```
1  f'str+{16.123:.2f}'
```

```
'str+16.12'
```

```
1  f'float_{16:f}, int_{16:d}'
```

```
'float_16.000000, int_16'
```

```
1  f'{65:c} {97:c}'
```

```
'A a'
```

```
1  ord('3')
```

```
51
```

```
1  f'{"asc":s} {7}'
```

```
'asc 7'
```

+ Code + Text

```
1  from decimal import Decimal
```

```
1  f'{Decimal(1000000000000000000.0):.3f}'
```

```
'1000000000000000000.000'
```

```
1  f'{Decimal(1000_000_000_000_000_000.0):.2e}'
```

```
'1.00e+18'
```

Field Widths and Alignment • Python right-aligns numbers and left-aligns other values. • Python formats float values with six digits of precision.

```
1  f'[{"hello":10}]'#10 karakterlik alanın var  varsayılan sağa hızalı
```

```
'[hello     ]'
```

```
1 print(f'[{27:10d}]')
2 f'[{3.5:10f}]'# '.' dahil 10 karakter belirtilmediği için defult olarak 6 sıfır yazılır
```

```
[        27]
'[  3.500000]'
```

Explicitly Specifying Left and Right Alignment in a Field • Can specify left and right alignment with < and >:

```
1 print(f'[{27: >15d}]')# hizalama yönü
2 f'[{27: <15d}]'
```

```
[            27]
'[27            ]'
```

Centering a Value in a Field

```
• Centering attempts to spread the remaining unoccupied character positions equally to the left and
value
• Python places the extra space to the right if an odd number of character positions remain
```

```
1 f'[{27:^7d}]'#ortaya koyma
2
```

```
'[  27   ]'
```

```
1 f'[{3.5:^7f}]'
```

```
'[3.500000]'
```

```
1 f'[{"hello":^7}]'
```

```
'[ hello ]'
```

Formatting Positive Numbers with Signs

- A + before the field width specifies that a positive number should be preceded by a +
- To fill the remaining characters of the field with es rather than spaces, place a e before the field width (and after the +if there is one)

```
1 f'[{27:+10d}]'
```

```
'[       +27]'
```

```
1 f'[{27:+010d}]'
```

```
'[+000000027]'
```

Using a Space Where a + Sign Would Appear in a Positive Value

• A space indicates that positive numbers should show a space character in the sign position

```
1 print (f'{27:d}\n{27: d}\n {-27: d}')
```

```
27
 27
 -27
```

Grouping Digits

```
1 f'{123345678:,d}'# ',' ile ayır ve integer tipinde yaz
```

```
'123,345,678'
```

```
1 f'{123345678:,.2f}'#2 sıfır
```

```
'123,345,678.00'
```

```
1
```

8.2.4 String's format Method

- f-strings were added to Python in version 3.6
- Before that, formatting was performed with the string method format
- f-string formatting is based on the format method's capabilities
- Call method format on a format string containing curly brace ({}) placeholders, possibly with format specifiers
- Pass to the method the values to be formatted
- If there's a format specifier, precede it by a colon (:)

```
1 '{:.2f}'.format(17.489)
```

```
'17.49'
```

Multiple Placeholders

- format string may contain multiple placeholders
- format method's arguments correspond to the placeholders from left to right

```
1 'ad : {} soyad : {}'.format('cafer', 'asi')
```

```
'ad : cafer soyad : asi'
```

Referencing Arguments By Position Number

- Format string can reference specific arguments by their position in the format method's argument list

```
1 '{0} {0} {1}'.format('Happy', 'Birthday')
```

```
'Happy Happy Birthday'
```

Referencing Keyword Arguments

- Can reference keyword arguments by their keys in the placeholders

```
1 '{first} {last}'.format(first='Amanda', last='Gray')
```

```
'Amanda Gray'
```

```
1 '{last} {first}'.format(first='Amanda', last='Gray')
```

```
'Gray Amanda'
```

```
1
```

