Python for Scientific Data Analysis

Basic Python

Section 6: Attributes; Finding, Replacing, Sorting, and Searching

Objects in Python typically have both attributes (other Python objects stored “inside” the object) and methods (functions associated with an object that can have access to the object’s internal data). Both of them are accessed via the syntax obj.attribute_name.

Say for example that you have a string variable called a which has a value of `foo':

`a='foo'`.

To get all the attribute names type `a.` and then press `tab` twice on your keyboard.

Here's a list of some of them that may emerge:

```
>>>a. #press tab twice
a.capitalize a.format a.isupper    a.rindex     a.strip
a.center     a.index   a.join      a.rjust      a.swapcase
a.count      a.isalnum a.ljust     a.rpartition a.title
a.decode     a.isalpha a.lower     a.rsplit     a.translate
a.encode     a.isdigit a.lstrip    a.rstrip     a.upper
a.endswith   a.islower a.partition a.split      a.zfill
a.expandtabs a.isspace a.replace   a.splitlines
a.find       a.istitle a.rfind     a.startswith
```

Most of these are self-explanatory; others you can just look up. But below we detail a few of them.

Finding and Replacing

Python lists several attributes that will allow you to find/replace elements of a string. First, `find` finds the first match between a substring you give it and a string. A simple example:

```
>>> txt="All good things can wait a bit longer"
>>> x=txt.find("good")
>>> x #4
>>> y=txt.find("things")
>>> y #9
```

There, the return value `4` corresponds to the element of the string. The full call for `find` is `string.find(value, start, end)`. Here are some examples:

```
>>> txt="All good things can wait a bit longer, good"
>>> x=txt.find("good",13,len(txt))
>>> x
39

>>> txt="All good things can be good if you wait a bit longer, good"
>>> x=txt.find("good",11,len(txt))
>>> x
23
>>> x=txt.find("good",41,len(txt))
>>> x
54
```

The property `replace` replaces a substring with another substring. E.g.

```
>>> c='one duck walks into a bar'
>>> c.replace('duck','cat')
'one cat walks into a bar'
```

Sorting

Python also allows sorting of lists: `reverse`, `sort` and other functions.

The `reverse` attribute reverses the order of a list. E.g. if `c=[4,5,6]` then `c.reverse()` is `[6,5,4]`. If you want to sort the list by an increasing value then do `c.sort()`; for decreasing, `c.sort(reverse=True)`.

Python also has another useful module called `bisect` that can be used after it is imported. It implements a binary search and insertion into a sorted list. E.g. `bisect.bisect` finds the locaitno where an element should be inserted to keep it sorted, while `bisect.insort` actually inserts the element into that location. E.g.

```
>>> import bisect
>>> c=[1, 2, 2, 2, 3, 4, 7]
>>> bisect.bisect(c, 2)
    4
>>> bisect.bisect(c, 3)
    5
>>> bisect.bisect(c, 5)
    6

    ##now, insort to insert into a sequence

>>> bisect.insort(c,6)
>>> c
[1, 2, 2, 2, 3, 4, 6, 7]
```

Python includes the ability to do string expression matching with the module `re`. You have to `import re` first to use it.

The formal documentation includes an exhaustive list of attributes for `re`: https://docs.python.org/3/library/re.html . Below are a few that are particularly useful:

`re.search` looks for the first instance of a string:

```
>>> startrek3string='Help,Jim.  Why did you leave me on Genesis? Help me,
 Spock.'
>>> key_word='Genesis'
>>> a=re.search(key_word,startrek3string)
>>> a.start() #gives the starting index position of the match
    35
>>> a.end() #gives the end index position
    42
```

`re.split` will split a string at every point there is a certain character:

```
>>> startrek3string='Help,Jim. Why did you leave me on Genesis? Help me,
Spock.'
>>> a= re.split('\s',startrek3string)
>>> a
['Help,Jim.', 'Why', 'did', 'you', 'leave', 'me', 'on', 'Genesis?', 'Help
', 'me,', 'Spock.']
```

`re.sub` will replace every instance of one string by another string:

Here's a simple example:

```
>>> mst3kstring='watch out for snakes'
>>> a=re.sub('snakes','Torgo',mst3kstring)
```

or this example where we replace spaces by 999:

```
>>> startrek3string='help,Jim. Why did you leave me on Genesis? help me,
Spock.'
>>> a=re.sub('\s','999',startrek3string)
>>> a
'help,Jim.999Why999did999you999leave999me999on999Genesis?999help999me,999
Spock.'
>>> b=re.sub('Genesis','Pluto',startrek3string)
>>> b
'Help,Jim. Why did you leave me on Pluto? Help me, Spock.
```

