullanımına izin verir, hepsini sıra ile yazdırır ve yazdırırken her biri arasına bir boşluk değeri bırakır. Bir programcı tek bir satırda çok ögeli yaklaşımı kullanarak birden fazla değer ataması yapabilir. Bu işleme **çoklu atama** denilir.

```
x, y, z = 100, -45, 0  
print("x =", x, " y =", y, " z =", z)
```

Çoklu atamada değerler virgüler ile ayrılmış olarak listelenir.  $x, y, z = 100, -45, 0$  ifadesinde  $x, y, z$  bir grup çoklu öğeyi;  $100, -45, 0$  ise diğer grup çoklu öğeyi ifade eder. Çoklu atama şu şekilde çalışır: Sol taraftaki çoklu öğe grubundaki ilk değişken, sağ taraftaki çoklu öğe grubunun ilk elemanı ile eşleştir ( $x = 100$ ). Benzer şekilde ikinci ve sonraki öğeler de birbiri ile eşleştir. İkinci öğelerin eşleşmesi,  $y = -45$  ve son öğelerin eşleşmesi,  $z = 0$  ile sonuçlanır. Bu işlemden sonraki durum aşağıdaki gibidir:

```
x = 100 y = -45 z = 0
```



Çoklu atama, sol taraftaki çoklu öge grubu ile sağ taraftaki çoklu öge grubunun sayıları eşit ise gerçekleşir. Atanan değer bir değişken ismini bir nesneye bağlar.



*Şekil 2.6: Değişkene değer atama*

Örnek olarak  $x = 2$  ifadesini inceleyelim (Şekil 2.6). Bir kutu değişkeni ifade eder ve bu, değişken ismi ile adlandırılır. Diğer kutudan nesneye doğru yönelen ok, değişkenin bağıldığı nesneyi gösterir. Bu durumda ok 2 değerini içeren başka bir kutuyu işaret eder. İkinci kutu 2 değerinin ikilik düzendeki karşılığını içeren hafıza yerini temsil eder.

Bilgisayar her bir program satırını işledikçe değişkenlerin değerlerinin nasıl değiştiğini gözleyelim.

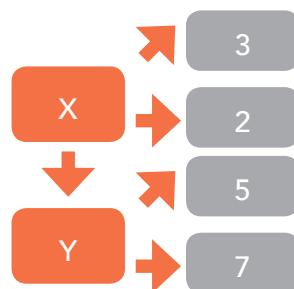
$x = 2$

$y = 5$

$x = 3$

$x = y$

$y = 7$



Burada özellikle  $x = y$  ifadesine dikkat ediniz. Bu atama hem  $x$  hem de  $y$  değerinin aynı değer ile eşleştiği anlamına gelmektedir. Sonra  $y$  değerinin değişmesi  $x$  değerini etkilemez.

Programın çalışması sırasında bir değişkenin yalnızca değeri değil, türü de değişimdir.

```
a = 10  
print("a değişkeninin ilk değeri", a, "ve tipi", type(a))  
a = "ABC"  
print("a değişkeninin yeni değeri", a, "ve tipi", type(a))
```

Bu ifade aşağıdaki çıktıyı oluşturur.

```
a değişkeninin ilk değeri 10 ve tipi <class "int">  
a değişkeninin yeni değeri ABC ve tipi <class "str">
```

Programcılar program akışı içinde bir değişkenin türünü nadiren değiştirmeye gerek duyarlar. Bir değişkenin program çalıştığı sürece belli bir anlamı ve rolü olmalıdır ki bu, genellikle değişmez. Herhangi bir değer atanmamış bir değişken, tanimsız değişken olarak ifade edilir. Böyle bir değişken program içerisinde kullanıldığında hata ile karşılaşılır. Nadiren daha önce tanımlanmış bir değişkeni tanimsız bir değişkene dönüştürmek isteriz. Bu işlemi del satırını kullanarak gerçekleştiririz.



```
>>> x = 2
>>> x
2
>>> del x
>>> x
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name "x" is not defined
```



Kullanılan del komutu sil anlamına gelir ve yorumlayıcı, içerisindeinden ya da program içerisinde de-ğişken tanımını siler. Örneğin a, b ve c önceden tanımlanmış değişkenler ise;

```
del a, b, c
```

ifadesi bütün değişkenlerin silinmesini sağlar.

### 2.3. Reel Sayılar

Pek çok hesaplamalı işlem, kesir parçası olan sayıları kullanır. Örneğin bir dairenin çevresini ya da alanını hesaplamak için  $\pi$  değerine ihtiyacımız vardır ve bu değer yaklaşık 3,14159 olarak ifade edilir. Python, bu şekilde noktalı sayılarla işlem yapar ve bu sayılarla reel ya da gerçek sayı denir. İsminden de anlaşılacığı gibi matematiksel hesaplamalar sırasında ondalık nokta, önemli sayıları ifade etmek için farklı basamakla-ra kayabilir. Python ile programlama yaparken bu tanımı yapmak için **float** kelimesi kullanılır.

```
>>> x = 5.62
>>> x
5.62
>>> type(x)
<class "float">
```

Python için reel sayıların alabileceği en küçük ve en büyük değer ile duyarlılık düzeyi, kullanılan makine ve uygulamaya göre değişiklik gösterebilir. Reel sayılar da tam sayılar gibi pozitif ve negatif olabilir. Yaygın olarak geçerli olan durum aşağıda görülmektedir.

Tanım	Bellek	Minimum Değer	Maksimum Değer	Duyarlılık
float	64 bit	$2,22507 \times 10^{-308}$	$1,79769 \times 10^{308}$	15 basamak

*Tablo 1: Float değeri*

Çoğu programlama editörü, alt ya da üst simge ve özel sembollerin kullanımını desteklemediği için sayıların gösterimi farklılaşmaktadır. Python ile kod yazarken  $6.022 \times 10^{23}$  yerine  $6.022e^{23}$  olarak yazmamız gereklidir. Burada “e” karakterinin solunda kalan kısım normal sayı; sağında kalan kısım ise 10 üzerindeki sayıdır. Simge olarak “e” yerine “E” de kullanılabilir.

Reel sayılarından farklı olarak tam sayılar kesirli ifadeleri içermeyez. Reel bir sayıyı bir tam sayıya dö-nüştürmenin iki temel yolu vardır:



- Yuvarlama: Reel sayıya en yakın tam sayıya ulaşmak için kesrin belirli bir miktarı eklenerek ya da çıkarılarak yuvarlama yapılır.
- Kesme: Sayının kesirli kısmı tamamen göz ardı edilir.

Yuvarlama ve kesme işlemlerinin sonuçlarının ne şekilde farklılığını gözlemleyebiliriz:

```
>>> 28.71  
28.71  
>>> int(28.71)  
28  
>>> round(28.71)  
29  
>>> round(19.47)  
19  
>>> int(19.47)  
19
```

Gördüğü gibi kesme, her zaman aşağı doğru yuvarlamaktadır.

Yuvarlama yapmak için round fonksiyonunu noktadan sonra belirli bir sayıda basamağı koruyarak sonuç elde etmek için de kullanabiliriz.

```
>>> x = 93.34836  
>>> x  
93.34836  
>>> round(x)  
93  
>>> round(x, 2)  
93.35  
>>> round(x, 3)  
93.348  
>>> round(x, 0)  
93.0  
>>> round(x, 1)  
93.3  
>>> type(round(x))  
<class "int">  
>>> type(round(x, 1))  
<class "float">  
>>> type(round(x, 0))  
<class "float">
```

Gördüğü gibi tek argüman tam sayı sonucu verirken iki argüman reel sayı sonucu vermektedir.

Kullanılan ikinci argüman negatif bir değerde olabilir: round(n, r) ifadesi n sayısını 10-r ile çarpmayı yapar. Örneğin round(n, -2) n sayısını 10-2 ile çarpmaktadır.

```
>>> x = 28793.54836  
>>> round(x)  
28794
```

```
>>> round(x, 1)
28793.5
>>> round(x, 2)
28793.55
>>> round(x, 0)
28794.0
>>> round(x, 1)
28793.5
>>> round(x, -1)
28790.0
>>> round(x, -2)
28800.0
>>> round(x, -3)
29000.0
```

Python için round fonksiyonu tam sayılar için de kullanılabilir. Burada ilk argüman tam sayı, ikinci argüman ise yuvarlama için sola doğru kayması beklenen basamak sayısını ifade eder. İkinci değer pozitif bir değerse sayının orijinal değeri elde edilir. Her durumda sonuç her zaman tam sayıdır.

```
>>> round(65535)
65535
>>> round(65535, 0)
65535
>>> round(65535, 1)
65535
>>> round(65535, 2)
65535
>>> round(65535, -1)
65540
>>> round(65535, -2)
65500
>>> round(65535, -3)
66000
>>> round(65535, -4)
70000
>>> round(65535, -5)
100000
>>> round(65535, -6)
0
```

## 2.4. Belirteçler

Matematik işlemlerinde genellikle x ve y tek karakterden oluşan değişkenler için kullanılır. Programcılar bunadan kaçınarak çok daha uzun, anlamlı ve açıklayıcı değişken isimleri seçmesi gereklidir. Bu nedenle t, at, y ve s gibi isimler yerine, toplam, araToplam, yükseklik ve süre gibi içeriğindeki değeri ifade eden değişken isimleri kullanmak çok daha etkilidir. Değişken ismi program içerisindeki kullanım amacına uygun olmalıdır. Değişken isimleri ne kadar doğru seçilirse program okuyan kişiler için o

kadar çabuk anlaşılır ve anlamlı olur.

Python, büyük küçük harf duyarlıdır ve değişken isimleri için kesin kurallar kullanır. Bir değişken ismi belirteç için bir örnektir. Belirteç, öğeleri isimlendirmek için kullanılan kelimedir. Belirteçler fonksiyon, sınıf ve metot gibi parçaları isimlendirmek için de kullanılır.

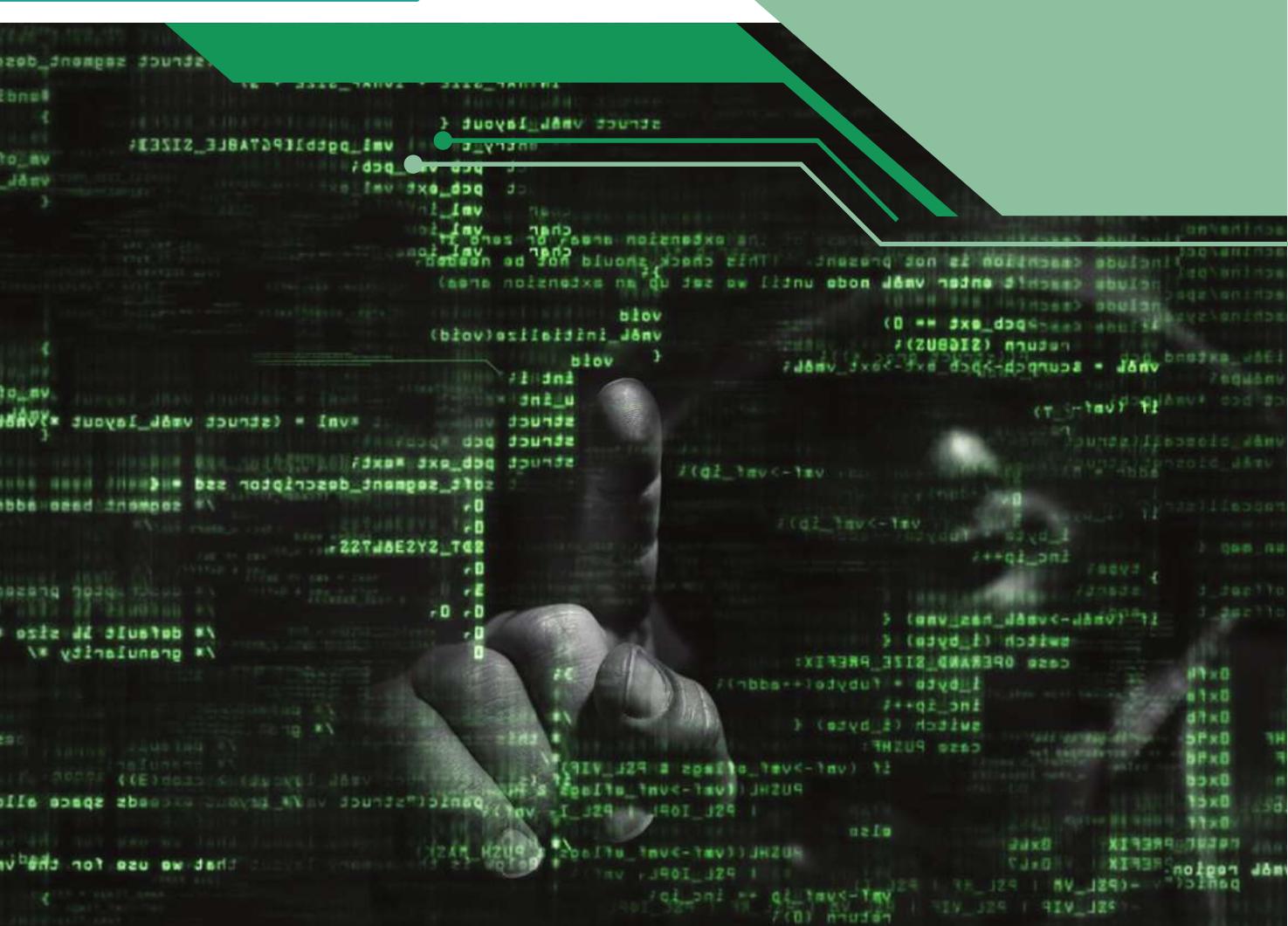
Belirteçlerin özellikleri şöyle sıralanabilir:

1. Bir belirteç en az bir karakter içermelidir.
2. Belirtecin ilk karakteri harf (küçük ya da büyük) ya da alt çizgi olmalıdır (ABCDEF<sup>GH</sup>IJKLM-NOPQRSTU<sup>VW</sup>XYZabcde<sup>fghijklm</sup>nopqrstuvwxyz\_).
3. Devam eden karakterler, harf (küçük ya da büyük), alt çizgi ya da sayı olabilir (ABCDEF<sup>GHI</sup>JKL-MN<sup>O</sup>PQRSTU<sup>VW</sup>XYZabcde<sup>fghijklm</sup>nopqrstuvwxyz\_0123456789).
4. Belirteçlerde boşluk dâhil diğer özel karakterler kullanılamaz.
5. Programlama diline ait ayrılmış kelimeler belirteç olarak kullanılamaz (Reserved Words).
6. Son olarak print, int, str ya da type gibi özel kelimeler kullanılabilmesine rağmen iyi bir program için bunlar asla önerilmmez.
7. Türkçe karakterler kullanılamaz.

Python için Ayrılmış Kelimeler			Python için Doğru Belirteçler	Python için Yanlış Belirteçler
and	del	from		<input checked="" type="checkbox"/> ara-toplam ( - sembolü kullanılamaz)
as	elif	global	<input checked="" type="checkbox"/> x	<input checked="" type="checkbox"/> ilk değer (boşluk kullanılamaz)
assert	else	if	<input checked="" type="checkbox"/> a2	<input checked="" type="checkbox"/> 4ogrenci (sayı ile başlayamaz)
break	except	import	<input checked="" type="checkbox"/> Toplam	<input checked="" type="checkbox"/> *2 ( * sembolü kullanılamaz)
class	False	in	<input checked="" type="checkbox"/> Toplam_Brut	<input checked="" type="checkbox"/> öğrenci (Türkçe karakter içeremez)
continue	finally	is	<input checked="" type="checkbox"/> Anahtar_10	<input checked="" type="checkbox"/> class (class ayrılmış kelime olduğu için kullanılamaz)
def	for	lambda		

*Sekil 2.7: Python için özel durumlar*

### 3. İFADELER VE ARİTMETİK İŞLEMLER



Bu bölümde;

- ✓ Python programlama dilinde kullanılan ifadeler ve aritmetik işlemler konusunda bilgi sahibi olacak,
- ✓ Operatör, işlem önceliği ve birleşim durumlarını açıklayabilecek,
- ✓ Python dilinde ortaya çıkan hata türlerini kavrayacaksınız.

### 3.1. Sabit Değerler

Sabit değerler (ör. 34) ve değişkenler (ör. x) basit ifadelerdir. Değerleri ve değişkenleri birleştirmek ve daha karmaşık ifadeler oluşturmak için operatörler kullanabiliriz. Aşağıdaki örnekte kullanıcı tarafından girilen 2 sayıyı toplamak için toplama operatörü kullanabiliriz.

```
deger1 = int(input("Bir sayı giriniz: "))

deger2 = int(input("Lütfen diğer sayıyı giriniz: "))

toplam = deger1 + deger2

print(deger1, "+", deger2, "=", toplam)
```

### 3.2. Python'da Sık Kullanılan Aritmetik İkili Operatörler

Python programlama dilinde kullanılan aritmetik operatörler aşağıdaki tabloda yer almaktadır:

İfade	Anlamı
x+y	Eğer x ve y rakamsa, x, y'ye eklenir. Eğer x ve y harf dizisi ise x ile y birleştirilir.
x-y	Eğer x ve y rakamsa, x'ten y çıkarılır.
x*y	Eğer x ve y rakamsa, x ile y çarpılır. Eğer x harf dizisi ise ve y rakamsa, x, y kez sıralanır. Eğer x rakamsa ve y harf dizisi ise y, x kez sıralanır.
x/y	Eğer x ve y rakamsa, x, y'ye bölünür.
x//y	Eğer x ve y rakamsa, x'in içinde y değişkenin katsayısını arar. Örn. $10 // 4 = 2$
x%y	Eğer x ve y rakamsa, bölüm işleminde x / y kalan kısmını verir.
x**y	Eğer x ve y rakamsa, x, y'nin kuvvetiyle çarpılır.

Tablo'da belirtilen operatörlerin ikili operatör olarak belirtilmesinin nedeni 2 işlenen (operand) üzerinde çalışmasıdır.  $x = y + z$  deyiminde, atama operatörünün sağ tarafında  $y + z$  bir toplamsal ifadedir.  $+$  operatörünün iki işleneni  $y$  ve  $z$ 'dir.

- $+, -, *, //, \%, \text{or } **$  operatörlerini 2 sayıya uyguladığımızda sonuç tam sayı olacaktır.
- `print(25//4, 4//25)` yazdığımızda 6 ve 0 sonuçlarını verecektir.
- Mod operatörü (%) sayı bölümünden kalanı hesaplar. `print(25%4, 4%25)` yazdığımızda 1 ve 4 sonuçlarını verecektir.
- $/$  operatörü 2 sayıya uygulandığında ondalıklı sayı döndürür. `print(25/4, 4/25)` yazdığımızda 6.25 0.16 sonuçlarını verecektir.



### 3.3. Karışık Türlü İfadeler

İfadeler tam sayı ve ondalıklı sayı değerlerini içerebilir. Örneğin,

```
x = 4  
y = 10.2  
toplam = x + y
```

X, bir tam sayıdır ve Y, bir ondalık sayıdır.  $x + y$  ifadesinin türü, / operatörü hariç, sadece tam sayılar içeren aritmetik ifadeler bir tam sayı sonuç üretir. Ondalık sayılarla uygulanan tüm aritmetik operatörler bir ondalıklı sonuç üretir.

### 3.4. Operatör Önceliği ve Birleşim

Farklı operatörler aynı ifadede yer alduğunda, aritmetiğin normal kuralları uygulanır. Tüm Python operatörlerinde öncelik (precedence) ve birleşme (associativity) vardır.

- Öncelik, bir ifade iki farklı türde operatörler içeriği zaman, hangisi ilk olarak uygulanacak?
- Birleşme, bir ifade aynı önceliğe sahip iki operatörleri içeriği zaman, hangisi ilk olarak uygulanacak?
- Çarpımsal operatörler (\*, /, // ve %) birbirleri ile eşit önceliğe sahiptir ve toplamsal operatörler (ikili + ve -) birbirleri ile eşit önceliğe sahiptir.
- Çarpımsal operatörleri, toplamsal operatörleri üzerinde önceliğe sahiptir.
- Standart aritmetikte olduğu gibi bir Python programcısı öncelik kurallarını geçersiz kilmak için ayrıcları kullanabilir ve çarpmadan önce toplama işleminin yapılmasını sağlayabilir.

Her satırdaki operatörler, altındaki operatörlerden daha yüksek bir önceliğe sahiptir. Bir satır içinde yer alan operatörler aynı önceliğe sahiptir.

Arity	Operatörler	Birleşim
İkili	**	Sağ
Tekli	+,-	
İkili	*, /, //, %	Sol
İkili	+,-	Sol
İkili	=	Sağ

### 3.5. İfadeleri Biçimlendirme

Python, aritmetik ifadeleri biçimlendirmek için önemli bir esneklik sunar;

```
3x + 2y-5
```

Cebirin aksine, Python dilinde örtülü hiçbir çarpma yoktur. Bu,  $3x$ 'i,  $3 * x$  olarak yazmak gereği anlamına gelir. \* operatörünü atamayabiliriz. Boşluk, operatör önceliğini etkilemez.

```
print(3*x + 2*y -5)
print(3*x+2*y-5)
print(3 * x + 2 * y - 5)
print(3 * x+2 * y-5)
print(3 * (x+2) * (y-5))
print(3*(x + 2)*(y - 5))
```

## 3.6. Yorumlar

Python programlama dilinde yazılan programlar uzadıkça karmaşık bir hâl alabilir. Bu da zamanla okumayı ve hatta hataları bulmayı zorlaştırabilir. Bu durumu ortadan kaldırmak ve programcıya yardımcı olması amacıyla programa küçük notlar hâlinde açıklama eklenmesi gerekebilir. # işaretile program içeresine yorum yazmak, açıklama eklemek mümkündür. Olası kullanım durumları aşağıda belirtilmiştir:

```
# Girilen dakika değerinin kaç saat olduğunu hesaplar.
yuzde=(dakika*100)/60
```

Bir satırın sonuna da yorum eklenebilir.

```
yuzde=(dakika*100)/60 # dikkat: bir saatten küçük değerler ondalıklı sonuç verecektir.
```

Yorum satırlarını derleyici, göz ardı eder ve ilgili satırın sonuna kadar programın çalışmasına herhangi bir etki etmez. Yorum satırları ile küçük hatırlatma yapmak veya daha başka programcılar için uyarılar eklemek program geliştirme sürecine yardımcı olabilir.

## 3.7. Hatalar

Python'da, üç genel hata türü vardır: söz dizimi hataları, çalışma zamanı istisnaları ve hataları. Yorumlayıcı tüm geçerli Python programlarını çalıştırılmak için tasarlanmıştır. Yorumlayıcı Python kaynak dosyasını okur ve yürütülebilir bir forma çevirir. Bu, çeviri aşamasıdır (Translation). Yorumlayıcı bir çeviri aşamasında geçersiz program deyimi algılsa bu, programın yürütülmesini sonlandıracak ve bir hata raporu verecektir. Bu tür hatalar programcının dili kötü kullanmasından kaynaklanmaktadır. Bir söz dizimi hatası, bir Python deyimi makine diline çevirmeye çalışırken yorumlayıcı tarafından algılanabilen yaygın bir hatadır. Aşağıdaki programda, yorumlayıcı bir hata mesajı verecektir. Hatalı bir atama işlemi yapmaya çalışmaktadır.

```
>>> y = 5
>>> x = y + 2
>>> y + 2 = x
```

```
File "error.py", line 3
y + 2 = x
^
SyntaxError: can't assign to operator
```

Diğer yaygın söz dizimi hataları şunlardır:

- Eşleşmeyen ayrıc gibi basit yazım hataları (`()3 + 4)`)
- Eşleşmeyen harf dizini tırnak işaretleri (`('hello')`)
- Hatalı girinti (faulty indentation)



Yorumlayıcı, söz dizimi hatalarını programı çalıştırılmaya başlamadan önce algılar ve bu nedenle söz dizimi hatalarını içeren bir programın herhangi bir parçasını çalışmaz.

### 3.7.1. Çalışma Zamanı Hataları

Doğru yazılmış bir Python programının hâlâ sorunları olabilir. Bazı dil hataları programın yürütülmemesi durumuna bağlıdır. Yorumlayıcı bir istisna yaratır. Çalışma zamanı istisnaları yorumcunun çeviri aşamasından sonra ve programın yürütme aşamasında ortaya çıkmaktadır.

Yorumlayıcı aşağıdaki gibi söz dizimsel olarak doğru bir ifade için bir istisna verebilir.

$x = y + 2 \text{ d}$

Eğer y değişkeni henüz atanmamışsa;

(NameError: name "y" is not defined) hatası mesajını verir.

Kullanıcıyı 32 (pay) ve 0 (payda) olarak yazdığınızda

Zero Division Error: division by zero

veya harf dizinini bölmeye kalktığınız takdirde

unsupported operand type(s) for /: "str" and "int" uyarısı alabilirsiniz.

### 3.7.2. Mantık Hataları

Böülünen/bölen ifadesi yerine bölen/böülünen olarak değiştirdiğinizde etkilerini düşünün. Program çalışır ve kullanıcı bölünene sıfır değeri girmedikçe, yorumlayıcı hata raporu vermeyecektir. Ancak he-saplanan cevap genel olarak doğru değildir. Program sadece bölünen ile bölen eşit olduğu zaman doğru cevap yazdıracaktır. Program bir hata içermektedir. Fakat yorumlayıcı sorunu algılayamaz. Bu tür bir hata, bir mantık hatası olarak bilinir. Yorumlayıcı, mantık hatalarının konumu için herhangi bir fikir sağlamakta güçsüzdür. Mantık hataları, bu nedenle, bulma ve onarmakta en zor olma eğilimindedir. Yorumlayıcı mantık hatalarına yönelik hiçbir yardımda bulunmaz.

## 3.8. Aritmetik Örnekler

**Örnek:** Sıcaklığı Fahrenheit derecesinden Celcius derecesine dönüştürmek istediğiniz varsayılmı.

```
>>> # Sıcaklık değerini okumak için
>>> dereceF = float (input ("Sıcaklığını F derece olarak girin:"))
>>> # Dönüşümü gerçekleştirin
>>> dereceC = 9/5 * (dereceF - 32)
>>> # Sonucu bildir
>>> print (dereceF, "derece F =", dereceC, "C derece")
Sıcaklığını F derece olarak girin: 212
212 ° F = 100.0 ° C
```

**Örnek:** Kullanıcının girdiği saniyeleri, saat, dakika ve saniye olarak parçalara ayıran programda tam sayı bölme ve modül kullanır.

```
>>> saniye = int (input ("saniye sayısını girin:"))
>>> saat = saniye // 3600 # 3600 saniye = 1 saat
```

```
>>> saniye = saniye% 3600  
>>> dakika = saniye // 60 # 60 saniye = 1 dakika  
>>> saniye = saniye% 60  
print (saat, "sa", dakika, "dk", saniye, "sn")
```

Kullanıcı 10000 girerse program 2 saat, 46 dakika, 40 saniye yazdırır.

**Örnek:** Dijital saat ekranlarında yapıldığı gibi dakika ve saniyelerde tek haneli değerlerin önüne sıfır koymak için bazı ekstra aritmetik gerekir.

```
>>> saniye = int (input ("saniye sayısını girin:"))  
>>> saat = saniye // 3600 # 3600 saniye = 1 saat  
>>> saniye = saniye% 3600  
>>> dakika = saniye // 60 # 60 saniye = 1 dakika  
>>> saniye = saniye% 60  
>>> print (saat, ":", sep = "", end = "")  
>>> onlar = dakika // 10  
>>> birler = dakika % 10  
>>> print (onlar, birler, ":", sep = "", end = "")  
>>> onlar = saniye // 10  
>>> birler = saniye% 10  
>>> print (onlar, birler, sep = "")
```

### 3.9. Aritmetik İfadeler

Python basit aritmetik aracıyla bir değişkeni değiştiren bir deyimin basitleştirilmesinde daha genel bir yol sağlar. Örneğin  $x = x + 5$  deyimi  $x += 5$  olarak kısaltılabilir. Bu ifade “ $x$ ’i 5 arttır.” anlamına gelir.  $x * = y + z$  deyimi ile  $x = x * (y + z)$  deyimi aynıdır.

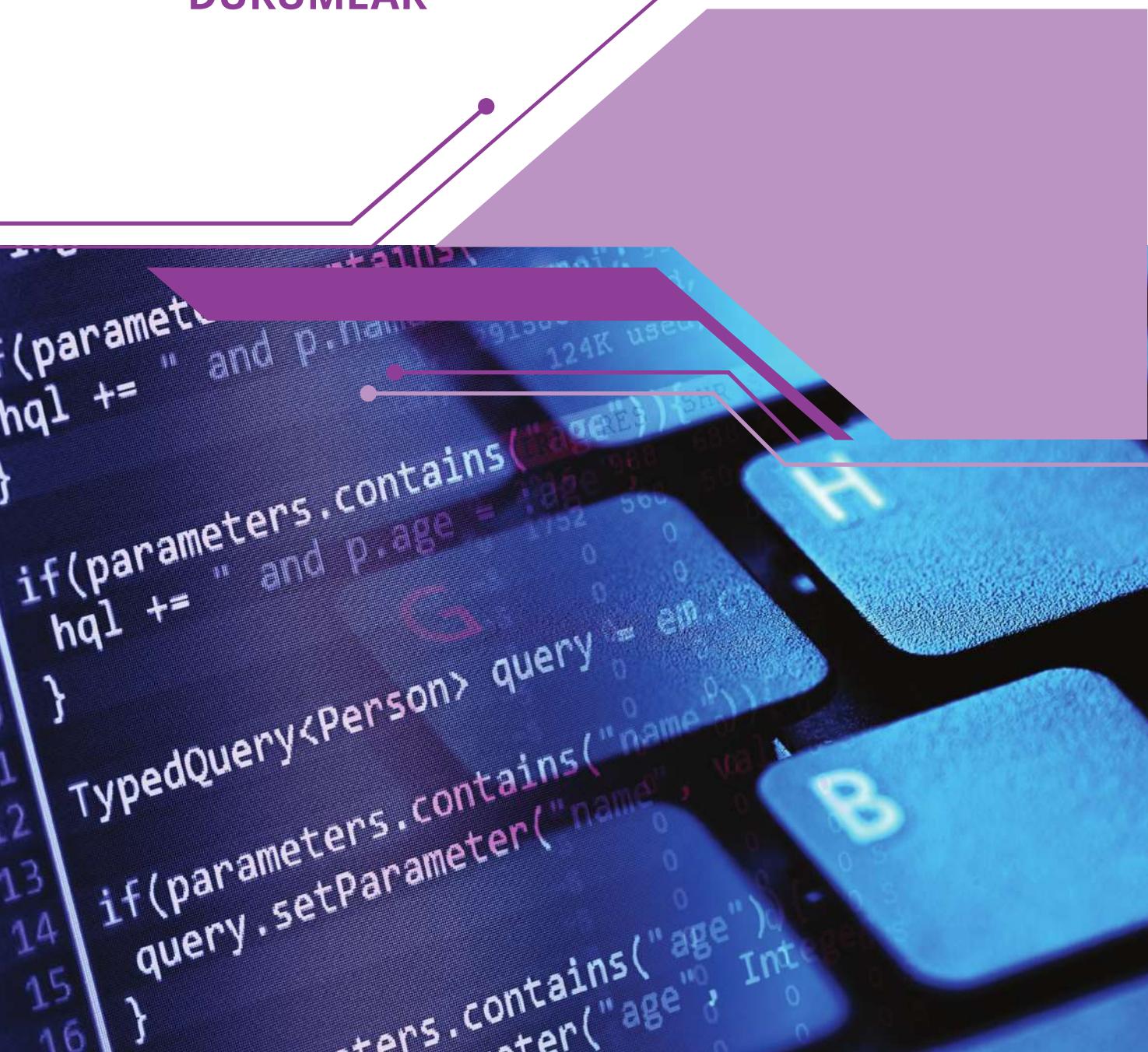
**Örnek:**

```
>>> y=5  
>>> y=y+15  
>>> x=5  
>>> x+=15  
>>> print("x-->",x," y-->",y)  
x--> 20 y--> 20
```

Bu örnekte ilk olarak  $x$  ve  $y$  değişkenlerine 5 değeri aktarılmış sonra aynı değişkenlerin üzerine 15 değeri farklı yöntemlerle eklenmiştir. Print komutuyla ekrana yazdırıldığından çıkan değerlerin aynı olduğu görülecektir.