8.3 Concatenating and Repeating Strings

- Previously, we used the operator to concatenate strings and the operator to repeat strings
- Also can perform these operations with augmented assignments
- Strings are immutable, so each operation assigns a new string object to the variable

```
1 s1 = 'happy'
2 s2 = 'birthday'
```

```
1 s1 += '  '+s2
```

```
1 s1
```

```
'happy  birthday'
```

```
1 symbol = '>'
2 symbol *= 5
```

```
1 symbol
```

```
'>>>>>'
```

8.4 Stripping Whitespace from Strings

- Methods for removing whitespace from the ends of a string each return a new string Removing Leading and Trailing Whitespace
- strip removes leading and trailing whitespace

```
1 sentence = '\t  \n n This is a test string. \t\t \n'
2
```

```
1 sentence.strip()
```

```
'n This is a test string.'
```

Removing Leading Whitespace

- lstrip removes only leading whitespace

```
1 sentence.lstrip()#sondaki boşlukları kaldırır
```

```
'n This is a test string. \t\t \n'
```

Removing Leading Whitespace • 1strip removes only leading whitespace sentence. lstrip()

Removing Trailing Whitespace

- rstrip removes only trailing whitespace

```
1 sentence.rstrip()
```

```
'\t  \n n This is a test string.'
```

8.5 Changing Character Case

Capitalizing Only a String's First Character

- Method capitalize returns a new string with only the first letter capitalized (sometimes called sentence capitalization)

```
1
2 'happy birthday' .capitalize()#ilk harf büyük yapar
3
```

```
'Happy birthday'
```

Capitalizing the First Character of Every Word in a String

• Method title returns a new string with only the first character of each word capitalized (sometimes called book-title capitalization)

```
1 'strings: a deeper look'.title()
```

```
'Strings: A Deeper Look'
```

8.6 Comparison Operators for Strings

- Strings may be compared with the comparison operators
- Strings are compared based on their underlying integer numeric values
- Can check integer codes with ord

```
1
```

Double-click (or enter) to edit

```
1 print (f'A: {ord("A")}; a: {ord("a")}')
2
```

```
A: 65; a: 97
```

- Compare the strings 'Orange' and 'orange' using the comparison operators

```
1 'Orange' == 'orange'
2
```

```
False
```

```
1 'Orange' != 'orange'
2
```

```
True
```

```
1 'Orange' < 'orange'
2
```

```
True
```

```
1 'Orange' <= 'orange'
2
```

```
True
```

```
1 'Orange' > 'orange'
```

```
False
```

```
1
```

8.7 Searching for Substrings

. Can search a string for a **substring to ■count number of occurrences ▪ determine whether a string contains a substring ▪ determine the index at which a substring resides in a string • Each method shown in this section compares characters lexicographically using their underlying numeric values Counting Occurrences • count returns the number of times its argument occurs in a string

```
1 sentence = 'to be or not to be that is the question'
2 sentence.count('to')
```

```
2
```

• If you specify as the second argument a start index, count searches only the slice_string_ [start_index:]

```
1 sentence.count('to', 12)#12. indexten sonra ,kaç tane 'to' olduğnu döndürür
2
```

```
1
```

• If you specify as the second and third arguments the start index and end index, count searches only the slice *string* [start_index: end_index]

```
1 sentence.count('that', 12,25)#12. ve 25. inedx arasınaki 'that' sayısına bakar
2
```

```
1
```

• Like count, the other string methods presented in this section each have start index and end index arguments

Locating a Substring in a String

• index searches for a substring within a string and returns the first index at which the substring is found; otherwise, a ValueError occurs:

```
1 sentence.index('be')
```

```
3
```

• rindex performs the same operation as index, but searches from the end of the string

```
1 sentence.rindex ('be')
2
```

```
16
```

• find and rfind perform the same tasks as index and rindex but return -1 if the substring is not found Determining Whether a String Contains a Substring • To check whether a string contains a substring, use operator in or not in

```
1 #sentence.rindex ('bee')
2
```

```
1 sentence.rfind ('bee')
```

```
1 'that' in sentence
2
```

```
True
```

```
1 'THAT' in sentence
2
```

```
False
```

```
1 'THAT' not in sentence
```

```
True
```

Locating a Substring at the Beginning or End of a String

• startswith and endswith return True if the string starts with or ends with a specified substring

```
1 sentence.startswith('to')#'to' ile başlıyor mu
```

```
True
```

```
1 sentence.startswith('be')
```

```
False
```

```
1 sentence.endswith('question')#'question' ile bitiyor mu
```

```
True
```

```
1 sentence.endswith('quest')
```

```
False
```

