06_python_appl

January 18, 2021

1 String manipulation

String manipulation is one of Python's strong suites. It comes built in with methods for strings, and the `re` module (for *regular expressions*) ups that power many fold.

Strings are objects that we typically see in quotes. We can also check if a variable is a string.

[1]:
```
a = 'Les Miserable'

type(a)
```

[1]:
```
str
```

Strings are a little funny. They look like they are one thing, but they can act like lists. In some sense they are really a container of characters. So we can have

[2]:
```
len(a)
```

[2]:
```
13
```

[3]:
```
a[:4]
```

[3]:
```
'Les '
```

[4]:
```
a[3:6]
```

[4]:
```
' Mi'
```

The rules are basically the same as lists. To make this explicit, let's consider the word 'bare'. In terms of positions, we can write this out.

index	0	1	2	3
string	b	a	r	e
neg index	-4	-3	-2	-1

We can also slices strings (and lists for that matter) in intervals. So, going back to `a`,

[5]:
```
a[::2]
```

[5]:
```
'LsMsrbe'
```

slices every other character.

Strings come with several methods to manipulate them natively.

[6]:
```
'White Knight'.capitalize()
"It's just a flesh wound".count('u')
'Almond'.endswith('nd')
'White Knight'.lower()
'White Knight'.upper()
'flesh wound'.replace('flesh','bullet')
' This is my song   '.strip()
'Hello, hello, hello'.split(',')
```

[6]:
```
['Hello', ' hello', ' hello']
```

One of the most powerful string methods is `join`. This allows us to take a list of characters, and then put them together using a particular separator.

[7]:
```
' '.join(['This','is','my','song'])
```

[7]:
```
'This is my song'
```

Also recall that we are allowed "string arithmetic".

[8]:
```
'g' + 'a' + 'f' + 'f' + 'e'

'a '*5
```

[8]:
```
'a a a a a '
```

1.0.1 String formatting

In older code, you will see a formal format statement.

[9]:
```
var = 'horse'
var2 = 'car'

s = 'Get off my {}!'

s.format(var)
s.format(var2)
```

[9]:
```
'Get off my car!'
```

This is great for templates.

[10]:
```
template_string = """
{country}, our native village
There was a {species} tree.
We used to sleep under it.
"""

print(template_string.format(country='India', species = 'banyan'))
print(template_string.format(country = 'Canada', species = 'maple'))
```

```
India, our native village
There was a banyan tree.
We used to sleep under it.


Canada, our native village
There was a maple tree.
We used to sleep under it.
```

We can easily expand this concept for use over a whole dataset, but that is left to the student to try.

In Python 3.6+, the concept of `f-strings` or formatted strings was introduced. They can be easier to read, faster and have better performance.

[11]:
```
country = 'USA'
f"This is my {country}!"
```

[11]:
```
'This is my USA!'
```

This is how all those email's that are personalized slightly for each person are made! Hate to get them, but this makes sending them very convenient

1.1 Regular expressions

Regular expressions are amazingly powerful tools for string search and manipulation. They are available in pretty much every computer language in some form or the other. I'll provide a short and far from comprehensive introduction here. The website regex101.com is a really good resource to learn and check your regular expressions.

1.1.1 Pattern matching

Syntax	Description
.	Matches any one character
^	Matches from the beginning of a string
$	Matches to the end of a string
*	Matches 0 or more repetitions of the previous character
+	Matches 1 or more repetitions of the previous character

Syntax	Description
`?`	Matches 0 or 1 repetitions of the previous character
`{m}`	Matches `m` repetitions of the previous character
`{m,n}`	Matches any number from `m` to `n` of the previous character
`\`	Escape character
`[ ]`	A set of characters (e.g. `[A-Z]` will match any capital letter)
`( )`	Matches the pattern exactly
`|`	OR

Remeber that `dir(module)` will tell you all the functions in that module

[7]:
```
import re
d = 'orange,3,2.25'



re.findall('[0-9]+',d) #find all digits of any length
```

[7]:
```
['3', '2', '25']
```

[8]:
```
re.findall('[0-9\.]+',d) #decimal is a reserved character so we need the \
```

[8]:
```
['3', '2.25']
```

[11]:
```
x = re.findall('[0-9\.]+',d) is None

type(x)
re.findall('[0-9\.]+',d) is None
```

[11]:
```
False
```

Regular expressions are very helpful in dealing with weird things that people enter into data sheets or different formating options of something like phone numbers which have lots of formats to give the 10 digits. Regular expressions can do a lot!! Google is your friend here. They are useful for string searching in a lot of complex ways and can help with nameing variables and files.

2 BioPython

BioPython is a package aimed at bioinformatics work. As with many Python packages, it is opinionated towards the needs of the developers, so might not meet everyone's needs.

You can install BioPython using `conda install biopython`.

We'll do a short example here and the `genomics_project.ipynb` uses this package for a more complete example of what can be done.

By looking at their documentation you can get a quick look at what you can do for it

[1]:
```
from Bio.Seq import Seq

#create a sequence object
my_seq = Seq("CATGTAGACTAG")

#print out some details about it
# this is another method to format strings: %s will add a string like my_seq␣
 ↪and %i will add an integer
print("seq %s is %i bases long" % (my_seq, len(my_seq)))
print("reverse complement is %s" % my_seq.reverse_complement())
print("protein translation is %s" % my_seq.translate())
```

```
seq CATGTAGACTAG is 12 bases long
reverse complement is CTAGTCTACATG
protein translation is HVD*
```

BioPython has capabilities for querying databases like `Entrez`, read sequences, do alignments using FASTA, and the like.

To do specific tasks in python there are lots of other packages that have been developed

2.0.1 Single Cell Analysis

One thing that a lot of people want to understand is Seurat

There is a python package that is basically a python implimentation of the R package, scanpy

This is nice becuase python can run this pipline much, much faster than Seurat in R

2.0.2 Image analysis

One that is built on `scikit` is scikit-image. Python is really good at analyzing arrays. It is really fast. There are ways to make python even faster at array analysis `Numba` is to `numpy` as `DASK` is to `Pandas`. These make huge datasets easier to analyze

2.0.3 Try not to use Excel!

It will make your workflow more reproducible and Excel leads to Excele errors. Save your data as txt files. They are more compressible and less error prone than Excel.

2.1 Futher Learning

These are paid options to practice 1. data camp - practice problems 2. data quest - more practice

Really the best way is to find a simple project to practice and use google to find resorces that are relevant to that area. Most things are open source and someone has had the error you have before. Programing is a language if you don't use it you lose it.

[ ]:
```

```