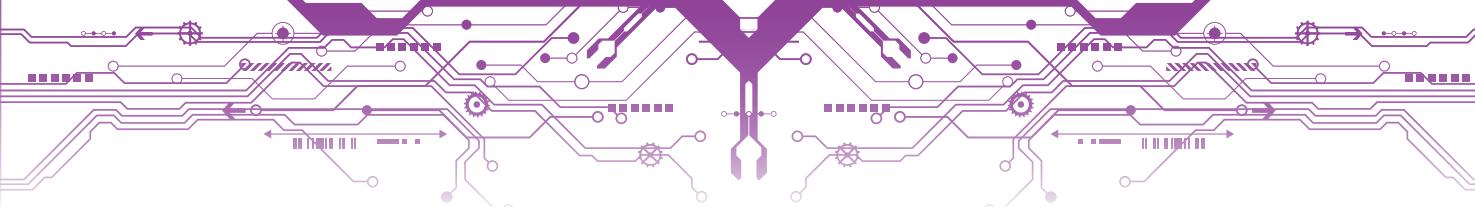
İki veya daha fazla ifadeyi karşılaştırmak gereken koşullu durumlarda koşul ifadeleri kullanılır. Koşul ifadeleri, program yazarken çok kullanılan ifadelerden biridir. Farklı türden durum ve/veya koşulları ifade etmek için kullanılan koşul ifadeleri sırayla açıklanmıştır.

## 4. KOŞULLU DURUMLAR



Bu bölümde;

- ✓ Program yazma sürecinde kullanılan koşullu durumlar konusunda bilgi sahibi olacak,
- ✓ İç içe koşul durumu ifade etmeyi kavrayacak,
- ✓ Koşul durumlarında yapılan hatalar konusunda fikir sahibi olacaksınız.



## 4.1. Boolean İfadesi

Bilgisayar bilimi temelde 0 ve 1 değerleri üzerine kurulmuştur; 0 değeri False(Yanlış), 1 değeri True(Doğru) demektir. Bu ifadelere Boolean (bool) İfadeleri denir. Doğru ve Yanlış değerleri korumak için kullanılan tipe bool adı verilmektedir. Sadece iki Boolean ifade değeri vardır:

- True (Doğru) (1)
- False (Yanlış) (0)

Python açısından büyük harfle başlamaları önemlidir. Boolean deyimi, İngiliz Matematikçi George Boole'in soyadından gelmektedir. Soyut matematiğin bir dalı olan Boole Cebiri (mantık cebiri), George Boole'nin mantıksal ifadelerle ilgili çalışmalarına ithaf edilmiştir.

## 4.2. Python'da İlişkisel Operatörler

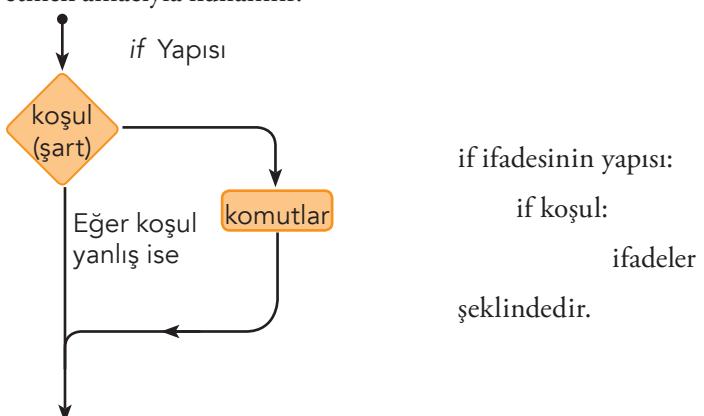
İfade	Anlamı
<code>x==y</code>	Eğer x ve y birbirine eşitse (matematiksel olarak) doğrudur, değilse yanlıştır.
<code>x&lt;y</code>	Eğer x, y'den küçükse doğrudur; değilse yanlıştır.
<code>x&lt;=y</code>	Eğer x, y'den küçük ya da eşitse doğrudur; değilse yanlıştır.
<code>x&gt;y</code>	Eğer x, y'den büyükse doğrudur; değilse yanlıştır.
<code>x&gt;=y</code>	Eğer x, y'den büyük ya da eşitse doğrudur; değilse yanlıştır.
<code>x!=y</code>	Eğer x, y'den farklı ise (büyük ya da küçük) doğrudur; değilse yanlıştır.

Python'da ilişkisel operatörlere örnekler

İfade	Değer
<code>10 &lt; 20</code>	True
<code>10 &gt;= 20</code>	False

## 4.3. "if" İfadesi

Türkçede EĞER anlamına gelen if ifadesi, adından da anlaşılacağı üzere, bir koşula bağlı durumları kontrol etmek amacıyla kullanılır.



Burada koşul "true" değer alıyorsa yani koşul sağlanıyorsa blok kısmındaki ifadeler gerçekleşecektir. Eğer koşul "false" değer alıyorsa yani koşul sağlanmıyorsa blok kısmındaki ifadeler gerçekleşmeden program devam edecektir.

```

print("Lütfen bölme için iki sayı giriniz.")
bolum=int(input("Lütfen bölme için ilk sayınızı giriniz:"))
bolen=int(input("Lütfen bölme için ikinci sayınızı giriniz:"))
if bolen!=0:
    print(bolum,"/",bolen,"=", bolum/bolen)

```

Ekran çıktısı aşağıdaki gibi olur:

```

Lütfen bölme için iki sayı giriniz.
Lütfen bölme için ilk sayınızı giriniz:32
Lütfen bölme için ikinci sayınızı giriniz:8
32/8=4.0

```

Başa değerler için test edecek olursak:

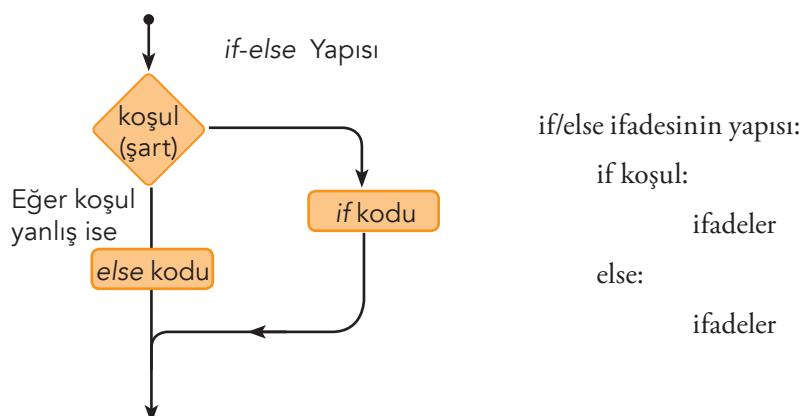
```

Lütfen bölme için iki sayı giriniz.
Lütfen bölme için ilk sayınızı giriniz:32
Lütfen bölme için ikinci sayınızı giriniz:0

```

#### 4.4. "if/else" İfadesi

"if/else" ifadesi, "if" ifadesi ile birlikte çalışır ve if koşullarının sağlanmadığı tüm durumları kapsar.

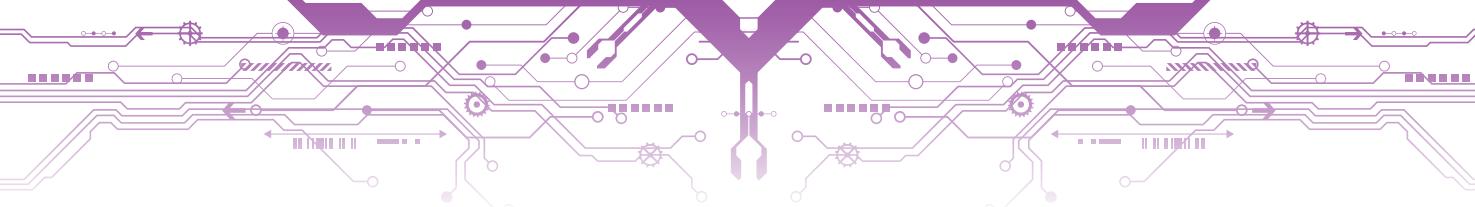


else, isteğe bağlı bir ifadedir ve if bloku ile birlikte sadece bir kez kullanılır.

```

print("Lütfen bölme için iki sayı giriniz.")
bolum=int(input("Lütfen bölme için ilk sayınızı giriniz:"))
bolen=int(input("Lütfen bölme için ikinci sayınızı giriniz:"))
if bolen!=0:
    print(bolum,"/",bolen,"=", bolum/bolen)
else
    print("Sıfıra bölme işlemi yapılamaz.")

```



## Ekran Çıktısı

```
Lütfen bölme için iki sayı giriniz.  
Lütfen bölme için ilk sayınızı giriniz:32  
Lütfen bölme için ikinci sayınızı giriniz:0  
Sıfıra bölme işlemi yapılamaz.
```

## Örnek

```
# Ağacın boyunu sorgulayan program.  
boy = input("Ağacın boyu kaç cm? ")  
if int(boy) > 150:  
    print ("Ağacın boyu uzun demek ki!")  
else:  
    print ("Ağacın boyu kısa")
```

## 4.5. Birleşik Boolean İfadeleri

Boolean ifadesi ile bir ilişkisel operatörle birleştirilerek daha karmaşık Boolean ifadeleri oluşturabilir. Bu tür durumlarda 3 farklı mantıksal operatörden yararlanılabilir: **and**, **or** ve **not**. Mantıksal operatörler aracılığı ile iki veya daha fazla Boolean ifadesinin kullanıldığı deyimlere **Birleşik Boolean İfadeleri** denir.

*Birleşik Boolean İfadeleri oluşturan mantıksal operatörler –e1 ve e2 örneği*

e1	e2	e1 and e2	e1 or e2	not e1
False (yanlış)	False (yanlış)	False (yanlış)	False (yanlış)	True (doğru)
False (yanlış)	True (doğru)	False (yanlış)	True (doğru)	True (doğru)
True (doğru)	False (yanlış)	False (yanlış)	True (doğru)	False (yanlış)
True (doğru)	True (doğru)	True (doğru)	True (doğru)	False (yanlış)

Mantıksal operatörlerden and ve or sola birleşmeli, not sağa birleşmelidir. Örneğin,

$x \leq y$  and  $x \leq z$  ifadesi

$(x \leq y)$  and  $(x \leq z)$  olarak işlem görür.

$x = 10$ ,  $y = 20$  olarak veriliyor. Buna göre aşağıda verilen kod örneklerini inceleyiniz.

$b = (x == 10)$	# b'ye True değerini atar.
$b = (x != 10)$	# b'ye False değerini atar.
$b = (x == 10 \text{ and } y == 20)$	# b'ye True değerini atar.
$b = (x != 10 \text{ and } y == 20)$	# b'ye False değerini atar.
$b = (x == 10 \text{ and } y != 20)$	# b'ye False değerini atar.
$b = (x != 10 \text{ and } y != 20)$	# b'ye False değerini atar.
$b = (x == 10 \text{ or } y == 20)$	# b'ye True değerini atar.
$b = (x != 10 \text{ or } y == 20)$	# b'ye True değerini atar.
$b = (x == 10 \text{ or } y != 20)$	# b'ye True değerini atar.
$b = (x != 10 \text{ or } y != 20)$	# b'ye False değerini atar.

## 4.6. Pass İfadesi

Pass ifadesi Python'da herhangi bir işlem yapmadan geçeceğimiz durumlarda kullanılır. Kısaca "Hiçbir şey yapmadan yola devam et!" anlamını katar.

```
if x == 2:  
    print(x)  
else:  
    pass # x 2'ye eşit değilse hiçbir şey yapma  
if x == 2:  
    print(x) # yalnızca x 2'ye eşitse yazdır
```

## 4.7. Kayan Noktalı Eşitlik

Eşitlik operatörü (==) gerçek eşitlik olup olmadığını kontrol eder. Ancak kayan noktalı sayılarla işlem yaparken bu durum sorun oluşturabilir. Örneğin,

```
d1 = 1.11 - 1.10  
d2 = 2.11 - 2.10  
print("d1 =", d1, " d2 =", d2)  
if d1 == d2:  
    print("Aynı")  
else:  
    print("Farklı")
```

Normalde, matematiksel işlem yapıldığında aşağıdaki gibi bir eşitliğin olduğu görülür.

$$1.11 - 1.10 = 0.01 = 2.11 - 2.10$$

Ancak; bilgisayar sistemlerinde tüm işlemler bitler ( 0 ve 1 ) ile yapıldığından sayıların hafızadaki değerleri de bu şekilde tutulur. Kayan noktalı sayılar bilgisayar sisteminde ikilik tabanda karşılığı ve bunun köküst şeklinde temsil edilir. Bu sebepledir ki;

$$d1 = 0.01000000000000009$$

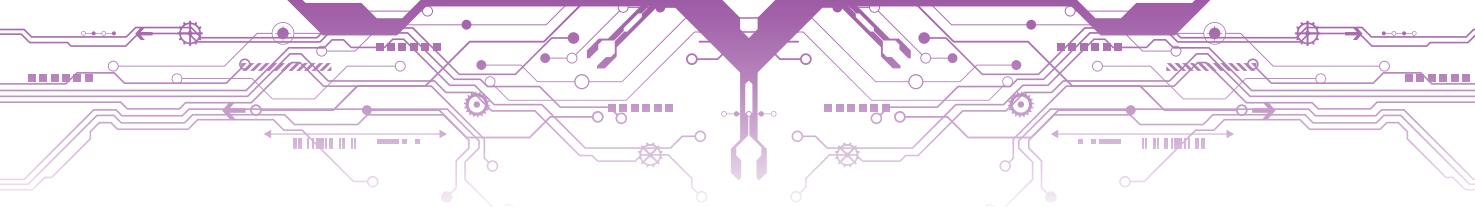
$$d2 = 0.00999999999999787$$

olduğu için, yazılan kod çalıştırıldığı zaman d1 ve d2'nin farklı olduğunu yazacaktır.

## 4.8. İç İçe Koşul İfadeleri

Karşılaştırma yapıları kullanırken bazı durumlarda istenilen koşulların birden fazla şartta aynı anda uyması istenebilir. Bu durumda koşul yapılarının birbirinin içinde kullanılması gereklidir. Bu şekilde bir yapı kullanıldığında istenilen komut veya komut kümelerinin yapılması için iki koşul ifadesinin de True olması gereklidir. Örneğin;

```
deger = int(input("Lütfen 0....10 aralığında bir tam sayı giriniz: ")  
if deger >= 0 and deger <= 10: # ikili koşul kontrolü  
    print("aralıkta")  
print("tamamlandı")
```



yerine, her koşul ayrı olacak şekilde daha basit bir iç içe koşul yazılabılır.

```
değer = int(input("Lütfen 0...10 aralığında bir tam sayı giriniz: "))
    if değer >= 0: # İlk kontrol
        if değer <= 10: # İkinci kontrol
            print("aralıkta")
        print("tamamlandı")
```

## 4.9. Çok Yönlü Karar İfadeleri

Basit if/else ifadesinde iki farklı koşul varken çok yönlü karar ifadelerinde daha fazla koşulun gerçekleşme durumuna göre işlem yapılır. Bunun için kod yazılırken iç içe if/else ifadeleri gerekir. Örneğin;

```
değer = int(input("Lütfen 0 yada 5 tam sayı değerlerinden birini
girin: "))
if değer < 0:
    print("çok küçük")
else:
    if değer == 0:
        print("sıfır")
    else:
        if değer == 1:
            print("bir")
        else:
            if değer == 5:
                print("beş")
            else:
                print("çok büyük")
print("Tamamlandı")
```

Python'da çok yönlü koşullu durumlar iç içe if ifadeleri yanında if/elif/else ifadesi ile birlikte de kullanılır. Elif ifadesi, else if ifadesinin kısaltmasıdır. Örneğin,

```
# Klavyeden girilen 0-5 arasındaki sayıların yazı karşılığını veren
program
değer = int(input("Lütfen 0 - 5 arasında bir değer girin: "))
if değer < 0:
    print("çok küçük")
elif değer == 0:
    print("sıfır")
elif değer == 1:
    print("bir")
elif değer == 2:
    print("iki")
elif değer == 3:
    print("üç")
```

```
elif deger == 4:  
    print("dört")  
elif deger == 5:  
    print("beş")  
else:  
    print("çok büyük")  
print("Tamamlandı")
```

## Örnek

```
if 9 > 5:  
    print("Evet, 9 5'ten büyük")  
if 9 != 5:  
    print("Evet, 9 5'e eşit değildir")  
# Başka bir örnek  
if not (10 == 4) and 9 > 5:  
    print("Tabii ki, çok basit bir karşılaştırma bu")  
else:  
    print(":(")
```

## 4.10. Çok Yönlü ve Zincirleme Durum İfadeleri

İkiden fazla olasılığın olduğu durumlarda ikiden fazla dallanmaya (yol) gereksinim duyuyoruz. Bu tür durumlarda zincirleme koşul ifadeleri kullanılır. Her koşul sırasıyla sınanır. İlk yanlış ise sonraki kontrol edilir ve yazılan kodun tamamı bu şekilde çalıştırılır. Koşullardan biri doğru ise ilgili dal yürütülür ve cümlenin işlevi biter. Eğer birden fazla koşul doğru ise sadece ilk karşılaşılan doğru dal çalışır.

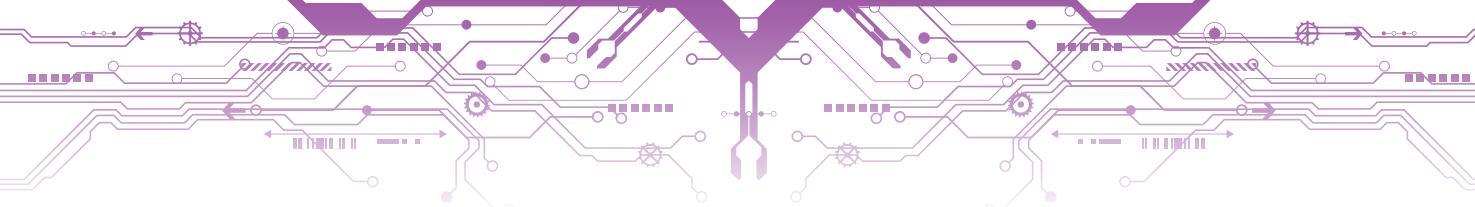
### Çok Yönlü Koşullu Durum İfadesi

```
value = int (input("Lütfen 0...3  
arasında bir tam sayı giriniz:"))  
if value==0: Kontrol 1  
  
    print("sıfır")  
elif value==1: Kontrol 2  
  
    print("bir") Sonuç  
elif value==2:  
    print("iki")  
elif value==3:  
    print("üç")  
else  
  
    print("çok büyük")  
print("Tamamlandı")
```

Diger kontroller  
atlanır.

### Zincirleme Durum İfadesi

```
value = int (input("Lütfen 0...3  
arasında bir tam sayı giriniz:"))  
if value==0: Kontrol 1  
  
    print("sıfır")  
if value==1: Kontrol 2  
  
    print("bir") Sonuç  
if value==2: Kontrol 3  
  
    print("iki")  
if value==3: Kontrol 4  
  
    print("üç")  
if value>3: Kontrol 5  
  
    print("çok büyük")  
print("Tamamlandı")
```



Yukarıdaki problemde kullanıcının klavyeden "1" girmesi durumunda işletilecek şartlar belirtilmiştir. Çok Yönlü Koşullu Durum yapısında Kontrol 1 kısmına girilecek fakat şart sağlanmadığı için sonuç kısmına girilmeyecek ve Kontrol 2 kısmına girilecektir. Geçilen bu aşamada şart sağlandığı için print() komutu işletilecek ve diğer tüm kontrol satırları atlanacaktır. Böylece fazla koşul kontrollerinden kaçınılmış olacaktır. Ancak Zincirleme Durum ifadesinde tüm şartlar sırasıyla kontrol edilecek, şartı sağlayanlar için print() satırları yapılacaktır.

## 4.11. Koşullu İfadeler

Koşullu ifadelerin genel yapısı aşağıdaki gibidir.

Birinci Durum

**if**

Koşul

**else**

İkinci Durum

Koşul doğru ise koşullu ifadenin sonucu birinci durumdur. Koşul, "if" ifadesinde de görülebilen Boolean ifadesidir. Eğer koşul yanlış ise koşullu ifadenin sonucu ikinci durumdur.

```
n = int(input("Bir sayı giriniz: "))
print("|", n, "| = ", (-n if n < 0 else n), sep="")
```

### Ekrان Çıktısı

```
Bir sayı giriniz: -34
|-34| = 34
Bir sayı giriniz: 100
|100| = 100
```

Örnekte n değişkenine klavyeden giriş alınmış ve kullanıcı -34 değerini girmiştir. print() komutu ile ifade yazdırılırken koşullu ifade kullanılmış şart olarak n değişkeninin 0 'dan küçük durumu (if n < 0) kontrol edilmiştir. Şart doğru ise -n işlemi, yanlış ise n değeri yazdırılacaktır.

sep Parametresi : print() komutunda birden fazla değer yazdırılırken, yazdırılan ifadeler arasında istenilen bir karakter eklemek için sep parametresi kullanılır.

```
>>> print("T","C",sep=".")
T.C
```

### 4.11.1. Koşullu İfadelerde Hatalar

```
deger = int(input("Lütfen 0 - 5 arasında bir değer girin: "))
cevap="aralıkta değil" #Varsayılan Cevap
if deger == 0:
    cevap="Sıfır"
elif deger == 1:
    cevap="Bir"
elif deger == 2:
```

```
cevap="iki"  
elif deger == 3:  
    cevap="üç"  
elif deger == 4:  
    cevap="dört"  
elif deger == 5:  
    cevap="beş"  
print("Girdiğiniz sayı",cevap)
```

```
Lütfen 0 - 5 arasında bir değer girin: 2  
Girdiğiniz sayı iki  
>>>  
Lütfen 0 - 5 arasında bir değer girin: 8  
Girdiğiniz sayı aralıkta değil
```



### Düşünelim/Deneyelim

Mantıksal operatörlerden and ve or operatörlerinin karışması durumu, en yaygın programlama hatasıdır. Programcılar, Python kaynak kodunu analiz etmek için Pylint'i (<http://www.pylint.org/>) kullanabilirler.

## 4.12. Mantık Karmaşası

Python, çok karmaşık durum/koşul ifadelerini oluşturmak için gerekli araçları sağlar. Ancak önemli olan, mantık karmaşasına yol açmadan kullanabilmektir. Boolean ifadeleri and ve not ile birlikte kullanılmak istendiğinde, karmaşık mantığa dayalı koşullar oluşturmamıza olanak sağlar. Örneğin aşağıda verilen 4 farklı Boolean ifade kodu çalıştırıldığı zaman aynı sonucu verecektir.

1. not (a == b and c != d)
2. not (a == b and not (c == d))
3. not (a == b) or not (c != d)
4. a != b or c == d

Ancak unutulmamalıdır ki;

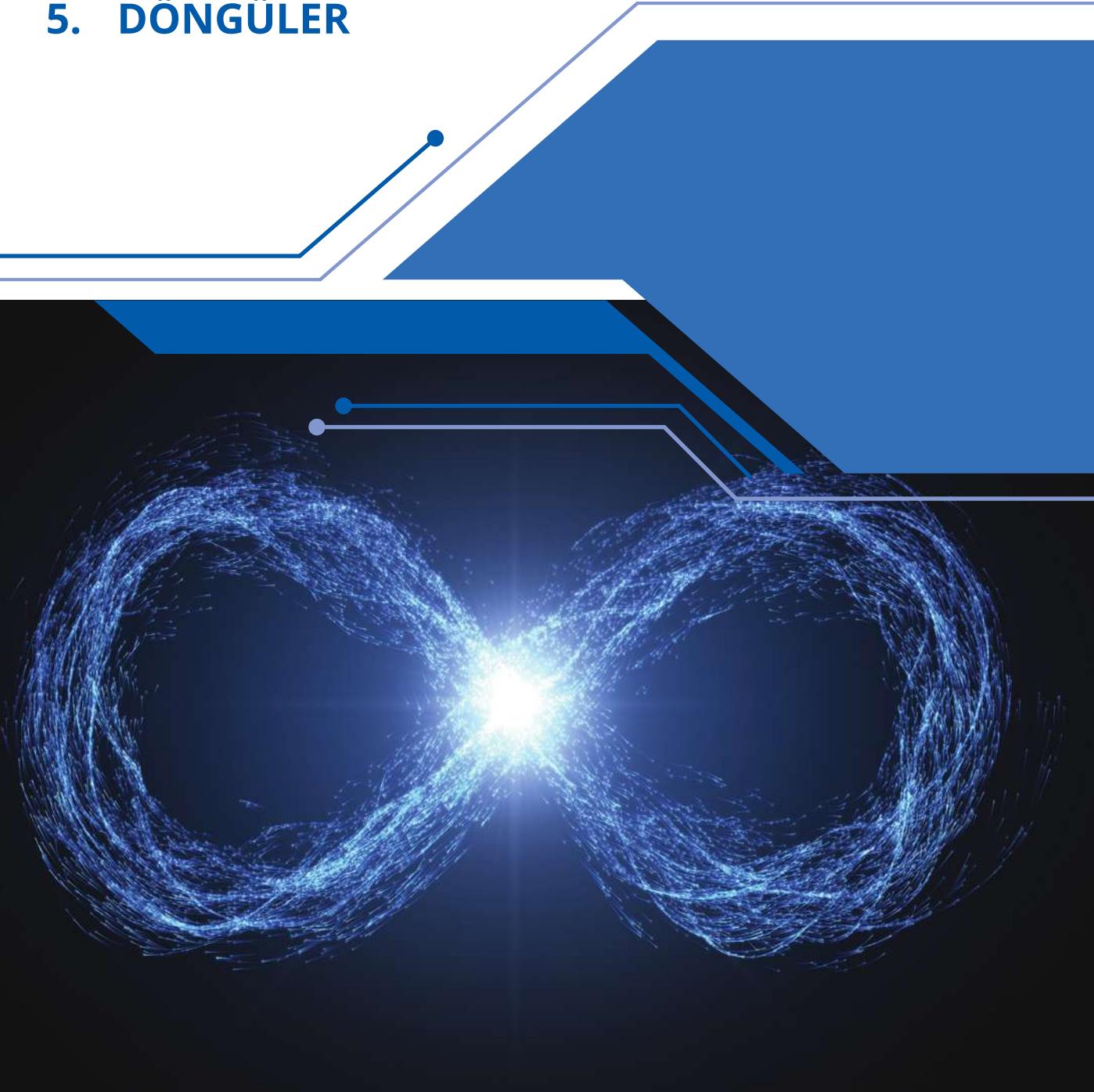
- Çalıştırılırken en verimli yöntem basit düzeydeki mantıksal ifadelerdir.
- Basit düzeydeki mantıksal ifadeleri yazmak ve çalıştmak daha kolaydır.
- Basit düzeydeki mantıksal ifadeler, çalıştırılırken de en verimli yöntemdir.
- Basit düzeydeki mantıksal ifadelerin değiştirilmesi, düzenlenmesi ve genişletilmesi de daha kolaydır.

## Tartışalım



1. Bir Boolean ifadesi hangi değerleri alabilir?
2. Boolean deyimi nereden gelmektedir?
3. Python'da True deyimi hangi tam sayıya denk gelmektedir?
4. Python'da False deyimi hangi tam sayıya denk gelmektedir?
5.  $x, y, z = 3, 5, 7$  olarak verilmektedir. Buna göre, aşağıda verilen işlemlerin sonucunu bulunuz.
  - (a)  $x == 3$
  - (b)  $x < y$
  - (c)  $x \geq y$
  - (d)  $x \leq y$
  - (e)  $x != y - 2$
  - (f)  $x < 10$
  - (g)  $x \geq 0 \text{ and } x < 10$
  - (h)  $x < 0 \text{ and } x < 10$
  - (i)  $x \geq 0 \text{ and } x < 2$
  - (j)  $x < 0 \text{ or } x < 10$
  - (k)  $x > 0 \text{ or } x < 10$
  - (l)  $x < 0 \text{ or } x > 10$

## 5. DÖNGÜLER



Bu bölümde;

- ✓ Python dilinde kullanılan döngü yapılarını kavrayacak,
- ✓ Döngü yapılarını ve kontrol yapılarını bir arada kullanabilecek,
- ✓ While ve for döngülerini içeren programlar geliştirebilecek,
- ✓ İç içe döngü yapılarını kullanan program yazabilecek,
- ✓ Döngü yapılarından çıkmak için kullanılan komutları kavrayacaksınız.



## 5.1. Döngü Yapıları

Döngüler, sıralı bir kod bloğunun istenilen sayıda tekrarlanmasıdır. Döngü ve karar yapıları, algoritma oluşturma ve programlamada birçok problemin çözümünde kullanılır.

Aşağıdaki kod bloku 1'den 5'e kadar sayıları ekrana yazar.

```
print(1)  
print(2)  
print(3)  
print(4)  
print(5)
```

Ekran Çıktısı

```
1  
2  
3  
4  
5
```

Ancak, 1'den 10.000'e kadar yazmak gerekirse böyle bir çözüm yolu doğru olmayacaktır!

Tekrar sayısı fazla olduğunda döngü yapıları tercih edilmelidir. Programlamada tekrar sayısının sayılması bir değişken yardımı ile elde edilebilir. Python dilinde döngü için while ve for döngü yapıları kullanılır.

## 5.2. For Döngüsü

For döngüleri belirli sayıda işlemlerin tekrarlanması için kullanılan döngülerdir. For döngüleri başlangıç ve bitiş değerleri arasında artım miktarına göre istenilen sayıda tekrar yapar.

Yukarıdaki örnek çıktı for döngüsüyle yapılmak istenseydi yazılması gereken kodlar şu şekilde olurdu:

```
for n in range(1,6)  
    print(n)
```

Örnek Kullanım

```
for n in range(1, 11):  
    print(n)
```

range (1,11) ifadesi, n değerinin hangi aralıkta çalışacağını gösterir. n'in alacağı değerler: 1, 2 , 3 , 4 , 5, 6, 7, 8, 9, 10 yani  $1 \leq n < 11$ 'dir. n'in ilk değeri 1 olup her çalışma anında sırayla artmaktadır.

### 5.2.1. For Döngüsü İçin Söz Dizimi

**range (başlangıç değeri, son değer, artırma/azaltma değeri) :**

**Başlangıç değeri:** Döngü değişkeninin alacağı ilk değerdir. Eğer boş bırakılırsa 0 olarak belirlenir.

**Son değer:** Döngü değişkeninin bitiş değeridir. Boş bırakılmamalıdır.

**Artırma/azaltma değeri:** Döngü değişkeninin artırma veya azaltma miktarını belirler. Eğer boş bırakılırsa, 1 olarak belirlenir.

Başlangıç, bitiş, artırma ve azaltma değerlerinin hepsi tam sayı olmalıdır. Ondalıklu değerler veya diğer veri türleri kullanılmaz. Bunun dışında range ifadesi esnek kullanımına sahiptir:

```
for n in range(21, 0, -3):
    print(n, end=" ")
```

#### Ekran Çıktısı

```
21 18 15 12 9 6 3
```

**end parametresi:** print() içerisinde kullanılan end, bir parametre olarak görev yapar. İşlevi ise yazdırılacak istenen ifadelerin sonuna hangi karakterin geleceğini belirler. Varsayılan olarak "\n" karakteri ile birlikte gelir. Yani yazılan ifade bitince bir alt satırda geçer.

#### Başka bir örnek

```
top= 0
for i in range(1, 100): # burada döngü değişkeni olarak i kullanılmıştır.
    top+= i
print(top)
```

#### Ekran Çıktısı

```
4950
```

range(1000) denildiğinde 999'a kadar olan sayıların toplamı işlemini yapar ve ekrana 4950 yazar. Çünkü range() komutunda bitiş değeri döngüye dahil değildir.

### 5.2.2. For Döngüsü İçin Farklı Örnekler

#### Örnek

```
range(10) → 0,1,2,3,4,5,6,7,8,9
range(1, 10) → 1,2,3,4,5,6,7,8,9
range(1, 10, 2) → 1,3,5,7,9
range(10, 0, -1) → 10,9,8,7,6,5,4,3,2,1
range(10, 0, -2) → 10,8,6,4,2
```



```
range(2, 11, 2) → 2,4,6,8,10
range(-5, 5) → -5,-4,-3,-2,-1,0,1,2,3,4
range(1, 2) → 1
range(1, 1) → ()
range(1, -1) → ()
range(1, -1, -1) → 1,0
range(0) → ()
```

- range içerisinde 1 değer varsa bitiş değerini gösterir. 0'dan başlayıp birer artarak çalışır.
- range içerisinde 2 değer varsa başlangıç ve bitiş değerini simgeler ve birer artarak ilerler.
- 3 değer varsa başlangıç, bitiş ve artma miktarını ifade eder.