8.8 Replacing Substrings

• A common text manipulation is to locate a substring and replace its value • replace searches a string for the substring in its first argument and replaces each occurrence with the substring in its second argument • Can receive an optional third argument specifying the maximum number of replacements

```
1 values= '1\t2\t3\t4\t5'
```

```
1 values.replace('\t', ', ') # \t yi , ile değiştirir
```

```
'1, 2, 3, 4, 5'
```

```
1
```

```
1
```

```
1
```

```
1
```

8.9 Splitting and Joining Strings

- Tokens typically are separated by whitespace characters such as blank, tab and newline, though other characters may used the separators are known as delimiters

Splitting Strings

• To tokenize a string at a custom delimiter, specify the delimiter string that split uses to tokenize the string

```
1 letters= 'A,B,C,D'
```

```
1 letters.split(',')
```

```
['A', 'B', 'C', 'D']
```

- Specify the maximum number of splits with an integer as the second argument
- Last token is the remainder of the string

```
1 letters.split(', ', 2)
```

```
['A,B,C,D']
```

- rsplit performs the same task as split but processes the maximum number of splits from the end of the string toward the beginning

Joining Strings

- join concatenates the strings in its argument, which must be an iterable containing only string values
- The separator between the concatenated items is the string on which you call join

```
1 letters_list = ['A', 'B', 'C', 'D']
2 ','.join(letters_list)
```

```
'A,B,C,D'
```

• Join the results of a list comprehension that creates a list of strings

```
1 ','.join([str(i) for i in range(10)])
```

```
'0,1,2,3,4,5,6,7,8,9'
```

String Methods partition and rpartition

String method partition splits a string into a tuple of three strings based on the method's separator argument

- the part of the original string before the separator
- the separator itself
- the part of the string after the separator

```
1 'Amanda: 89, 97, 92'.partition (': ')# : öncesi ayrı  3e bölüyor
```

```
('Amanda', ': ', '89, 97, 92')
```

• To search for the separator from the end of the string, use method rpartition

```
1 url = 'http://www.deitel.com/books/PyCDS/table_of_contents.html'
```

```
1 rest_of_url, separator, document = url.rpartition('/')
```

```
1 rest_of_url, separator, document
```

```
('http://www.deitel.com/books/PyCDS', '/', 'table_of_contents.html')
```

String Method splitlines

- splitlines returns a list of new strings representing lines of text split at each newline character in the original string

```
lines="""This is line 1
This is line2
This is line3"""
```

```
lines
```

```
'This is line 1\nThis is line2\nThis is line3'
```

```
lines.splitlines() # \n e göre bölüyor
```

```
['This is line 1', 'This is line2', 'This is line3']
```

```
lines.split('\n')
```

```
['This is line 1', 'This is line2', 'This is line3']
```

- Passing True to splitlines keeps the newlines

```
lines.splitlines (True)# daha sonra birleştirilecekse sorun çıkmaması için '\n' leri sil
```

```
['This is line 1\n', 'This is line2\n', 'This is line3']
```

8.10 Characters and Character-Testing Methods

- in Pythan a character is simply a one character string
- Python provides string methods for testing whether a string matches certain chatic
- isdigit returns True if the string or which you call the method contains only the digit characters (0-9)
    - Useful for val dating data

```
a=int(input("sayi girin"))
```

```
sayi girin12
```

```
'-27'.isdigit()

```

```
False
```

```
1 '27'.isdigit()
2
```

```
True
```

8.10 Chacters and Character-Testing Methods (cont.)

- Isalnum returns True it the string or which you call the method is aphanumeric (only cats and letters)

```
1 'A9876'.isalnum();
```

```
1
```

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found

rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

```
1 # methodları incelemek için kullanılır
2 str.*?
```

```
1 str.isalnum?
```

```
1 str.isalpha?
```

