7. Giriş ve Çıkış

Bir programın çıktısını sunmanın birkaç yolu vardır; veriler insan tarafından okunabilir bir biçimde yazdırılabilir veya ileride kullanılmak üzere bir dosyaya yazılabilir. Bu bölüm bazı olasılıkları tartışacaktır.

7.1. Büyüleyici Çıktı Biçimlendirme

Şimdiye kadar değer yazmanın iki yolu ile karşılaştık: *ifade ifadeleri* ve print()işlev. (Üçüncü bir yol da write()dosya nesneleri yöntemini kullanmaktır ; standart çıktı dosyasına başvurulabilir sys.stdout. Bu konuda daha fazla bilgi için Kütüphane Referansı'na bakın.)

Genellikle çıktınızın biçimlendirmesi üzerinde yalnızca boşlukla ayrılmış değerler yazdırmaktan daha fazla kontrol istersiniz. Çıktıyı biçimlendirmenin birkaç yolu vardır.

- Biçimlendirilmiş dize değişmezlerini kullanmak için , açılış tırnak işaretiyle veya üçlü tırnak işaretiyle fveya Föncesinde bir dizeye başlayın . Bu dize İçinde, aralarında bir Python ifade yazabilirsiniz {ve } değişken veya değişmez değerler başvurabilirsiniz karakter.

```
>>>
>>> year = 2016
>>> event = 'Referendum'
>>> f'Results of the {year} {event}'
'Results of the 2016 Referendum'
```

- Dize str.format()yöntemi daha fazla manuel çaba gerektirir. Bir değişkenin nerede değiştirileceğini ve işaretlemek için yine de kullanırsınız {ve }ayrıntılı biçimlendirme yönergeleri sağlayabilir, ancak biçimlendirilecek bilgileri de sağlamanız gerekir.

```
>>>
>>> yes_votes = 42_572_654
>>> no_votes = 43_132_495
>>> percentage = yes_votes / (yes_votes + no_votes)
>>> '{:-9} YES votes  {:2.2%}'.format(yes_votes, percentage)
' 42572654 YES votes  49.67%'
```

- Son olarak, hayal edebileceğiniz herhangi bir düzen oluşturmak için dize dilimleme ve birleştirme işlemlerini kullanarak tüm dize işlemlerini kendiniz yapabilirsiniz. Dize türü, belirli bir sütun genişliğine kadar dizeleri doldurmak için yararlı işlemler gerçekleştiren bazı yöntemlere sahiptir.

Fantezi çıktıya ihtiyacınız olmadığında, ancak hata ayıklama amacıyla bazı değişkenlerin hızlı bir şekilde görüntülenmesini istediğinizde, repr()veya str()işlevleriyle herhangi bir değeri bir dizeye dönüştürebilirsiniz .

str()İse fonksiyon, oldukça insan tarafından okunabilir değerleri temsillerini geri kastedilmektedir repr()yorumlayıcı tarafından okunabilir (veya zorlar gösterimini geri olan SyntaxErrorherhangi bir sözdizim var ise). İnsan tüketimi için belirli bir temsili olmayan nesneler

için str(), aynı değeri döndürür repr(). Sayılar veya listeler ve sözlükler gibi yapılar gibi birçok değer, her iki işlevi kullanarak aynı temsile sahiptir. Özellikle dizgilerin iki farklı temsili vardır.

Bazı örnekler:

```
>>>

>>> s = 'Hello, world.'

>>> str(s)

'Hello, world.'

>>> repr(s)

"'Hello, world.'"

>>> str(1/7)

'0.14285714285714285'

>>> x = 10 * 3.25

>>> y = 200 * 200

>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'

>>> print(s)

The value of x is 32.5, and y is 40000...

>>> # The repr() of a string adds string quotes and backslashes:

... hello = 'hello, world\n'

>>> hellos = repr(hello)

>>> print(hellos)

'hello, world\n'

>>> # The argument to repr() may be any Python object:

... repr((x, y, ('spam', 'eggs')))

"(32.5, 40000, ('spam', 'eggs'))"
```

stringModül içeren Templateteklifler başka bir yol gibi yer tutucular kullanılarak dizeleri içine değerleri yerine o sınıfı $xve bir sözlükten değerlerle bunların yerine, ancak teklifler biçimlendirme çok daha az kontrolü.