## Örnek

10'un katlarını yazmak için aşağıdaki kod satırları kullanılabilir.

```
for i in range(16):
    print("{0:3} {1:16}".format(i, 10**i))
```

## Ekran Çıktısı

0	1
1	10
2	100
3	1000
4	10000
5	100000
6	1000000
7	10000000
8	100000000
9	1000000000
10	10000000000
11	100000000000
12	1000000000000
13	10000000000000
14	100000000000000
15	1000000000000000

**format() metodu:** print() komutunda çıktı verilirken yazdırılmak istenilen değerlerde hizalama, ekranda istenildiği yere yazdırma gibi biçimlendirme işlemlerinde kullanılan bir metotdur.

## Örnek

```
>>> print("{} {}yi seviyor!".format("Ali", "Ayşe"))
"Ali Ayşe"yi seviyor!
>>> print("{} {} yaşında bir {}dur".format("Ahmet", "18", "futbolcu"))
"Ahmet 18 yaşında bir futbolcudur"
```



## Örnek

```
for i in range(10):
    print(i, end=" ")
    if i == 5:
        i = 20
    print("({})".format(i), end=" ")
print()
```

## Ekran Çıktısı

```
0 (0) 1 (1) 2 (2) 3 (3) 4 (4) 5 (20) 6 (6) 7 (7) 8 (8) 9 (9)
```

## Örnek

Aşağıdaki örnek girilen ifadenin harfleri üzerinde işlem yapar.

```
kelime= input("Bir kelime yaz: ")
for harf in kelime:
    print (harf)
```

## Ekran Çıktısı

```
Bir kelime yaz: Python
P
Y
t
h
o
n
```

## Örnek

Girilen kelimenin sesli harflerinin bulunmasını sağlayan program

```
kelime = input("Cümle Giriniz: ")
sesliHarfSayisi = 0
for c in kelime:
    if c == "A" or c == "a" or c == "E" or c == "e" \
    or c == "I" or c == "ı" or c == "İ" or c == "i" \
    or c == "O" or c == "o" or c == "Ö" or c == "ö" \
    or c == "U" or c == "u" or c == "Ü" or c == "ü":
        print(c, ",", sep=" ", end=" ")
        sesliHarfSayisi += 1
print(", sesliHarfSayisi, "sesli)", sep=" ")
```



### Ekran Çıktısı

```
Cümle Giriniz: Bugün hava çok güzel  
u , ü , a , a , o , ü , e , ( 7 sesli)
```

## 5.3. İç İçe Döngüler

İf ifadelerinde olduğu gibi while ve for blokları, başka döngü yapılarını içerebilir. İç içe döngülerin çalışma mantığını anlayabilmek için programın bir çarpım tablosu ürettiğini varsaymak gereklidir. Benzer şekilde satır ve sütun değerleri olan bir tablo gibi düşünülebilir.

### Örnek

```
# Satır oluşturmak için  
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))  
for satir in range(1, sayi + 1):  
    print("Satır #", satir)
```

### Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10  
Satır # 1  
Satır # 2  
Satır # 3  
Satır # 4  
Satır # 5  
Satır # 6  
Satır # 7  
Satır # 8  
Satır # 9  
Satır # 10
```

Satırların yanına sütun eklemek istenirse kodlar aşağıdaki şekilde düzenlenmesi gereklidir.

```
# Hem satır hem de sütun oluşturmak için :  
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))  
for satir in range(1, sayi + 1):  
    for sutun in range(1, sayi + 1):  
        deger = satir*sutun  
        print(deger, end=" ")  
    print()
```



## Ekran Çıktısı

Lütfen tablo ölçüsünü giriniz: 10										
1	2	3	4	5	6	7	8	9	10	
2	4	6	8	10	12	14	16	18	20	
3	6	9	12	15	18	21	24	27	30	
4	8	12	16	20	24	28	32	36	40	
5	10	15	20	25	30	35	40	45	50	
6	12	18	24	30	36	42	48	54	60	
7	14	21	28	35	42	49	56	63	70	
8	16	24	32	40	48	56	64	72	80	
9	18	27	36	45	54	63	72	81	90	
10	20	30	40	50	60	70	80	90	100	

>>>

Tablonun formatını hizalı yapmak için

```
# Hem satır hem de sütun oluşturmak için :  
sayi = int(input("Lütfen tablo ölçüsünü giriniz: "))  
for satir in range(1, sayi + 1):  
    for sutun in range(1, sayi + 1):  
        deger = satir*sutun  
        print("{0:4}".format(deger), end="")  
    print()
```

## Ekran Çıktısı

Lütfen tablo ölçüsünü giriniz: 10										
1	2	3	4	5	6	7	8	9	10	
2	4	6	8	10	12	14	16	18	20	
3	6	9	12	15	18	21	24	27	30	
4	8	12	16	20	24	28	32	36	40	
5	10	15	20	25	30	35	40	45	50	
6	12	18	24	30	36	42	48	54	60	
7	14	21	28	35	42	49	56	63	70	
8	16	24	32	40	48	56	64	72	80	
9	18	27	36	45	54	63	72	81	90	
10	20	30	40	50	60	70	80	90	100	

>>>

## Örnek

```
sayi = int(input("Lütfen tablo ölçüsünü giriniz:"))
print(" ",end=" ")
for sutun in range(1, sayi + 1):
    print("{0:4}".format(sutun), end=" ")
print()
print("+", end=" ")
for sutun in range(1, sayi + 1):
    print("----", end=" ")
print()
for satir in range(1, sayi + 1):
    print("{0:3}|".format(satir), end=" ")
    for sutun in range(1, sayi + 1):
        deger = satir*sutun # Çarpım sonucunun hesaplanması
        print("{0:4}".format(deger), end=" ")
    print()
```

## Ekran Çıktısı

```
Lütfen tablo ölçüsünü giriniz: 10
      1   2   3   4   5   6   7   8   9   10
+-----+
1 |  1   2   3   4   5   6   7   8   9   10
2 |  2   4   6   8   10  12  14  16  18  20
3 |  3   6   9   12  15  18  21  24  27  30
4 |  4   8   12  16  20  24  28  32  36  40
5 |  5   10  15  20  25  30  35  40  45  50
6 |  6   12  18  24  30  36  42  48  54  60
7 |  7   14  21  28  35  42  49  56  63  70
8 |  8   16  24  32  40  48  56  64  72  80
9 |  9   18  27  36  45  54  63  72  81  90
10 | 10  20  30  40  50  60  70  80  90  100
>>>
```

İç içe döngülerde önemli olan, döngü dışında kalacak kod satırlarının belirlenmesidir. Hangi satırların bir kere, hangi satırların döngü içerisinde olacağını belirlemek gerekir. İç içe döngüde satır (satır) dışındaki kontrol döngüsü olup sütun (sutun) döngüsü ise içteki döngüdür. İçteki döngü, dışındaki döngünün her tekrarında en baştan son bitiş değerine ulaşıcaya kadar kendini yineler.

## 5.4. İç İçe 3'lü Döngü

```
# ABC harflerinin farklı permütasyonu:  
for ilk in "ABC":  
    for ikinci in "ABC":  
        if ikinci != ilk:  
            for ucuncu in "ABC":  
                if ucuncu != ilk and ucuncu != ikinci:  
                    print(ilk + ikinci + ucuncu)
```

### Ekran Çıktısı

```
ABC  
ACB  
BAC  
BCA  
CAB  
CBA
```

## 5.5. While Döngüsü

```
sayac = 1 # Başlangıç değeri kontrol değişkenine atanır.  
while sayac <= 5: #İstenilen değere ulaşıp ulaşmadığını kontrol eder.  
    print(sayac)      # Sayaç değerini ekrana yazar.  
    sayac+= 1         # Sayaç değerini 1 arttırır.
```

While ifadesi, aşağıdaki ifadeleri 5 kere tekrar eder.

```
>>>print(sayac)  
>>>sayac += 1
```

Sayac değişkeninin değerini sürekli olarak ekrana yazar. Yazma işlemi sonrasında değişkenin değerini 1 artırır. Bu işlemden sonra sayac değerinin 5'ten küçük veya eşit olması durumuna göre yazma işlemine devam eder. Şart sağlanmadığında ilgili kod blokunun tekrarlanması duracaktır.

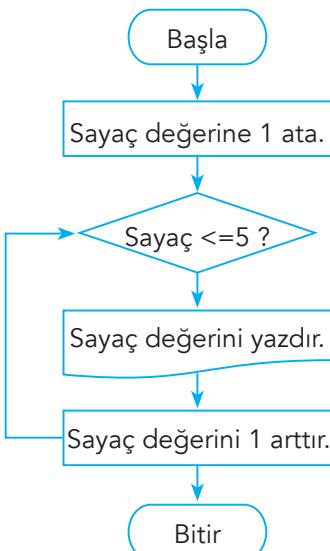
### 5.5.1. While Döngüsü İçin Söz Dizimi

While ifadesi, ilgili kod satırlarının çalıştırılıp çalıştırılmayacağını belirler. Şart doğru olduğu sürece kod blokunu tekrar çalıştırır. Şart yanlış olduğunda ise döngü sonlanır. While için söz diziminde ilk önce while ifadesi yazılır. Şart ifadesi sonrasında : işaretini ile yazılmalı ve bu şartla bağlı olarak çalışacak kod satırları alt alta yazılmalıdır.

**while koşul :**

**komutlar**

### 5.5.2. While Döngüsü İçin Akış Şeması



### While Döngüsü İçin Örnek Soru

Dışarıdan N/n karakteri girilinceye kadar döngünün kaç kere döndüğünü ekrana yazan while döngüsü

```
sayac = 0
giris= "Y"
while giris!="N" and giris!="n":
    print(sayac)
    giris= input("Devam etmek için " Y" - çıkmak için "N" giriniz: ")
    if giris == "Y" or giris == "y":
        sayac += 1
    elif giris != "N" and giris != "n":
        print(""" + giris + "" geçerli bir giriş kodu değil")
```

Ekrana çıktıları aşağıdaki gibi olur.



```
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
1
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
2
Devam etmek için " Y" - çıkmak için "N" giriniz: y
3
Devam etmek için " Y" - çıkmak için "N" giriniz: q
" q" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: r
"r" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: W
"W" geçerli bir giriş kodu değil
3
Devam etmek için " Y" - çıkmak için "N" giriniz: Y
4
Devam etmek için " Y" - çıkmak için "N" giriniz: y
5
Devam etmek için " Y" - çıkmak için "N" giriniz: n
```

### While Döngüsü İçin Örnek Soru 2

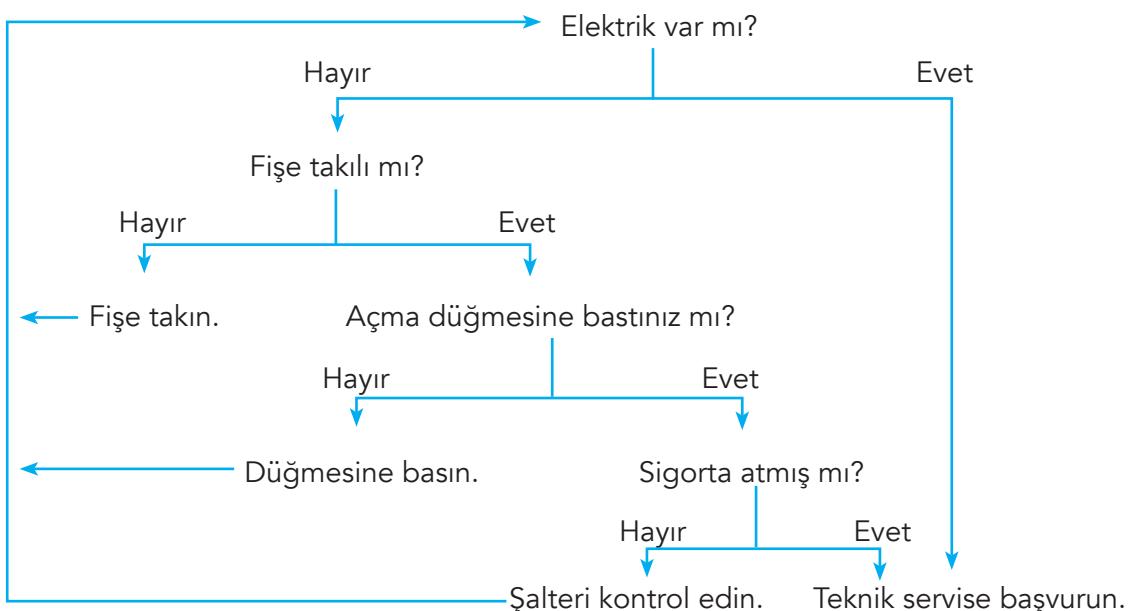
Dışarıdan negatif sayı girilinceye kadar sayıları toplayan while döngüsü

```
sayi = 0
toplam = 0
print("Bir sayı giriniz, negatif sayı döngüyü sonlandırır:")
while sayı >= 0:
    sayı= int(input())
    toplam += sayı
print("Toplam=", toplam)
```



### While Döngüsü İçin Örnek Soru 3

Çalışmayan bir bilgisayar için sorun çözme adımlarını gösteren while döngüsü



#### # Bilgisayar Arıza Çözüm Programı

```
print("Yardım Edin! Bilgisayarım Çalışmıyor")
cozum = False
while not cozum:
    print("Bilgisayardan herhangi bir ses geliyor mu (fans vb) ")
    secim = input("Veya herhangi bir ışık yanıyor mu? (y/n):")
    if secim == "n":
        secim = input("Fişe takılı mı (y/n):")
        if secim == "n":
            print("Fişe takın")
        else:
            secim = input("Açma düğmesine bastınız mı (y/n):")
            if secim == "n":
                print("Açma düğmesine basın.")
            else:
                secim = input("Sigorta atmış mı? (y/n):")
                if secim == "n":
                    secim = input("Şalter inmiş mi (y/n):")
                    if secim == "n":
                        print("Şalteri kontrol edin veya yenişi ile değiştirin. ")
                    else:
```



```
print("Teknik servise başvurun.")  
cozum = True  
else:  
    print("Sigortayı kontrol edin. ")  
else:  
    print("Teknik servise başvurun")  
cozum = True
```

### Çözümün Açıklaması

While döngüsünü, mantıksal değeri olan değişken kontrol ediyor. Çözüm değişkeni yanlış olduğu sürece döngü, çalışmaya devam ediyor. Bu soruda kullanılan çözüm değişkeni, bayrak (flag) olarak kullanılmıştır. Bayrak aşağı indiğinde değer yanlış; yukarı kalktığında ise değer doğru olarak değişir. Bu örnek çözümde ise bayrak yukarı doğru kaldırılmış ve döngü sonlandırılmıştır. Ayrıca, bu soruda kullanılan “not cozum” ifadesi de önemlidir. Döngünün çalışması sorunun çözümüne bağlı olduğu için kontrol şartı, mantıksal değil olarak belirlenmiştir. Bu durum değişkenin değerini değiştirmeden kontrol edilmesini sağlayacaktır.

Python, tam sayı değeri olan 0 ve ondalıklı sayı olan 0,0 değerlerini yanlış (false); diğer tüm değerleri ise (pozitif ve negatif olanlar da dahil) doğru (true) olarak kabul eder.

## 5.6. Belirli ve Belirsiz Döngüler

Döngünün tekrar sayısının bilindiği veya bilinmediği durumlar olabilir. Belirli döngülerde döngünün kaç defa döneceği, kaç kere çalışacağı kestirilebilir. Çünkü şart ifadesi bu konuda bilgi verir. Ancak bu durumun aksine kullanıcı girişine göre değişiklik gösteren ve farklı sayıda çalışan döngüler olabilir. Böyle durumlarda kullanılan döngülere de **belirsiz döngü** denir.

### Belirli döngü örneği 1

```
n = 1  
while n <= 10:  
    print(n)  
    n += 1
```

### Belirli döngü örneği 2

```
n = 1  
karar= int(input())  
while n <= karar:  
    print(n)  
    n += 1
```

### Belirsiz döngü örneği 3

```
karar = False  
while not karar:  
    giris = int(input())  
    if giris == 999:  
        karar = True  
    else:  
        print(giris)
```

## 5.7. Döngü'den Çıkma Komutları

While döngüsü, şart sağlandığı sürece ilgili kod satırlarını çalıştırır. Benzer şekilde for döngüsü de aralık içerisinde ilgili işlemleri gerçekleştirir. Ancak bazı değerler için döngü yapısından çıkmak, başka bir deyişle döngü işleyişinde değişiklik yapmak mümkündür. Bunun için **break** ve **continue** komutları kullanılabilir.

```
giris = 0
toplam = 0
print("Lütfen bir sayı giriniz, negatif sayılar döngüyü sonlandırır:")
while True:
    giris = int(input())
    if giris < 0:
        break # Döngüden çıkış
    toplam += giris
print("Toplam =", toplam)
```

### Örnek

```
# Metin içerisindeki sesli harfleri bulma
kelime = input("Lütfen bir metin giriniz (Çıkış için X / x): ")
sesliHarfSayisi = 0

for c in kelime:
    if c == "A" or c == "a" or c == "E" or c == "e" \
    or c == "I" or c == "ı" or c == "O" or c == "o" \
    or c == "U" or c == "u" or c == "Ö" or c == "ö" \
    or c == "Ü" or c == "ü" or c == "İ" or c == "i":
        print(c, ", ", end=" ", sep="")
        sesliHarfSayisi += 1
    elif c == "X" or c == "x":
        break

print(" (", sesliHarfSayisi, " adet sesli harf)", sep="")
```



## Ekran Çıktısı

```
Lütfen bir metin giriniz (Çıkış için X / x): AaEeİiİiOoÖöUuÜü  
A, a, E, e, İ, i, O, o, Ö, ö, U, u, Ü, ü, (16 adet sesli harf)
```

### 5.7.2. Continue ifadesi

Break ifadesi kullanıldığında ilgili kod satırları çalıştırılmadan atlanırken continue ifadesi kullanıldığında döngü başı yapılarak bir sonraki yeni değer için işlem yapılır.

#### Örnek

```
# 999 girilene kadar girilen pozitif sayıların toplamını alan program  
  
# < girilen negatif sayılar işleme alınmayacağıdır. >  
toplam = 0  
durum = False  
  
while not durum:  
    deger = int(input("Lütfen pozitif tam sayı giriniz (Çıkış için 999):"))  
    if deger < 0:  
        print("Negatif değer girildi, ", deger, "değeri işleme alınmadı")  
        continue  
    if deger != 999:  
        print("Eklenen değer", deger)  
        toplam += deger  
    else:  
        durum = (deger == 999)  
print("Toplam =", toplam)
```

## Ekran Çıktısı

```
Lütfen pozitif tam sayı giriniz (Çıkış için 999):1  
Eklenen değer 1  
Lütfen pozitif tam sayı giriniz (Çıkış için 999):2  
Eklenen değer 2  
Lütfen pozitif tam sayı giriniz (Çıkış için 999):3  
Eklenen değer 3  
Lütfen pozitif tam sayı giriniz (Çıkış için 999):6
```



Örnekte negatif değerde döngü başı yapılır.

### 5.7.3. While/else ve for/else

Python döngüler için opsiyonel else bloku kullanımını destekler. Break ifadesine rağmen döngünün terkedilmediği durumlarda döngüye ait else ifadesi kullanılabilir.

#### While / else

```
# Girilen 5 sayının ortalamasını alan program
# Negatif sayı girildiğinde program sonlandırılır
sayac = toplam = 0
print("Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz")
while sayac < 5:
    sayı = float(input("Sayı giriniz: "))
    if sayı < 0:
        print("Negatif sayılar kabul edilmemektedir. Çıkılıyor")
        break
    sayac += 1
    toplam += sayı
else:
    print("Ortalama =", toplam/sayac)
```

#### Ekran Çıktısı

```
Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz
Sayı giriniz: 6
Sayı giriniz: 7
Sayı giriniz: 8
Sayı giriniz: 6
Sayı giriniz: 5
Ortalama = 6.4
>>>
===== RESTART:
Lütfen Ortalama hesaplamak için 5 pozitif sayı giriniz
Sayı giriniz: -1
Negatif sayılar kabul edilmemektedir. Çıkılıyor
```

Örnekte beş sayı girilene kadar while döngüsü dönecek şart yanlış olduğunda else satırına geçilerek ortalama değeri ekrana yazdırılacaktır.

## 5.8. Döngü Örnekleri

### Faktöriyel Hesaplama

```
# Program girilen sayının faktöriyelini hesaplar
faktoriyel=1
sayac=1
sayi=int(input("Lütfen bir sayı giriniz.."))
while sayac<=sayi:
    faktoriyel*=sayac
    sayac+=1
print(sayı," sayısının foktöriyeli:",faktoriyel)
```

### Ekrان Çıktısı

```
Lütfen bir sayı giriniz..5
5 sayısının foktöriyeli: 120
```

Faktöriyel hesaplamak için klavyeden girilen sayıdan 1'e kadar döngü kurulmuş ve her döngü değeri çarpılma işlemine alınarak (faktöriyel\*=sayac) amaca ulaşılmıştır.

### Ağaç Çizimi

```
# Girilen değere göre "*" karakterinden ağaç çizen program
yukseklik = int(input("Çizilecek ağaçın yüksekliğini giriniz: "))
satir = 0
while satir < yukseklik:
    sayac = 0
    while sayac < yukseklik - satir:
        print(end=" ")
        sayac += 1
    sayac = 0
    while sayac < 2*satir + 1:
        print(end="*")
        sayac += 1
    print()
    satir += 1
```



## Ekran Çıktısı

**Çizilecek ağacın yüksekliğini giriniz:** 12

The image shows a decorative pattern of stars arranged in a grid-like structure. The stars are black and vary in size. They are arranged in horizontal rows, with each row containing more stars than the one above it, creating a pyramid-like shape. The stars are set against a white background.

#### **Asal Sayıları Listeleme**

```
sonDeger = int(input("Kaça kadar asal sayıları görmek istersiniz? "))
sayi = 2
while sayi <= sonDeger:
    kontrol = True
    gecici = 2
    while gecici < sayi:
        if sayi % gecici == 0:
            kontrol = False
            break
        gecici += 1
    if kontrol:
        print(sayi, end= " ")
    sayi += 1
print()
```

Ekran Çıktısı

Kaç kadar asal sayıları görmek istersiniz? 77

2    3    5    7    11    13    17    19    23    29    31    37    41    43    47    53    59    61    67    71    73

## İstenilen Sayıdaki Fibonacci Sayılarını Listeleyen Program

```
# İstenilen sayıdaki Fibonacci Sayılarını Listeleyen Program  
deger1=0  
deger2=1  
sayac=2  
sonDeger=int(input("Listelemek istediğiniz Fibonacci Sayıları adetini giriniz...:"))  
print(str(deger1) + " " +str(deger2),end=" ")  
while sayac<=sonDeger:  
    deger3=deger1+deger2  
    print(deger3,end=" ")  
    deger1=deger2  
    deger2=deger3  
    sayac+=1
```

### Ekran Çıktısı

```
Listelemek istediğiniz Fibonacci Sayıları adedini giriniz...:15  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

### Fibonacci Sayıları:

Her sayının kendisinden önce gelen iki sayının toplamı şeklinde yazılıp devam ettiği sayı dizisine Fibonacci Sayı Dizisi denir.

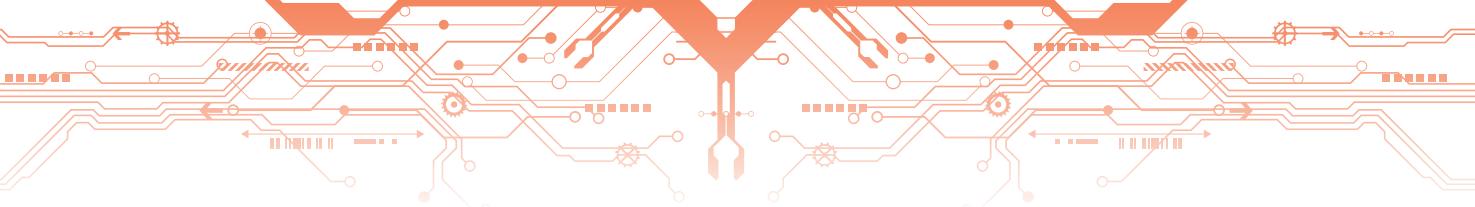
1,1,2,3,5,8,13,21,34,55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...

## 6. FONKSİYONLAR 1



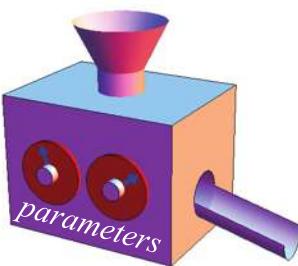
Bu bölümde;

- ✓ Fonksiyon kullanmanın gerekliliklerini öğrenebilecek,
- ✓ Python programlama dilinde kullanılan standart fonksiyonları kullanabilecek,
- ✓ Fonksiyon türleri konusunda bilgi sahibi olabileceksiniz.



## 6.1. Neden Fonksiyonlar?

Büyük ve kapsamlı bir program yazdığınımız ve program kapsamında pek çok kez aynı işlemi yapmamız gerektiğini düşünelim. Örneğin karekökü hesaplamak. Matematik bağlamında iki geometrik nokta  $(x_1, y_1)$  ve  $(x_2, y_2)$  arasındaki uzaklığını hesaplamamız gerekebilir. İlkinci dereceden bir denklemin  $(ax^2+bx+c = 0)$  çözüm kümesini bulmamız istenebilir. Elektrik mühendisliği ya da fizik bağlamında bir dizi değerin, sayıların karelerinin ortalamasının karekökünü bulmamız beklenebilir.



$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \sqrt{\frac{a_1^2 + a_2^2 + a_3^2 + \dots + a_n^2}{n}}$$

Bu durumda her formül için karekök hesaplama yapan program satırlarını tekrar tekrar gereken yerlere kopyalamamız mı gereklidir? Peki ya karekök bulma işlemi pek çok farklı program tarafından kullanılan bir hesaplama ise o zaman bütün programların içerisinde yine bu satırları eklememiz mi gereklidir? Acaba bu şekilde tekrarlayan işlemler için bu kodu paketleyip tekrar kullanmamızı sağlayan bir yöntem var mıdır?

## 6.2. Fonksiyon Nedir?

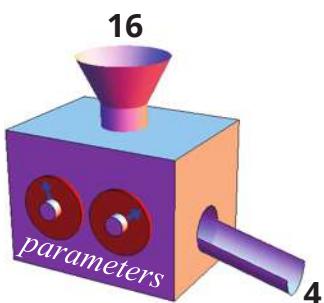
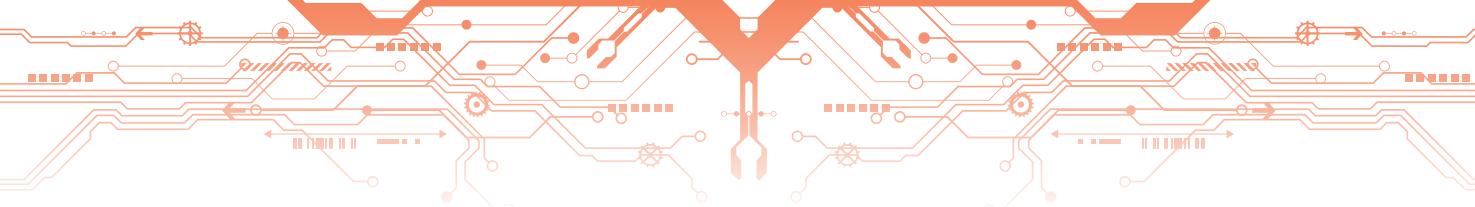
Bu kodu paketleyerek tekrar tekrar kullanmamızı sağlayan yaklaşımlardan biri “fonksiyonlar”dır. Bir fonksiyon, tekrar kullanılabilen kod parçasıdır. Kendimiz fonksiyon yazabileceğimiz gibi önceden yazılmış ve kullanıma hazır fonksiyonları da kullanabiliriz. Diğer programlama dillerinde olduğu gibi Python kapsamında da standart fonksiyonların bulunduğu bir kütüphane vardır. Programcılar, “modül” olarak adlandırılan bu fonksiyonları kendi kodları içinden çağrıarak kullanabilirler.

## 6.3. Fonksiyonlara Giriş

Aslında biz ilk bölümden itibaren fonksiyonları kullanmaya başlamıştık: `print`, `input`, `int`, `float`, `str`, ve `type`. Sıkça kullanılan işlemlerin çoğu için Python kütüphanesinde pek çok fonksiyon bulunmaktadır. Fonksiyon kavramını anlatmak için karekök bulma ve kare alma işlemlerini kullanalım. Fonksiyonlar, belirli bir işlemi gerçekleştiren kod blokudur. İlgili işlemi gerçekleştirmek için fonksiyon çağrıılır. Python kütüphanesinde `sqrt` isimli bir fonksiyon karekök alma işlemini yapmaktadır. Bu fonksiyon tam sayı ya da reel sayı kabul etmektedir. Örneğin parametre olarak 16 gönderildiğinde 4 değeri geri dönmektedir.

$$\sqrt{16} = 4$$

Fonksiyonları kapalı bir kutu olarak düşünebiliriz. İçerisindeki kodlama detaylarını bilmemize gerek olmadan kolayca çağrııp işlem yaptırmak için kullanabiliriz. Dolayısıyla işlemi nasıl yaptığından daha çok ne yaptığı bilmemiz yeterli olur.



### 6.3.1. sqrt() Fonksiyonu

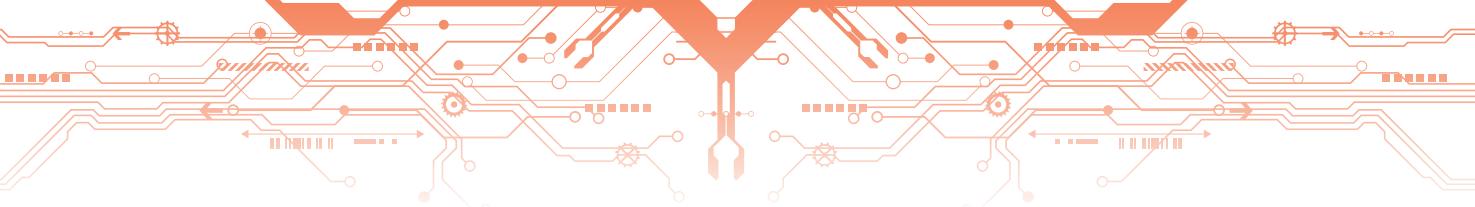
Burada `sqrt(sayı)` komutu ilgili “fonksiyonu çağırırmak” için kullanılmaktadır. Önceden kullandığımız fonksiyonlar gibi sık kullanılan fonksiyonların küçük koleksiyonu kapsamı dışındadır. Bu fonksiyon standart kütüphane içerisinde ayrı bir modül olarak düşünülebilir. Bu nedenle “import” anahtar kelimesi kullanılarak ve “math” yani matematik kütüphanesinden çağrılarak kullanılır. Böylece `sqrt()` fonksiyonu programa tanıtılmış olur. “sayı” ise fonksiyona gönderilecek parametredir. Parametreler fonksiyona işlem yapması için ihtiyaç duyduğu değerleri göndermek ve bilgi alışverişini sağlamak için kullanılır.

```
from math import sqrt  
# Kullanıcıdan değer alınıyor  
sayı = float(input("Sayı Giriniz: "))  
# Karakök hesaplanarak kök değişkenine aktarılıyor  
kok = sqrt(sayı)  
# Sonuçlar yazdırılıyor  
print(sayı," sayısının karekökü" "=", kok)
```

`sqrt()` fonksiyonu sadece sayısal değerleri kabul eder. O yüzden farklı bir değer göndermek hata ile karşılaşmasına neden olur. Örneğin `sqrt("16")` söz dizimi hatalı bir yapıdır.