8.11 Raw Strings

- Backslash characters in strings introduce *escape sequences* like \n for newline and \t for tab
    - To include a backslash in a string, use two backslash characters
- Makes some strings difficult to read
    - Consider a Microsoft Windows file location:

```
1
```

```
1 s
```

```
"""
alfa nümerik ve küçük harf
merhaba
-------
"""

veri = input("")
if veri.isalnum():
    if veri.islower():
        print(veri)
        print('-' * len(veri))

```

```
as
as
--
```

8.11 Raw Strings (cont.)

- Faw strings—preceded by the character —are more convenient
- They teat each backstash as a regular character, rather than the beginning ofan escape sequence

```

File_path = r'C:\tyFolder\WySubFolder\WyFile.txt'
```

```
File_path#windows formatın açevirir
```

```
'C:\\tyFolder\\WySubFolder\\WyFile.txt'
```

8.12.1 re Module and Function fullmatch

- fuldmateh checks whether the entire string in its second argument matches the pattem in its first argument

```
import re

```

```
# Matching Literal Characters

```

```
pattern = "92215"

"Match" if re.fullmatch(pattern, '02215') else "No match"


```

```
'No match'
```

```
"Match" if re.fullmatch(pattern, '92215') else "No match"

```

```
'Match'
```

- First argument is the regular expression patterh to match
  - Any string can be a regular expression
  - literal characters match themselves in the specified order
- Second argument is the string that should entirely match the pattern
- Ifthe second argument matches the pattern in the first argument, fullmatch retums an object containing the matching text, which evaluates to True
- Forno match, fullmatch retums None , which evaluates to False

Metacharacters, Character Classes and Quantifiers

- Regular expressions typically contain various special symbols called metacharacters:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	
$	Capture and group	"planet$$"

- \ metacharacter begins each predefined character class
- Each matches a specific set of characters

```
'Valid' if re.fullmatch(r'\d{5}', '2215') else "Invalid"

```

```
'Invalid'
```

- In \d{5}, \d isa character class representing a digit (0-9)
- Acharacter class is a regular expression escape sequence that matches one character

- To match more than one, follow the character class with a quantifier
- {5} repeats \d five times to match five consecutive digits

```
import re

# ^ (Caret): Satırın başlangıcına karşılık gelir.
# Örnek: "hello" kelimesinin bir satırın başında olup olmadığını kontrol edin.
pattern = r"^hello"
text = "hello world"
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found.")

```

```
Match found!
```

```

# $ (Dollar): Satırın sonuna karşılık gelir.
# Örnek: "world" kelimesinin bir satırın sonunda olup olmadığını kontrol edin.
pattern = r"world$"
text = "hello world"
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found.")

```

```
Match found!
```

```

# . (Dot): Herhangi bir tek karakteri eşleştirir.
# Örnek: "cat" kelimesinin bir "c" karakteri, herhangi bir karakter ve bir "t" karakteriy
pattern = r"c.t"
text = "The cat in the hat"
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found.")

```

```
Match found!
```

```

# * (Asterisk): Önceki karakterin sıfır veya daha fazla tekrarını eşleştirir.
# Örnek: "a" harfinin sıfır veya daha fazla tekrarını eşleştirin.
pattern = r"a*"
text = "banana"
match = re.findall(pattern, text)
```

```
print(match)

```

```
['', 'a', '', 'a', '', 'a', '']
```

```

# + (Plus): Önceki karakterin bir veya daha fazla tekrarını eşleştirir.
# Örnek: "a" harfinin bir veya daha fazla tekrarını eşleştirin.
pattern = r"a+"
text = "banana"
match = re.findall(pattern, text)
print(match)

```

```
['a', 'a', 'a']
```

```

# ? (Question Mark): Önceki karakterin sıfır veya bir kez tekrarını eşleştirir.
# Örnek: "color" veya "colour" kelimesini eşleştirin.
pattern = r"colou?r"
text1 = "color"
text2 = "colour"
match1 = re.search(pattern, text1)
match2 = re.search(pattern, text2)
if match1:
    print("Match found in text1!")
if match2:
    print("Match found in text2!")