7.1.1. Biçimlendirilmiş Dize Değişmezleri

Biçimlendirilmiş dize değişmezleri (kısaca f-dizeleri olarak da adlandırılır), dizenin önünefveyaile ön ek koyarakFve bir ifadeyi yazarak Python ifadelerinin değerini bir dizenin içine dahil etmenizi sağlar{expression}.

İsteğe bağlı bir format belirleyici ifadeyi takip edebilir. Bu, değerin nasıl biçimlendirildiği üzerinde daha fazla kontrole izin verir. Aşağıdaki örnek, pi'yi ondalık işaretinden sonra üç yere yuvarlar:

>>>

>>> import math

>>> print(f'The value of pi is approximately *{math.pi:.3f}*.')

The value of pi is approximately 3.142.

':'Will değerinden sonra bir tam sayı iletilmesi o alanın minimum sayıda karakter genişliğine neden olur. Bu, sütunları hizalamak için kullanışlıdır.

>>>

>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}

>>> for name, phone **in** table.items():

... print(f'*{name:10}* ==> *{phone:10d}*')

...

Sjoerd ==> 4127

Jack ==> 4098

Dcab ==> 7678

Değeri biçimlendirilmeden önce dönüştürmek için diğer değiştiriciler kullanılabilir. '!a'uygular ascii(), '!s'uygular str()ve '!r' uygular repr():

>>>

>>> animals = 'eels'

>>> print(f'My hovercraft is full of *{animals}*.')

My hovercraft is full of eels.

>>> print(f'My hovercraft is full of *{animals!r}*.')

My hovercraft is full of 'eels'.

Bu format spesifikasyonları hakkında referans için Format Spesifikasyonu Mini Dili referans kılavuzuna bakın .

7.1.2. Dize biçimi () Yöntemi

str.format()Yöntemin temel kullanımı şöyle görünür:

>>>

>>> print('We are the *{}* who say "*{}*!"'.format('knights', 'Ni'))

We are the knights who say "Ni!"

İçlerindeki köşeli ayraçlar ve karakterler (biçim alanları olarak adlandırılır), str.format()yönteme iletilen nesnelerle değiştirilir . Parantez içindeki bir sayı, str.format()yönteme iletilen nesnenin konumuna atıfta bulunmak için kullanılabilir .

>>>

>>> print('*{0}* and *{1}*'.format('spam', 'eggs'))

spam and eggs

>>> print('*{1}* and *{0}*'.format('spam', 'eggs'))

eggs and spam

str.format()Yöntemde anahtar kelime bağımsız değişkenleri kullanılıyorsa , değerlerine bağımsız değişken adı kullanılarak başvurulur.

>>>

>>> print('This *{food}* is *{adjective}*.'.format(

... food='spam', adjective='absolutely horrible'))

This spam is absolutely horrible.

Konumsal ve anahtar kelime bağımsız değişkenleri isteğe bağlı olarak birleştirilebilir:

>>>

>>> print('The story of *{0}*, *{1}*, and *{other}*.'.format('Bill', 'Manfred',

other='Georg'))

The story of Bill, Manfred, and Georg.

Bölmek istemediğiniz gerçekten uzun bir biçim dizeniz varsa, konuma göre biçimlendirilecek değişkenlere başvurabilirseniz iyi olur. Bu sadece dikseli geçerek '[]'ve tuşlara erişmek için köşeli parantez kullanarak yapılabilir

>>>

>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}

>>> print('Jack: *{0[Jack]:d}*; Sjoerd: *{0[Sjoerd]:d}*; '

... 'Dcab: *{0[Dcab]:d}*'.format(table))

Jack: 4098; Sjoerd: 4127; Dcab: 8637678

Bu, tabloyu '**' gösterimi ile anahtar kelime bağımsız değişkenleri olarak ileterek de yapılabilir.