#### 6.3.1.1. sqrt() Fonksiyonunun Farklı Kullanımıları

```
# Bu program sqrt() fonksiyonunun farklı kullanımlarını gösterir.  
from math import sqrt  
x = 16  
# İstenilen sabit değerin karekökünün alınması  
print(sqrt(16.0))  
# Değişkenin karekökünün alınmasını sağlar  
print(sqrt(x))  
# sqrt() fonksiyonunun içerisinde işlem kullanımı  
print(sqrt(2 * x - 5))
```



```
# İşlem sonucu geri dönen değerin değişkene aktarılması
y = sqrt(x)
print(y)

# İçerisinde işlem kullanılan sqrt() fonksiyonunun dönen değerinin
# işleme tabi tutulması
y = 2 * sqrt(x + 16) - 4
print(y)

# İç içe sqrt() fonksiyonunun kullanılması
y = sqrt(sqrt(256.0))
print(y)
print(sqrt(int("45")))
```

Fonksiyonlar kendilerini çağrıırken gönderilen parametreleri genellikle değiştirmez. Sonuçları fonksiyon adında ya da birden fazla parametre gönderildiği durumda sonucu içeren parametre ile iletir. Değer çağrıran kişi tarafından bir değişkene atanmadığı sürece değişkenin değeri değişmez.

```
>>> from math import sqrt
>>> x = 2
>>> sqrt(x)
1.4142135623730951
>>> x
2
>>> x = sqrt(x)
>>> x
1.4142135623730951
```

### 6.3.2. Fonksiyonların Bölümleri

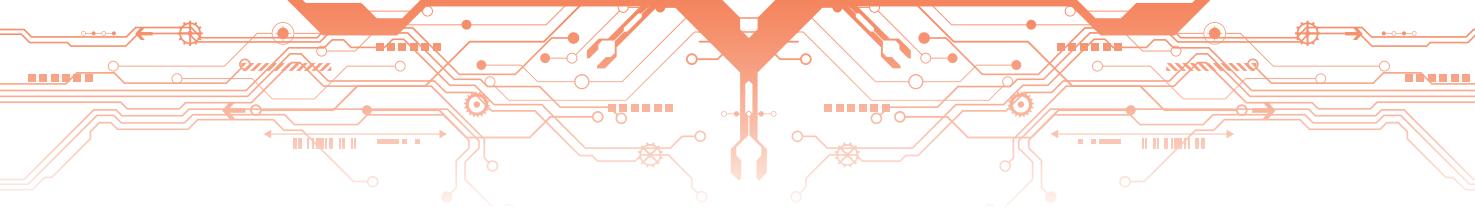
Çağıran kişi açısından fonksiyonun 3 önemli bölümü vardır:

**İsmi:** Her fonksiyonun, nasıl bir işlem yapılacağını ifade eden bir adı vardır. Değişkenleri isimlendirdikten dikkat ettiğimiz kurallar, fonksiyon isimleri için de geçerlidir.

**Parametreler:** Bir fonksiyon belli sayıda parametre ile çağrırlır ve her birinin doğru türde olması gereklidir. Beklenenden daha az ya da çok sayıda parametre göndermek hataya neden olur.

**Sonuç Türü:** Fonksiyon kendini çağrıran programa bir değer döndürür. Bu değer beklenen veri türü ile aynı olmalıdır.

```
>>> sqrt(10)
3.1622776601683795
>>> sqrt()
```



```
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
    sqrt()
TypeError: sqrt() takes exactly one argument (0 given)
>>> sqrt(10, 20)
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
    sqrt(10, 20)
TypeError: sqrt() takes exactly one argument (2 given)
>>> sqrt(16)
4.0
>>> sqrt("16")
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
    sqrt("16")
TypeError: a float is required
>>> type(sqrt(16.0))
<class "float">
```

### 6.3.3. Parametresiz Fonksiyonlar

Bazı fonksiyonlar parametre kabul etmez. Örneğin rastgele bir sayı oluşturmamızı sağlayan random fonksiyonu bu duruma bir örnektir. Bu fonksiyonu çağırarak rastgele bir sayı değeri elde ederiz. Bu fonksiyonu parametre ile çağırmak hataya neden olacaktır.

```
>>> from random import random
>>> random()
0.9595266948278349
>>> random(20)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: random() takes no arguments (1 given)
```

### 6.3.4. Değer Döndürmeyen Fonksiyonlar

Bazı fonksiyonlarda parametre beklemelerine rağmen sonuç, değeri döndürmeyebilir. Örneğin print fonksiyonunun işlevi, hesaplama yapmak değil, ekranda görüntülemek olduğu için bu fonksiyonun sonucu diğer fonksiyonlara göre farklıdır.

```
>>> print(print(4))
```

```
4
```

```
None
```

Bu örnekte içerisindeki parametre ekrana 4 yazdırırken dıştaki print, içerisindeki fonksiyon değerini döndürür. Ayrıca bir değişkene “None” değeri atayabiliriz. Bu durumda hiçbir değer atanmamış demektir.

## 6.4. Fonksiyon ve Modüller

Bir Python modülü Python kodları içeren bir dosyadır. Dosyanın adı modülün adına işaret eder. Örneğin math.py isimli bir dosya standart matematik modülünde yer alan fonksiyonları içerir. Python standart kütüphanesinde 230 modül kapsamında yer alan binlerce fonksiyon vardır. Bu modüllerin geniş bir uygulama alanı bulunmaktadır. Örneğin built-ins (yerleşik işlevler) isimli modül (\_\_builtins\_\_), daha önce kullandığımız fonksiyonları kapsamaktadır: print, input vb. Bu yerleşik işlevler standart kütüphanenin çok küçük bir kısmını oluşturmaktadır. Geri kalan tüm diğer fonksiyonlara ulaşmak için programcılar program ya da yorumlayıcı içinden öncelikle “import” komutunu kullanarak ilgili kütüphaneye erişim sağlamaları gereklidir.

**from**

modül adı

**import**

fonksiyon adı

Bu komut ile ilgili kod parçası bilgisayarın sabit diskinde bir yerde saklanır. Böylece program, ihtiyaç duyduğunda bu kodları çalıştırmak için nereden çalıştıracağını bilir.

Python, bir modülden fonksiyon çağrılmak için farklı yollar sunar. Bunlardan çok yaygın kullanılan ikisini inceleyelim.

```
from math import sqrt
```

Eğer birden fazla fonksiyon çağrılmamız gerekirse örneğin yaygın logaritma ve trigonometri kapsamındaki cos fonksiyonuna da ihtiyacımız varsa söz dizimi şu şekilde olacaktır:

```
from math import sqrt, log10, cos
```

Böylece her 3 fonksiyon da program açısından erişilebilir ve kullanılabilir duruma gelir. “math” modülü pek çok farklı fonksiyonu da içermektedir. Eğer çok sayıda modül kullanmamız gerekiyorsa birkaç modülün spesifik olarak ismini belirtmek yerine

```
import math
```

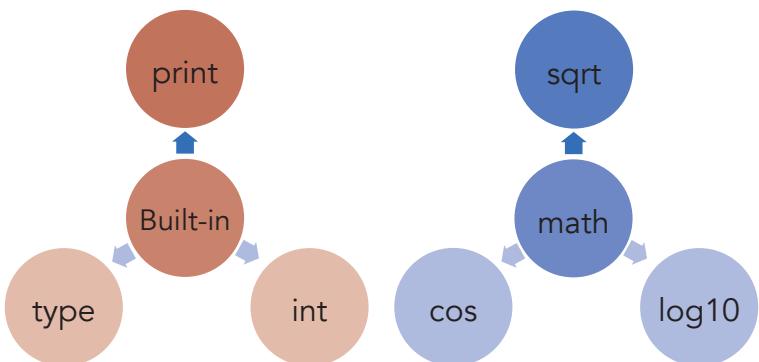
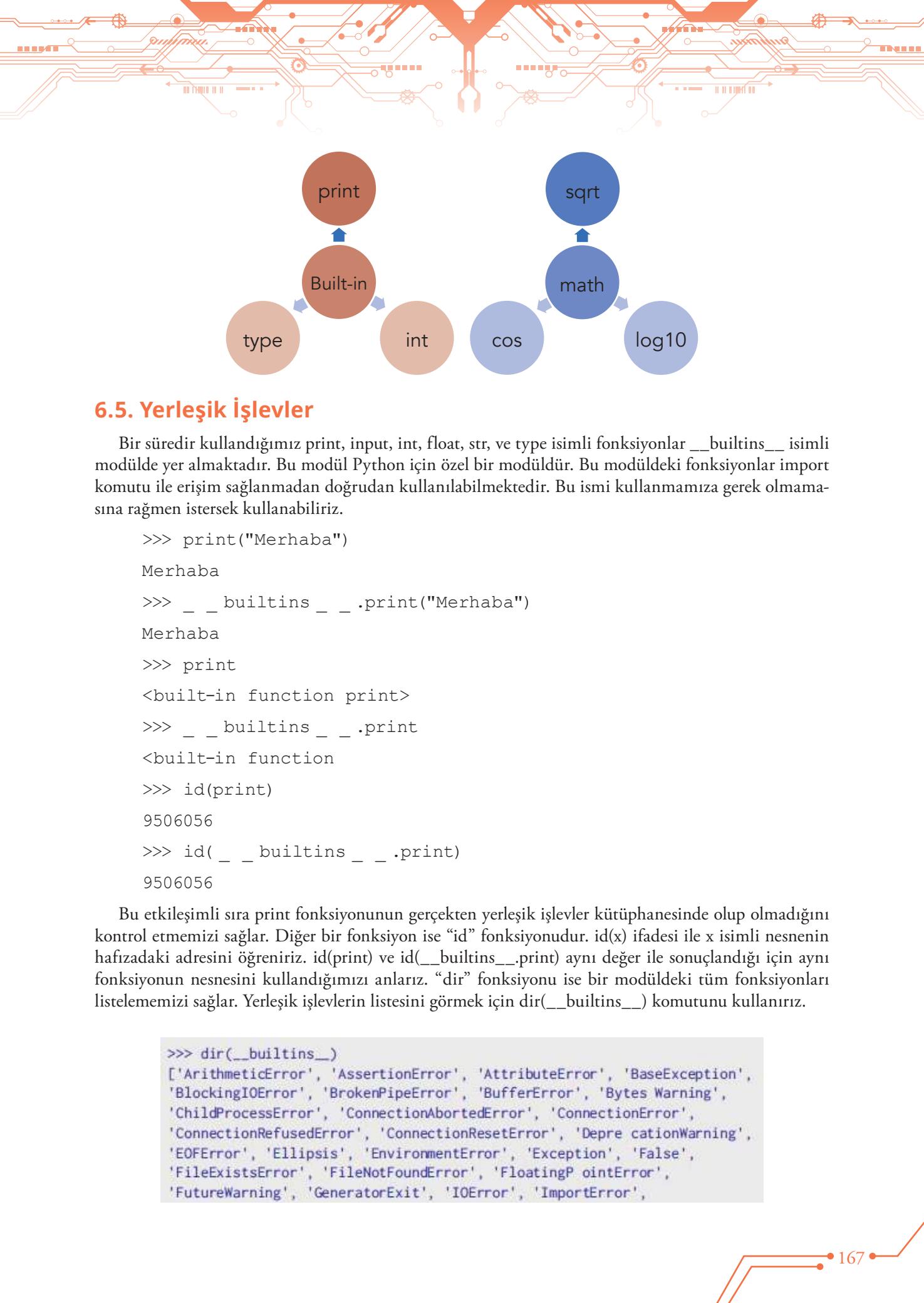
ifadesini kullanarak “math” modülünün tamamına erişim sağlarız.

**import**

modül adı

Böylece “math” modülündeki tüm fonksiyonlar erişilebilir hâle geldi. Ancak ilgili fonksiyonu çağrıırken modülün adını da belirtmemiz gereklidir. Modül adının “.” ile fonksiyon adlarına birleştirilerek kullanıldığına dikkat ediniz. Bu yapıya birleşik (modül adı.fonksiyon adı) yapı diyoruz. Programcılar çoğu fonksiyon çağrıma işleminde bu yapıyı tercih eder çünkü bu yapı programı daha basit ve anlaşılır kılmaktadır.

```
y = math.sqrt(x)  
print(math.log10(100))
```



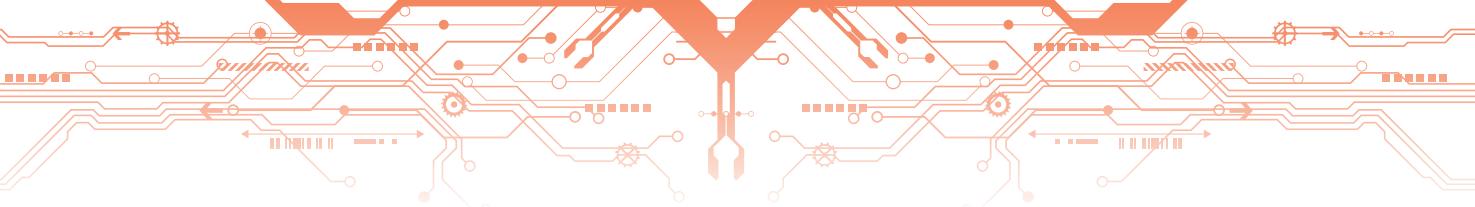
## 6.5. Yerleşik İşlevler

Bir süredir kullandığımız `print`, `input`, `int`, `float`, `str`, ve `type` isimli fonksiyonlar `__builtins__` isimli modülde yer almaktadır. Bu modül Python için özel bir modüldür. Bu modüldeki fonksiyonlar `import` komutu ile erişim sağlanmadan doğrudan kullanılabilmektedir. Bu ismi kullanmamıza gerek olmasına rağmen istersek kullanabiliriz.

```
>>> print("Merhaba")
Merhaba
>>> __builtins__.__print("Merhaba")
Merhaba
>>> print
<built-in function print>
>>> __builtins__.print
<built-in function
>>> id(print)
9506056
>>> id(__builtins__.print)
9506056
```

Bu etkileşimli sıra `print` fonksiyonunun gerçekten yerleşik işlevler kütüphanesinde olup olmadığını kontrol etmemizi sağlar. Diğer bir fonksiyon ise “`id`” fonksiyonudur. `id(x)` ifadesi ile `x` isimli nesnenin hafızadaki adresini öğreniriz. `id(print)` ve `id(__builtins__.print)` aynı değer ile sonuçlandığı için aynı fonksiyonun nesnesini kullandığımızı anlarız. “`dir`” fonksiyonu ise bir modüldeki tüm fonksiyonları listelememizi sağlar. Yerleşik işlevlerin listesini görmek için `dir(__builtins__)` komutunu kullanırız.

```
>>> dir(__builtins__)
['ArithmetricError', 'AssertionError', 'AttributeError', 'BaseException',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
 'ConnectionRefusedError', 'ConnectionResetError', 'Depre cationWarning',
 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
 'FileExistsError', 'FileNotFoundException', 'FloatingP ointError',
 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
```



`__builtins__` modülü bir de `help` (yardım) fonksiyonu içermektedir. Etkileşimli yorumlayıcıda spefik fonksiyonlar hakkında anlaşılabilir bilginin sunulması sağlanır. Böylece her fonksiyona ilişkin ayrıntılı bilgi edinilebilir.

```
>>> help(input)
Help on built-in function input in module builtins:

input(...)
    input([prompt]) -> string

    Read a string from standard input. The trailing newline is stripped.
    If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError.
    On Unix, GNU readline is used if enabled. The prompt string, if given,
    is printed without a trailing newline before reading.

>>> help(sqrt)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> help(math.sqrt)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
>>> import math
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

## 6.6. Standart Matematik Fonksiyonları

### *math Modülü*

Fonksiyon Adı	Açıklama	Örnek Kullanım ve Çıktı
<code>sqrt()</code>	İstenilen değerin karekök'ünün bulunmasını sağlar.	<code>math.sqrt(16)</code> → 4
<code>exp()</code>	e (Euler sabiti) sayısının istenilen kuvvetinin alınmasını sağlar.	<code>math.exp(2)</code> → 7.389056
<code>log()</code>	<code>log(x,y)</code> fonksiyonumuz iki parametre alır. İlk parametremiz olan x logaritması, alınacak sayı; ikinci parametre olan y taban sayısını temsil etmektedir.	<code>math.log(2,2)</code> → 1.0
<code>log10()</code>	<code>log(x,y)</code> fonksiyonundan tek farkı taban olarak 10 sayısının sabit olmasıdır.	<code>math.log10(10)</code> → 1.0
<code>cos()</code>	<code>cos(x)</code> , x derecesinin kosinüs değerini verir.	<code>math.cos(45)</code> → 0.5253
<code>pow()</code>	<code>pow(x,y)</code> fonksiyonu x sayısının y. kuvvetinin alınmasını sağlar.	<code>math.pow(2,2)</code> → 4
<code>degress()</code>	<code>degress(x)</code> fonksiyonu x açısını radyandan dereceye çevirmeye yarar.	<code>math.degress(45)</code> → 2578.310078088
<code>radians()</code>	<code>radians(x)</code> fonksiyonu x açısını dereceden radyana çevirmeye yarar.	<code>math.radians(45)</code> → 0.7853981633
<code>fabs()</code>	<code>fabs(x)</code> fonksiyonu x değerinin mutlak değerinin alınması işlemini gerçekleştirir.	<code>math.fabs(-5)</code> → 5.0

```
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
['__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float',
'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help',
'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit',
'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
'vars', 'zip']
>>>
```

Standart "math" modülü fonksiyonel bir hesap makinesi ile yapılabilen işlemlerin çoğunu içerir. Ayrıca pi (p) ve e (e) değerlerini de tanımlar. "math" modülünün içeriği tüm fonksiyonları dir (math) komutu ile görebiliriz.

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>>
```

Çağırın tarafından gönderilen parametre gerçek parametre olarak bilinir. Fonksiyon tarafından tanımlanan parametre ise resmi parametredir. Bir fonksiyon çağrılığında ilk gerçek parametre ilk resmi parametre değerine; ikinci gerçek parametre ikinci resmi parametre değerine atanır. Bu nedenle parametreleri doğru sırada göndermek çok önemlidir. Örneğin,

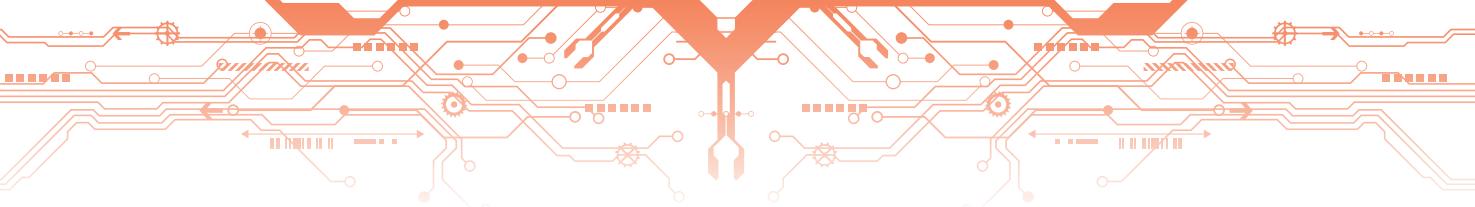
```
math.pow(8,2)
```

8 değerinin karesini alıp 64 sonucunu döndürürken

```
math.pow(2,8)
```

2 üzeri 8 değerini hesaplar ve 256 değerini döndürür.

Karekök fonksiyonunu asal sayıları belirlemek için de kullanabiliriz. Bütün olasılıkları (n ve n-1 için) denemek yerine n değerinin karekök sonucuna kadar olan değerleri denememiz yeterli olacaktır.



```
from math import sqrt
sonDeger = int(input("Hangi sayıya kadar asal sayılar listelensin?"))
deger = 2 # En küçük asal sayı
while deger <= sonDeger:
    # İstenilen değere kadar dönmesi için döngü kuruluyor
    kontrol = True # Başlangıç aşamasında kontrol değişkeni True olarak belirlenir
    # 2 ile -1 arasındaki tüm değerlerin kontrolünün yapılması
    geciciDeger= 2
    kok = sqrt(deger) # Döngüde sırası gelen değerin karakökü hesaplanıyor
    while geciciDeger <= kok:
        if deger % geciciDeger == 0:
            kontrol = False # Asal sayı özelliği yitiriliyor ve kontrol False oluyor
            break # Kontrol döngüsünden çıkılıyor.
        geciciDeger += 1 # Bir sonraki kontrol sayısına geçiş
    if kontrol:
        print(deger, end= " ") # Şarta uyan değer Asal olarak kabul edildip yazdırılıyor.
    deger += 1 # Asal sayı kontrolü için sonraki sayı
print() # Kursor bir sonraki satıra alınıyor
```

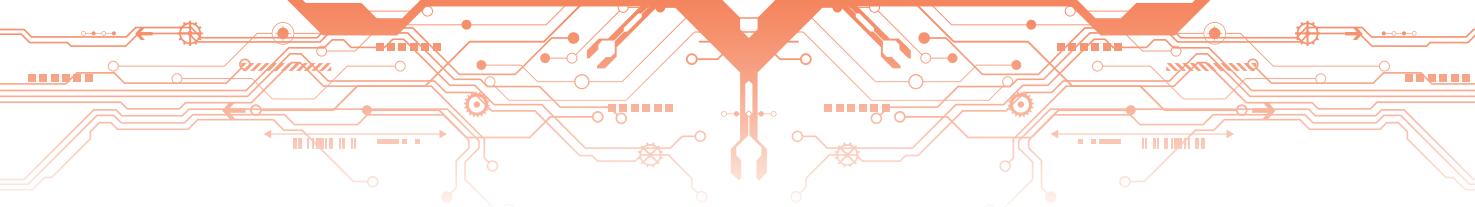
### Ekran Çıktısı

```
Hangi sayıya kadar asal sayılar listelensin? 51
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

## 6.7. time Fonksiyonları

Zamanla ilgili bilgi ve işlemlerin yer aldığı modül, “time” modülüdür. Bunlardan ikisi clock ve sleep fonksiyonlarıdır. Time.clock fonksiyonu ile programın belli bölümlerinin çalışma süresini ölçebiliriz. Programın ilk çağrıldığı andan itibaren geçen süreyi saniye olarak verir.

```
from time import clock
print("Adınızı Giriniz: ", end="")
baslangicZamani = clock()
```



```
ad = input()
zaman = clock() - baslangicZamani
print(ad, "bilgilerinizi", zaman, "zamanda girdiniz")
```

### Ekran Çıktısı

```
Adınızı Giriniz: Oğuz
Oğuz bilgilerinizi 0.0041149999999998 zamanda girdiniz
```

Bir Python programının 1.000.000'a kadar bütün sayıların toplamını hesaplamasının ne kadar sürdüğünü öğrenmek için yine aynı fonksiyon kullanılır.

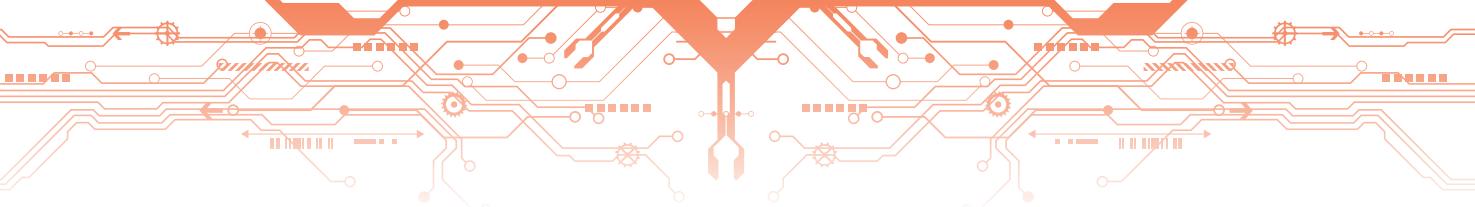
```
from time import clock
toplam = 0 # Toplam değişkeni tanımlanıp ilk değer olarak 0 veriliyor
basla = clock() # İşlem süresinin hesaplanması için süre başlatılıyor
for n in range(1, 1000001): # Toplamı alınacak sayılar için 1.000.000'e kadar döngü kuruluyor
    toplam += n
gecenZaman = clock() - basla # Geçen zaman hesaplanıyor
print("Toplam:", toplam, "Geçen Süre:", gecenZaman) # Sonuçlar yazdırılıyor
```

### Ekran Çıktısı

```
Toplam: 500000500000 Geçen Süre: 0.16708000000000003
```

Bir Python programının 10.000'e kadar bütün asal sayıların toplamını hesaplamasının ne kadar sürdüğünü öğrenmek için yine aynı fonksiyon kullanılır.

```
# 10.000'e kadar olan asal sayıların adetini ve geçen zamanı bulan program
from time import clock
sonDeger = 10000
sayac = 0
zaman = clock() # Süre başlatılıyor
# En küçük asal sayı olan 2 den istenilen değere kadar döngü kuruluyor
for deger in range(2, sonDeger + 1):
    # Sırayla sayılar ele alınıyor
```



```
kontrol = True # Değerlerin kontrol edilmesi için ilk değer True veriliyor

# Asal olma özelliğinin kontrolü için bölenlerinin döngüsü kuruluyor

for bolenSayi in range(2, deger):

    if deger % bolenSayi == 0:

        kontrol = False # Tam bölme işlemi oluştuysa kontrol False yapılıyor

        break # ve döngü sonlandırılıyor

    if kontrol:

        sayac += 1 # Asal olma özelliği sağlanmışsa sayac arttırılıyor

print() # Yeni satır başı

gecenZaman = clock() - zaman # İşlem tamamlandıktan sonra süre sonlandırılıyor

print("Adet:", sayac, " Geçen Zaman:", gecenZaman, " saniye")
```

### Ekran Çıktısı

```
Adet: 1229 Geçen Zaman: 0.8711089999999999 saniye
```

Time.sleep() fonksiyonu ise programın çalışması sırasında belirtilen süre kadar durmasını sağlar. Örneğin geriye sayımda her sayıdan sonra 1 saniye beklemek için aşağıda görülen kod kullanılır.

```
from time import sleep

for sayac in range(10, -1, -1): # Range 10, 9, 8, ..., 0
    print(sayac) # Sayac yazdırılıyor
    sleep(1) # 1 saniye bekleme işlemi yapılıyor
```

## 6.8. Rastgele Sayılar

Rastgele sayılar; birçok programlama dilinde, oyun ve simülasyonlarda kullanılır. Bütün rastgele sayılar üreten algoritmalar, aslında gerçek rastgele sayılar üretmez. Sözde rastgele sayılar üreten bu algoritmalar uzun süre kullanımından sonra aynı seriyi üretmeye başlar. Gerçek rastgele değerler, farklı sıralamada gelir ve bu sıralamayı tekrarlamaz. Python standart kütüphanesinde, Mersenne Twister algoritmasına dayalı olarak çalışan sözde rastgele değer üretmek mümkündür.

Mersenne Twister algoritmasına ilişkin daha fazla bilgi almak için <https://docs.python.org/2/library/random.html> adresini ziyaret ediniz.

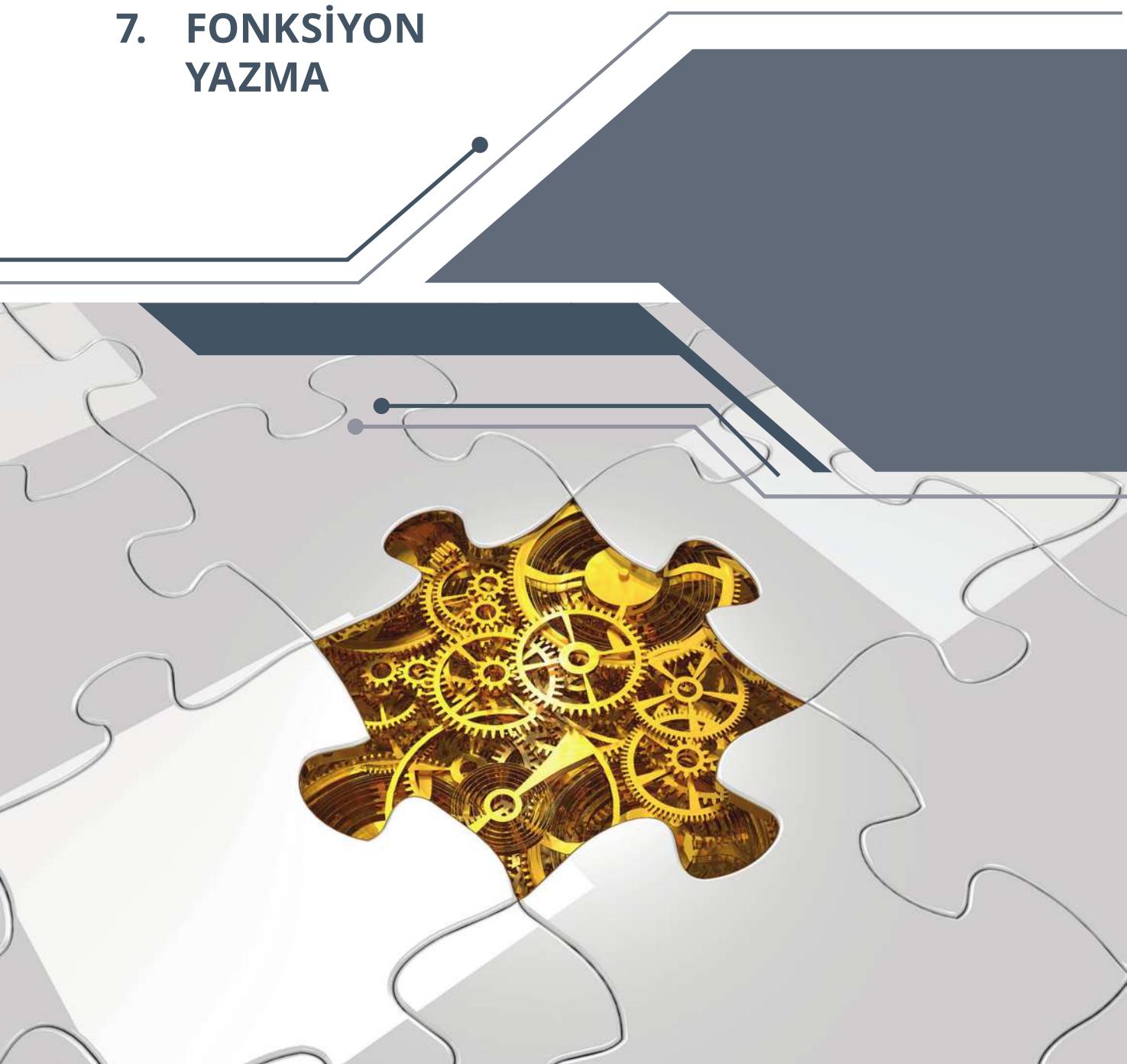
Aşağıdaki tabloda rastgele değer üreten fonksiyonlar ve özellikleri açıklanmıştır.

Rastgele sayı üretme fonksiyonları	
random	0<= x <1 arasında sözde rastgele ondalık bir değer döndürür.
randrange	Belirli bir aralıktı sözde rastgele tam sayı bir değer döndürür.
seed	Rastgele değer dizisi için bir değer alır.
choice	Değerler arasından rastgele bir değer seçer.

random.seed fonksiyonu, üretilecek sözde rastgele değerler için bir başlangıç değeri belirler. random.random veya random.randrange fonksiyonları her çağrıldığında bir sonraki sözde değerler için yeni bir değer döndürür. Aşağıdaki örnek, 1 ile 100 arasında rastgele değer döndürür.

```
from random import randrange, seed
for i in range(0, 100): # 100 adet rastgele sayı için döngü kuruluyor
    print(randrange(1, 1001), end=" ") # 1..1001 aralığında üretilen
    rastgele sayı yazdırılıyor
print()
```

## 7. FONKSİYON YAZMA



Bu bölümde;

- ✓ Python programlama dilinde programları alt programlar hâlinde yazmayı öğrenebilecek,
- ✓ Fonksiyon tanımla ve çağrıma süreçlerini kavrayacak,
- ✓ Fonksiyonlara değer gönderme ve fonksiyonları döndürme süreçlerini uygulayabileceksiniz.



## 7.1. Fonksiyon Kavramı

Program yazarken kod satırları uzayabilir ve yazılan program karmaşık bir hâl alabilir. Bu durumu ortadan kaldırmak için problemi alt problemler hâlinde ele almak ve fonksiyon yazmak gereklidir. Böylece bir çözüm yolu birçok yarar sağlar:

- Programın yönetimi kolaylaşır.
- Daha doğru çözüm üretilebilir.
- Daha kolay hata ayıklama yapılabilir.
- Kod satırlarını değiştirmek/genişletmek kolaylaşır.

Python programlama dilinde bir fonksiyon için iki durum söz konusudur:

- Fonksiyon tanımlama: Fonksiyonun nasıl davranışacağını tanımlayan kod satırları.
- Fonksiyon çağrıma: Program içinde fonksiyonun çağırılması ile kod satırlarının çalışması.

Her fonksiyonun bir kez tanımlanması ancak farklı şekillerde çağrılmaması söz konusudur.

### 7.1.1. Fonksiyon Tanımlama

Fonksiyon tanımlamak için dikkat edilmesi gereken dört durum vardır:

- def: Bu ayrılmış sözcük ile fonksiyon tanımlama başlar.
- İsim: Fonksiyon için bir isim verilmelidir. Aynı değişken tanımlamada olduğu gibi.
- Parametre: Fonksiyon içinde kullanılan değerleri ifade eder.
- Gövde: Fonksiyon için gerekli olan kod blokundan oluşur.



### 7.1.2. Fonksiyon Yazma

Aşağıdaki örnekte def kelimesi fonksiyon tanımlama için kullanılmıştır. Fonksiyon ismi double olarak belirlenmiştir. Kullanıcıdan n ile bir değer istenmiştir. Fonksiyona ait kod bloku bir satırdan oluşmakta olup bu, girinti ile öteleme olmuştur.