You can control the number of replacements with the `count' keyword:

```
>>>startrek3string='help,Jim. Why did you leave me on Genesis? help me, S
pock.  Will you help?'
>>> b=re.sub('help','Waffles',startrek3string,2)
>>> b
'Waffles,Jim. Why did you leave me on Genesis? Waffles me, Spock.  Will y
ou help?'
```

Another attribute of `re` -- `.search` -- combined with `span` will give you the indices of the first string match while combining with `group` will give you the first string where there is a match.

```
>>>txt= "The needs of the many outweigh the needs of the few"
>>>x=re.search("outweigh",txt)
>>>x.span()
   (22,30)

>>> txt= "The Needs of the many outweigh the Nags of the few"
>>> y=re.search(r"\bN\w+",txt) #searches at the beginning \b for containi
ng any word character N
>>> y.group()
'Needs'
>>> y=re.search(r"\bNa\w+",txt) #searches at the beginning \b for contain
ing any word characters Na
>>> y.group()
'Nags'
```

Instead of `.search`, you can also use `.findall`. `.findall` will actually return the instances themselves. E.g.

```
>>> txt= "The needs of the many outweigh the needs of the few"
>>> x=re.search(r"outweigh",txt)
>>> x
<re.Match object; span=(22, 30), match='outweigh'>
>>> x=re.findall(r"outweigh",txt)
>>> x
['outweigh']
```

You can use brackets -- `[ ]` -- and backslashes -- `\` -- and other metacharacters, special sequences, and sets to do complex string searches. Here are some of them

Metacharacters

Examples of metacharacters can be found here
https://www.w3schools.com/python/gloss_python_regex_metacharacters.asp

```
e.g. given a string called "hello world" ...

[]  A set of characters     "[a-m]"
\   Signals a special sequence (can also be used to escape special charac
ters)  "\d"
.   Any character (except newline character)    "he..o"
^   Starts with     "^hello"
$   Ends with   "planet$"
*   Zero or more occurrences    "he.*o"     #Search for a sequence that s
tarts with "he", followed by 0 or more  (any) characters, and an "o":
+   One or more occurrences     "he.+o"     #Search for a sequence that s
tarts with "he", followed by 1 or more  (any) characters, and an "o":
?   Zero or one occurrences     "he.?o"     #Search for a sequence that s
tarts with "he", followed by 0 or 1 characters, and an "o":
{}  Exactly the specified number of occurrences #Search for a sequence th
at starts with "he", followed excactly 2 (any) characters, and an "o": "h
e.{2}o"
|   Either or   "falls|stays"
()  Capture and group
```

Special Sequences

Examples of special sequences can be found here `https://www.w3schools.com/python/gloss_python_regex_sequences.asp`

```
e.g. given a string called ... txt = "The rain in Spain"

\A  Returns a match if the specified characters are at the beginning of t
he string  "\AThe"     #Check if the string starts with "The":
\b  Returns a match where the specified characters are at the beginning o
r at the end of a word
(the "r" in the beginning is making sure that the string is being treated
 as a "raw string")    r"\brain" #Check if "rain" is present at the begin
ning of a WORD:; e.g. x=re.search(r"\brain",txt)
r"ain\b"    #Check if "ain" is present at the end of a WORD

\B  Returns a match where the specified characters are present, but NOT a
t the beginning (or at the end) of a word
(the "r" in the beginning is making sure that the string is being treated
 as a "raw string")    # e.g. r"\Bain"
r"ain\B"

#for these examples do ...
#eightiessong = "Jenny's number is 8675309"

#Check if the string contains any digits (numbers from 0-9):


\d  Returns a match where the string contains digits (numbers from 0-9)
   "\d"    #x = re.findall("\d", eightiessong)
\D  Returns a match where the string DOES NOT contain digits    "\D"    #
x = re.findall("\D", eightiessong)
\s  Returns a match where the string contains a white space character   "
\s"
\S  Returns a match where the string DOES NOT contain a white space chara
cter   "\S"
\w  Returns a match where the string contains any word characters (charac
ters from a to Z, digits from 0-9, and the underscore _ character)     "\
w"
\W  Returns a match where the string DOES NOT contain any word characters
   "\W"
\Z  Returns a match if the specified characters are at the end of the str
ing    "Spain\Z"
```

Sets

```
[arn]   Returns a match where one of the specified characters (a, r, or n
) is present
[a-n]   Returns a match for any lower case character, alphabetically betw
een a and n
[^arn]  Returns a match for any character EXCEPT a, r, and n
[0123]  Returns a match where any of the specified digits (0, 1, 2, or 3)
 are present
[0-9]   Returns a match for any digit between 0 and 9
[0-5][0-9]  Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]    Returns a match for any character alphabetically between a an
d z, lower case OR upper case
[+]     In sets, +, *, ., |, (), $,{} has no special meaning, so [+] mean
s: return a match for any + character in the string
```