

1.1.1. format() Metodu

Karakter dizilerinin içinde değerlerin yazdırılması büyük bir ihtiyaçtır. Karakter dizisinin metotlarından bir olan **format()** metodu değerlerin karakter dizisi içinde yazdırılması için kullanılabilecek en kullanışlı yollardan biridir.

format() metodu için kapsamlı kaynakları bulabileceğiniz adresler;

<https://docs.python.org/3.5/library/string.html#formatspec>

<https://pyformat.info/>

```
>>> "{} , {} ve {} pozitif tek sayılardır".format(3,5,7)
'3,5 ve 7 pozitif tek sayılardır'
```

format() metodu karakter dizisinin içinde bulunan süslü parantezlerin **{}** yerine sırasıyla aldığı argümanların değerlerini yerleştirir.

```
"{} , {} ve {} pozitif tek sayılardır".format(3,5,7)
```

Karakter dizileri üzerinde diğer tüm verilerde olduğu gibi doğrudan çalışılabilir. Karakter dizisinden sonra nokta ile **format()** metodu çağrılır. Çağrılan **format()** metodu doğrudan kendi karakter dizisi üzerinde etkili olacaktır.

```
"{} , {} ve {} pozitif tek sayılardır".format(3,5,7)
```

.format(3,5,7) format metodunun içinde bulunan argümanlar karakter dizisi içinde bulunan süslü parantezlerin **{}** yerlerine sırasıyla yerleştirilir.

```
'3,5 ve 7 pozitif tek sayılardır'
```

format() metodu aldığı argümanların değerlerini karakter katarına yerleştirir ve karakter dizisi son haliyle ekrana yazdırılır.

```
>>> adı,yaş,boy = 'Can',15,1.85
>>> print("{} {} yaşında {} boyundadır".format(adı,yaş,boy))
Can 15 yaşında 1.85 boyundadır
```

Üsteki örnek incelendiğinde karakter katarı içinde bulunan **{}** süslü parantezler yerine, **format** metodunda bulunan **adı, yaş ve boy** değişkenlerinin değerleri sırasıyla yerleştirilmiş. Son durumda karakter dizisi değişkenlerin değerleriyle birlikte kullanıcı için anlamlı bir şekilde ekrana yazdırılır.

```
>>> "{} {} {}".format(1,2)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    "{} {} {}".format(1,2)
IndexError: tuple index out of range
```

format() metodunun argüman sayısı **{}** süslü parantez sayısından az olursa hata oluşur.

```
>>> "{} {} {}".format(1,2,3,4,5,6)
'1 2 3'
```

Eğer **{}** süslü parantezden fazla değer **format()** metoduna argüman olarak girilirse, sırasıyla **{}** süslü parantez sayısı kadar argüman kullanılır. Üsteki örnekte **format()** metodunun ilk üç argümanı karakter dizisine yerleştirilmiş.

```
>>> '{0} {1} {0}'.format('Müdür','Öğretmen')
'Müdür Öğretmen Müdür'
```

format() metodu içindeki argümanlar karakter dizisi içinde tekrar edebilir. format() argümanları metin içinde tekrar edilecekse süslü parantez içinde argüman indeks numarası kullanılmalıdır.

```
>>> a,b = 5,7
>>> metin = "{} + {} = {}".format(a,b,a+b)
>>> sonuc = metin
>>> print(sonuc)
5 + 7 = 12
```

Şimdiye kadar kullanılan örneklerin tümünde karakter dizisi üzerinde doğrudan çalışıldı. Bu örnekte karakter dizisi **metin** değişkenine atanıyor. **metin** değişkeninde bulunan karakter dizisi **format()** metodunun argümanlarıyla şekillendirilerek **sonuc** değişkenine atanıyor.

