Tips and trics for Python

Python things

List comprehensions

List comprehensions is a concise way of creating lists based on functions on other iterables.
Basic syntax: myList = [expression forloops conditions].
To create a generator instead of a list, use ().
To create sets, use {}
Examples:

```
import numpy as np
matrix = np.array([ [x,y]
  for x in range(4)
  for y in range(5)
]) # Creates a 20x2 matrix
```

OS utilities

Path

One of the most prominent OS libraries is the Path from pathlib. Path is used for path strings that are OS independent.

```
from pathlib import Path
currentLocation = Path()
fileLocation = Path().joinpath('data', 'Locations.txt')
```

When running a file, Python creates a variable called `__file__`, that can be used to get the files path. Use the command:

```
filepath = Path(__file__).parent
```

where parent can be used instead of '..'.

Plotting

Basic library is matplotlib.pyplot.
Another good library is seaborn.

3D plotting

Create a 3d plot using pyplot

```
import matplotlib.pytplot as plt
fig = figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X, Y, Z)
plt.show()
```

Parsing

Regex

Some sites:

- An site for testing your regex: www.regex101.com

- Basics of python regex: basics

Basic library is re

Basics

- Any character: Use . to match any character (but not newline), and `[\s\S]*` to match anything including new line.
- Use .* to match 0 or more times and .+ to match one or more times.
- Use ? for nongreedy search, e.g., `a.*?b` stops after first b and `a.*b` stops at last b.
- use r to make the string raw (or triple quotes which keep new lines)
- For whitespace type characters use `\s` instead of ' ', and for nonwhitespace `\S`
- For digits use `\d`

Capturing groups

We can capture everything between (including ends) using parenthesis as follows:

```
import re
myString = '-hereisstuff;hereismorestuff'
find_between = re.compile(r"-(.*);") # Matches between - and ; any character
match_object = find_between.match(myString)
match_object.group(0) # Returns the full match.
#Out: '-hereisstuff;'
```

Remarks:

- 0 returns the full match including the - and ;, where as 1 would give it without them ('hereisstuff')

You can also name the groups. In the previous example:

```
import re
myString = '-hereisstuff;hereismorestuff'
find_between = re.compile(r"-(?P<variableName>.*);") # Matches between - and ; any charac
match_object = find_between.match(myString)
match_object.group('variableName') # Works like 1 on previous example
#Out: 'hereisstuff'
```

Remarks:

- Great example of using groupings is first answer in this_link

Pandas

Pandas is the main library for data slicing and dicing in Python.

read_csv
This is the workhorse of pandas, used to read any text file.

- To read a string, use io.StringIO(str). This command makes the string into a file-like stream which can be used by functions that take as inputs files.

- option `delim_whitespace=True` is the same as `sep=\s+`, i.e., the separator can be any mixture of whitespace/tab combination.

- header=None in the case there is no header.

- giving dtype explicitly makes the reading faster.

Concurrent programming

Article on the basics: ConcurrentProgramming_with_Python
Main libraries are:

- multiprocessing, uses various processors and thus good for cpu-bound problems

- threading, creates many threads on one cpu but no control of changing execution, I/O-bound

- asyncio, knows where the program is going, best choice for I/O (more complex than threading)