>>>

>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}

>>> print('Jack: *{Jack:d}*; Sjoerd: *{Sjoerd:d}*; Dcab: *{Dcab:d}*'.format(**table))

Jack: 4098; Sjoerd: 4127; Dcab: 8637678

Bu, özellikle vars()tüm yerel değişkenleri içeren bir sözlük döndüren yerleşik işlevle birlikte kullanışlıdır .

Örnek olarak, aşağıdaki satırlar tamsayılar ve bunların kareleri ve küplerini veren düzenlenmiş bir sütun kümesi oluşturur:

>>>

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

İle dize biçimlendirmesine tam bir genel bakış için str.format()bkz . Dize Sözdizimini Biçimlendirme .

7.1.3. Manuel Dize Biçimlendirme

İşte aynı biçimlendirilmiş kareler ve küpler tablosu:

```
>>>
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
```

10 100 1000

(Her sütun arasındaki boşluğun print()çalışma şekliyle eklendiğini unutmayın : her zaman argümanları arasına boşluk ekler.)

str.rjust()Dize yöntem soldaki boşluklar ile doldurup sağ haklı çıkarmaktadır, belirli bir genişlikteki bir alanda bir dizi nesneleri. Benzer yöntemler str.ljust()ve vardır str.center(). Bu yöntemler hiçbir şey yazmaz, sadece yeni bir dize döndürür. Giriş dizesi çok uzunsa, kısaltmazlar, ancak değiştirmezler; bu sütun düzeninizi bozar, ancak bu genellikle bir değer hakkında yalan söyleyen alternatiften daha iyidir. (Gerçekten kesme istiyorsanız, her zaman olduğu gibi bir dilim işlemi ekleyebilirsiniz x.ljust(n)[:n].)

str.zfill()Solda sıfırlarla sayısal bir dize dolduran başka bir yöntem var . Artı ve eksi işaretlerini anlar:

```
>>>
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

7.1.4. Eski dize biçimlendirme

%Operatör ayrıca dize biçimlendirme için kullanılabilir. Sol bağımsız sprintf()değişkeni, sağ bağımsız değişkene uygulanacak bir -style biçim dizesi gibi yorumlar ve bu biçimlendirme işleminden kaynaklanan dizeyi döndürür. Örneğin:

```
>>>
>>> import math
>>> print('The value of pi is approximately %5.3f.' % math.pi)
The value of pi is approximately 3.142.
```

Daha fazla bilgi printf-style String Formatting bölümünde bulunabilir.

7.2. Dosyaları Okuma ve Yazma

open()Bir döndüren dosya nesnesi ve genellikle iki argüman ile kullanılır: .open(filename, mode)

```
>>>
>>> f = open('workfile', 'w')
```

İlk argüman dosya adını içeren bir dizedir. İkinci argüman, dosyanın nasıl kullanılacağını açıklayan birkaç karakter içeren başka bir dizedir. *modu*'r' , dosyanın yalnızca yazılacağı zaman okunacağı zaman olabilir 'w' (aynı ada sahip mevcut bir dosya silinir) ve 'a'dosyayı eklemek üzere açar; dosyaya yazılan tüm veriler otomatik olarak sona eklenir. 'r+'dosyayı hem okuma hem de yazma için açar. *Modu* bağımsız değişkeni isteğe bağlıdır; 'r'atlanırsa varsayılır.

Normalde, dosyalar *metin modunda* açılır , yani belirli bir kodlamada kodlanmış olan dosyadan ve dosyaya dizeler okur ve yazarsınız. Kodlama belirtilmezse, varsayılan değer platforma bağlıdır (bkz. open()). 'b'moda eklenen dosya *ikili* modda açılır : şimdi veriler bayt nesneleri şeklinde okunur ve okunur. Bu mod metin içermeyen tüm dosyalar için kullanılmalıdır.

Metin modunda, okuma sırasında varsayılan, platforma özgü satır sonlarını (\nUnix'te, \r\nWindows'ta) adil olarak dönüştürmektir \n. Metin modunda yazarken, varsayılan değer \ngeri dönüşleri platforma özgü satır sonlarına dönüştürmektir. Dosya verilerindeki bu perde arkasındaki değişiklik metin dosyaları için iyidir, ancak dosya JPEGveya EXEdosyalardaki gibi ikili verileri bozar . Bu tür dosyaları okurken ve yazarken ikili modu kullanmaya çok dikkat edin.

withDosya nesneleriyle uğraşırken anahtar kelimeyi kullanmak iyi bir uygulamadır . Avantajı, bir noktada bir istisna olsa bile dosyanın paketi bittikten sonra düzgün bir şekilde kapatılmasıdır. Kullanımı withda eşdeğer yazılı çok daha kısadır try- finallyblokları:

```
>>>
>>> with open('workfile') as f:
...     read_data = f.read()


>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

withAnahtar kelimeyi kullanmıyorsanız, f.close()dosyayı kapatmak için arayın ve hemen kullandığı sistem kaynaklarını boşaltın. Bir dosyayı açıkça kapatmazsanız, Python'un çöp toplayıcı sonunda nesneyi yok eder ve açık dosyayı sizin için kapatır, ancak dosya bir süre açık kalabilir. Diğer bir risk, farklı Python uygulamalarının bu temizlemeyi farklı zamanlarda yapmasıdır.

Bir dosya nesnesi bir withifade veya çağrı ile kapatıldıktan sonra, dosya nesnesini f.close()kullanma girişimleri otomatik olarak başarısız olur.

```
>>>
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

7.2.1. Dosya Nesneleri Yöntemleri

Bu bölümdeki örneklerin geri kalanı adlı bir dosya nesnesinin fönceden oluşturulduğunu varsayacaktır .

Bir dosyanın içeriğini okumak için f.read(size), bir miktar veri okuyan ve onu bir dize (metin modunda) veya bayt nesnesi (ikili modda) olarak döndüren çağrı . *boyut* isteğe bağlı sayısal bir bağımsız değişkendir. Zaman *boyutu* ihmal edilebilir ya da negatif bir dosyanın tüm içeriği okumak ve

döndürülür; dosya makinenizin belleğinden iki kat daha büyükse, bu sizin probleminizdir. Aksi takdirde, çoğu *boyutta* (metin modunda) veya *boyut* baytlarında (ikili modda) okunur ve döndürülür. Dosyanın sonuna gelindiğinde f.read()boş bir dize ('') döndürür .

```
>>>
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

f.readline()dosyadan tek bir satır okur; \ndizenin sonunda bir yeni satır karakteri ( ) kalır ve yalnızca dosya yeni satırla bitmezse dosyanın son satırında atlanır. Bu, dönüş değerini açık hale getirir; eğer f.readline()boş bir satır ile temsil edilirken getiriler boş bir dize, dosyanın sonuna ulaşıldı '\n'sadece tek bir yeni satır içeren bir dize.

```
>>>
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Bir dosyadan satır okumak için dosya nesnesi üzerinde döngü yapabilirsiniz. Bu bellek tasarruflu, hızlıdır ve basit koda yol açar:

```
>>>
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Listedeki bir dosyanın tüm satırlarını okumak isterseniz list(f)veya tuşunu da kullanabilirsiniz f.readlines().

f.write(string)*dizenin* içeriğini dosyaya yazar ve yazılan karakter sayısını döndürür.

```
>>>
>>> f.write('This is a test\n')
```

Diğer nesne türlerinin yazılmadan önce (metin modunda) veya bayt nesnesine (ikili modda) dönüştürülmesi gerekir:

```
>>>
>>> value = ('the answer', 42)
>>> s = str(value)  # convert the tuple to string
>>> f.write(s)
18
```

f.tell() ikili moddayken dosyanın başından itibaren bayt sayısı olarak temsil edilen dosyadaki dosya nesnesinin geçerli konumunu ve metin modundayken opak bir sayı veren bir tamsayı döndürür.