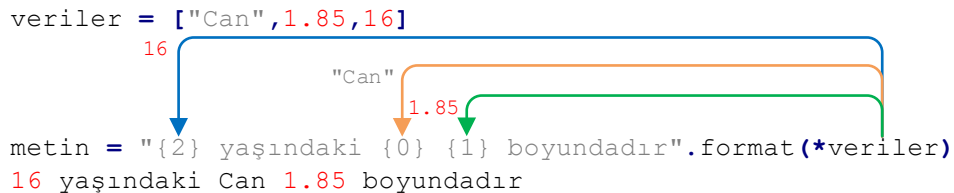
```
>>> adi,yaş,boy = 'Can',15,1.85
>>> print("{} yaşındaki {} {} boyundadır".format(adi,yaş,boy))
15 yaşındaki Can 1.85 boyundadır
```

format() argümanları süslü parantezlerin içine sıra numarası belirtilerek istenen sırada karakter dizisine yerleştirilebilir.

```
>>> veriler = ["Can",1.85,16]
>>> metin = "{} yaşındaki {} {} boyundadır".format(*veriler)
>>> print(metin)
16 yaşındaki Can 1.85 boyundadır
```

Fonksiyonlara ve metotlara girilen **list**, **tuple** ve **karakter** verisi tipinde argümanların başına ***** operatörü getirilirse argümanlar üzerinde **sekans açma işlemi** yapılır.

```
veriler = ["Can",1.85,16]
16
"Can"
1.85
metin = "{} yaşındaki {} {} boyundadır".format(*veriler)
16 yaşındaki Can 1.85 boyundadır
```



Sekans açma işlemiyle dizi elemanlarına ayrılır. Dizi elemanları süslü parantezlerin **{}** içine sıra numarası belirtilerek karakter dizisine yerleştirilir.

```
# değeri girildiği gibi tırnaklarıyla yazdırma
>>> "Değeri Tırnak ile Birlikte Yazdırma : {!r} ".format('Değer')
'Değeri Tırnak ile Birlikte Yazdırma : 'Değer' '
```

{!r} şeklinde kullanımda karakter değeri tanımlandığı tırnak karakterleri ile birlikte yazdırılır.

```
# değeri tırnaksız yazdırma yazdırma. {} ile yanı işlem
>>> "Değeri Tırnak Olmadan Yazdırma : {!s} ".format('Değer')
'Değeri Tırnak Olmadan Yazdırma : Değer '
```

{!s} içerisinde kullanılan **!s** değer string olduğunu ve tırnaksız yazdırılacağını belirtir. **{}** kullanımı ile aynı etkiyi yaratır.

```
>>> '{x} * {y} = {sonuc}'.format(x=3,y=5,sonuc=3*5)
'3 * 5 = 15'
```

Küçük karakter dizileri içinde verileri {} ile görüntülemek kafanızı karıştırmaz. Fakat büyük bir metin ve çok sayıda değişken değeri görüntülediğiniz bir karakter dizisi kullanıyorsanız metin içinde kaybolmanız çok doğaldır. format() metodu içinde değerleri değişkenlere atayarak süslü parantez {} ile kullanabilirsiniz.

```
>>> adı = 'Tuna'
>>> soyadı= 'Potur'
>>> '{adı} {soyadı}'.format(adı=adı,soyadı=soyadı)
'Tuna Potur'
```

Değişkenleri {} içinde doğrudan kullanamazsınız. {} içinde değişkenleri kullanmak için format() metodu içinde değişkeni kendisine atamalısınız.

```
>>> veriler = [2,4,8,16,32]
>>> '1.indeks = {v[1]} ve 3.indeks = {v[3]}'.format(v=veriler)
'1.indeks = 4 ve 3.indeks = 16'
```

format() metodunda dizileri kolay kullanmak için dizileri tek harflik bir ifadeye atayabilirsiniz. Üsteki örnekte veriler dizisi v harfine atanmış. Karakter dizisinde süslü parantezler içinde {v[1]} ve {v[3]} kullanımı ile veriler dizisi 1 ve 3 indeksindeki elemanlar kullanılıyor.

```
>>> kisi = {'adı':'Ahmet','soyadı':'Çelik','babaAdı':'Cenk'}
>>> print('Adı: {adı} {soyadı}\nBaba Adı: {babaAdı}'.format(**kisi))
Adı: Ahmet Çelik
Baba Adı: Cenk
```