```
def double(n):
    return 2 * n
x = double(3)
print(x)
```

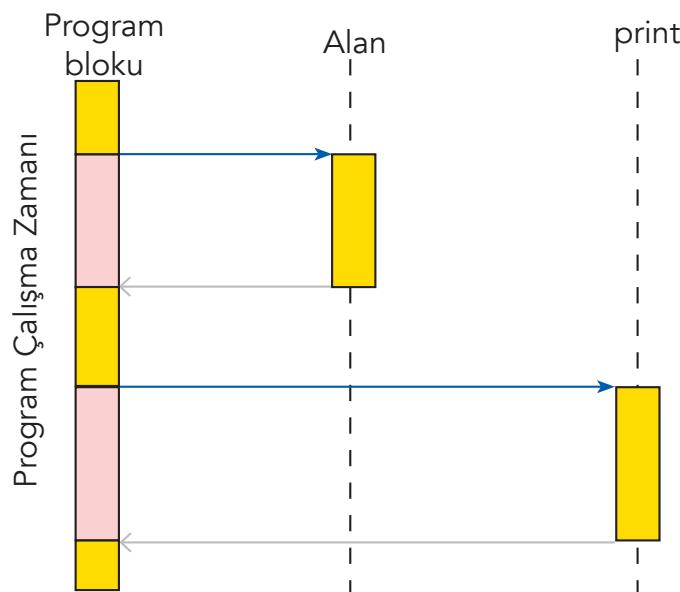
### 7.1.3. Fonksiyon Çağırma

Aşağıdaki örnekte fonksiyon, 5 değeri ile a değişkeni içerisine çağrılmıştır. return komutu ile gelen değer, print() komutu ile yazılmıştır.



```
# karenin alanını hesaplayan program
def Alan(a)
    return a*a
hesapla=Alan(5)
print(hesapla)
```

**Fonksiyon şu şekilde çalışır**



Program alan alt programına 5 değerini göndererek dallanma yapar. Alan alt programından hesapla değişkenine return komutu ile gelen değer aktarılır ve bu değer print komutu ile ekrana yazdırılır.

## 7.2. Fonksiyon Kullanımı Örnekleri

**Örnek**

```
def say():
    for i in range(1, 11):
        print(i, end=" ")
    print()
print("10'a kadar sayılıyor. . .")
say()
print("Tekrar 10'a kadar sayılıyor. . .")
say()
```

**Ekran Çıktısı**

```
10'a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
Tekrar 10'a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
```



### Düşünelim/Deneyelim

Bir fonksiyon yazarak, program içinde kaç defa çağrılabileceğini deneyiniz.

#### Örnek

```
def say_n(n):
    for i in range(1, n + 1):
        print(i, end=" ")
    print()
print("10'a kadar sayılıyor. . .")
say_n(10)
print("5'e kadar sayılıyor. . .")
say_n(5)
```

#### Ekran Çıktısı

```
10'a kadar sayılıyor. . .
1 2 3 4 5 6 7 8 9 10
5'e kadar sayılıyor. . .
1 2 3 4 5
```



### Düşünelim/Deneyelim

Sabit değerle çağırılan bir fonksiyon yazınız.

#### Örnek

```
def say_n(n):
    for i in range(1, n + 1):
        print(i, end=" ")
    print()
for i in range(1, 10):
    say_n(i)
```

#### Ekran Çıktısı

```
1
1 2
1 2 3
1 2 3 4
```



```
1 2 3 4 5  
1 2 3 4 5 6  
1 2 3 4 5 6 7  
1 2 3 4 5 6 7 8  
1 2 3 4 5 6 7 8 9
```



### Düşünelim/Deneyelim

Bir fonksiyon yazarak bir döngü içinde kaç defa çağrılabileceğini deneyiniz.

## 7.3. Değer Gönderme ile İlgili Olası Sorunlar

say\_n()

Fonksiyon çağrıldığında eksik parametre hatası verir.

say\_n(3, 5)

Fonksiyon çağrıldığında fazla parametre hatası verir.

say\_n(3.2)

Tam sayı olmadığı için çalışma zamanı hatası verir.

## 7.4. Çoklu Değer Gönderme Örnekleri

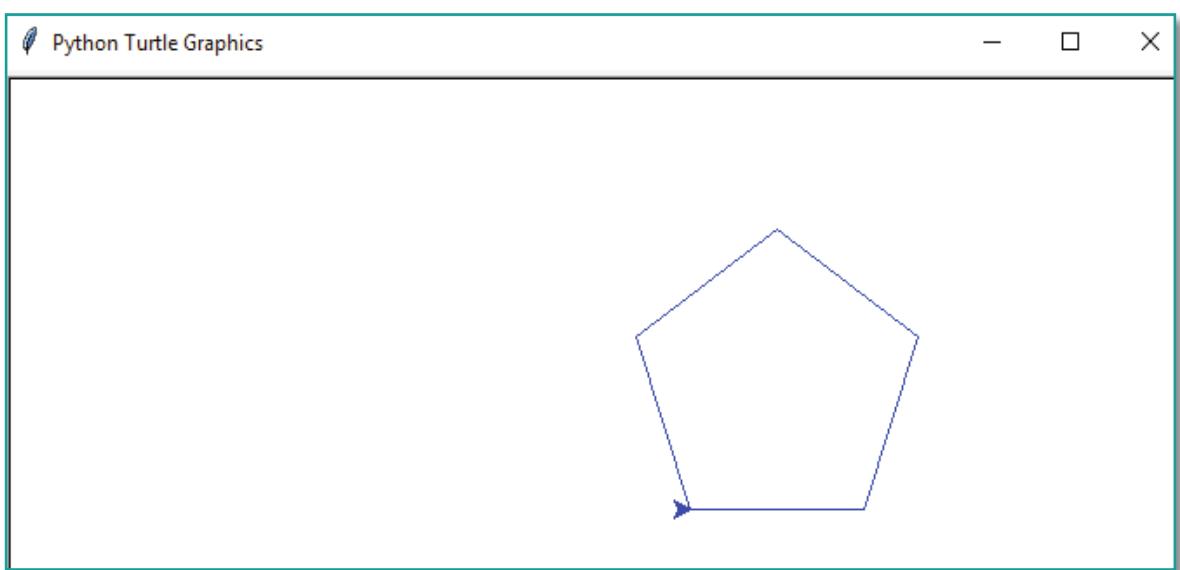
### 7.4.1. Grafik Ortamda Beşgen Çizimi

```
# turtle modülünü kullanarak beşgen çizimi  
import turtle  
import random  
# turtle modülü programa ekleniyor  
def polygon(sides,length,x,y,color):  
    #polygon adında ve içerisinde 5 değer alabilen bir fonksiyon  
    # tanımlanıyor  
    turtle.penup()  
    # Çizimi yapacak kalemin yönleri belirleniyor  
    turtle.setposition(x,y)  
    # Parametre olarak gelen x,y değerleri başlangıç noktası olarak  
    # belirleniyor  
    turtle.pendown()  
    # Çizimi yapacak kalemin yönleri belirleniyor  
    turtle.color(color)  
    # Parametre olarak gelen renk atanıyor
```



```
turtle.begin_fill()
for i in range(sides):
    turtle.forward(length)
    turtle.left(360//sides)
turtle.end_fill()
polygon(5,100,50,"blue") # Tanımlanan fonksiyonun istenilen parametrelerle çağırılması
```

### Ekran Çıktısı



Polygon fonksiyonu, 4 parametre alıyor: kenar sayısı, her kenarın uzunluğu, x ve y ise polygonun koordinatları ve polygonun rengi.

### 7.4.2. En Büyük Ortak Çarpan Fonksiyonu

```
# Girilen 2 sayıdan en büyük ortak bölen program
sayi1=int(input("Lütfen ilk sayıyı giriniz: "))
sayi2=int(input("Lütfen ikinci sayıyı giriniz: "))
def gcd(s1,s2):
    min=s1 if s1<s2 else s2
    ebop=1
    for i in range(1,min+1):
        if s1%i==0 and s2%i==0:
            ebop=i # En büyük ortak bölen aktarılıyor
    return ebop
```

## Ekran Çıktısı

```
Lütfen ilk sayınızı giriniz: 24  
Lütfen ikinci sayınızı giriniz: 18  
En büyük ortak çarpan : 6
```

## 7.5. Yerel Değişken

Yerel değişkenler fonksiyonların içinde tanımlanıp sonlandırılan değişken türüdür. Fonksiyon içerasına girildiğinde tanımlanıp hafızada yer ayıırlar ve fonksiyondan çıktılarında hafızadan silinirler.

```
x=2  
  
print("1. x =",x)  
  
def fun1():  
    x=10  
    print("2. x =",x)  
  
print("3. x =",x)  
  
def fun2():  
    x=20  
    print("4. x =",x)  
  
print("5. x =",x)  
  
fun1()  
  
fun2()  
  
print("6. x =",x)
```

## Ekran Çıktısı

```
1.x=2  
3.x=2  
5.x=2  
2.x=10  
4.x=20  
6.x=2
```



## 7.6. Fonksiyon Yazarken Fonksiyon Sıralamasını Belirleme

Bir programın içerisinde fonksiyon tanımı, kullanımından önce ifade edilmelidir. Aksi takdirde program hata verecektir. Python yorumlayıcısı, bir kod blokunu satır satır çalıştırır. Fonksiyonu çalıştırmadan önce yukarıda tanımına rastlamaz ise program çalışmayaçaktır.

### 7.6.1. Girilen İki Değerin En Büyük Ortak Böleni

```
def gcd(sayı1,sayı2):
    min=sayı1 if sayı1<sayı2 else sayı2
    ebop=1
    for i in range(1,min+1):
        if sayı1 % i== 0 and sayı2 % i== 0:
            ebop=i # En büyük ortak bölen aktarılıyor
    return ebop
def SayiGir():
    return int(input("Lütfen bir sayı giriniz : "))
def main():
    s1=SayıGir()
    s2=SayıGir()
    print("gcd(",s1, ", ",s2, " ) = ",gcd(s1,s2),sep="")
main()
```

#### Ekran Çıktısı

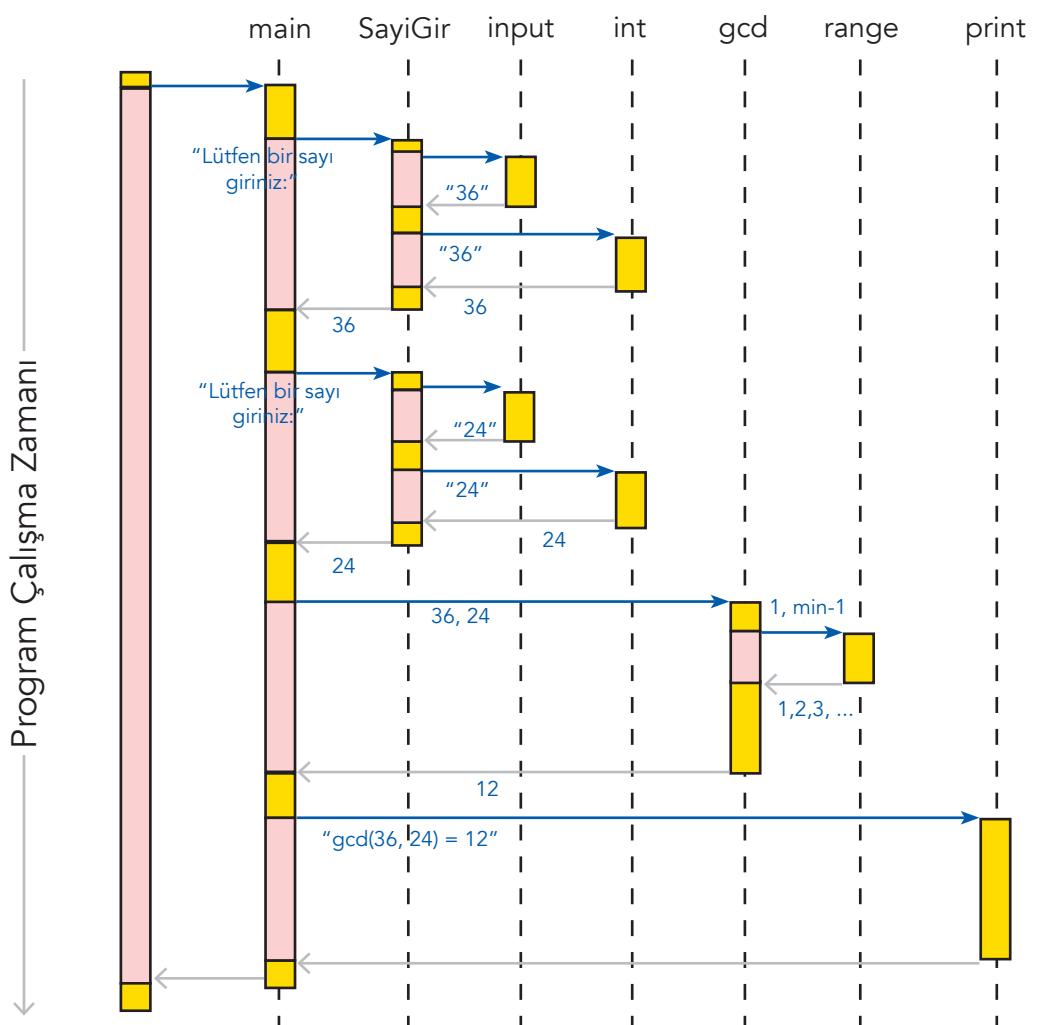
```
Lütfen bir sayı giriniz: 36
Lütfen bir sayı giriniz: 24
gcd (56,32) = 12
```

Kodlarda main() fonksiyonunda EBOB'u bulunacak iki sayı kullanıcıdan istenmiş ve bu iki sayı gcd() fonksiyonuna gönderilmiştir. gcd() fonksiyonu içerisinde küçük olan sayıya kadar döngü kurulmuş ve döngü içerisinde iki sayıya aynı anda bölünen sayı EBOB olarak belirlenmiştir.

main () ifadesi, ana fonksiyonu ve diğer fonksiyonları çağırır.



## Programın Çalışma Sırası



Program çalıştığında ilk olarak main() fonksiyonu çalıştırılır. SayıGir() fonksiyonuna iki kere daldırma yaparak s1 ve s2 değişkenlerine girilen sayıları aktarır ( $s1=36$ ,  $s2=24$ ). Üçüncü aşamada gcd fonksiyonuna s1 ve s2 değişkenleri parametre olarak gönderilir. for döngüsü içerisinde bu parametreler kullanılarak en büyük ortak bölen değeri bulunur ve return deyimi ile çağırıldığı yere gönderilir. print komutu ile gönderilen değer ekrana yazdırılır.

## 7.7. Parametre Gönderme

Bir fonksiyonun içerisinde parametre alacağı belirtilmişse mutlaka parametre gönderilmelidir.

```
def artir(x):
    print("Değişkenin artırılması正在被执行, x =",x)
    x+=1
    print("Artırma sonucu değer, x =",x)
    return x
```

```
def main():
    x=5
    artir(x)
    print("Artırma sonrası, x =",x)
    print("Artırma sonrası, x =",artir())
main()
```

### Ekran Çıktısı

```
Değişkenin artırılması yapılıyor, x = 5
Artırma sonucu değer, x = 6
Artırma sonrası, x = 5
Traceback (most recent call last):
  File "/Users/macbook/Desktop/Fwd __ Bilgisayar _ Bilimi _ Taslak _
Kitap/Pyton Kitap Kodlar/artirmaFonk.py", line 11, in <module>
    main()
  File "/Users/macbook/Desktop/Fwd __ Bilgisayar _ Bilimi _ Taslak _
Kitap/Pyton Kitap Kodlar/artirmaFonk.py", line 10, in main
    print("Artırma sonrası, x =",artir())
TypeError: artir() missing 1 required positional argument: "x"
>>>
```

Program başladığında `artir()` fonksiyonuna parametre olarak 5 değeri gönderilir ve işlem yaptırılır. İkinci kez `artir()` fonksiyonu çağrıldığında parametre verilmemişinden hata kodları alınır.

## 7.8. Fonksiyon Yazarken Tanımlayıcı Bilgileri Ekleme

Kod satırları için açıklamalar eklemek programı geliştirirken yarar sağlayacaktır.

- Fonksiyonun amacı
- Alınacak parametrenin görevi
- Geri dönüş değeri, açıklama satırlarında belirtilebilir.

Ayrıca, fonksiyon yazarı, değiştirilme tarihi varsa referanslar da eklenebilir.

```
# Yazar: Oğuz İŞIK
# Son düzenleme: 2017-01-06
# Yayınlanan bir örnekten uyarlanmıştır
def gcd (s1,s2)
    # Girilen iki değer arasındaki EBOB'u bulur
```

## 7.9. Fonksiyon Örnekleri

### 7.9.1 Asal Sayıların Bulunması

```
from math import sqrt
def asal(n):
    kok=round(sqrt(n))+1

    for deneme in range(2,kok):
        if n % deneme==0:
            return False
        else:
            return True

def main():
    en _ buyuk=int(input("Asal sayıları hangi değere kadar gösterelim? "))
    for deger in range(2,en _ buyuk+1):
        if asal(deger):
            print(deger,end=" ")
            print()
main()
```

#### Ekran Çıktısı

```
Asal sayıları hangi değere kadar gösterelim? 15
3
5
7
9
11
13
15
```

### 7.9.2 Hesaplama Yapan Fonksiyon

```
def yardim():
    print("Topla : Girilen iki sayıyı toplar")
    print("Fark Al : Girilen iki sayının farkını alır")
    print("Yazdır : İşlem yapılan en son değeri ekrana yazdırır")
    print("Yardım : Bu ekranı görüntüler")
    print("Çıkış : Programdan çıkışını sağlar")
def menu():
```



```

return input("===(T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===")
def main():
    result=0.0
    done= False
    while not done:
        secim=menu()
        if secim=="T" or secim=="t":
            sayil=float(input("Sayı 1: "))
            sayi2=float(input("Sayı 2: "))
            sonuc=sayil+sayi2
            print(sonuc)
        elif secim=="F" or secim=="f":
            sayil=float(input("Sayı 1: "))
            sayi2=float(input("Sayı 2: "))
            sonuc=sayil-sayi2
            print(sonuc)
        elif secim=="Y" or secim=="y":
            print(sonuc)
        elif secim=="A" or secim=="a":
            yardım()
        elif secim=="Ç" or secim=="ç":
            done= True
    main()

```

### Ekran Çıktısı

```

==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===T
Sayı Giriniz #1: 12
Sayı Giriniz #2: 13
25.0
==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===F
Sayı Giriniz #1: 23
Sayı Giriniz #2: 21
2.0
==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===Ç

```

Bu programda dört işlem harf seçimine göre yapılmaktadır. `main()` fonksiyonu çağrıldığında yapılacak işlem secim değişkenine aktarılarak kontrol edildikten sonra istenilen işlem yapılacaktır.

### 7.9.3 Kısıtlı Veri Girişi

```
def arasindami(ilk,son):
    if ilk>son:
        ilk,son=son,ilk
    deger=int(input("Lütfen belirtilen aralıkta bir değer girin " \
                    +str(ilk)+ "..." +str(son)+ ": "))
    while deger<ilk or deger>son:
        print(deger, "Bu değerler arasında değil",ilk, "...",son)
        deger=int(input("Tekrar deneyin: "))
    return deger
def main():
    print(arasindami(10,20))
    print(arasindami(20,10))
    print(arasindami(5,5))
    print(arasindami(-100,100))
main()
```

### Ekran Çıktısı

```
Lütfen belirtilen aralıkta bir değer girin 10...20: 4
4 Bu değerler arasında değil 10 ... 20
Tekrar deneyin: 21
21 Bu değerler arasında değil 10 ... 20
Tekrar deneyin: 16
16
Lütfen belirtilen aralıkta bir değer girin 10...20: 10
10
Lütfen belirtilen aralıkta bir değer girin 5...5: 4
4 Bu değerler arasında değil 5 ... 5
Tekrar deneyin: 6
6 Bu değerler arasında değil 5 ... 5
Tekrar deneyin: 5
5
Lütfen belirtilen aralıkta bir değer girin -100...100: -101
-101 Bu değerler arasında değil -100 ... 100
Tekrar deneyin: 101
101 Bu değerler arasında değil -100 ... 100
Tekrar deneyin: 0
0
```

Programda istenilen değerler arası veri girilmediğinde hata mesajı, girildiğinde onay mesajı verilmektedir.

#### 7.9.4 Ağaç Çizimi

```
def tree(yukseklik):
    satir=0 # Ağacın çizilmesine başlanıyor
    while satir<yukseklik: # Girilen yükseklik değerine kadar döngü
        kuruluyor
            sayac=0
            while sayac<yukseklik-satir:
                print(end=" ")
                sayac+=1
            sayac=0
            while sayac<2*satir+1:
                print(end="*")
                sayac+=1
            print()
            satir+=1
def main():
    yukseklik=int(input("Ağacın yüksekliğini giriniz: "))
    tree(yukseklik)
main()
```

#### Ekran Çıktısı

```
Ağacın yüksekliğini giriniz: 4
*
 ***
 *****
 *****
```

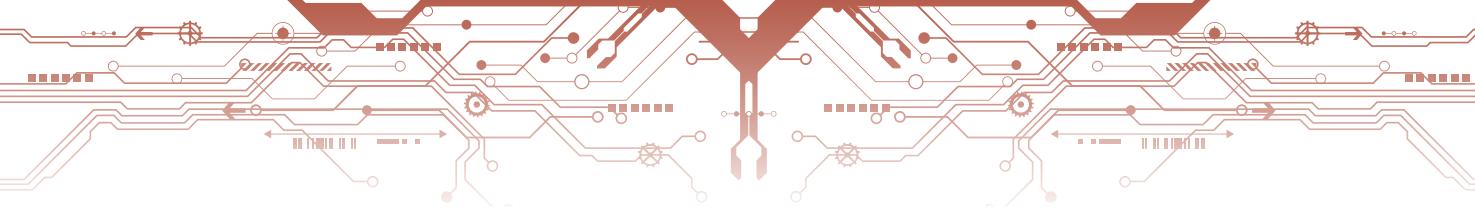
Çizilmesi istenilen ağaç yüksekliğinin parametre olarak alan tree() fonksiyonu while döngüsüyle problemin çözümünü gerçekleştirir.

## 8. FONKSİYONLAR 2



Bu bölümde;

- ✓ Fonksiyonlarda kullanılan global ve yerel değişkenler hakkında bilgi sahibi olabilecek,
- ✓ Fonksiyonların tekrarlı kullanımlarını kavrayabileceksiniz.



## 8.1. Global Değişkenler

Fonksiyonların içerisinde tanımlanan değişkenler yerel değişkenlerdir. Bu değişkenlerin bazı özelilikleri vardır:

- Bu değişkenler, hafızada aktif kullanıldıkları sürece yani sadece fonksiyon çalışırken korunur. Fonksiyondan çıktılığında bu değişkenler de program tarafından unutulur ve hafızada ayrılan yerde başka değişkenler tarafından kullanılır.
- Aynı değişken adı çakışma olmaksızın farklı fonksiyonlarda kullanılabilir. Bir fonksiyon sonlanmadan diğerini çalışmaya başlayamayacağı için aynı değişken kullanılması sorun yaratmaz.
- Yerel değişkenler geçicidir; fonksiyonlar çağrıldığında kaybolur. Bazen bu çalışma sürecinden bağımsız bir değişkene ihtiyaç duyulur.
- Global değişken: Ne zaman hangi fonksiyon çağrırlırsa çağrılsın, program tarafından tanımlanır ve hafızada sürekli yeri olan bir değişken.
- Bir değişken, bir nesneye atandığı zaman tanımlanır. Bir fonksiyona atanmış olsalar da, o fonksiyon için yereldir. Ancak global olarak tanımlanır ise programın tümü tarafından tanımlanır ve kullanılır.

## 8.2. Örnekler

### 8.2.1. Hesap Makinesi Örneği

```
# Kullanıcı seçimine göre Toplama ve Çıkarma işlemi yapan program kodları

def Yardim():
    print("Topla : Girilen iki sayıyı toplar")
    print("Fark Al : Girilen iki sayının farkını alır")
    print("Yazdır : İşlem yapılan en son değeri ekrana yazdırır")
    print("Yardım : Bu ekranı görüntüler")
    print("Çıkış : Programdan çıkışını sağlar")

def Menu():
    return input("===(T)opla (F)ark Al (Y)azdır (Y(A)rdım (Ç)ıkış ===")

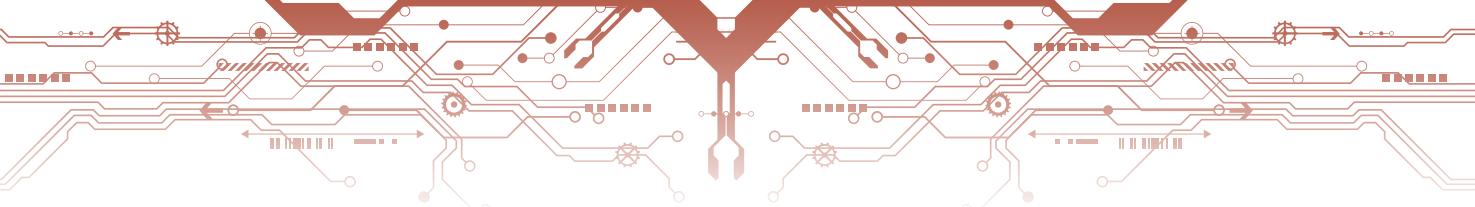
# Programda kullanılmak üzere global değişken tanımlaması

sonuc = 0.0
sayil = 0.0
sayi2 = 0.0

def SayiGir():
    global sayil, sayi2 # sayil ve sayi2 nin global değişken olarak bildirilmesi
    sayil = float(input("Sayı Giriniz #1: "))
    sayi2 = float(input("Sayı Giriniz #2: "))

def Yazdir():
    print(sonuc)

def Topla():
```



```
global sonuc
sonuc = sayil + sayi2
def FarkAl():
    global sonuc
    sonuc = sayil - sayi2
def main():
    durum = False
    while not durum:
        secim = Menu() # İşlem yapılacak Menü Tasarımı
        if secim == "T" or secim == "t": # Toplama
            SayiGir()
            Topla()
            Yazdir()
        elif secim == "F" or secim == "f": # Çıkarma
            SayiGir()
            FarkAl()
            Yazdir()
        elif secim == "Y" or secim == "y": # Yazdırma
            Yazdir()
        elif secim == "A" or secim == "a": # Yardım
            Yardim()
        elif secim == "Ç" or secim == "ç": # Çıkış
            durum = True
main()
```

### Ekran Çıktısı

```
==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===T
Sayı Giriniz #1: 12
Sayı Giriniz #2: 43
55.0
==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===F
Sayı Giriniz #1: 44
Sayı Giriniz #2: 32
12.0
==== (T)opla (F)ark Al (Y)azdır Y(A)rdım (Ç)ıkış ===Ç
```

### 8.2.2. Varsayılan (Default) Parametreler

Parametreli fonksiyonlar çağırıldığında bir değer gönderilmesi gereklidir. Bazı durumlarda bu değer gönderilmeden de fonksiyon çalıştırılmak istenirse, fonksiyonun tanımlama aşamasında gönderilmesi istenen parametreye varsayılan olarak bir değer verilmesi gereklidir. Örneğin `a=input()` ya da `a=input("Adınızı giriniz")`

```
def gerisayim(n=5):
    for sayac in range(n, -1, -1):
        print(sayac)

gerisayim()
print()
gerisayim(8)
```

#### Ekran Çıktısı

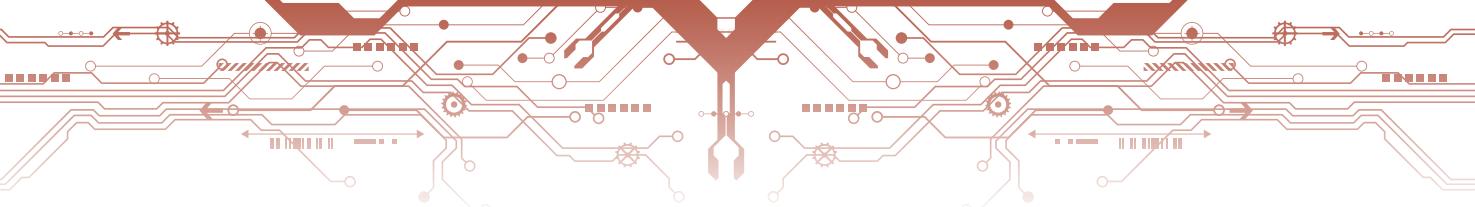
```
5
4
3
2
1
0

8
7
6
5
4
3
2
1
0
```

Yukarıdaki programda `gerisayim()` fonksiyonu iki kere çağrılmıştır. İlk çağrımda parametre verilmemiş için fonksiyonun tanımında verilen default değer ile işlem yapılmış ve 5'ten geriye doğru sayma işlemi gerçekleşmiş ve ekrana yazılmıştır. İkinci çağrılmamasında ise 8 değeri parametre olarak gönderilmiş ve bu değer ile işlem yapılmıştır.

### 8.2.3. Varsayılan ve Diğer Parametreler

```
def AralikTopla(n, m=100): # Tek değer varsayılan atama
    toplam = 0
    for deger in range(n, m + 1):
        toplam += deger
```



Yukarıdaki AralikTopla() fonksiyonunun ikinci (m) parametresine değer girilmez ise varsayılan olarak 100 değeri alınır.

```
def AralikTopla(n=0, m=100): # İki değer varsayılan atama
    toplam = 0
    for deger in range(n, m + 1):
        toplam += deger
```

Yukarıdaki AralikTopla() fonksiyonunun birinci (n), ikinci (m) parametrelerine değer girilmez ise varsayılan olarak sırasıyla 0 ve 100 değerleri alınır.

```
def AralikTopla(n=0, m): # Tek değer varsayılan atama
    toplam = 0
    for deger in range(n, m + 1):
        toplam += deger
```

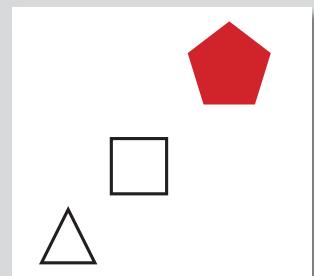
Yukarıdaki AralikTopla() fonksiyonunun birinci (n) parametresine değer girilmez ise varsayılan olarak 0 değeri alınır.

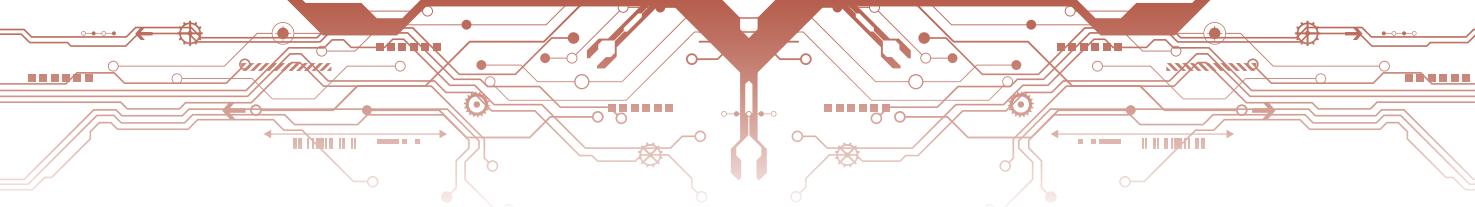
#### 8.2.4. Gelişmiş Poligon Çizimi

```
import turtle
import random

# Program fonksiyona gönderilen parametreler ile çokgen çizer.
# Uzunluk parametresi girilerek herbir kenarın uzunluğu belirlenir.
# Çizim x ve y parametrelerine girilen koordinat noktalarından başlar.
# Bir sonraki parametre çizimin kenar rengini belirler. (Varsayılan değer olarak siyah).
# Çizilen çokgenin içine dolgu olup olmayacağı belirlenir(Varsayılan False).

def Cokgen(kenarSayisi, uzunluk, x, y, renk="black", dolgu=False):
    turtle.penup()
    turtle.setposition(x, y)
    turtle.pendown()
    turtle.color(renk)
    if dolgu:
        turtle.begin_fill()
    for i in range(kenarSayisi):
        turtle.forward(uzunluk)
        turtle.left(360//kenarSayisi)
    if dolgu:
        turtle.end_fill()
```





```
# Adım adım çizim işlemi iptal edilerek çizim hızlandırılıyor
turtle.hideturtle()
turtle.tracer(0)

# Fonksiyonlar örnek çizimler için kullanılıyor
Cokgen(3, 30, 10, 10) # Üçgen çizimi
Cokgen(4, 30, 50, 50, "blue") # Kenar rengi mavi olan Kare çizimi
Cokgen(5, 30, 100, 100, "red", True) # Dolgusu kırmızı olan beşgen
çizimi
turtle.update()
turtle.exitonclick() # Fare tuşuna tıklandığında çıkış işlemi
yapılacaktır.
```

### 8.2.5. Öz Yineleme

Bir fonksiyonun kendisini çağrıarak çözüme gitmesine özyineleme, böyle çalışan fonksiyonlara da özyinelemeli fonksiyonlar denilir. Özyinelemeli algoritmalarla, tekrarlar fonksiyonun kendi kendisini kopyalayarak çağırması ile elde edilir. Bu kopyalar işlerini bitirdikçe kaybolur. Bu yönteme en uygun örnek faktöriyel problemidir.