```

```
Match found in text1!
Match found in text2!
```

```
# [] (Square Brackets): Belirli bir karakter kümesini eşleştirir.
# Örnek: "a", "e" veya "i" karakterlerinden herhangi birini eşleştirin.
pattern = r"[aei]"
text = "The quick brown fox jumps over the lazy dog"
match = re.findall(pattern, text)
print(match)

```

```
['e', 'i', 'e', 'e', 'a']
```

```

# [^] (Caret inside Square Brackets): Belirli bir karakter kümesi dışındaki herhangi bir
# Örnek: "The" ile başlayan ve "over" kelimesini içeren herhangi bir karakter dizisini eş
pattern = r"^The[^over]*over.*$"
text = "The quick brown fox jumps over the lazy dog"
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found")
```

```
Match not found
```

```
# () (Parantheses): Bir grup karakteri eşleştirir.
# Örnek: "cat" veya "bat" kelimelerini eşleştirin.
pattern = r"(c|b)at"
text = "The cat in the hat chased the bat"
match = re.findall(pattern, text)
print(match)
```

```
['c', 'b']
```

```


# {n} (Curly Braces): Önceki karakterin tam olarak n kez tekrarını eşleştirir.
# Örnek: "a" harfini üç kez tekrar eden karakter dizilerini eşleştirin.
pattern = r"a{3}"
text = "banana"
match = re.findall(pattern, text)
print(match)
```

```
[]
```

```


# {m,n} (Curly Braces with two parameters): Önceki karakterin en az m ve en fazla n kez t
# Örnek: "a" harfini en az 2, en fazla 4 kez tekrar eden karakter dizilerini eşleştirin.
pattern = r"a{2,4}"
text = "baananaaa"
match = re.findall(pattern, text)
print(match)

```

```
['aa', 'aaa']
```

```

# | (Vertical Bar): Alternatif karakterler arasında seçim yapar.
# Örnek: "cat" veya "dog" kelimelerini eşleştirin.
pattern = r"cat|dog"
text = "I have a cat and a dog"
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found.")

```

```
Match found!
```

```

# \ (Backslash): Metacharacterleri kaçırmak için kullanılır.
# Örnek: "I am 5'9\" tall" cümlesindeki tırnak işaretini eşleştirin.
pattern = r"5'9\\"
text = "I am 5'9\" tall"
```

```
text2="I am 5'9\\\" tall"#text2 içinn Match found! dödürecektir
match = re.search(pattern, text)
if match:
    print("Match found!")
else:
    print("Match not found.")
print(pattern)
print(text)
```

```
Match not found.
5'9\\
I am 5'9" tall
```

```
'Valid'  if re.fullmatch(r'\d{5}', '9876') else "Invalid"

```

```
'Invalid'
```

Other Predefined Character Classes

- To match any metacharacter as its literal value, precede it by a backslash (\ )
  - For example, \\ matches a backslash (\) and \\$ matches a dollar sign ($)

```
##################
```

\d Any digit (0-9) \D Any character that is not a digit.

\s Any whitespace character (such as spaces, tabs and newlines) \S Any character that is not a vinitespace character.

\w Any word character (also called an alphanumeric character}—that is, any uppercase or lowercase letter, any digit or an underscore

\W Any character that is not a word character.

Custom Character Classes

- Square brackets [] , define a custom character class that matches a single character
- [aeibu) matches a lowercase vowel
- [A-Z] matches an uppercase letter
- [a-z] matches a lowercase letter
- [a-zA-Z] matches any lowercase or uppercase letter

```

```

```
"Valid" if re.fullmatch("[A-z][a-z]*", 'Wally') else "Invalid"


```

```
'Valid'
```

```
"Valid" if re.fullmatch("[A-z][a-z]*", 'eva') else "Invalid"

```

```
'Valid'
```

```
"Match" if re.fullmatch('[^a-z]', 'A') else 'No match'

```

```
'Match'
```

```
"Match" if re.fullmatch('[^a-z]', 'a') else 'No match'

```

```
'No match'
```

```
"Match" if re.fullmatch("[*+$]", '*') else 'No match'

```

```
'Match'
```

```
"Match" if re.fullmatch('[*+$]', '!') else 'No match'

```

```
'No match'
```

* vs. + Quantifier

- + matches at least one occurrence of a subexpression
- * and + are greedy—they match as many characters as possible

```
'Valid' if re.fullmatch( '[A-Z][a-z]+' , 'Wally') else 'Invalid'