İleride göreceğimiz sözlük verisi format metodunda ** operatörü yardımıyla kolay bir şekilde kullanılabilir. Sözlük veri tipinde indeksler yerine anahtarlar kullanılır. Veriler sözlük içinden anahtarlar yardımıyla çağrılır. Sözlük içindeki 'adı', 'soyadı' ve 'babaAdı' sözlük verisinin anahtarlarıdır. Anahtarların sağında : ile ayrılan alandakilerde verilerdir. Üsteki örnekte görüldüğü gibi format içinde sözlük değişkeni üzerinde ** kullanıldığında sekans açmaya benze bir şekilde veriler metin içine kullanılabilir. Metin içinde sözlük verilerine { } içine anahtar kelimeleri {adı} gibi yazılarak erişir.

1.1.1.1. Karakter Dizisi İçine Girilen Verilen Şekillendirilmesi

{:} Süslü parantez içinde iki nokta üst üste karakteri ile verilerin karakter dizisi içinde nasıl görüleceğe belirtilir. İki nokta üst üste : karakteri format metoduna değerlerin nasıl şekillendirileceğiyle ilgili büyük kontrol imkanı verir.

```
>>> "{:.2f} {:.3f} {:.4f}".format(3.14159,2.7182818,1.145793)
'3.14 2.718 1.1458'
```

: karakterinin en sık kullanılacağı yerler float sayıların kaç ondalıkla kullanılacağını belirttiği uygulamalardır. Üsteki örnekte : karakterinden sonra .2f, .3f ve .4f ile float sayıların 2, 3 ve 4 ondalık hanesine sahip olacağı belirtilmiş. f karakteri float sayılar için düzenleme yapılacağını belirtir.

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004

>>> "{:.2f}".format(0.2 + 0.1)
'0.30'
>>> "{:.2f}".format(3 * 0.1)
'0.30'
```

float sayılar arasında yapılan işlemlerde bazen kullanışlı olmayan değerler çıkabilir. Üsteki örnekte {:.2f} ifadesiyle format() metodu işlem sonuçları 2 ondalık olarak düzgün şekilde görüntüler.

```
# değerler olduğu gibi görüntülenir
>>> '{:f}; {:f}'.format(3.14, -3.14)
'3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)
'3.140000; -3.140000'
# değerler işaretleriyle görüntülenir
>>> '{:+f}; {:+f}'.format(3.14, -3.14)
'+3.140000; -3.140000'
# değerler eğer pozitifse önü boş negatifse önü - ile görüntülenir.
>>> '{: f}; {: f}'.format(3.14, -3.14)
' 3.140000; -3.140000'
```

Üsteki örnekte görüldüğü gibi değerler işaretleriyle birlikte görüntülenebilir.

```
>>> # format binary sayıları da destekler
>>> "int:{0:d}; float:{0:f}; hex:{0:x}; oct:{0:o}; bin:{0:b}".format(41)
'int:41; float:41.000000; hex:29; oct:51; bin:101001'
```

: ile birlikte değer **tam**, **float**, **hexadecimal**, **octal** ve **binary** sayılara çevrilerek düzenlenebilir.

```
>>> # binary sayıların ön takıları 0x, 0o, ve 0b # ile kullanılabilir
>>> "hex:{0:#x}; oct:{0:#o}; bin:{0:#b}".format(41)
'hex:0x29; oct:0o51; bin:0b101001'
```

kullanarak, **hexadecimal**, **octal** ve **binary** sayıların ön takıları yazdırılır.

