Strings

Python dilinin gerçekten öne çıktığı alanlardan biri de dizelerin işlenmesidir. Bu bölümde, Python'ın yerleşik dize yöntemlerinden ve biçimlendirme işlemlerinden bazıları ele alınacaktır.

Bu tür dize işleme kalıpları, veri bilimi çalışmaları bağlamında sıklıkla karşımıza çıkar.

## **String syntax**

Önceki derslerde örneklerde birçok dize gördünüz, ancak özetlemek gerekirse, Python'da dizeler tek veya çift tırnak işareti kullanılarak tanımlanabilir. İşlevsel olarak eşdeğerdirler.

x = 'Pluto is a planet'

y = "Pluto is a planet"

x == y

True

Diziniz tek tırnak işareti içeriyorsa (örneğin, bir kesme işaretini temsil ediyorsa) çift tırnak işaretleri kullanışlıdır.

Benzer şekilde, tek tırnak işaretleri içine alırsanız çift tırnak işareti içeren bir dize oluşturmak kolaydır:

print("Pluto's a planet!")

print('My dog is named "Pluto"')

Pluto's a planet!

My dog is named "Pluto"

Tek tırnaklı bir dizenin içine tek tırnak karakteri koymaya çalışırsak Python'un kafası karışır:

'Pluto's a planet!'

File "/tmp/ipykernel\_19/1561186517.py", line 1

'Pluto's a planet!'

^

SyntaxError: invalid syntax

Tek tırnak işaretini ters eğik çizgiyle "kaçırarak" bunu düzeltebiliriz.

'Pluto**\'**s a planet!'

"Pluto's a planet!"

Aşağıdaki tablo ters eğik çizgi karakterinin bazı önemli kullanımlarını özetlemektedir.

| What you type... | What you get | example | print(example) |
| --- | --- | --- | --- |
| \' | ' | 'What\'s up?' | What's up? |
| \" | " | "That's \"cool\"" | That's "cool" |
| \\ | \ | "Look, a mountain: /\\" | Look, a mountain: /\ |
| \n |  | "1\n2 3" | 1 2 3 |

Son dizi olan \n, yeni satır karakterini temsil eder. Python'ın yeni bir satıra başlamasını sağlar.

hello = "hello**\n**world"

print(hello)

hello

world

Ayrıca, Python'un dizeler için kullandığı üçlü tırnak sözdizimi, yeni satırları gerçek anlamıyla eklememize olanak tanır (yani, özel '\n' dizisini kullanmak yerine klavyeden 'Enter' tuşuna basarak). Bunu, fonksiyonlarımızı belgelemek için kullandığımız belge dizelerinde zaten gördük, ancak bunları bir dize tanımlamak istediğimiz her yerde kullanabiliriz.

triplequoted\_hello = """hello

world"""

print(triplequoted\_hello)

triplequoted\_hello == hello

hello

world

True

print() fonksiyonu, end anahtar sözcüğü için varsayılan '\n' değerinden farklı bir değer belirtmediğimiz sürece otomatik olarak yeni satır karakteri ekler:

print("hello")

print("world")

print("hello", end='')

print("pluto", end='')

hello

world

hellopluto

## **Strings are sequences[¶](https://www.kaggle.com/code/colinmorris/strings-and-dictionaries" \l "Strings-are-sequences" \t "_self)**

Dizeler, karakter dizileri olarak düşünülebilir. Bir listeye yapabileceğimiz hemen hemen her şeyi bir dizeye de yapabiliriz.

*# Indexing*

planet = 'Pluto'

planet[0]

'P'

*# Slicing*

planet[-3:]

'uto'

*# How long is this string?*

len(planet)

5

*# Yes, we can even loop over them*

[char+'! ' for char **in** planet]

['P! ', 'l! ', 'u! ', 't! ', 'o! ']

Ancak listelerden en büyük farkları, değiştirilemez olmalarıdır. Onları değiştiremeyiz.

planet[0] = 'B'

*# planet.append doesn't work either*

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/2683731249.py in <module>

----> 1 planet[0] = 'B'

**2** # planet.append doesn't work either

TypeError: 'str' object does not support item assignment

## **String methods[¶](https://www.kaggle.com/code/colinmorris/strings-and-dictionaries" \l "String-methods" \t "_self)**

Liste gibi, str tipinin de çok sayıda kullanışlı metodu vardır. Burada sadece birkaç örnek göstereceğim.

*# ALL CAPS*

claim = "Pluto is a planet!"

claim.upper()

'PLUTO IS A PLANET!'

*# all lowercase*

claim.lower()

'pluto is a planet!'

*# Searching for the first index of a substring*

claim.index('plan')

11

claim.startswith(planet)

True

*# false because of missing exclamation mark*

claim.endswith('planet')

False

### **Going between strings and lists: .split() and .join()**[**¶**](https://www.kaggle.com/code/colinmorris/strings-and-dictionaries#Going-between-strings-and-lists:-.split()-and-.join())

str.split(), bir dizeyi varsayılan olarak boşluklara göre keserek daha küçük dizelerden oluşan bir listeye dönüştürür. Bu, sizi büyük bir dizeden bir kelime listesine taşımak için oldukça kullanışlıdır.

words = claim.split()

words

['Pluto', 'is', 'a', 'planet!']

Bazen boşluk dışında bir şey üzerinde bölme yapmak isteyeceksiniz:

datestr = '1956-01-31'

year, month, day = datestr.split('-')

str.join() bizi diğer yöne götürür, çağrıldığı dizeyi ayırıcı olarak kullanarak bir dizi dizeyi tek uzun bir dizeye diker.