```
#Özyineleme ile faktöriyel hesaplama
def faktoriyel(n):
    #Gelen n değerinin faktöriyeli alır.
    if n == 0:
        return 1
    else:
        return n * faktoriyel(n-1)
def main():
    # Fonksiyonumuza çeşitli değerler ile test edelim
    print(" 0! = ", faktoriyel(0))
    print(" 1! = ", faktoriyel(1))
    print(" 6! = ", faktoriyel(6))
    print("10! = ", faktoriyel(10))
main()
```

### Sonuç Ekranı

```
0! = 1
1! = 1
6! = 720
10! = 3628800
```

## Çalışma Prensibi

```
faktoriyel(6) = * faktoriyel(5)
                  = 6 * 5 * faktoriyel(4)
                  = 6 * 5 * 4 * faktoriyel(3)
                  = 6 * 5 * 4 * 3 * faktoriyel(2)
                  = 6 * 5 * 4 * 3 * 2 * faktoriyel(1)
                  = 6 * 5 * 4 * 3 * 2 * 1 * faktoriyel(0)
                  = 6 * 5 * 4 * 3 * 2 * 1 * 1
                  = 6 * 5 * 4 * 3 * 2
                  = 6 * 5 * 4 * 6
                  = 6 * 5 * 24
                  = 6 * 120
                  = 720
```

Öz yinelemeli fonksiyonlar kendilerini çağırarak işlem yapan fonksiyonlardır. Yukarıdaki örnekte parametre olarak gelen değerden başlayarak birer azaltıp kendini çağrıma işlemi yapar. Bu çağrıma işlemi, döngü parametre değeri, 0 olasıya kadar devam eder. 0 olduğunda kendini çağrıma işlemi durur ve geriye doğru değerler çarpılarak sonuç bulunur.

### 8.2.6. Öz Yineleme Olmadan Faktöriyel Hesaplama

```
def faktoriyel(n):
    sonuc = 1
    while n:
        sonuc *= n
        n -= 1
    return sonuc
def main():
    print(" 0! = ", faktoriyel(0))
    print(" 1! = ", faktoriyel(1))
    print(" 6! = ", faktoriyel(6))
    print("10! = ", faktoriyel(10))
main()
```

## Sonuç Ekranı

```
0! = 1
1! = 1
6! = 720
10! = 3628800
```

### 8.2.7. Fibonacci Sayıları

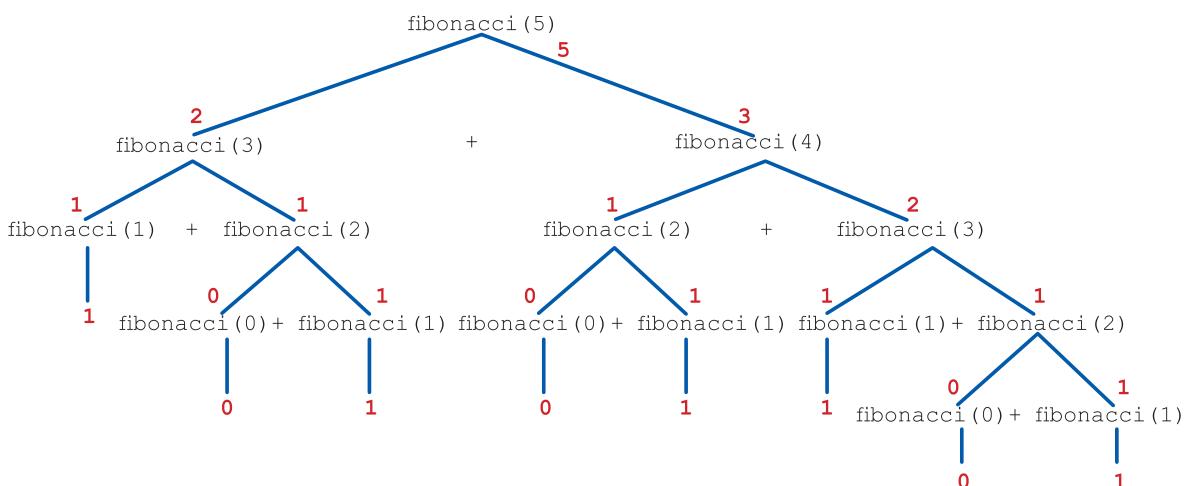
```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 2) + fibonacci(n - 1)
sira=int(input("Görmek istediğiniz fibonacci sıra numarasını giriniz.:"))
print(fibonacci(sira))
```

#### Sonuç Ekranı

Görmek istediğiniz fibonacci sıra numarasını giriniz.:22  
17711

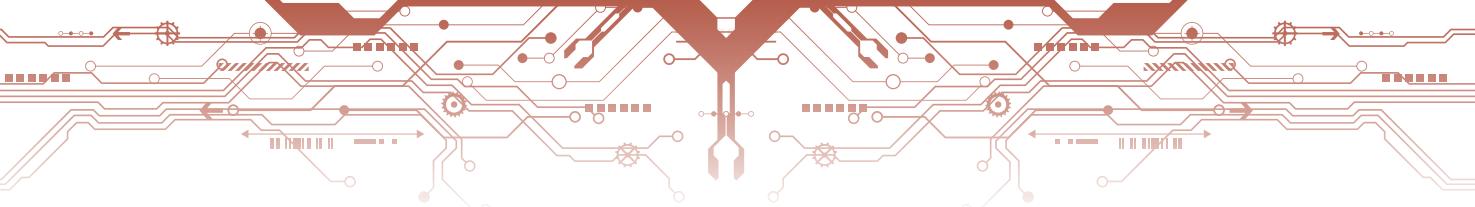
**Fibonacci Sayıları:** Her sayının kendisinden önce gelen iki sayının toplamı şeklinde yazılıp devam ettiği sayı dizisine Fibonacci Sayı Dizisi denir.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...



### 8.3. Fonksiyonları Tekrar Kullanılabilir Yapma

Fonksiyonlar programlar içerisinde defalarca kullanılabilir. Aynı amaçlı fonksiyonların birden fazla program tarafından da kullanılması istenebilir. Bu durumda tanımlanan fonksiyon, amacına uygun çağrışım yapacak bir isim verildikten sonra kullanılmak istenen programlarda “from” komutu ile dosya adı yazılarak çağırılabilir.



## Kullanımı

```
from <dosya adı> import <fonksiyon adı>
```

### Örnek

```
# Asal sayının kontrol edildiği fonksiyon tanımlama
from math import sqrt

def AsalKontrol(n):
    # Fonksiyona gelen değer asal ise geriye True, değilse False döner.
    bolen= 2
    kok = sqrt(n)
    while bolen <= kok:
        if n % bolen == 0: # Kalan kontrolü yapılıyor
            return False # Tam bölünme işlemi gerçekleşti. Asal Değil
        bolen += 1 # Bir sonraki bölen değerine geçiliyor.
    return True # Tüm değer kontrollerinden sonra kalanlı bölme
gerçekleşmediğinde, True değeri dönüyor.
```

Yukarıda yazılan kodlar, gönderilen sayının asal olup olmadığını kontrol eder. Gelen sayı asal ise geriye True, değilse False gönder. Kodlar yazıldıktan sonra dosya Kontrol.py adı ile kaydedilmelidir.

Aşağıdaki kod, kendisi ile aynı dizinde bulunan Kontrol.py dosyasında bulunan AsalKontrol fonksiyonunu kullanıyor.

```
from Kontrol import AsalKontrol
# Kontrol dosyasındaki AsalKontrol fonksiyonu programa ekleniyor
sayi = int(input("Bir sayı giriniz.: "))
if AsalKontrol(sayi):
    print(sayi, "ASAL")
else:
    print(sayi, "ASAL değil")
```

Diğer bir yol ise şu şekildedir:

```
import Kontrol
sayi = int(input("Bir sayı giriniz.: "))
if Kontrol.AsalKontrol(sayi):
    print(num, "sayısı ASAL sayıdır.")
else:
    print(num, "sayı ASAL değildir.")
```

## 9. NESNELER



Bu bölümde;

- ✓ Python dilinde kullanılan nesneler hakkında bilgi sahibi olacak,
- ✓ Nesneler içeren program yazabilecek,
- ✓ Dizi nesneleri kullanabilecek,
- ✓ Dosya nesneleri ile dosyalarda veriler üzerinde işlemler yapabilecek,
- ✓ Grafik nesneleri ile programlar geliştirebileceksiniz.

## 9.1. Nesne Kavramı

Nesne, kendine has özellikler olan ve bu özellikler doğrultusunda bulunduğu duruma bağlı olarak çeşitli tutumlar sergileyen somut ya da soyut varlıklardır. Bu tanım, "somut" kısmıyla gerçek yaşamdaki nesneler için de geçerlidir. Donanım açısından baktığımızda kişisel bir bilgisayar; ana kart, işlemci, video kart, sabit disk ve kontrol ünitesi, bunları barındıran bir kasa, klavye, fare ve ekrandan oluşur. Video kart; çip, hafıza ve diğer elektronik bileşenleri barındıran karmaşık bir yapıdır. Somut kısmıyla nesnelere, donanım birimleri örnek olarak verilebilir.

Ancak gerçek yaşamda "nesne" olarak nitelendirdiğimiz fonksiyon, değişken, dizi gibi kavamlar programlama ortamında nesne olarak tanımlanabilirler. Bu da tanımın "soyut" diye ifade ettiği bölümü oluşturur.

Günümüzde yazılım geliştirme daha çok yazılımların donanım gibi kullanıldığı bileşenlere dayanmaktadır. Bir yazılım sistemi var olan yazılım oluşturma blokları üzerine inşa edilir. Python, farklı yapıdaki blokları ve bileşenleri desteklemektedir.

Python nesne yönelimli bir programlama dilidir. Nesne yönelimli programlama dili programcının nesneleri tanımlamasına, oluşturmaya ve yönetmesine olanak sağlar. Nesneler, veri ve fonksiyonları bir araya toplar. Diğer değişkenler gibi Python nesnelerinin de tipi ve sınıfı vardır.

## 9.2. Nesneleri Kullanmak

Bir nesne bir sınıfa örnek olarak verilebilir. Aslında başından beridir nesneleri kullanıyoruz ama detaylı bir biçimde şimdi inceleyeceğiz. Tam sayılar, reel sayılar, diziler ve fonksiyonlar Python için birer nesnedir. Fonksiyonların dışında genel olarak nesneleri pasif veri olarak kullandık. Bir değişkene tam sayı atayarak daha sonra o değişkenin değerini kullanabiliyoruz. "+" operatörü ile iki reel sayıya ya da iki kelimeyi toplayabiliyoruz. Fonksiyonlara nesne yollayabilir ve sonucu nesne olarak alabiliyoruz.

Tipik bir nesne iki bölümden oluşur: veri ve metotlar. Örnek değişken ait olduğu nesne tarafından temsil edilen değişken anlamına gelir ve nesne de bir sınıf örneğidir. Örnek değişkenler için diğer isimler, özellik ve alanları da kapsamaktadır. Metotlar fonksiyon gibidir ve operatör olarak da ifade edilir. Bir nesne için örnek değişkenler ve metotlar nesnenin üyeleri olarak bilinir. "Nesneyi kullanan kod, nesne istemcisidir ve nesne istemcilere servis sunmaktadır." şeklinde açıklanır. Bir nesne tarafından sunulan servisler basit fonksiyonlara göre daha ayrıntılıdır. Bunun nedeni ise nesneler değişken içinde kolay veri korur çünkü nesnelerin örnek değişkenler içinde veri koruması daha kolaydır.

## 9.3. Dizi Nesneleri

Dizi nesnelerinin nasıl oluşturulacağını str örneği ile inceleyelim. Nesneler veri ve diziyi oluşturan veri diziyi oluşturan karakterlerin sıralaması hâline gelir.

Şimdi str metotlarını inceleyelim.

```
ad = input("Adını yaz: ")
print("Merhaba " + ad.upper() + ", nasılsın?")
```

Bu kodlama ile kullanıcı tarafından girilen dizideki bütün karakterler büyük harfe çeviriliyor.

```
Adını yaz: Filiz
Merhaba FİLİZ, nasılsın?
```



print ifadesinde yer alan ad.upper() komutu ile bir metot çağrılmaktadır. Genel olarak bir metot çağrımanın biçimini şu şekildedir:

nesne adı • metot adı ( parametre listesi )

- Bir önceki örnekte “name” bir dizi nesnesini çağrılmaktadır.
- “.” (nokta) nesneye bağlı olarak çağrıılacak metot ile bağ kurulduğunu ifade eder.
- “metot adı” çağrıılan ve çalışacak olan metot adıdır.
- “parametre listesi” metoda gönderilen ve virgülle ayrılmış parametre listesidir. Bazı metotlar parametre istemez, o zaman bu liste boş olabilir.

Metoda gönderilen parametre listesi, fonksiyona gönderilen parametre listesi ile tamamen aynı biçimde davranıştır. Bu nedenle metotlar da geri değer döndürebilir.

Aşağıdaki örnekte rjust komutu metni sağa hizalamak için kullanılmaktadır.

```
kelime = "ABCD"
print(kelime.rjust(10, "*"))
print(kelime.rjust(3, "*"))
print(kelime.rjust(15, ">"))
print(kelime.rjust(10))*****ABCD
```

### Ekran Çıktısı

```
*****ABCD
ABCD
>>>>>>>>>ABCD
ABCD
```

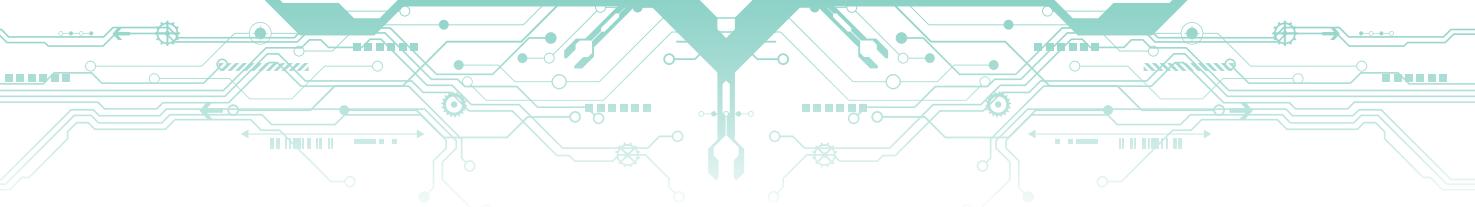
- kelime.rjust(10, “\*”) 10 karakterlik bir alan içinde boşluklara “\*” değeri vererek “ABCD” dizisini sağa dayalı yazar.
- kelime.rjust(3, “\*”) komutu belirtilen değer dizi boyutundan daha küçük olduğu için bir işlem yapamaz.
- kelime.rjust(15, “>”) 15 karakterlik bir alan içinde boşluklara “>” değeri vererek “ABCD” dizisini sağa dayalı yazar.
- kelime.rjust(10) ise boş alanlara basılacak varsayılan karakterin boşluk olduğunu belirtmektedir.

Aşağıdaki örnek, metodu dizi kullanarak çağrıabildiğimizi göstermektedir:

```
>>> "aBcDeFgHiJ".upper()
"ABCDEFGHIJ"
>>> "PYTHON çok güzel bir dildir.".rjust(35,"-")
"-----PYTHON çok güzel bir dildir."
```

Bu söz dizimi daha önce kullandığımız bazı örneklerle benzerlik göstermektedir.

```
>>> "{0} {1}".format(23, 9)
```



```
"23 9"  
>>> s = "{0} {1}"  
>>> s.format(23, 9)  
"23 9"
```

### 9.3.1 str Nesnesi İçin Yöntemler

Yöntem	Açıklama
upper	Metnin tüm harflerini büyük harfe dönüştürür.
lower	Metnin tüm harflerini küçük harfe dönüştürür.
rjust	Karakter dizisini sağa yaslar.
ljust	Karakter dizisini sola yaslar.
center	Karakter dizisini ortalar.
strip	Karakter dizisinin sağında ve solunda bulunan boşluk karakterlerini yok etmemizi sağlar.
startswith	Karakter dizilerinin ön ekini sorgulamamızı sağlar.
endswith	Karakter dizilerinin ön ekini sorgulamamızı sağlar.
count	Bir karakter dizisi içinde belli bir karakterin kaç kez geçtiğini sorgular.
find	Bir karakter dizisi içinde belli bir karakteri bulur.
format	Birimlendirilmiş değerleri, dizi konum parametrelerine yerleştirir.

#### strip ve count Fonksiyon Örneği

```
# Ön ve arka plandaki boşlukları ve sayıç alt dizilerini ayırır  
s = " ABCDEFGHBCDIJKLMNOPQRSTUVWXYZ "   
print("[", s, "]", sep="")  
s = s.strip()  
print("[", s, "]", sep="")  
# Alt dizinin sayılarını sayar "BCD"  
print(s.count("BCD"))
```

#### Ekran Çıktısı

```
[ ABCDEFGHBCDIJKLMNOPQRSTUVWXYZ ]  
[ABCDEFGHIJKLMNPQRSTUVWXYZ]  
3
```

### 9.3.2 getitem Kullanımı

“str” sınıfı `__getitem__` isimli bir metot ile karakterin dizideki sırasını verir. Metodun isminin “`__`” ile başlıyor olması, bu metodun dâhilî kullanımı olduğu ve istemcilerin kullanamayacağı anlamına gelir. İstemciler, bu metodu özel bir söz dizimi ile kullanabilirler.

```
>>> s = "ABCEFGHI"  
>>> s  
"ABCEFGHI"  
>>> s. __getitem__ (0)  
"A"  
>>> s. __getitem__ (1)  
"B"  
>>> s. __getitem__ (2)  
"C"  
>>> s[0]  
"A"  
>>> s[1]  
"B"  
>>> s[2]  
"C"
```

Dizi nesnelerinde ilk karakterin konumu ya da indeks değeri “0” olduğundan her bir karakterin konumu, köşeli ayraç (`[ ]`) içerisinde gösterilir.

Dizilerle kullanılabilen başka bir metot ise “`__len__`” metodudur. Böylece dizideki karakter sayısı elde edilir.

```
>>> s  
"ABCEFGHI"  
>>> s = "ABCEFGHI"  
>>> s  
"ABCEFGHI"  
>>> len(s)  
8  
>>> s. __len__ ()  
8
```

“`len(s)`” ve “`s.__len__()`” söz dizimleri fonksiyonel olarak aynıdır. İstemciler çağrıırken global “`len`” fonksiyonunu kullanmalıdır.

#### Karakter Yazma Örneği

```
s = "Hayatta En Hakiki Mürşit İlimdir"  
print(s)  
  
for i in range(len(s)):  
    print("[", s[i], "]", sep="", end="")  
  
print() # Yeni satır başı
```

```
for ch in s:  
    print("<", ch, ">", sep="", end="")  
print() # Yeni satır başı
```

### Ekran Çıktısı

```
Hayatta En Hakiki Mürşit İlimdir  
[H][a][y][a][t][t][a][ ][E][n][ ][H][a][k][i][k][i][ ][M][ü][r][ş][i][t][ ][İ][l]  
[i][m][d][i][r]  
<H><a><y><a><t><t><a>< ><E><n>< ><H><a><k><i><k><i><  
><M><ü><r><ş><i><t>< ><İ><l><i><m><d><i><r>
```

Diziler değişmez nesnelerdir. Bu nedenle bir dizi nesnesinin içeriğini değiştiremeyiz.

```
s = "ABCDEFGHIJKLMN"  
s[3] = "S" # Kurala aykırı, dizi sabit
```

Dizi değişmezliği, bir şeridin belirli bir diziyi değiştirmediği bir yöntem anlamına gelir.

```
s = " ABC "  
s.strip() # s değişmez  
print("<" + s + ">") # < ABC > yazılır , <ABC> değil
```

Ön ve arka plandaki boşlukları s değişkenine bağlı dizi kadar çıkarmak için yeniden atamanız gereklidir:

```
s = " ABC "  
s = s.strip() # Yeniden atama yapılır  
print("<" + s + ">") # <ABC> yazılır
```

“strip” metodu yeni bir dizi döndürür ancak mevcut dizi değiştirilmez. Bir diziden boşlukları etkili bir biçimde temizleyebilmek için istemci, strip metoduna gönderilen değişkene tekrar atama yapmak zorundadır. Mantıksal olarak değerlendirildiğinde boş dizi (“”) yanlış; tüm diğer diziler ise doğru olarak işlenir.

## 9.4. Dosya Nesneleri

Şu ana kadar çalıştığımız tüm programlar, sonlarındakılarda tüm verilerini kaybetti. Oysa bazı durumlarda bu verilerin saklanması gerekebilir. Örneğin kaydetmenize olanak sağlayamayan bir kelime işlemci programı düşünün. Bu durumda tekrar erişip düzenleme, çıktı alma vb. hiçbir işlemi yapmak mümkün olmaz.

Sistemlerin çoğu saklanması gereken veriyi dosya biçiminde kaydeder ve her bir programın da dosya türünü belirten özel uzantısı vardır.

Python kapsamında veri saklama ve geri çağrıma işlemleri “file (dosya)” nesnesi ile gerçekleştiririz. “io” modülündeki “TextIOWrapper” ile bu işlemi yaparız. Dosya işlemleri yaygın kullanıldığı için “io” modülü dâhilinde kullanılan fonksiyon ve sınıflar, “import” komutu kullanılmadan çalıştırılabilir.



```
f = open("dosyam.txt", "r")
```

İfadesi bir dosya oluşturarak “f” isimli bir dosya nesnesi döndürür. İlk parametre dosya adını; ikinci parametre ise dosyanın durumunu ifade eder. Dosya durumu aşağıdaki gibi olabilir.

- ‘r’ yalnızca okunabilir.
- ‘w’ dosyayı yazmak için açar, yeni dosya oluşturur.
- ‘a’ dosyaya yeni veri eklenerek değiştirme yapılabilir.

“f” isimli bir dosya nesnesi oluşturmak ve “dosyam.txt” isimli dosya içeriğini okuyabilmek için

```
f = open("dosyam.txt", "r")
```

söz dizimi kullanılır. Eğer dosya yoksa ya da programı kullanan kişinin dosyaya erişim için gereken izinleri bulunmuyorsa bu komut hata verecektir.

“f” isimli bir dosya nesnesi oluşturmak ve “dosyam.txt” isimli dosyaya yazabilmek için

```
f = open("dosyam.txt", "w")
```

söz dizimi kullanılır. Dosya yoksa fonksiyon, disk üzerinde yeni bir dosya oluşturur. Aynı isimli bir dosya zaten varsa dosyadaki eski veriler yenileri ile değişecektir. Bu, dosyanın içerisinde önceden oluşturulmuş içeriğin silineceği anlamına gelir. “f” isimli bir dosya nesnesi oluşturmak ve “dosyam.txt” isimli dosyaya erişmek ve veri ekleyebilmek için

```
f = open("dosyam.txt", "a")
```

söz dizimi kullanılır. Dosya yoksa yeni bir dosya oluşturulur. Aynı isimli bir dosya varsa bu dosya tekrar düzenlemek için erişime açılır. Böylece dosyanın mevcut içeriği korunmuş olur. Bu fonksiyon ikinci parametre unutularak çağırılırsa varsayılan değer olarak “r” atanır.

```
f = open("dosyam.txt") ile f = open("dosyam.txt", "r")
```

aynı işlemi gerçekleştirir. “w” ya da “a” izni ile erişilmiş ve yazabileceğiniz bir dosya nesneniz varsa write metodunu kullanarak dosya üzerinde işlem yapabilirsiniz.

```
f.write("kaynak")
```

komutu ‘kaynak’ verisini dosya içerişine yazar. Aşağıdaki 3 komut

```
f.write("kaynak")
```

```
f.write("dosya")
```

```
f.write("veri")
```

dosyaya ’kaynakdosyaveri’ verisini ekler. Veriyi ayırarak saklamak istiyorsak ona göre düzenlememiz gerekir.

```
f.write("kaynak\n")
```

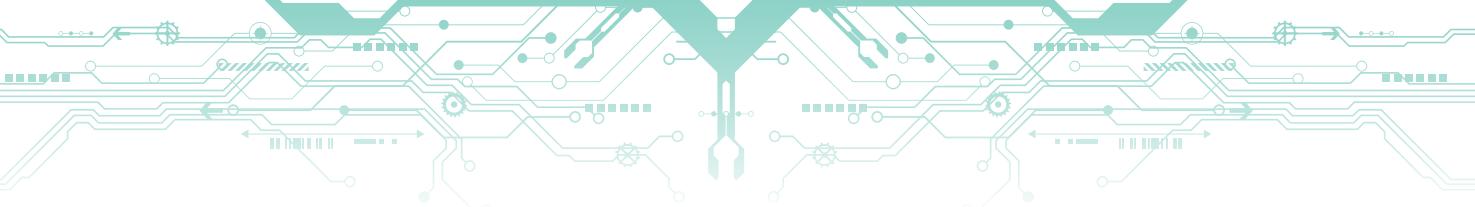
```
f.write("dosya\n")
```

```
f.write("veri\n")
```

Bu işlem her kelimeyi ayrı bir satırda saklar. Böylece okuma işlemi yapmamız gerekiğinde işimiz kolaylaşmış olur. Dosyaya okuma izni ile erişilmişse

```
print(line.strip())
```

komutu, dosyadaki her bir satırı okur ve yazdırır. Dosya nesnesinin ok komutunu kullanarak bir dosyanın tüm içeriğini tek bir komutla bir dizi içerişine aktarabiliriz.



```
icerik = f.read()
in = open("bilgiTerimleri.txt", "r")
```

“open” metodu, bir dosyayı okuma ve yazma için açar; böylece programın dosya ile etkileşimi sağlanır. Dosya ile işi bittikten sonra programın dosyayı uygun biçimde kapatması gereklidir. Daha önce kapatılmamış bir dosyaya tekrar erişim sırasında sorun yaşanabilir. Bu nedenle, açılan her dosya işlemler bittikten sonra mutlaka “close” metodu kullanılarak kapatılmalıdır.

#### 9.4.1. Dosya Okuma ve Yazma İşlemleri

```
f = open("veriler.dat") # f adında dosya nesnesi
for line in f: # Her satırı metin olarak oku
    print(line.strip()) # Sondaki yeni satır karakterini sil
f.close() # Dosyayı kapat
with open("veriler.dat") as f: # f adında dosya nesnesi
    for line in f: # Her satırı metin olarak oku
        print(line.strip()) # Sondaki yeni satır karakterini sil
    f.close() # Dosyayı kapat
```

#### 9.4.2. Dosya Okuma ve Yazma İşlemlerinde with/as kullanımı

“with/as” ifadesi ile nesnelere ilişkin işlemler yürütülür.

- “object-creation” ifadesi bir nesne oluşturur ve döndürür. Bu işlem başarısız olursa söz dizimi çalışmaya devam etmez.
- “as” ifadesi yaratılan nesne ile değişkenin bağlantısını kurar.
- “object” ile yaratılan nesne ilişkilendirilir.
- “block” ifadesi kapsamında farklı kodlar bulunur.

**with**      nesne oluşturma    **as**      nesne      :      kodlar

“with/as” ifadesi “TextIOWrapper” gibi sınıflarla çalışabilir; böylece başlama ve bitiş için belirli bir protokol sağlanmış olur. Sadece belirli sınıflar bu işlemi yürütmek için uygunudur. Bu sınıfların ilk değer atama için “\_\_enter\_\_” ve sonlandırma için “\_\_exit\_\_” metotları vardır.

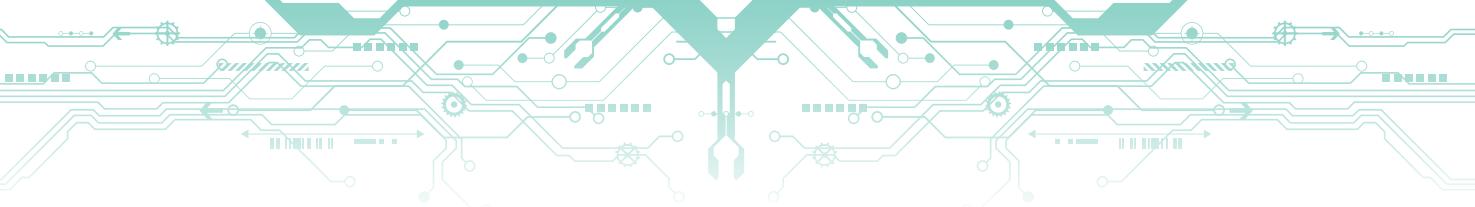
##### Sayıları Kaydetme Örneği

```
# Python "da dosyaya yazma ve dosyadan okuma programı
def Listeleme(dosyaAdi):
    # Parametre olarak gelen dosyada bulunan kayıtları listeleye.
    # Okumak için dosyanın açılması
    with open(dosyaAdi) as f: # f adında bir dosya nesnesi oluşturuldu
        for satir in f: # Satır satır okuma işlemi için döngü kuruldu
            print(int(satir)) # int veri türüne dönüştürme ve ekrana yazdırma
def Kaydet(dosyaAdi):
```

```
# Parametre olarak gelen dosyada bulunan kayıtları kaydetme.
with open(dosyaAdi, "w") as f: # f adında yazma modunda bir dosya
nesnesi oluşturuldu
    sayı = 0
    while sayı != 999: # Kullanıcı 999 giresiye kadar dönen döngü
kuruluyor
        sayı = int(input("Lütfen sayı giriniz..(Çıkış için 999):"))
        if sayı != 999:
            f.write(str(sayı) + "\n") # String veri türüne dönüşüm ve
dosyaya kayıt
        else:
            break # Döngüden çıkış
def main():
# Ana Program başlangıcı. Menülü seçim işlemleri ve çıkış.
    kontrol = False
    while not kontrol:
        secim = input("K)aydet L)isteleme S)onlandır: ")
        if secim == "K" or secim == "k":
            Kaydet(input("Kayıt Edilecek Dosya Adı Girin:"))
        elif secim == "L" or secim == "l":
            Listeleme(input("Kayıtların Okunacağı Dosya Adını Girin<:"))
        elif secim == "S" or secim == "s":
            kontrol = True
main()
```

## Ekran Çıktısı

```
K)aydet L)isteleme S)onlandır: k
Kayıt Edilecek Dosya Adı Girin:sayılar
Lütfen sayı giriniz..(Çıkış için 999):1
Lütfen sayı giriniz..(Çıkış için 999):54
Lütfen sayı giriniz..(Çıkış için 999):66
Lütfen sayı giriniz..(Çıkış için 999):44
Lütfen sayı giriniz..(Çıkış için 999):23
Lütfen sayı giriniz..(Çıkış için 999):999
K)aydet L)isteleme S)onlandır: l
Kayıtların Okunacağı Dosya Adını Girin<:sayılar
1
54
66
44
23
K)aydet L)isteleme S)onlandır: s
```



“io” modülünden seçilen Python dosya sınıfını “TextIOWrapper” olarak görüyoruz. Bu sınıfta işlem gören dosyalar metin türündedir. Metin dosyaları, karakter veri saklar ve basit bir editörle kolayca oluşturulup düzenlenebilir. Python kapsamında dizi ve dosya nesnelerini bir arada kullanarak güçlü dosya işleme programları yazabiliriz. Bir dosyayı açıp, içeriğini okuyup tamamen değiştirip başka bir dosyaya yazabiliriz.

```
from convertupper import capitalize  
capitalize("declaration.txt")
```

#### 9.4.3. TextIOWrapper Yöntemleri

open	Metin dosyasını açma komutu
read	Metin dosyasına bir dizi okuyan komut
write	Metin dosyasına bir dizi yazan komut
close	Dosyayı kapatma komutu. Dosyaya yazarken close yöntemi kullanılırsa dosyaya gönderilen tüm verilerin dosyaya kaydedilmesi sağlanır.

#### Örnek

Nesneler metodların yanı sıra veri de içerir. “TextIOWrapper” nesneleri tam sayı, dizi ve mantıksal ifadeler saklayabilir.