```
>>> # virgöl binlik haneler için kullanılır
>>> '{:,}'.format(1500250125)          >>> '{:,.2f}'.format(1500250125.6492)
'1,500,250,125'                      '1,500,250,125.65'
```

Türkçe 'de virgöl ondalık sayılarda, nokta binlik haneleri ayırmakta kullanılır. Fakat İngilizce 'de ve birçok diğer dilde bu durum tam tersidir. Üstteki örnekte büyük sayıların anlaşılması için sayının virgöl ile binlik haneleri ayırmakta nasıl kullanıldığı gösterilmiş.

```
>>> '{:.2%}'.format(19/22)          >>> '{: .2%}'.format(0.8636)
'86.36%'                          ' 86.36%'
```

% kullanarak ondalıklı sayılar yüzde olarak ifade edilir.

```
>>> '{:06d}'.format(57)             >>> '{:06d}'.format(57)
'      57'                          '000057'
```

Ekrana yazdırılacak sayılar başlarına boşluk veya 0 eklenerek yazdırılabilir.

```
>>> '{:10}'.format('deneme')
'deneme      '
```

: karakterinin sağına girilen rakam format() metodunun gönderdiği değer kaç karakter içine yazdırılacağını belirtir. Üsteki örneklerde **{:10}** kullanımında 'deneme' karakteri sola dayalı olarak 10 karakterlik alana yazdırılır.

```
>>> '{:<10}'.format('deneme')        >>> "{:*<10}".format('deneme')
'deneme      '                      'deneme****'
```

Normal kullanımda karakter yazılımı sola dayalı olarak yapılır. : karakterinden sonra < kullanmakta karakterin sola dayalı olacağını belirtir. < karakterin önüne yazılan karakter format() metodunun gönderdiği karakterden boş kalan yerlere yazdırılır. Örnekte * ile 'deneme' metninden sonra boşluklar '****' ile doldurulmuş.

```
>>> "{:>10}".format('deneme')        >>> "{:*>10}".format('deneme')
'      deneme'                      '****deneme'
```

format() metodunun gönderdiği metni sağa dayalı yazdırmak için > karakteri kullanılır. > karakterinin önüne yazılan karakter ile boşluklar doldurulur. Örnekte *> ile boşluklar 'deneme' metninden önce '****' ile doldurulmuş.

```
>>> '{:^10}'.format('deneme')           >>> '{:*^10}'.format('deneme')
'deneme'                               '***deneme**'
```

format() metodunun gönderdiği metni ortalarak yazdırmak için ^ karakteri kullanılır. ^ karakterinin önüne yazılan karakter ile boşluklar doldurulur. Örnekte *^ ile boşluklar '**' ile doldurulmuş.

```
>>> '{:*^6}'.format('iki')
'*iki**'
```

Eğer ortalamak istediğiniz metin belirttiğiniz alanın tam ortasına yerleşmiyorsa fazla karakter sağ tarafa eklenir.

```
>>> '{:.4}'.format('Lüleburgaz')         >>> '{:10.4}'.format('Lüleburgaz')
'Lüle'                                  'Lüle'
>>> '{:*>10.4}'.format('Lüleburgaz')     >>> '{:*^10.4}'.format('Lüleburgaz')
'*****Lüle'                           '***Lüle***'
```

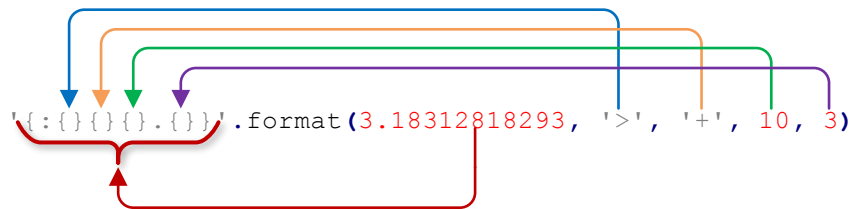
: karakterinden sonra .4 şeklinde yazılan sayı format() metodu içinde ekrana yazdırılacak karakterin baştan kaç harfinin yazdırılacağını belirtir. {:.4} 'Lüleburgaz' kelimesinin ilk dört harfi olan 'Lüle' kelimesini ekrana yazdırır. Eğer : karakterinden sonra float sayılara benzer şekilde nokta ile bir sayı yazılırsa belirtilen boşluk içinde kaç harf yazdırılacağı belirtilir. Üsteki örnekte {:10.4} ifadesinde noktanın solundaki 10 sayısı yazdırılacak değerin kaç boşluk içine yazdırılacağını belirtir. 4 değeri ise değerin kaç karakterinin yazdırılacağını belirtir. {:*>10.4} ve {:*^10.4} şeklindeki kullanımda belirtilen boşlukların yerine * karakteri yerleştirilir.

