Strings in Python

PairProgramming Exercise, DSE5002

Carrie Beauzile-Milinazzo

String variables in Pythons are arrays, we can access the letters in them by index

We will import the re package, that allows us to carry out string manipulation using regular expression (regex), that we will see more about later

Helpful source

https://www.w3schools.com/python/python_ref_string.asp

In [5]:
```
import re
```

In [6]:
```
my_string="String variables in Python are used to store sequences of characters."
```

In [7]:
```
# accessing a region of the string

my_string[7:15]
```

Out[7]:
```
'variable'
```

Question/Action

access the word store in my_string

In [14]:
```
my_string[39:44]
```

Out[14]:
```
'store'
```

comparing strings

can be done with standard operations

In [8]:
```
"alpha"=="beta"
```

Out[8]:
```
False
```

In [9]:
```
"alpha"<"beta"
```

Out[9]:
```
True
```

string methods

These are a number of standardized ways of modifying strings

In [15]:
```
myword="Python rocks!"

print(myword.lower())
print(myword.upper())
print(myword.title())
```

```
python rocks!
PYTHON ROCKS!
Python Rocks!
```

find, replace, strip, split and index

These are basic operations on strings. The built-in string functions in Python are very limited,they act like the find/relace commands in Word or Excel, rather than using the more sophisticated regular expression patterns we will be discussing shortly

In [16]:
```
# find returns the index location where the pattern string is found

myword.find("rock")
```

Out[16]:
```
7
```

In [17]:
```
# find will return a -1 if the word is not in the string

myword.find("chill")
```

Out[17]:
```
-1
```

In [18]:
```
#replace replaces a target word with another word
```

In [19]:
```
myword.replace('Python',"C")
```

Out[19]:
```
'C rocks!'
```

In [20]:
```
myword.replace('Rust',"C")
```

Out[20]:
```
'Python rocks!'
```

Strip

Strip removes excess leading or trailing white space

Handy for cleaning up a string

In [21]:
```
yourword=" Once upon a regex manual "

yourword.strip()
```

Out[21]:
```
'Once upon a regex manual'
```

split

This splits a string into pieces based on a delimited, the delimiter is removed

In [23]:
```
theirstring="Alpha, Beta, Gamma"

theirstring.split(",")
```

Out[23]:
```
['Alpha', ' Beta', ' Gamma']
```

In [24]:
```
theirstring.split(" ")
```

Out[24]:
```
['Alpha,', 'Beta,', 'Gamma']
```

In [31]:
```
#index,  it's not clear to me how index differs from find

# Both index() and find() are methods used to locate substrings within a string in
# However, they differ primarily in how they handle the case where the substring is
# index() raises a ValueError when a substring is not found
# find() returns -1 when a substring is not found

# use index() when you expect the substring to be present in the string
# use find() when the presence of the substring is uncertain or optional

myword.index("roc")
```

Out[31]:
```
7
```

Regular Expressions, or Regex

In regex, we create a search pattern that is used in

-find

-replace

-string splitting

In [26]:
```
import re

#re is a python package for searching with regex

my_string="Hey, where are the anchovies?"
```

```
pattern="are"

re.search(pattern,my_string)
```

Out[26]:
```
<re.Match object; span=(11, 14), match='are'>
```

This means that the pattern "are" can be found from the location 27 to 30 in my_string

If the pattern is not in the string, we get a -1

In [32]:
```
pattern="arc"

re.search(pattern,my_string)
```

Wildcards . - this is a wildcard for a single character * - this is a wildcard for any number of characters

In [37]:
```
pattern="anc.ovies"

res=re.search(pattern,my_string)

print(res)

print(res.span())



pattern="e.*e"
res=re.search(pattern,my_string)
print(res)
print(res.span())
```

```
<re.Match object; span=(19, 28), match='anchovies'>
(19, 28)
<re.Match object; span=(1, 27), match='ey, where are the anchovie'>
(1, 27)
```

Splitting

In [29]:
```
pattern=" "
re.split(pattern,my_string)
```

Out[29]:
```
['Hey,', 'where', 'are', 'the', 'anchovies?']
```

In [30]:
```
# replacement

pattern="Hey"
newpattern="Wow"

re.sub(pattern, newpattern, my_string)
```

Out[30]:
```
'Wow, where are the anchovies?'
```