```
>>> f = open("temp.dat", "w")  
>>> f.name  
"temp.dat"  
>>> f._CHUNK_SIZE  
8192  
>>> f.mode  
"w"  
>>> f.encoding  
"cp1252"  
>>> f.line_buffering  
False
```

name”, “\_CHUNK\_SIZE”, “encoding” ve “line\_buffering” ifadeleri “f” nesnesinin örnek değişkenleridir. Bu değişkenlerin önceden kullandıklarımızdan farkı “.” ile belirli bir nesne ile ilişkilendirilmiş olmalarıdır. Bu isimler metod değil veriyi ifade ettiği için sonunda ayrı kullanılmamaktadır. “f” ve “g” isimli iki farklı nesnemiz varsa bu nesneler birbirinden farklı davranışlı olabilir.

```
x = 2
```

ifadesinde x değişkenine “2” değeri atanırken

```
obj.x = 2
```

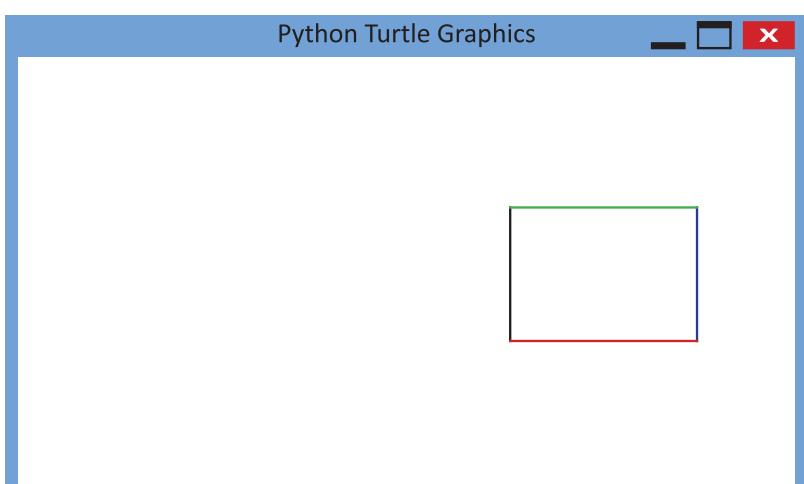
ifadesinde “obj” isimli nesnenin örnek x değişkenine atama yapmaktadır.

## 9.5. Turtle Grafik Nesneleri

Grafik çizme işleminde “Turtle” nesnesinin, çizim işlemi için kalemi modellediğini biliyoruz.

Kendi “Turtle” nesnelerimizi tasarlayarak kullanabiliriz. Birden fazla kalem kullanmamız gerekirse bu, çok kullanışlı olacaktır.

```
# Pencere içerisinde diktörtgen bir kutu çizimi
from turtle import Turtle, mainloop
t = Turtle() # Adı t olan yeni bir Turtle() nesnesi oluşturuluyor
t.pencolor("red") # t nesnesinin çizim kalemi rengi kırmızı olarak
ayarlanıyor
t.forward(200) # t nesnesinin kalemi 200 birim ileri oynatılıyor ve
dikdörtgenin alt kenarı çiziliyor
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("blue") # t nesnesinin rengi mavi olarak değiştiriliyor
t.forward(150) # Sağ kenarın çizilmesi için Turtle 150 birim yukarı
ilerletiliyor
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("green") # Turtle nesnesinin rengi yeşil olarak değiştiriliyor
t.forward(200) # Dikdörtgenin üst kenarının çizilmesi için Turtle
200 birim ilerletiliyor.
t.left(90) # Turtle 90 derece sola döndürülüyor
t.pencolor("black") # Turtle nesnesinin rengi siyah olarak değiştiriliyor
t.forward(150) # Son kenar olan sol kenarın çizilmesi için Turtle
150 birim ilerletiliyor.
t.hideturtle() # Turtle gizleniyor
mainloop() # Kullanıcıdan veri bekleniyor
```



### 9.5.1. Turtle Grafik & tkinter Nesneleri

“tkinter” modülü Tk araç kiti ile grafiksel kullanıcı arayüzleri oluşturma sürecinde farklı sınıflar sunar. Tk Microsoft Windows, Apple Mac, and Linux işletim sistemleri ile kullanılabilir. “tkinter” modülü “Turtle” modülünden daha kapsamlı ve karmaşıktır. Aslında “Turtle” modülü “tkinter” modülünün sunduğu bileşenler üzerine inşa edilmiştir.

```
from tkinter import Tk, Button  
  
sayac = 0 # Tıklama sayısının hafızada tutulacağı değişken  
tanımlanıyor  
  
def update():  
  
    # Grafikte bulunan butona tıklandığında sayaç artırma işlemi  
    global sayac, b  
    sayac += 1  
    b.config(text="Tıklama Sayısı = " + str(sayac))  
    print("Güncelleniyor")  
  
root = Tk()  
b = Button(root)  
b.configure(background="yellow", text="Tıklama Sayısı = 0",  
command=update) # Ekrana buton nesnesi oluşturuluyor  
  
b.pack()  
root.mainloop()
```



### 9.5.2. Buton Test Etme

**Tk:** Bu sınıf bir grafik pencereyi temsil eder.

```
root = Tk()
```

İfadeleri, “root” isimli bir nesne oluşturur. Bu nesne, uygulamanın ana grafik penceresini ifade eder.

```
root.mainloop()
```

İfadeleri, pencerenin yanı sıra grafik programını başlatmak için “mainloop” metodunu çağırır. Bu metot, hareket sürecini başlatarak kullanıcının görsel döngü almasına olanak sağlar.



**Button:** Kullanıcının basabileceği bir grafik butonu temsil eden sınıfıtır. Bir buton, araç kitinin sağladığı pek çok grafikten biridir.

```
b = Button(root)
```

ifadesi, "b" isimli bir buton nesnesi oluşturur ve nesneyi "root" pencere ile ilişkilendirir.

```
b.configure(background="yellow", text="Tıklama Sayısı = 0", command=update)
```

ifadesi, butonu sarı arka plan, metin rengi ve basıldığından gerçekleşeceğin işlem açısından konfigüre eder.

"Button" nesnesi ile yazı tipi ve rengi, sola ya da sağa hizalı yazma, yatay ya da dikey konumlama ve çerçeve kalınlığı gibi seçenekleri de değiştirebiliriz.

```
b.pack()
```

ifadesi, butonun pencerede görünür iyi bir noktada yer olmasını sağlar.

```
b = Button(root, background ="yellow", text="Tıklama Sayısı = 0", command=update)
```

ifadesinde gerekli olan pencere parametresi ile birlikte (root) 3 parametrenin daha gönderildiğini görüyoruz. Bu örnekte "sayac" değişkeni global olarak tanımlanmalıdır çünkü fonksiyonda tekrar atama yapılmaktadır. Ayrıca, "b" değişkeni de global olmalıdır.

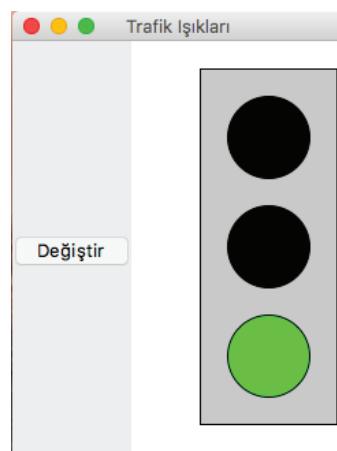
### Trafik Işıkları Örneği

```
# "Değiştir" butonuna basıldığında trafik ışıklarını sırasıyla yakan program
from tkinter import Tk, Canvas
from tkinter.ttk import Button, Frame
def ButonaBasildiginda(): # Her bir tıklamada ışıkların sırasıyla yanması
    global renk
    if renk == "red":
        renk = "green"
        canvas.itemconfigure(kirmiziLamba, fill="black") # Kırmızı ışık kapatılıyor
        canvas.itemconfigure(yesilLamba, fill="green") # Yeşil ışık yanıyor
    elif renk == "green":
        renk = "yellow"
        canvas.itemconfigure(yesilLamba, fill="black") # Yeşil ışık kapatılıyor
        canvas.itemconfigure(sariLamba, fill="yellow") # Sarı ışık yanıyor
    elif renk == "yellow":
        renk = "red"
        canvas.itemconfigure(sariLamba, fill="black") # Sarı ışık kapatılıyor
```

```

        canvas.itemconfigure(kirmiziLamba, fill="red") # Kırmızı
ışık yanıyor
# Kullanılacak değişkenlerin tanımlanması
renk = "red" # Açık olarak gelecek ilk trafik ışıığı
renk root = Tk() # Ana Pencere'nin oluşturulması
root.title("Trafik Işıkları") # Pencere başlığı
frame = Frame(root) # Nesnelerin birlikte tutulması için grafiksel
bileşen ( widget ) oluşturuluyor
frame.pack() # Pencere içerisinde frame yerleştiriliyor
# Grafiksel bileşenlerin yerleştirileceği çizim alanı ( canvas )
oluşturuluyor
canvas = Canvas(frame, width=150, height=300)
# frame'in içerisinde çizim arayüzü oluşturuluyor
# Trafik ışıkları oluşturuluyor, zemin rengi gri olarak ayarlanıyor
canvas.create_rectangle(50, 20, 150, 280, fill="gray")
# Kırmızı Lamba
kirmiziLamba = canvas.create_oval(70, 40, 130, 100, fill="red")
# Sarı Lamba
sariLamba = canvas.create_oval(70, 120, 130, 180, fill="black")
# Yeşil Lamba
yesilLamba = canvas.create_oval(70, 200, 130, 260, fill="black")
# Grafiksel butonun oluşturulması ve işlevsellik kazandırılması
Butona Basıldığında # Fare ile tıklama yapıldığında fonksiyon
çağrılıyor
button = Button(frame, text="Değiştir", command=ButonaBasildiginda)
# Oluşturulan katmanın ilk satır ve ilk sütünuna buton,
# birinci satır ikinci sütunununa da çizim alanı yerleştiriliyor.
button.grid(row=0, column=0)
canvas.grid(row=0, column=1)
# Grafiksel arayüz oluşturuluyor
root.mainloop()

```





Mevcut kodlama içinde tkinter paketinden ve tkinter.ttk paketinden ayrı sınıflar kullanılmaktadır. "Tk" sınıfı bir grafik penceresi sunar.

```
root = Tk()
```

komutu "root" isimli bir Tk nesnesi oluşturur ve "root" nesnesi uygulamanın ana grafik penceresi ile bağlantı kurar.

```
root.title("Trafik Işıkları")
```

pencerenin başlık çubuğundaki metni belirler.

```
root.mainloop()
```

"mainloop" yöntemini çağırarak grafik programı başlatır.

"Frame" sınıfı diğer grafik nesnelerini barındırmak için görünmeyen bir depo (widget) oluşturur.

```
frame = Frame(root)
```

komutu bir çerçeve nesnesi oluşturarak nesneyi grafik penceresi ile ilişkilendirir.

```
frame.pack()
```

komutu ise grafik penceresinin tamamını doldurur. "widget" görsel programlamada bir kütüphanedeki grafik bileşenlere verilen isimdir.

"Canvas" sınıfı grafik pencere içinde bir çizim alanı oluşturur.

```
canvas = Canvas(frame, width=300, height=300)
```

komutu çerçeveyen deposunda yer alan "canvas" isimli bir nesne oluşturur. "canvas" nesnelerinin boyutları 300 px x 300 px olarak yükseklik ve genişlik anahtar kelimeleri ile belirlenir. Canvas üzerindeki koordinat sisteminin merkezi (0;0), pencerenin çizim alanının sol üst köşesinde yer alır ve y ekseni yukarı değil aşağı doğru şekillenir. Diğer bir ifade ile, soldan sağa doğru x değeri arttıkça, y değeri yukarıdan aşağıya doğru artmaktadır.

## 9.6. Nesne Değişkenliği ve Örtüşme

```
from fractions import Fraction
# Bazı kesir tanımlamaları yapılıyor
f1 = Fraction(1, 2)
f2 = Fraction(1, 2)
f3 = f1
# İlişkilendirmeler
print("f1 =", f1)
print("f2 =", f2)
print("f3 =", f3)
# Pay ve paydalar ayrı ayrı inceleniyor
print("f1--> Pay, Payda:", f1.numerator, f1.denominator)
print("f2--> Pay, Payda:", f2.numerator, f2.denominator)
print("f3--> Pay, Payda:", f3.numerator, f3.denominator)
# Kesirler karşılaştırılıyor
```

```
print("f1 == f2?", f1 == f2)
print("f1 == f3?", f1 == f3)
print("f1 ile f2 aynı değer mi?", f1 is f2)
print("f1 ile f3 aynı değer mi?", f1 is f3)
```

## Ekran Çıktısı

```
f1 = 1/2
f2 = 1/2
f3 = 1/2
f1--> Pay, Payda: 1 2
f2--> Pay, Payda: 1 2
f3--> Pay, Payda: 1 2
f1 == f2 ? True
f1 == f3 ? True
f1 aynı f2 değer mi? False
f1 aynı f3 değer mi? True
```

```
f1 = Fraction(1, 2)
```

Komutu “Fraction (kesir)” sınıfını çağırarak yeni bir kesir nesnesi oluşturur. Bu komut ile paya 1 ve paydaya 2 değeri atanır. “f1” değişkeni de bu yeni kesir nesnesine atanır.

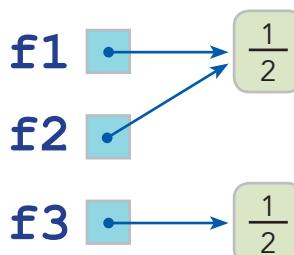
```
fraction object. The statement
```

```
f2 = Fraction(1, 2)
```

komutu aynı şekilde davranarak “f2” nesnesi de bu kesir nesnesine atanır.

```
f3 = f1
```

komutu ile “f3” değişkeni diğerleri ile aynı kesir nesnesine atanır. Ancak, söz dizimi “Fraction” sınıf yapısını içerdiginden yeni bir kesir nesnesi oluşturmaz. Bu aşamada iyi kesir nesnemiz ve buna bağlı 3 değişkenimiz vardır. Aşağıda nesneler ve kesir arasındaki ilişki görülmektedir.



Gördüğü gibi f1 ve f3 aynı nesneyi işaret etmektedir. Bu durum “örtüşme” olarak ifade edilir yani f1 ile f3 örtüşmektedir (f1 aliases f3). Kesirlerin mantıksal olarak eşitliğini karşılaştırmak için kesir nesnesine ait \_\_eq\_\_ yöntemi “==” operatörü ile kullanılır.

```
print("f1 == f2?", f1 == f2)
print("f1 == f3?", f1 == f3)
```



komutlarının üçü de mantıksal olarak birbirine eş değer değişkenler olduğunu açığa çıkarmaktadır. “`__eq__`” yöntemi (`==`) ile kesirlerin pay ve payda değerleri eşitlik konusunda karar verebilmek için ayrı ayrı karşılaştırılmaktadır.

Bazen mantıksal eşitlik yeterli olmaz ve her iki değişkenin de aynı nesneye ait olup olmadığını bilmek isteriz.

```
print("f1 ile f2 aynı değer mi?", f1 is f2)
print("f1 ile f3 aynı değer mi?", f1 is f3)
```

komutları `f1` ve `f2`'nin iki farklı nesneyi, `f1` ve `f3`'ün ise aynı nesneye işaret ettiğini gösterir. Bu durumda `f1` ve `f3` örtüşmektedir.

Python “`id`” isimli bir fonksiyonu vardır ve bu fonksiyon, her bir nesne için özel oluşturulmuş bir tam sayı değeri döndürür (Çoğu Python uygulamalarında bu değer programın nesneyi yerleştirdiği hafızanın başlangıç adresidir.). `a` ve `b` nesne ise bu nesnelerin eş değerliği aşağıdaki biçimde sorgulanır.

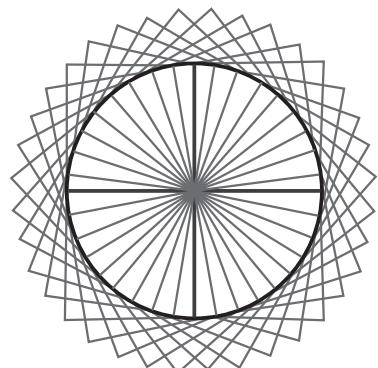
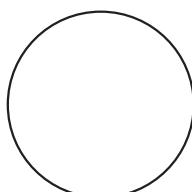
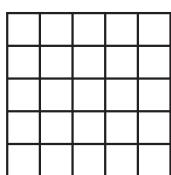
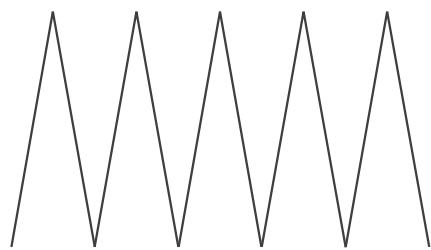
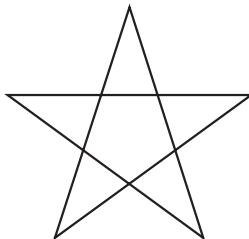
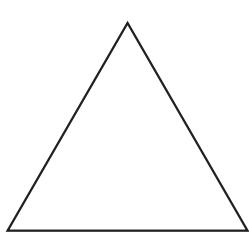
```
id(a) == id(b)
```

Nesnelerin örtüşüp örtüşmediğine bakılırken değişkenlerin türü önemli değildir. Tam sayı değeri olarak 3 her zaman 3'tür. Dizi olarak “Fred” kelimesi “Free” olarak değişmez. Kesir sınıfı örnekleri de bu şekilde değişkendir. Değişken nesneler için örtüşme önemli bir konu olabilir. Python'un Turtle grafik kütüphanesindeki nesneler değişkendir. Programcılar kaplumbağa nesnesini hareket ettirebilir ve yönünü ve çizim rengini değiştirebilirler. Her bir işlem kaplumbağanın durumunu değiştirir ve grafik pencere içinde kaplumbağanın çizim sürecini etkiler.



### Düşünelim/Deneyelim

Python Turtle grafikleri kullanarak aşağıdaki şekilleri oluşturunuz.

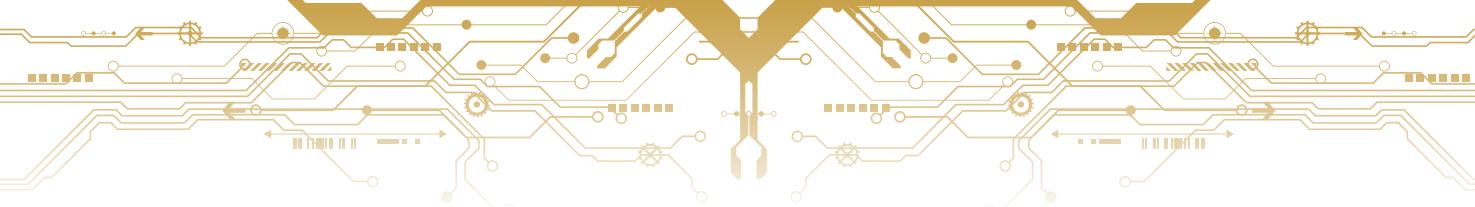


## 10. LİSTELER



Bu bölümde;

- ✓ Listelerin kullanım amacı konusunda bilgi sahibi olacak,
- ✓ Listeleri içeren programları yazabilecek,
- ✓ Listeler üzerinde işlemler yapabilecek,
- ✓ Listeler ve fonksiyonları bir arada kullanan programlar geliştirebilecek,
- ✓ Farklı boyutları olan listeleri içeren programları yazabileceksiniz.



## 10.1. Liste Kavramı

Bu konuya kadar kullanılan değişkenler, bir değeri temsil ediyordu. Ancak listeler ile daha fazla değeri bir değişkene aktarmak mümkün. Aşağıdaki örnekte beş değer için beş değişken kullanılmıştır.

```
def main():
    print("Lütfen 5 adet sayı giriniz: ")
    n1=float(input("1. sayı: "))
    n2=float(input("2. sayı: "))
    n3=float(input("3. sayı: "))
    n4=float(input("4. sayı: "))
    n5=float(input("5. sayı: "))
    print("Girdiğiniz Sayılar : ",n1,n2,n3,n4,n5,sep=" - ")
    print("Ortalama : ",(n1+n2+n3+n4+n5)/5)
main()
```

### Ekrان Çıktısı

```
Lütfen 5 adet sayı giriniz:
1. sayı: 23
2. sayı: 35
3. sayı: 55
4. sayı: 67
5. sayı: 98
Girdiğiniz Sayılar : - 23.0 - 35.0 - 55.0 - 67.0 - 98.0
Ortalama : 55.6
```

5 değil de 25 değere ihtiyaç duyulsa idi bir önceki soruya 20 değer daha eklemek gerekecekti. Oysaki değişken kullanımı konusunda alternatif yaklaşımlar ile bu durum daha pratik bir şekilde çözülebilir.

```
def main():
    toplam=0.0
    girilecekSayıAdeti=5
    print("Lütfen ",girilecekSayıAdeti, " adet sayı giriniz: ")
    for i in range(0, girilecekSayıAdeti):
        sayı=float(input("Lütfen " +str(i+1)+ ". sayıyı giriniz: "))
        toplam+=sayı
    print("Ortalama : ",toplam/girilecekSayıAdeti)
main()
```

## Ekran Çıktısı

```
Lütfen 5 adet sayı giriniz:  
Lütfen 1. sayıyı giriniz: 12  
Lütfen 2. sayıyı giriniz: 23  
Lütfen 3. sayıyı giriniz: 34  
Lütfen 4. sayıyı giriniz: 65  
Lütfen 5. sayıyı giriniz: 76  
Ortalama : 42.0
```

### 10.1.1. Listeleri Kullanmak

Listeler, belirli bir sırada çoklu değer tutmak için kullanılır. Listeler bir anlamda string veri türüne benzemektedir. Ancak, string veri türünün aksine herhangi bir liste içerisinde herhangi bir Python nesnesi korunabilir. Bir liste, tüm değişkenler gibi yerel ve global olarak kullanılabilir. Listenin kullanılmadan önce tanımlanması gerekmektedir.

Liste kullanırken değerler köşeli ayraç içerisinde virgül ile ayrılarak yazılır.

```
lst=[2,-3,0,4,-1]  
A=[]
```

#### Örnek

```
lst=[2,-3,0,4,-1]  
print([2,-3,0,4,-1])  
print(lst)
```

## Ekran Çıktısı

```
[2,-3,0,4,-1]  
[2,-3,0,4,-1]
```

#### Örnek

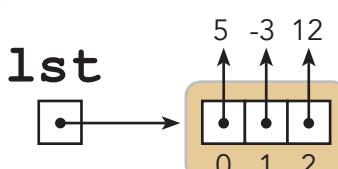
```
lst=[2,-3,0,4,-1]  
lst[0]=5  
print(lst[1])  
lst[4]=12  
print(lst)  
print([10,20,30][1])
```

## Ekran Çıktısı

```
-3  
[5,-3,0,4,12]  
20
```

Listelerde,

- Köşeli ayraç içerisindeki sayıya indeks denmektedir.
- `lst[0]` değeri ilk değerdir.
- `lst[1]` değeri serideki ikinci değerdir.
- 0, başlangıç indeksi olduğu için n elemanlı bir serinin indeks değeri  $n-1$  olur.



- `A[-1]` değeri, `A` dizisindeki son elemanı ifade eder.
- `A[-2]` ise sondan bir önceki değerini ifade eder. Bu şekilde devam eder.

## Örnek

```
def main():  
    data=[10,20,30,40,50,60]  
    print(data[-1])  
    print(data[-2])  
    print(data[-3])  
    print(data[-4])  
    print(data[-5])  
    print(data[-6])  
main() # Ana program çalıştırılıyor
```

## Ekran Çıktısı

```
60  
50  
40  
30  
20  
10
```



### 10.1.2. Listelerin Farklı Kullanımları

```
karisikListe=[24.2,4,"Python","Bilisim",19,-0.03,"kelime"]
```

Örnekte görüldüğü gibi bir listede tam sayı, ondalık sayı, string ve hatta fonksiyonlar olabilir. Liste içerisinde liste bile yer alabilir.

```
col=[23,[9.3,11.2,99.0],[23],[],4,[0,0]]
```

```
print(col)
```

Köşeli ayraç içerisinde ilgili değerler ile işlem yapılabilir.

```
nums=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10 , 11 , 12, 13, 14, 15]
```

```
print(nums[3])
```

```
nums[2]=(nums[0]+nums[9])/2;
```

```
nums[1], nums[4]=sqrt(x), x+2*y
```

String içerisindeki değerlere de köşeli ayraçlar ile erişilebilir.

```
>>>s= "ABCDEFGHI"
```

```
>>>s[0]
```

```
"A"
```

```
>>>s[1]
```

```
"B"
```

Köşeli ayraç içerisindeki değer bir tam sayı olmalıdır.

```
a[x]
```

Köşeli ayraç içerisinde aritmetik bir işlem yapılabilir.

```
a[x+3]
```

Geri dönen değer de kullanılabilir. Bunlar da tam sayı olmalıdır.

```
a[max(x,y)]
```

### 10.1.3. Değerler Arası Gezinme

Bir listedeki değerlerin her birini ziyaret etmeye **gezinme** denmektedir. Bir liste aslında tekrarlanan bir nesne gibidir ve bu nedenle elemanları arasında gezinmek için for döngüsünü kullanabiliriz.

```
karisikListe=[24.2,4,"Python","Bilisim",19,-0.03,"kelime"]
```

```
for item in collection:  
    print(item) # Her bir eleman yazdırılıyor
```

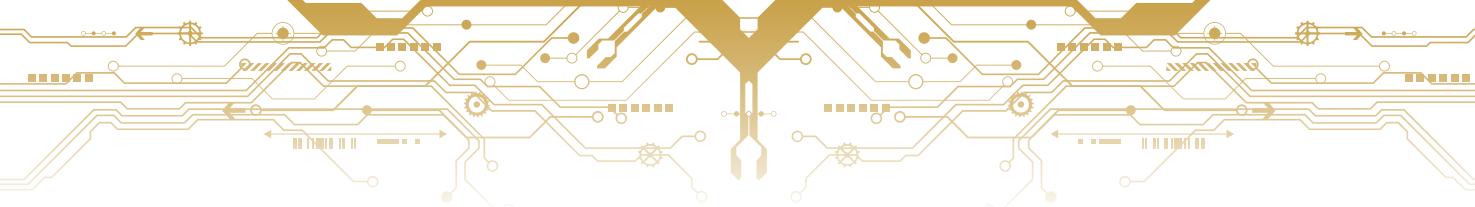
#### Örnek

```
print(len([2,4,6,8]))
```

```
a=[10,20,30]
```

```
print(len(a))
```

```
>>>4
```



```
>>>3 değerlerini ekrana yazar.
```

### Örnek

```
nums=[2,4,6,8]
for i in range(len(nums)-1,-1,-1):
print(nums[i])
>>>8
>>> 6
>>> 4
>>> 2 değerlerini ekrana yazar.
```

### Örnek

```
nums=[2,4,6,8]
for i in reversed(nums):
print(nums[i])
```

### Örnek

```
def my_reversed(lst):
for i in range(len(lst)-1,-1,-1):
yield(lst[i])
```

Len(lst)-1 listedeki son değerin indeks değeridir. İkinci argüman ise döngüyü sonlandırmak içindir. Son -1 değeri ise geri saymak içindir.

## 10.2. Liste Oluşturmak

Python, liste oluşturmak için birçok yöntem sunmaktadır. Birleştirme yöntemi ile var olan iki liste tek bir liste yapabilir. Bunun için + operatörü kullanılır.

```
>>> a = [2, 4, 6, 8]
>>> a
[2, 4, 6, 8]
>>> a + [1, 3, 5]
[2, 4, 6, 8, 1, 3, 5]
>>> a
[2, 4, 6, 8]
>>> a = a + [1, 3, 5]
>>> a
[2, 4, 6, 8, 1, 3, 5]
>>> a += [10]
```

```
>>> a  
[2, 4, 6, 8, 1, 3, 5, 10]  
>>> a += 20  
Traceback (most recent call last):  
  File "<pyshell#14>", line 1, in <module>  
    a += 20  
TypeError: "int" object is not iterable
```

a = [2, 4, 6, 8] ifadesi ile a listesine değer atanır.

a + [1, 3, 5] ifadesi ile değerler [2, 4, 6, 8, 1, 3, 5] şeklini alır. Ancak atama gerçekleşmez.

a = a + [1, 3, 5] ifadesi ile değerler değişir.

a += [10] ifadesi ile liste [2, 4, 6, 8, 1, 3, 5, 10] şeklini alır.

a += 20 ifadesi bir liste olmadığı ve tam sayı olduğu için işlem yapmaz.

Bir değişkenin değeri listeye eklenmek istenirse köşeli ayraç kullanılmalıdır.

```
>>> x = 2  
>>> a = [0, 1]  
>>> a += [x]  
>>> a  
[0, 1, 2]
```

## Örnek

```
# Kullanıcıdan alınan sayılardan liste oluşturan program  
def ListeOlustur():  
    sonuc = []  
    girilenSayi = 0  
    while girilenSayi >= 0:  
        girilenSayi= int(input("Sayı giriniz ( Çıkış için -1): "))  
        if girilenSayi>= 0:  
            sonuc += [girilenSayi]  
    return sonuc  
def main():  
    sutun = ListeOlustur()  
    print(sutun)  
main()
```

## Ekran Çıktısı

```
Sayı giriniz ( Çıkış için -1): 2  
Sayı giriniz ( Çıkış için -1): 3  
Sayı giriniz ( Çıkış için -1): 4  
Sayı giriniz ( Çıkış için -1): 55
```

```
Sayı giriniz ( Çıkış için -1): 66
Sayı giriniz ( Çıkış için -1): 77
Sayı giriniz ( Çıkış için -1): -1
[2, 3, 4, 55, 66, 77]
```

### 10.3. List Fonksiyonu ile Tam Sayı Liste Yapmak

Liste oluşturulmak istenirken bazen kullanıcılarından veri almak yerine liste elemanları otomatik olarak oluşturulmak istenebilir. Bu durumlarda list() fonksiyonuyla beraber range() fonksiyonu kullanılabilir. list() fonksiyonu otomatik olarak belirlenen değerlerden liste oluştururken range() fonksiyonu ise başlangıç ve bitiş değerleri arasında artım miktarına göre değerler oluşturur.

```
def main():
    a = list(range(0, 10))
    print(a)
    a = list(range(10, -1, -1))
    print(a)
    a = list(range(0, 100, 10))
    print(a)
    a = list(range(-5, 6))
    print(a)
main()
```

#### Ekran Çıktısı

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

### 20 ile 50 Arasındaki Asal Sayılardan Liste Yapmak

```
from math import sqrt
def AsalKontrol(n):
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    geciciDeger = 3
```



```
kok = sqrt(n)
while geciciDeger <= kok:
    if n % geciciDeger == 0:
        return False
    geciciDeger += 2
return True

def AsalAralikBelirleme(ilk, son):
    for deger in range(ilk, son + 1):
        if AsalKontrol(deger):
            yield deger

def main():
    asal = list(AsalAralikBelirleme(20, 50))
    print(asal)

if __name__ == "__main__":
    main() # Programı çalıştır
```

### Ekran Çıktısı

```
[23, 29, 31, 37, 41, 43, 47]
```

## 10.4. \* Operatörü ile Liste Oluşturma

Daha önceki konularımızda \* operatörü matematikteki çarpma işlemini yerine getirmek için kullanılıyordur. Liste oluşturulurken \* operatörü solundaki liste elemanını sağındaki adet kadar çoğaltır.

\* Operatörü string'ler ile benzer sonuç verir.

```
>>> 'abc' * 3
'abcabcabc'
```

```
def main():
    a = [0] * 10
    print(a)
    a = [3.4] * 5
    print(a)
    a = 3 * ["ABC"]
    print(a)
    a = 4 * [10, 20, 30]
    print(a)
    n = 3 # çarpım değişkeni tanımlanıyor
```

```
a = n * ["abc", 22, 8.7]
print(a)
main()
```

### Ekran Çıktısı

```
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[3.4, 3.4, 3.4, 3.4, 3.4]
["ABC", "ABC", "ABC"]
[10, 20, 30, 10, 20, 30, 10, 20, 30]
["abc", 22, 8.7, "abc", 22, 8.7, "abc", 22, 8.7]
```

### Örnek

Listedeki değerlerin ortalamasını bulmak.