```
>>> from datetime import datetime
>>> '{:%Y-%m-%d %H:%M:%S}'.format(datetime(2017,12,3,7,5,9))
'2017-12-03 07:05:09'
```

{:%Y-%m-%d %H:%M} kullanımıyla tarih verili şekillendirilir. %Y yıl, %m ay, %d gün, %H saat, %M dakika ve %S saniye verileri için kullanılır. from datetime import datetime ifadesi datetime modülünün kodunuzun içine yüklenmesi için kullanılır. Modüller konusu ileride görülecek.

```
>>> '{:>+10.3}'.format(3.18312818293)
'      +3.18'
>>> # format metodundan düzenleme bilgisinin gönderilmesi
>>> '{:({}{}){}}'.format(3.18312818293, '>', '+', 10, 3)
'      +3.18'
```

Metin içerisinde şekillendirilecek değerin nasıl şekillendirileceği iç içe süslü parantez {} kullanılarak yapılabilir.



Süslü parantez içine iki noktadan sonra kullanılan süslü parantezlere değerin nasıl şekillendirileceği bilgisi format metodu içinden sırasıyla gönderilir. Üsteki şekilde görüldüğü {:000.0} süslü parantezi format metodundan değeri alır. Süslü parantez içinde iki noktadan sonra girilen {000.0} parantez grubuna format metodunda değerdan sonra gelen değerler sırasıyla gönderilir.

```
>>> '{:>{{isaret}}}.{}'.format(3.18312818293, '>', 10, 3, isaret='+')
'      +3.18'
```

format() metodu içinden değerleri metne değişkenle gönderebildiğimiz gibi şekillendirme bilgilerini de gönderebiliriz. Süslü parantez içinde kullanılan `{{isaret}}` kullanılan süslü paranteze değişken adı yazılarak format metodu içindeki değişken tanımından `isaret='+'` düzenleme bilgisi gönderilebilir.

```
>>> '{:*^15}'.format('deneme')
'****deneme*****'
>>> # Değişkenler ile formatlama
>>> '{:{krktr}{yer}{boy}}'.format('deneme', krktr='*', yer='^', boy='15')
'****deneme*****'
```

Değişkenler yardımıyla karakterler şekillendirilebilir. `{{krktr}{yer}{boy}}` ifadesiyle `:` karakterinden sonra süslü parantez içine format metodunda kullanılan metni şekillendirecek değişkenler yazılır.

```
>>> '{:08.4f}'.format(3.1482193)
'003.1482'
>>> # Değişkenler ile formatlama
>>> '{:{krktr}{yer}.{boy}f}'.format(3.1482193, krktr=0, yer=8, boy=4)
'003.1482'
```

Değişkenler yardımıyla float sayısal ifadeleri üste örnekte görüldüğü gibi şekillendirilir.

```
>>> '{0:.{boy}}={1:.{boy}f}'.format('Lüleburgaz', 3.1482953, boy=4)
'Lüle=3.1483'
```

Üste değişkenler yardımıyla karakter ve sayısal ifadelerin birlikte şekillendirilmesi örnek verilmiş. `{boy}` değişken adının öndü nokta olduğuna dikkat edin. Nokta kullanımı kaç karakter kullanılacağını belirtir.