```

```
'Valid'
```

```
'Valid' if re.fullmatch("[A-Z][a-z]+", 'Wally') else 'Invalid'

```

```
'Valid'
```

Other Quantifiers

- ? quantifier matches zero or one occurrences of a subexpression

```
'match' if re.fullmatch('labell?ed', 'labelled') else 'No match'

```

```
'match'
```

```
'match' if re.fullmatch('labell?ed', 'labeled') else 'No match'

```

```
'match'
```

```
'match' if re.fullmatch('labell?ed', 'labellled') else 'No match'

```

```
'No match'
```

```
1
```

8.12.2 Replacing Substrings and Splitting Strings

- sub function replaces patterns in a string
- split function breaks a string into pieces, based on patterns

Function sub—Replacing Patterns

- sub function replaces all occurrences of a pattern with the replacement text you specify

```
import re



re.sub(r'\t', ',', '1\t2\t3\t4')


```

```
'1,2,3,4'
```

- Three required arguments:
  - the pattern to match (the tab character *\t")
  - the replacement text (" , ") and
  - the string to be searched (1\t2\t3\t4')

- Keyword argument count can be used to specify the maximum number of replacements

```
1 re.sub(r'\t', ', ', '1\t2\t3\t4', count=2)
2
3
```

```
'1, 2, 3\t4'
```

Function split

- split function fokenizes a string, using a regular expression to specify the delimiter
- Retums a list of strings

```
1 re.split(r',\s*', "1, 2, 3,4,  5,6,7,8")
```

```
['1', '2', '3', '4', '5', '6', '7', '8']
```

- Keyword argument maxsplit specifies maximum number of splits

```
1 re.split(r',\s*', '1, 2, 3,4,   5,6,7,8', maxsplit=3)
```

```
['1', '2', '3', '4,   5,6,7,8']
```

```
1
```

Function search —Finding the First Match Anywhere in a String

- search looks in a string for the first occurrence of a substring that matches a regular expression and returns a match
- object (of type SRE_Match) that contains the matching substring
- Match object's group method returns that substring

```
1 import re
```

```
1 result = re.search('Python', 'Python is fun')
2
```

```
1 result.group() if result else 'not found'
```

```
'Python'
```

- search retums None if the string does not contain the pattern

```

```

```
1
2 result2 = re.search('fun!', 'Python is fun')
3
4 result2.group() if result2 else 'not found'
```

```
'not found'
```

Ignoring Case with the Optional flags Keyword Argument

- Many re module functions receive an optional flags keyword argument
- Changes how regular expressions are matched

```
1 result3 = re.search('Sam', 'SAM WHITE', flags=re.IGNORECASE)
2
3
4 result3.group() if result3 else 'not found'
5
```

```
'SAM'
```

Metacharacters That Restrict Matches to the Beginning or End of a String

- metacharacter at the beginning of a regular expression (and not inside square brackets) is an anchor
- Indicaties that the expression matches only the beginning of a string

```
1
2 result = re.search('Python', 'Python is fun')
3
4 result.group() if result else 'not found'
```

```
'Python'
```

```
1 result = re.search('fun', 'Python is fun')
2
3 result.group() if result else 'not found'
```

```
'fun'
```

- $ metacharacter at the end of a regular expression is an anchor indicating that the expression matches only the end of asstring

```
1
2 result = re.search('Python$', 'Python is fun')
3
4 result.group() if result else 'not found'
```

```
'not found'
```

```
result = re.search('fun$', 'Python is fun')

result.group() if result else 'not found'
```

```
'fun'
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

Function findall and finditer —Finding All Matches in a String

- findall finds every matching substring in a string
- Retums a list of the matching substrings

```
contact = "Wally White, Home: 555-555-1234, Work: 555-555-4321"

re. findall(r'\d{3}-\d{3}-\d{4}', contact)

```

```
['555-555-1234', '555-555-4321']
```

- finditer works like findall , but returns a lazy iterable of match objects

```
for phone in re.finditer(r"\d{3}-\d{3}-\d{4}", contact):
    print(phone.group())

```

