Data Science Cheat Sheet

Python - Intermediate

KEY BASICS, PRINTING AND GETTING HELP

This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet

`s` - A Python string variable
`i` - A Python integer variable
`f` - A Python float variable
`l` - A Python list variable
`d` - A Python dictionary variable

LISTS

`l.pop(3)` - Returns the fourth item from **l** and deletes it from the list
`l.remove(x)` - Removes the first item in **l** that is equal to **x**
`l.reverse()` - Reverses the order of the items in **l**
`l[1::2]` - Returns every second item from **l**, commencing from the **1**st item
`l[-5:]` - Returns the last 5 items from **l** specific axis

STRINGS

`s.lower()` - Returns a lowercase version of **s**
`s.title()` - Returns **s** with the first letter of every word capitalized
`"23".zfill(4)` - Returns **"0023"** by left-filling the string with **0**'s to make it's length **4**.
`s.splitlines()` - Returns a list by splitting the string on any newline characters.

Python strings share some common methods with lists

`s[:5]` - Returns the first **5** characters of **s**
`"fri" + "end"` - Returns **"friend"**
`"end" in s` - Returns **True** if the substring `"end"` is found in **s**

RANGE

Range objects are useful for creating sequences of integers for looping.

`range(5)` - Returns a sequence from **0** to **4**
`range(2000,2018)` - Returns a sequence from `2000` to `2017`
`range(0,11,2)` - Returns a sequence from **0** to **10**, with each item incrementing by **2**
`range(0,-10,-1)` - Returns a sequence from **0** to **-9**
`list(range(5))` - Returns a list from **0** to **4**

DICTIONARIES

`max(d, key=d.get)` - Return the key that corresponds to the largest value in **d**
`min(d, key=d.get)` - Return the key that corresponds to the smallest value in **d**

SETS

`my_set = set(l)` - Return a `set` object containing the unique values from `l`
`len(my_set)` - Returns the number of objects in `my_set` (or, the number of unique values from **l**)
`a in my_set` - Returns **True** if the value **a** exists in `my_set`

REGULAR EXPRESSIONS

`import re` - Import the Regular Expressions module
`re.search("abc","abcde")` - Returns a `match` object if the regex **"abc"** is found in the string, otherwise **None**
`re.sub("abc","xyz",s)` - Returns a string where all instances matching regex **"abc"** are replaced by **"xyz"**

LIST COMPREHENSION

A one-line expression of a for loop

`[i ** 2 for i in range(10)]` - Returns a list of the squares of values from **0** to **9**
`[s.lower() for s in l_strings]` - Returns the list `l_strings` with each item having had the `.lower()` method applied
`[i for i in l_floats if i < 0.5]` - Returns the items from **l_floats** that are less than **0.5**

FUNCTIONS FOR LOOPING

```
for i, value in enumerate(l):
  print("The value of item {} is {}".
format(i,value))
```

- Iterate over the list **l**, printing the index location of each item and its value

```
for one, two in zip(l_one,l_two):
  print("one: {}, two: {}".format(one,two))
```

- Iterate over two lists, `l_one` and `l_two` and print each value

```
while x < 10:
x += 1
```

- Run the code in the body of the loop until the value of **x** is no longer less than **10**

DATETIME

`import datetime as dt` - Import the `datetime` module
`now = dt.datetime.now()` - Assign `datetime` object representing the current time to **now**
`wks4 = dt.datetime.timedelta(weeks=4)` - Assign a **timedelta** object representing a timespan of 4 weeks to **wks4**
`now - wks4` - Return a **datetime** object representing the time 4 weeks prior to **now**
`newyear_2020 = dt.datetime(year=2020, month=12, day=31)` - Assign a `datetime` object representing December 25, 2020 to **newyear_2020**
`newyear_2020.strftime("%A, %b %d, %Y")` - Returns `"Thursday, Dec 31, 2020"`
`dt.datetime.strptime('Dec 31, 2020',"%b %d, %Y")` - Return a `datetime` object representing December 31, 2020

RANDOM

`import random` - Import the module `random`
`random.random()` - Return a random float between **0.01.0**
`random.randint(0,10)` - Return a random integer between **0** and **10**
`random.choice(l)` - Returns a random item from the list **l**

COUNTER

`from collections import Counter` - Import the `Counter` class
`c = Counter(l)` - Assign a `Counter` object with the counts of each unique item from `l`, to **c**
`c.most_common(3)` - Return the 3 most common items from **l**

TRY/EXCEPT

Catch and deal with Errors

`l_ints = [1, 2, 3, "", 5]` - Assign a list of integers with one missing value to `l_ints`

```
l_floats = []
for i in l_ints:
try:
l_floats.append(float(i))
except:
l_floats.append(i)
```

- Convert each value of **l_ints** to a float, catching and handling `ValueError: could not convert string to float:` where values are missing.