```
>>> '{0:{boy}}={1:{boy}}'.format('Lüleburgaz', 3.1482193, boy='.4')
'Lüle=3.148'
```

Üsteki örneğin bir önceki örneğe göre farkı `'4'` ile nokta bilgisi değişken içinde gönderilmiş.

```
>>> from datetime import datetime
>>> dt = datetime(2017,12,3,4,5)
>>> '{:{dfmt} {tfmt}}'.format(dt, dfmt='%Y-%m-%d', tfmt='%H:%M')
'2017-12-03 04:05'
```

Zaman bilgisinin formatlanmasında da değişkenler kullanılabilir.

```
>>> değerler = '{0:2d} {1:3d} {2:4d}'
>>> for x in range(1, 11):
    print(değerler.format(x, x*x, x*x*x))

>>> değerler = '{0:02d} {1:03d} {2:04d}'
>>> for x in range(1, 11):
    print(değerler.format(x, x*x, x*x*x))
```

1	1	1	01	001	0001
2	4	8	02	004	0008
3	9	27	03	009	0027
4	16	64	04	016	0064
5	25	125	05	025	0125
6	36	216	06	036	0216
7	49	343	07	049	0343
8	64	512	08	064	0512
9	81	729	09	081	0729
10	100	1000	10	100	1000

Üste karesi ve küpü alınan sayıların format() metodu ile düzgün bir şekilde ekrana yazdırılması örnek verilmiştir. Her iki örnekte de her sütun için en büyük sayının hanesi kadar boşluk ve `'0'` karakteri önceden hazırlanmıştır, boşluk ve `'0'` karakterleri işlem sonuçlarından arta kalan alanlara yazdırılmıştır. Birinci sütun için 2, ikinci sütun için 3 ve üçüncü sütun için 4 alanlık boşluk ve `'0'` işlem sonuçlarından arta kalan alanlara yazdırılmıştır.

1.1.2. zfill() Metodu

```
>>> '-3.14'.zfill(7)
'-003.14'
>>> '12'.zfill(5)
'00012'
>>> 'Tuna'.zfill(10)
'000000Tuna'
>>> 'Ahmet Tuna'.zfill(10)
'Ahmet Tuna'
>>> len('Ahmet Tuna')
10
```

```
>>> '3.14'.zfill(7)
'0003.14'
>>> '3.14159'.zfill(5)
'3.14159'
>>> 'Ahmet'.zfill(10)
'00000Ahmet'
>>> 'Merhaba Dünya'.zfill(10)
'Merhaba Dünya'
>>> len('Merhaba Dünya')
13
```

zfill() metodu karakter dizinin kaç karakterlik alan içinde oluşturulacağını belirtir. Eğer karakter dizisi **zfill()** metoduna girilen değerden az sayıda karaktere sahipse, karakter dizisinden arta kalan alanlara **sıfır(0)** karakteri yerleştirilir. Yerleştirilen **0** karakterleri karakter dizisinin sol başına yerleştirilir. Karakter dizisi **zfill()** metodunda belirtilenden çok karaktere sahipse **0** değerleri yerleştirilmez. Örnekte verilen **'12'.zfill(5)** koduna bakalım; **zfill(5)** metodunda karakter dizisinin 5 karakterlik alana yazılacağı belirtilmiş. '12' Karakter dizisi 2 elemanlı $5 - 2 = 3$ sıfır(0) karakteri karakter dizisinin sol başına yerleştirilerek '00012' karakter dizisi elde edilir.

```
>>> değerler = [1, 17, 27, 47, 139, 1098]
>>> for i in değerler:
    str(i).zfill(4)
```

```
'0001'
'0017'
'0027'
'0047'
'0139'
'1098'
```

Üsteki örnekte **değerler** isimli sayı listesi var. İleride göreceğimiz **for** döngüsü yardımıyla sayı listesinin elemanları tek tek başlarına **0** eklenerek **4** haneli şekilde yazdırılmış.