```
555-555-1234
555-555-4321
```

```

text = 'Charlie Cyan, e-mail: demol@deitel.com'


```

```
```

```
pattern = r'([A-Z][a-z]+ [A-Z][a-z]+), e-mail: (\w+@\w+\.\w{3})'

```

```
result = re.search(pattern, text)
```

```
result.groups()
```

```
('Charlie Cyan', 'demol@deitel.com')
```

- The regular expression specifies two substrings to capture, each denoted by the metacharacters (and)
- (and) do not affect whether the pattern is found in the string text
- match function returns a match object only if the entire pattern is found in the string text
- match object's groups method returns a tuple of the captured substrings

```
result. groups()
```

```
('Charlie Cyan', 'demol@deitel.com')
```

```
1
```

```
1
```

Data Validation

```
import pandas as pd
zips=pd.Series({'deston':'02215','Miami':'3310'})
```

```
zips
```

```
deston    02215
Miami      3310
dtype: object
```

Data Validation (cont.)

- The secon column represents the Series ZIP Code valves (ram the distonay's vale)
- The at column" represents her incves (hom the dicionary's keys)
- Can use regular expression wih Pandas o validate data
- The str attribute ofa Series proves sting processing and varius regular expression methods
- Use the str atiibate's match method io chackwinethar each ZIP Code is valid

```
1
2 zips.str.match(r'\d{5}')
```

```
deston     True
Miami     False
dtype: bool
```

- match apples he regular exoression \4(5} to esch Series slement
- Returns a new Series containing True for each valid element

Data Validation (cont.)

- Somatines, rather han matching an entre value pattem, youll want fo know whether a value contains a substnng that matches the patton
- Use matrod contains instead of watch

```
1
2 cities = pd.Series(["Boston, MA 97215", "miami, FL 33101"])
3
```

```
1 cities
```

```
0    Boston, MA 97215
1     miami, FL 33101
dtype: object
```

```
1
2 cities.str.contains(r'[A-z]{2}')
3
```

```
0    True
1    True
dtype: bool
```

```
1 cities.str.match(r'[A-z]{2}')
2
```

```
0    True
1    True
dtype: bool
```

Reformatting

```
contacts = [
    ['Mike Green','demo1@deitel.com','5555555555'],
     ['sue brown','demo2@deitel.com','555 555 5535']
     ]
```

```
myColumns=['Name','Mail','Phone']
```

```
contactsDF=pd.DataFrame(contacts,columns=myColumns)
```

```
contactsDF
```

	Name	Mail	Phone
0	Mike Green	demo1@deitel.com	5555555555
1	sue brown	demo2@deitel.com	555 555 5535

```
contactsDF[contactsDF.columns[2]]
```

```
0      5555555555
1    555 555 5535
Name: Phone, dtype: object
```

```
1
```

Reformatting

```
def get_formatted_phone(value):
  result = re.fullmatch(r'(\d{3})(\d{3})(\d{4})',value)# veriyi 3- 3-4 olarak grouplanaca
  return '-'.join(result.groups()) if result else value
```

```
formatted_phone=contactsDF['Phone'].map(get_formatted_phone)
```

```
contacts
```

```
[['Mike Green', 'demo1@deitel.com', '5555555555'],
 ['sue brown', 'demo2@deitel.com', '555 555 5535']]
```

```
formatted_phone
```

```
0    555-555-5555
1    555 555 5535
Name: Phone, dtype: object
```

```
contactsDF['Phone']=formatted_phone
```

```
"""
kullanıcıdan string iki değer alın ve karşılaştır
```

```
kullanıcıya soralım ASCII - LENGTH
"""
```

```
'\nkullanıcıdan string iki değer alın ve karşılaştır\nkullanıcıya soralım
ASCII - LENGTH\n'
```

```
"""

SIKIŞTIRMA İSTİYORUZ

AAAAAAAAAAAAABCBCBCBCBCAAAAAAAAAADEFDEFDEFDEF

"""
```

```
'\n\nSIKIŞTIRMA İSTİYORUZ\n\nAAAAAAAAAAAAABCBCBCBCBCAAAAAAAAAADEFDEFDEFD
EF\n\n'
```

```
a="AAAA AAAA AAAA BCBC BCBC BCBC AAAA AAAA DEFDEF DEFDEF"
```