'/'.join([month, day, year])

'01/31/1956'

*# Yes, we can put unicode characters right in our string literals :)*

' 👏 '.join([word.upper() for word **in** words])

'PLUTO 👏 IS 👏 A 👏 PLANET!'

### **Building strings with .format()**

Python, + operatörüyle dizeleri birleştirmemize olanak tanır.

planet + ', we miss you.'

'Pluto, we miss you.'

Eğer herhangi bir dize olmayan nesneyi eklemek istiyorsak, öncelikle onlar üzerinde str()'yi çağırmaya dikkat etmeliyiz.

position = 9

planet + ", you'll always be the " + position + "th planet to me."

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/1802981568.py in <module>

**1** position = 9

----> 2 planet + ", you'll always be the " + position + "th planet to me."

TypeError: can only concatenate str (not "int") to str

planet + ", you'll always be the " + str(position) + "th planet to me."

"Pluto, you'll always be the 9th planet to me."

Bunu okumak zorlaşıyor ve yazmak can sıkıcı hale geliyor. str.format() imdada yetişiyor.

"**{}**, you'll always be the **{}**th planet to me.".format(planet, position)

"Pluto, you'll always be the 9th planet to me."

Çok daha temiz! Eklemek istediğimiz Python değerlerinin {} yer tutucularıyla temsil edildiği bir "biçimlendirme dizesi" için .format() çağırıyoruz.

İnt türünden bir konumu dönüştürmek için str()'yi çağırmamıza bile gerek kalmadığına dikkat edin. format() bunu bizim için hallediyor.

format()'ın yaptığı tek şey bu olsaydı bile, yine de inanılmaz derecede faydalı olurdu. Ancak görünen o ki, çok daha fazlasını yapabiliyor. İşte küçük bir örnek:

pluto\_mass = 1.303 \* 10\*\*22

earth\_mass = 5.9722 \* 10\*\*24

population = 52910390

*# 2 decimal points 3 decimal points, format as percent separate with commas*

"**{}** weighs about **{:.2}** kilograms (**{:.3%}** of Earth's mass). It is home to **{:,}** Plutonians.".format(

planet, pluto\_mass, pluto\_mass / earth\_mass, population,

)

"Pluto weighs about 1.3e+22 kilograms (0.218% of Earth's mass). It is home to 52,910,390 Plutonians."

*# Referring to format() arguments by index, starting from 0*

s = """Pluto's a **{0}**.

No, it's a **{1}**.

**{0}**!

**{1}**!""".format('planet', 'dwarf planet')

print(s)

Pluto's a planet.

No, it's a dwarf planet.

planet!

dwarf planet!

Muhtemelen sadece str.format hakkında kısa bir kitap yazabilirsiniz, bu yüzden burada duracağım ve daha fazla okuma için sizi pyformat.info ve resmi belgelere yönlendireceğim.

# Dictionaries[¶](https://www.kaggle.com/code/colinmorris/strings-and-dictionaries" \l "Dictionaries" \t "_self)

Sözlükler, anahtarları değerlere eşlemek için kullanılan yerleşik bir Python veri yapısıdır.

numbers = {'one':1, 'two':2, 'three':3}

Bu durumda 'bir', 'iki' ve 'üç' anahtarlardır ve 1, 2 ve 3 bunlara karşılık gelen değerlerdir.

Değerlere, liste ve dizelere indekslemeye benzer şekilde köşeli parantez sözdizimi aracılığıyla erişilir.

numbers['one']

1

Başka bir anahtar, değer çifti eklemek için aynı sözdizimini kullanabiliriz

numbers['eleven'] = 11

numbers

{'one': 1, 'two': 2, 'three': 3, 'eleven': 11}

Veya mevcut bir anahtarla ilişkili değeri değiştirmek için

numbers['one'] = 'Pluto'

numbers

{'one': 'Pluto', 'two': 2, 'three': 3, 'eleven': 11}

Python'da, bir önceki derste gördüğümüz liste kavrayışlarına benzer bir sözdizimi olan sözlük kavrayışları vardır.

planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

planet\_to\_initial = {planet: planet[0] for planet **in** planets}

planet\_to\_initial

{'Mercury': 'M',

'Venus': 'V',

'Earth': 'E',

'Mars': 'M',

'Jupiter': 'J',

'Saturn': 'S',

'Uranus': 'U',

'Neptune': 'N'}

in operatörü bize bir şeyin sözlükte anahtar olup olmadığını söyler

'Saturn' **in** planet\_to\_initial

True

'Betelgeuse' **in** planet\_to\_initial

False

Bir sözlük üzerindeki for döngüsü, anahtarları üzerinde döngü oluşturacaktır

for k **in** numbers:

print("**{}** = **{}**".format(k, numbers[k]))

one = Pluto

two = 2

three = 3

eleven = 11

Tüm anahtarların veya tüm değerlerin bir koleksiyonuna sırasıyla dict.keys() ve dict.values() ile erişebiliriz.

*# Get all the initials, sort them alphabetically, and put them in a space-separated string.*

' '.join(sorted(planet\_to\_initial.values()))

'E J M M N S U V'

Çok kullanışlı dict.items() metodu, bir sözlüğün anahtarları ve değerleri üzerinde aynı anda yineleme yapmamızı sağlar. (Python jargonunda, bir öğe bir anahtar-değer çiftini ifade eder.)

for planet, initial **in** planet\_to\_initial.items():

print("**{}** begins with **\"{}\"**".format(planet.rjust(10), initial))

Mercury begins with "M"

Venus begins with "V"

Earth begins with "E"

Mars begins with "M"

Jupiter begins with "J"

Saturn begins with "S"

Uranus begins with "U"

Neptune begins with "N"

help(dict)