```
def main():
    toplam = 0.0
    girilecekSayiAdeti = 5
    sayilar = []
    print("Lütfen", girilecekSayiAdeti, " adet sayı giriniz :")
    for i in range(0, girilecekSayiAdeti):
        sayı = float(input(str(i+1) + " . sayıyı giriniz : "))
        sayilar += [sayı]
        toplam += sayı

    for sayı in sayilar:
        print(end="Girilen Sayılar: ")
        print(sayı, end="")
        print() # Alt satır geçiş
    print("Ortalama :", toplam/girilecekSayiAdeti)
main()
```

### Ekran Çıktısı

```
Lütfen 5 adet sayı giriniz :
1 . sayıyı giriniz : 12
2 . sayıyı giriniz : 23
3 . sayıyı giriniz : 43
4 . sayıyı giriniz : 54
```



```
Girilen Sayılar: 12.0
Girilen Sayılar: 23.0
Girilen Sayılar: 43.0
Girilen Sayılar: 54.0
Girilen Sayılar: 55.0
Ortalama : 37.4
```

## 10.5. Liste Üyeliği

in operatörü ile listenin elemanları üzerinde işlem yapılabilir. True veya false değeri döndürür.

```
liste = list(range(0, 21, 2))
for i in range(-2, 23):
    if i in liste:
        print(i," eleman, ", liste," listesinin bir üyesidir.")
    if i not in liste:
        print(i," eleman ", liste, "listesinin bir üyesi değildir.")
```

### Ekran Çıktısı

```
-2 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
-1 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
0 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
1 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
2 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
3 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
4 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
5 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
6 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
7 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
8 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
9 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
10 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
11 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
12 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
13 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
14 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
15 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.
16 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.
```

17 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.  
18 eleman, [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesidir.  
19 eleman [[0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.  
20 eleman, [0,2,4,6,8,10,12,14,16,18,20]] listesinin bir üyesidir.  
21 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.  
22 eleman [0,2,4,6,8,10,12,14,16,18,20] listesinin bir üyesi değildir.

## 10.6. Listeye Değer Atama ve Eşitleme

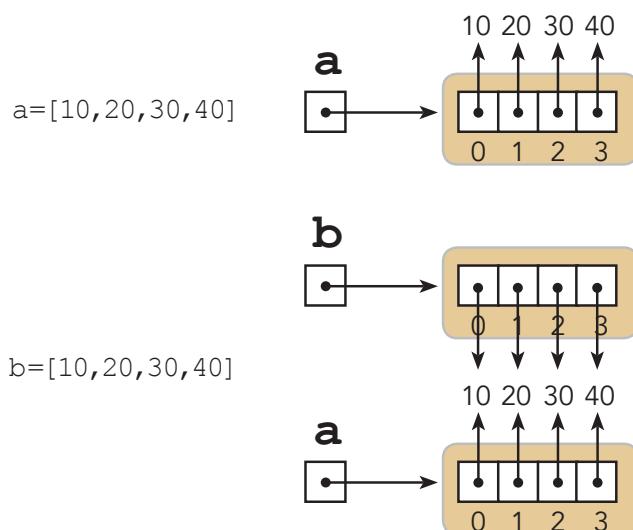
### Örnek

```
a=[10,20,30,40]
b=[10,20,30,40]
print("a =",a)
print("b =",b)
b[2]=35
print("a =",a)
print("b =",b)
```

### Ekran Çıktısı

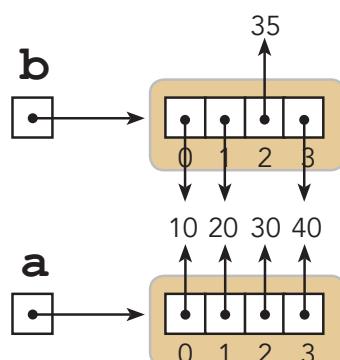
```
a=[10,20,30,40]
b=[10,20,30,40]
a=[10,20,30,40]
b=[10,20,35,40]
```

Programın çalışma süreci aşağıda verilen resimdeki gibidir.





$b[2] = 35$



### Örnek

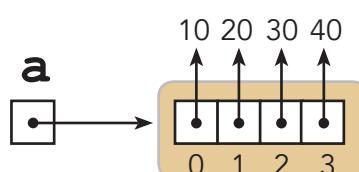
```
a=[10,20,30,40]
b=a
print("a =",a)
print("b =",b)
b[2]=35
print("a =",a)
print("b =",b)
```

### Ekran Çıktısı

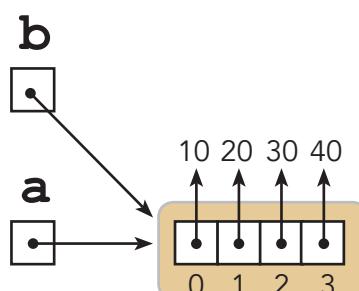
```
a=[10,20,30,40]
b=[10,20,30,40]
a=[10,20,30,40]
b=[10,20,35,40]
```

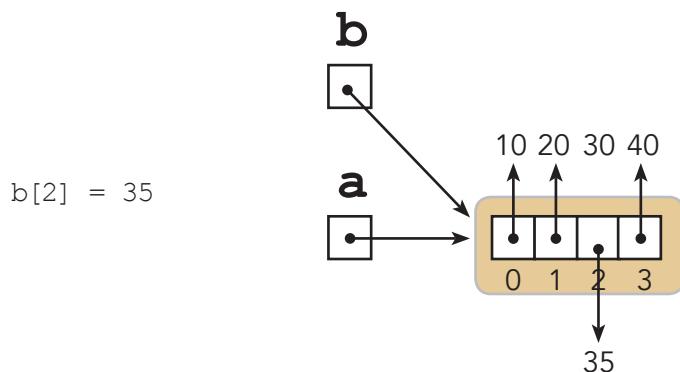
Programın çalışma süreci aşağıda verilen resimdeki gibidir.

$a=[10,20,30,40]$



$b=a$





### Örnek

```

a=[10,20,30,40]
b=[10,20,30,40]
print(a,"değeri ",b," değerine eşit mi?",sep="",end=" ")
print(a==b)
print(a,"değeri ",b," değeri ile aynı mı?",sep="",end=" ")
print(a is b)
c=[100,200,300,400]
d=c
print(c,"değeri ",d," değerine eşit mi?",sep="",end=" ")
print(c==d)
print(c,"değeri ",d," değeri ile aynı mı?",sep="",end=" ")
print(c is d)
    
```

### Ekran Çıktısı

```

[10, 20, 30, 40]değeri [10, 20, 30, 40] değerine eşit mi? True
[10, 20, 30, 40]değeri [10, 20, 30, 40] değeri ile aynı mı? False
[100, 200, 300, 400]değeri [100, 200, 300, 400] değerine eşit mi? True
[100, 200, 300, 400]değeri [100, 200, 300, 400] değeri ile aynı mı?
True
    
```

İki değişkenin değerlerinin aynı olması (`==` ile karşılaştırma) ile değişkenlerin aynı olması (`is` ile karşılaştırma), farklı durumlardır. Örnekte `a` ve `b` değişkenlerinin değerleri aynıdır dolayısıyla `==` ile karşılaştırma yapıldıklarında `True` değerini verir. Ancak değişken değerlerinin atamaları ayrı yapıldığı için `is` ile karşılaştırma yapıldığında `False` değeri verir. Bu durum `c` ile `d` değişkenlerinde ayrılır. Çünkü `d` değişkenine `c` değişkeni atanmış ve birbirlerine eşitlenmiştir. Dolayısıyla `is` ile karşılaştırma yapıldığında `True` değerini döndürür.

## Örnek

```
def ListeKopyala(lst):
    result=[]
    for item in lst:
        result+=[item]
    return result
def main():
    a=[10,20,30,40]
    b=ListeKopyala(a)
    print("a =",a, " b =",b)
    print(a, "listesi ile ",b, "listesi değerleri eşit mi?",sep="",end="")
    print(a==b)
    print(a, " ile ",b, " listeleri aynı mı?",sep="",end="")
    print(a is b)
    b[2]=35
    print("a =",a, " b =",b)
main()
```

## Ekran Çıktısı

```
a = [10, 20, 30, 40]   b = [10, 20, 30, 40]
[10, 20, 30, 40]listesi ile [10, 20, 30, 40]listesi değerleri eşit
mi?True
[10, 20, 30, 40] ile [10, 20, 30, 40] listeleri aynı mı?False
a = [10, 20, 30, 40]   b = [10, 20, 35, 40]
```

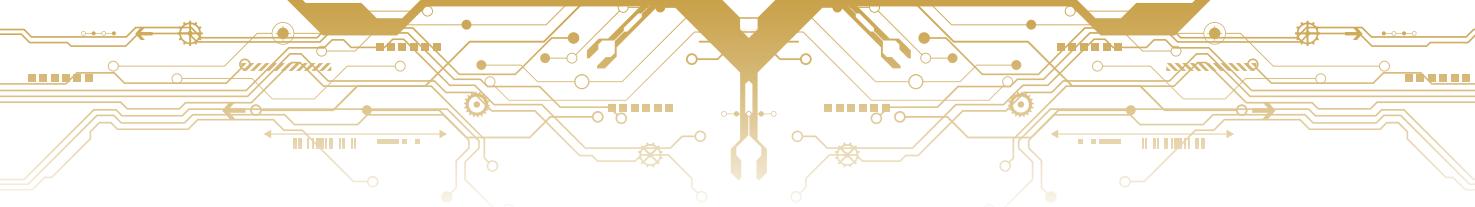
Bu örnekte a adında bir liste tanımlanmış ve bu listenin ListeKopyala() fonksiyonu ile kopyası b değişkenine oluşturulmuştur. İki liste değerleri birbirine eşittir. Fakat birbirlerine eşitlenme durumu False'dur.

## Örnek

```
print(list(range(11)))
print(list(range(10,101,10)))
print(list(range(10,-1,-1)))
```

## Ekran Çıktısı

```
[0,1,2,3,4,5,6,7,8,9,10]
[10,20,30,40,50,60,70,80,90,100]
[10,9,8,7,6,5,4,3,2,1,0]
```



## 10.7. Listenin Sınırları

a=[10,20,30,40] listesinde a[0], a[1], a[2] ve a[3] değerleri bulunurken a[4] değeri bulunmamaktadır.

a=[10,20,30,40]

print(a[4]) # listenin sınırı dışından bir erişim yapılmaya çalışılmaktadır.

Bu durumda IndexError hatası oluşur.

```
# Değerleri 0 olan 100 elemanlı liste oluşturuluyor
v=[0]*100
x=int(input("Bir sayı giriniz: "))
# Girilen değer liste sınırları içerisinde mi?
if 0<=x<len(v):
    v[x]=1 # Girilen indis değeri 1 olarak değiştiriliyor
else:
    print("Girdiğiniz değer liste sınırları arasında değil")
```

## 10.8. Dilimleme

Bir liste dilimlenerek başka bir liste oluşturulabilir. Bir listeyi dilimlemek için aşağıdaki ifadeyi kullanmak gereklidir.

**list [ başlangıç : bitiş : artım miktarı ]**

- List: liste değişkenini,
- başlangıç: listenin başlangıç indeksini,
- bitiş: listenin son elemanın indeksini,
- artım miktarı: verilen aralıktaki eleman sayısını belirtir.

```
Lst= [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst)          # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[0:3])    # [10, 20, 30]
print(Lst[4:8])    # [50, 60, 70, 80]
print(Lst[2:5])    # [30, 40, 50]
print(Lst[-5:-3])  # [80, 90]
print(Lst[:3])     # [10, 20, 30]
print(Lst[4:])     # [50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[:])      # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[-100:3]) # [10, 20, 30]
print(Lst[4:100])  # [50, 60, 70, 80, 90, 100, 110, 120]
print(Lst[2:-2:2]) # [30, 50, 70, 90]
print(Lst[::])     # [10, 30, 50, 70, 90, 110]
```

```
print(Lst[::-1])    # [120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

### 10.8.1. Dilimleme İçin Farklı Kullanımlar

```
>>>a=[34,-19,20,8,12]
>>>print(a)
[34,-19,20,8,12]
>>>print(a[0:1])
[34]
>>>print(a[0])
34
>>>b=[[2,5],7,[11,4]]
>>>print(b)
[[2,5],7,[11,4]]
>>>print(b[0:1])
[[2,5]]
>>>print(b[0])
[2,5]
```

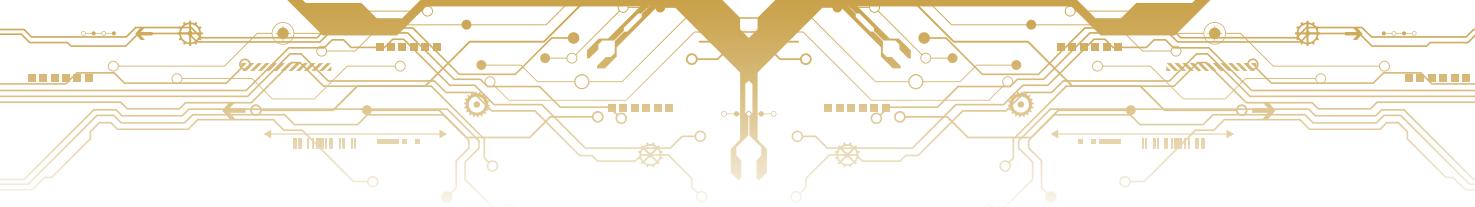
### 10.8.2. Dilimleme Örnekleri

#### Örnek

```
a=[1,2,3,4,5,6,7,8]
print("Listenin Soldan Başlanarak Dilimlenmesi",a)
for i in range(0,len(a)+1):
    print("<",a[0:i], ">",sep="")
print("-----")
print("",a)
for i in range(0,len(a)+1):
    print("<",a[i:len(a)+1], ">",sep="")
```

#### Ekran Çıktısı

```
Litenin Soldan Başlanarak Dilimlenmesi [1, 2, 3, 4, 5, 6, 7, 8]
<[]>
<[1]>
<[1, 2]>
<[1, 2, 3]>
```



```

<[1, 2, 3, 4]>
<[1, 2, 3, 4, 5]>
<[1, 2, 3, 4, 5, 6]>
<[1, 2, 3, 4, 5, 6, 7]>
<[1, 2, 3, 4, 5, 6, 7, 8,>
-----
Litenin Sağdan Başlanarak Dilimlenmesi [1, 2, 3, 4, 5, 6, 7, 8]
<[1, 2, 3, 4, 5, 6, 7, 8,>
<[2, 3, 4, 5, 6, 7, 8,>
<[3, 4, 5, 6, 7, 8,>
<[4, 5, 6, 7, 8,>
<[5, 6, 7, 8,>
<[6, 7, 8,>
<[7, 8,>
<[8,>
<[]>

```

### Örnek

```

lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:5]=["a", "b", "c"] #[30, 40, 50] değerleri ["a", "b", "c"]
değerleri ile değiştiriliyor
print(lst)
print("-----")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:6]=["a","b"] #[30, 40, 50, 60] değerleri ["a", "b"] değerleri ile
değiştiriliyor
print(lst)
print("-----")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:2]=["a", "b", "c"] # 2 nolu indexten başlayarak ["a", "b", "c"]
değerleri ekleniyor
print(lst)
print("-----")
lst=[10,20,30,40,50,60,70,80]
print(lst) # Liste yazdırılıyor
lst[2:5]=[] # [30, 40, 50] değerleri yerine [] atanıyor. (değerler
siliniyor)
print(lst)

```

## Ekran Çıktısı

```
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", "c", 60, 70, 80]
=====
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", 70, 80]
=====
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, "a", "b", "c", 30, 40, 50, 60, 70, 80]
=====
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 60, 70, 80]
=====
```

Listeler üzerinde değerleri değiştirme, değerler ekleme veya değerleri silme işlemleri yapılabilir.

## 10.9. Listeden Eleman Çıkarma

Listeden bir eleman silmek için del komutu kullanılır.

### Örnek

```
>>>a=list(range(10,51,10))
>>>a
[10,20,30,40,50]
>>>del a[2]
>>>a
[10,20,40,50]
```

### Örnek

```
>>>b=list(range(20))
>>>b
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
>>>del b[5:15]
>>>b
[0,1,2,3,4,15,16,17,18,19]
```

### Örnek

```
>>>c=list(range(20))
>>>c
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
>>>del c[1],c[18]
>>>c
[0,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
```

## 10.10. Listeler ve Fonksiyonlar

Liste elemanları üzerinde, Python diline ait fonksiyonları aynı zamanda programcının yazdığı kodları kullanarak işlem yapabiliriz. Bu örneğimizde random() fonksiyon sınıfına ait metodlar ile eleman ataması yapıldı. Bu işlemler de programcı tarafından yazılan fonksiyonlar ile gruplanmıştır.

```
import random
def Topla(lst):
    sonuc=0
    for eleman in lst:
        sonuct+=eleman
    return result
def SifirAta(lst):
    for i in range(len(lst)):
        lst[i]=0
def RastgeleDegerAta(n):
    sonuc=[]
    for i in range(n):
        RastgeleDeger=random.randrange(100)
        sonuct+=[RastgeleDeger]
    return sonuc
def main():
    a=[2,4,6,8]
    print(a)
    print(sum(a))
    SifirAta(a)
    print(a)
    print(sum(a))
    a=[]
    print(a)
    print(sum(a))
    a=RastgeleDegerAta(10)
    print(a)
    print(sum(a))
main()
```

### Ekran Çıktısı

```
2, 4, 6, 8]
20
[0, 0, 0, 0]
0
[]
0
[97, 28, 13, 38, 32, 59, 94, 60, 17, 55]
493
```



## 10.11. Listedeki Kullanılan Yöntemler

Listede Kullanılan Yöntemler	
<b>count</b>	Bir listede eleman sayısını geri döndürür. Listeyi değiştirmez.
<b>insert</b>	Verilen indeks numarasına yeni bir eleman yerleştirir. Listenin uzunluğunu bir artırır. Listeyi değiştirir.
<b>append</b>	Listenin sonuna bir eleman ekler. Listeyi değiştirir.
<b>index</b>	Listede verilen elemanın en düşük indeks numarasını geri döndürür. Eğer eleman listede yoksa hata üretir. Listeyi değiştirmez.
<b>remove</b>	Listedeki ilgili elemanı siler. İlgili eleman bulunmaz ise hata verir. Listeyi değiştirir.
<b>reverse</b>	Listedeki elemanları fiziksel olarak ters çevirir. Listeyi değiştirir.
<b>sort</b>	Artan değer şeklinde listeyi sıralar. Listeyi değiştirir.

Listelerde, değişen veri yapıları olduğu için `__getitem__` and `__setitem__` yöntemleri de kullanılır.

```
x=lst[2] ifadesi x=list.__getitem__(lst,2) ile  
lst[2]=x ifadesi list.__setitem__(lst,2,x)
```

ile benzer işlem yapar.

```
lst=["one","two","three"]  
lst+=["four"]
```



```
lst=["one","two","three"]  
lst.append("four")
```

## 10.12. Çok Boyutlu Listeler

Bir liste aslında tek boyutlu bir veri yapısıdır. İki boyutlu listeler, dikdörtgen olarak elemanlar dizisidir ve matriks olarak bilinir.

```
100 14 8 22 71  
0 243 68 1 30  
90 21 7 67 112  
115 200 70 150 8
```

```
matrix=[[100, 14, 8, 22, 71],  
[0, 243, 68, 1, 30],  
[90, 21, 7, 67, 112],  
[115, 200, 70, 150, 8]]
```

```
>>>print(matrix)  
[[100, 14, 8, 22, 71],[0, 243, 68, 1, 30],[90, 21, 7, 67, 112],  
[115, 200, 70, 150, 8]]
```

```
print (matrix [2][3])
```

## 10.13. Çok Boyutlu Diziler

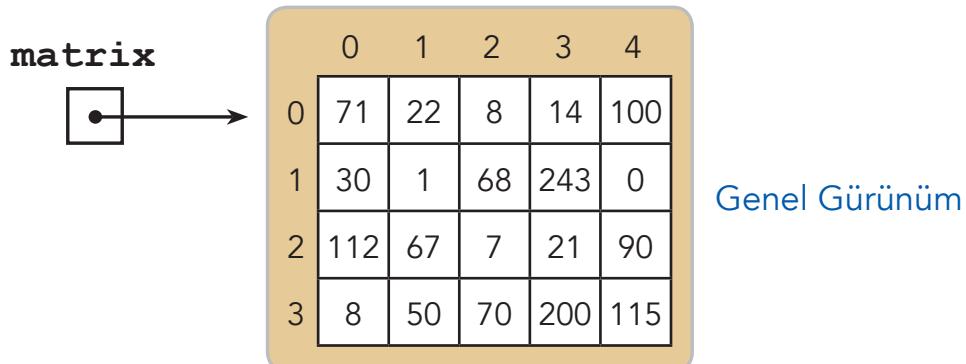
Aşağıdaki örnek iki boyutlu bir diziyi ekranaya yazdırmaktadır.

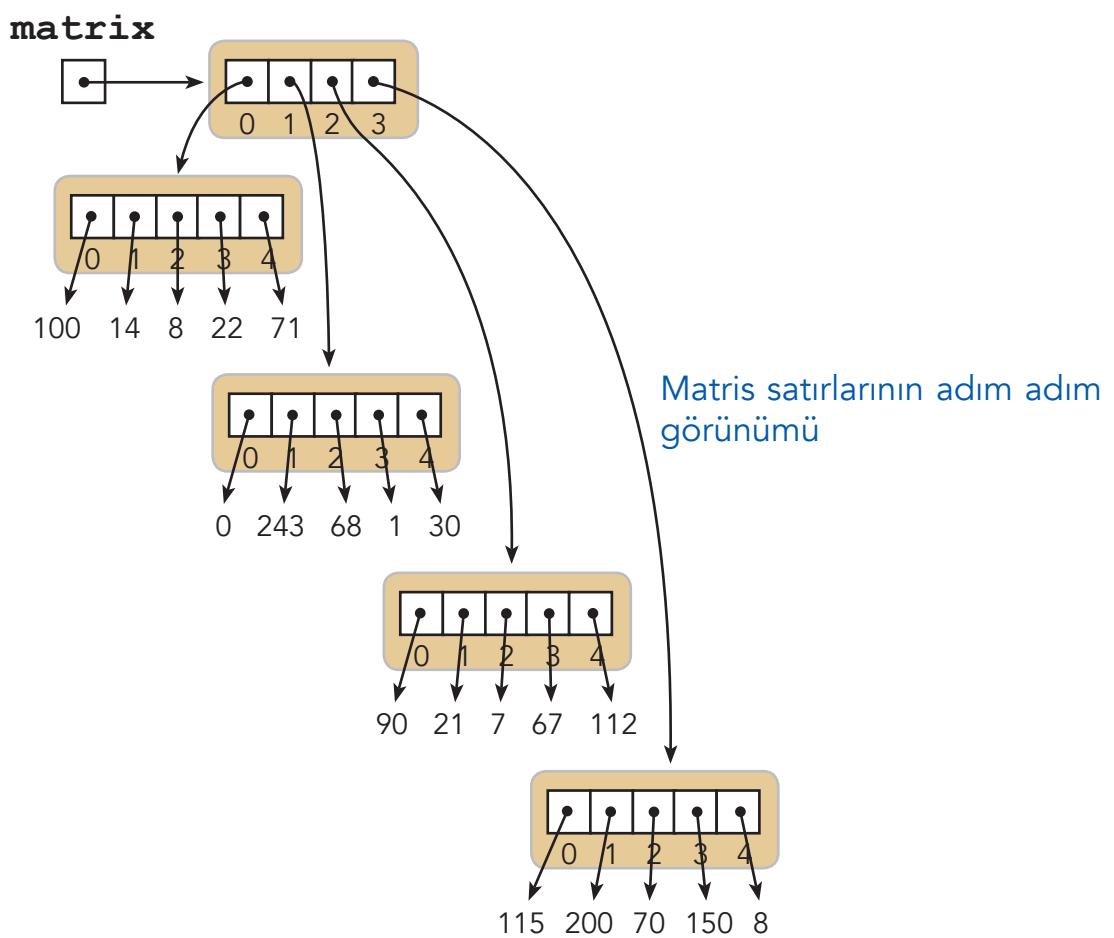
```
matrix= [[100,14,8,22,71],  
         [0,243,68,1,30],  
         [90,21,7,67,112],  
         [115,200,70,150,8]]  
  
for row in matrix:  
    for elem in row:  
        print("{:>4}".format(elem),end="")  
  
    print()
```

### Ekran Çıktısı

100	14	8	22	71
0	243	68	1	30
90	21	7	67	112
115	200	70	150	8

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.





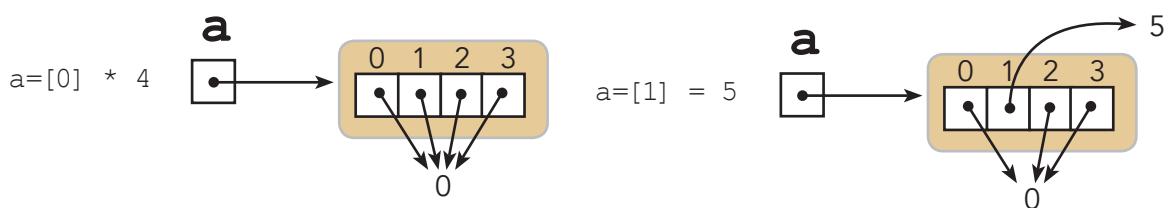
### Örnek

```
a=[[0]*4]*3
print(a)
a[1][2]=5
print(a)
```

### Ekran Çıktısı

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 5, 0], [0, 0, 5, 0], [0, 0, 5, 0]]
```

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.



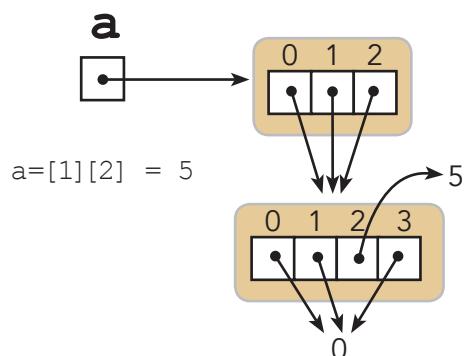
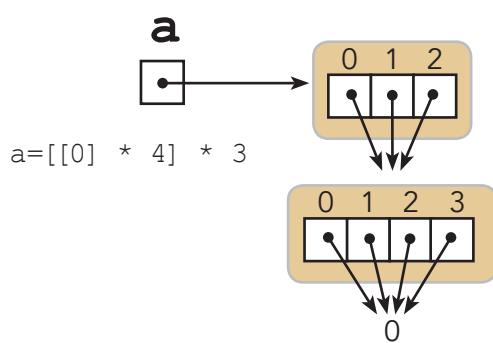
## Örnek

```
A=[[0] * 4] * 3  
for x in range(3):
```



```
a=[]  
for x in range(3):  
    a.append([0]*4)
```

Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.



## 10.14. \_ Sembolü

Kullanım durumları, aşağıdaki örneklerde gösterilmektedir.

```
a=[[0]*4 for _ in range(3)]
```

veya

```
a=[[0 for _ in range(4)] for _ in range(3)]
```

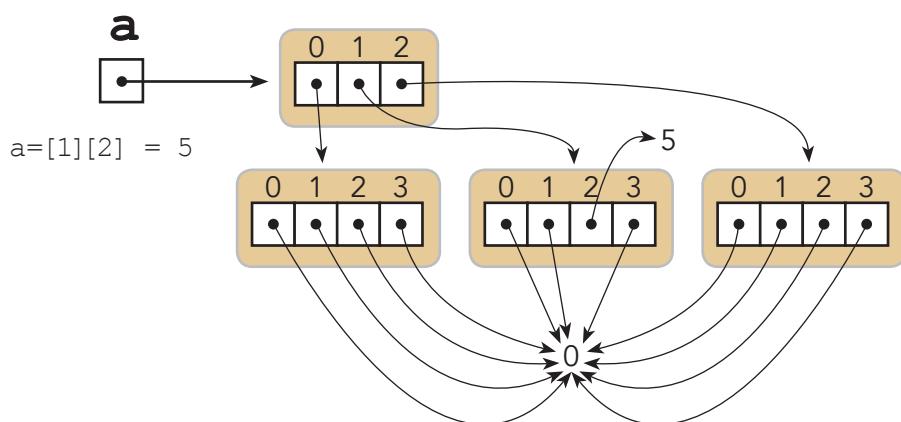
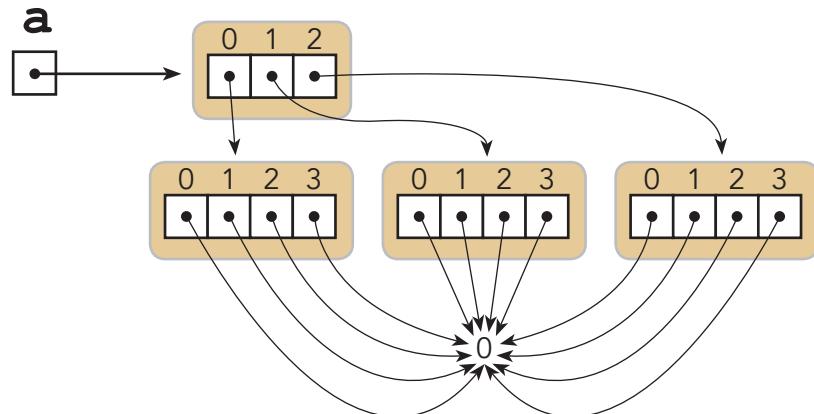
## Ekran Çıktısı

```
>>>10+4  
14  
>>>_  
14  
>>>100-60  
40  
>>>2+_  
42  
>>>
```



Dizi ve değer gösterimi, aşağıda verilen resimlerdeki gibidir.

```
a=[[0 for _ in range(4)] for _ in range(3)]
```



## 10.15. Liste Oluşturma Tekniklerinin Özeti

- Sıralı atama
  - `L=[2,4,6,8,10,12,14,16,18,20]`
- Düzensiz atama
  - `L=[]`
  - `for i in range(2,21,2):`
  - `L+=[i]`
  - `L=[2,4,6,8,10,12,14,16,18,20]`
- Range kullanımı
  - `L=list(range(2,21,2))`
- Liste oluşturma
  - `L=[x for x in range(1,21) if x%2==0]`
- Yöntemlerin birleşimi
  - `L=list(range(2,9,2))+[10,12,14]+[x for x in range(16,21,2)]`

## SÖZLÜK

### -A-

- algoritma** : Bir problem durumunu çözmek için ilgili adımların mantıksal sıralanması.
- anlam bilimsel hata** : Programın, programcının istediğiinden farklı bir şey yapmasına neden olan hata.

### -B-

- bug** : Programda oluşan bir hata.

### -C-

- çalışma zamanı hatası** : Program çalıştırılana kadar ortaya çıkmayan, çalıştırıldıktan sonra oluşan ve programın çalışmasına devam etmesini engelleyen hata.

### -D-

- derleme** : Yüksek seviyeli bir dilde yazılmış programı, düşük seviyeli bir dile çevirme işlemidir.
- düşük seviyeli dil** : Bilgisayarın kolay bir şekilde yürütmesi için tasarlanmış programlama dili. "Makine dili" veya "birleştirici dil" adı da verilmektedir.

### -H-

- hata ayıklama** : Herhangi bir programlama hatasını bulma ve ortadan kaldırma sürecidir.
- hedef kod** : Derleyicinin programı (kaynak kodu) çevirmesiyle ortaya çıkan kod serisi.

### -K-

- kaynak kod** : Yüksek seviyeli bir dildeki programın derlemeden önceki hali.

### -P-

- print** : Python yorumlayıcısının bir değeri ekranda göstermesini sağlayan komut.
- problem çözme** : Problemi formüle etme, probleme çözüm bulma ve problemin çözümü ifade etme süreci.
- program** : Bilgisayar tarafından gerçekleştirilecek eylemleri ve hesaplamaları belirtip art arda gelen komutlar.

### -S-

- söz dizimi** : Programın yapısı.
- söz dizimi hatası** : Programın ayrıştırılmasını imkânsız hale getiren hata (Dolayısıyla yorumlanması da imkânsız hale getirir.).

### -T-

- taşınabilirlik** : Bir programın birden fazla bilgisayar türünde yürütülebilmesi özelliği.
- token** : Programın söz dizimsel yapısının temel öğelerinden biri. Doğal dillerdeki kelimeye benzetilebilir.

### -Y-

- yorumlama** : Yüksek seviyeli dilde yazılmış bir programı satır satır çevirerek yürütme.
- yüksek seviyeli dil** : İnsanlar tarafından rahatlıkla okunabilmesi, yazılması ve anlaşılması için geliştirilmiş Python benzeri programlama dili.