Dosya nesnesinin konumunu değiştirmek için tuşunu kullanın . Konum, bir referans noktasına *ofset* eklenmesinden hesaplanır ; referans noktası *nereye* argümanı tarafından seçilir . Bir *nereden* dosyası, 1 kullanımları mevcut dosya pozisyonu başlangıcını nereden 0 ve 2 kullanımlar referans noktası olarak dosya sonu değeri. *nereye* referans noktası olarak dosyanın başı kullanılarak çıkarılmıştır ve varsayılan 0 edilebilir.f.seek(offset, whence)

```
>>>
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)     # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)  # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

Metin dosyaları (bu bir olmadan açılmış ise bmod dizesinde), sadece izin verilen (istisna ile çok dosya sonuna kadar arama olmanın dosyanın başlangıcına göre çalışmaktadır) ve tek geçerli *ofset* değerleri olanlardan döndürülür veya sıfır. Diğer herhangi bir *ofset* değeri tanımlanmamış davranış üretir.seek(0, 2)f.tell()

Dosya nesnelerinin daha az kullanılan isatty()ve gibi bazı ek yöntemleri truncate()vardır; dosya nesneleriyle ilgili eksiksiz bir kılavuz için Kütüphane Referansı'na bakın.

7.2.2. ile yapılandırılmış verileri kaydetmejson

Dizeler bir dosyaya kolayca yazılabilir ve dosyadan okunabilir. read()Yöntem biraz daha fazla çaba sarf eder, çünkü yöntem sadece gibi bir işleve iletilmesi gereken dizeleri döndürür, bu da int()bir dizeyi

alır '123' ve 123 değerini sayısal olarak döndürür. İç içe listeler gibi daha karmaşık veri türlerini kaydetmek istediğinizde, sözlükler, elle ayrıştırma ve serileştirme karmaşık hale gelir.

Python, kullanıcıların karmaşık veri türlerini dosyalara kaydetmek için sürekli kod yazma ve hata ayıklama yerine, JSON (JavaScript Nesne Gösterimi) adı verilen popüler veri değişim formatını kullanmanıza izin verir . Çağrılan standart modül jsonPython veri hiyerarşilerini alabilir ve bunları dize gösterimine dönüştürebilir; bu işleme *serileştirme* denir . Dize gösteriminden verilerin yeniden yapılandırılmasına *serileştirme* denir . Serileştirme ve serileştirme arasında, nesneyi temsil eden dize bir dosya veya veride depolanmış olabilir veya ağ bağlantısı üzerinden uzaktaki bir makineye gönderilebilir.

Not

JSON formatı, veri alışverişine izin vermek için modern uygulamalar tarafından yaygın olarak kullanılmaktadır. Birçok programcı zaten ona aşinadır, bu da birlikte çalışabilirlik için iyi bir seçimdir.

Bir nesneniz xvarsa, JSON dizesi temsilini basit bir kod satırıyla görüntüleyebilirsiniz:

```
>>>
>>> import json
>>> json.dumps([1, 'simple', 'list'])
'[1, "simple", "list"]'
```

dumps()Fonksiyonun başka bir varyantı denir dump(), nesneyi bir metin dosyasına serileştirir . Dolayısıyla f, bir metin dosyası nesnesi yazmak için açılmışsa , bunu yapabiliriz:

```
json.dump(x, f)
```

Nesnenin kodunu tekrar çözmek için, eğer okumak için açılan fbir metin dosyası nesnesiyse:

```
x = json.load(f)
```

Bu basit serileştirme tekniği listeleri ve sözlükleri işleyebilir, ancak JSON'da rasgele sınıf örneklerini serileştirmek biraz fazla çaba gerektirir. jsonModül referansı bunun bir açıklamasını içerir.

Ayrıca bakınız

pickle - turşu modülü

Aksine JSON , *turşu* keyfi karmaşık Python nesne seri sağlayan bir protokoldür. Bu nedenle, Python'a özgüdür ve diğer dillerde yazılmış uygulamalarla iletişim kurmak için kullanılamaz. Ayrıca varsayılan olarak güvenli değildir: güvenilmeyen bir kaynaktan gelen turşu verilerinin serileştirilmesi, veriler yetenekli bir saldırgan tarafından hazırlanmışsa rastgele kod yürütebilir.