TrendMiner Zeppelin Notebook - Python Cheat Sheet

Use `!pip install <module>` to install any additional libraries you might need in your code

Lists = [ ] or list( )

A list is a collection of items in a particular order. Lists are defined by square brackets [] and can include numbers, strings or both.

```
metals = ['lead', 'copper', 'gold', 'silver']
print(metals[0])
print(f"The last element of the list is {metals[-1]}")
```

```
lead
The last element of the list is silver
```

You can edit an existing list (or an empty list) by adding (append( )) deleting (remove( ) or del list_name[index]) or editing the elements of the list as follows:

```
non_metals = []
non_metals.append('oxygen')
non_metals.append('chlorine')
non_metals.append('iodine')
print(non_metals)

non_metals[0] = '-carbon-'
print(non_metals)
```

```
['oxygen', 'chlorine', 'silver']
['-carbon-', 'chlorine', 'silver']
```

```
metals.remove('lead')    OR    del metals[0]
print(metals)
```

```
['copper', 'gold', 'silver']
```

You can loop through a list as follows:

```
for element in metals:
    print(elements)
```

```
lead
copper
gold
silver
```

Tuples = ( )

A tuple, similar to a python list, is an object used to store a collection of items. Unlike lists, tuples are immutable so you can't add, remove or replace any elements. You can define a tuple by comma separating the elements wrapped around parentheses ( ).

```
my_tuple = (1, 9, 4, 2)
print(type(my_tuple))
```

```
<class 'tuple'>
```

Control Flow

```
if df['month'] => 3 and df['month'] <= 10:
    print('Remove C4s from gasoline')
else:
    print('Add C4s to gasoline')
```

```
for element in metals:
    print(elements)
```

```
lead
copper
gold
silver
```

While loop - repeat block of code until some condition is met ("indefinite iteration")

```
reactor_temperature = [56, 72, 76, 81, 83, 84, 84, 85]
while reactor_temperature < 80:
    print('Increase set point')
```

```
!=   ->  not equal to
==   ->  equal to
<=   ->  smaller than or equal to
>=   ->  greater than or equal to
```

Functions

Use functions to repeat a group commands a number of times without having to repeat the same chunk of code several times in your script. Define a function at the beggining of your script as follows:

```
import math

def lmtd(h_in, h_out, c_out, c_in=30):
    dT1 = h_in - c_out
    dT2 = h_out - c_in
    LogMeanTD = (dT1 - dT2 / math.log(dT1/dT2))
    return LogMeanTD
```

```
TD = lmtd(100, 85, 55)
print(str(round(TD,1)) + ' degF')
```

```
49.8 degF
```

Dictionaries = { }

A dictionary in Python is a collection of key-value pairs. Each key must be unique and has an associated value. You can use a key to access each corresponding value using the square bracket notation. Alternatively, you can use the *get* method which allows you to include a message in case that key does not exist as part of the dictionary.

```
atomic_number = {'hydrogen':1, 'hydrogen':1, 'carbon':6, 'nitrogen':7,
                 'silicon':14}
print(atomic_number['hydrogen'])
print(atomic_number.get('hydrogen', 'key does not exist'))
```

```
1
14
```

Given an empty dictionary, one can add items to the dictionary, delete key-value pairs and edit existing values as follows:

```
myDictionary = {}
myDictionary['color'] = 'green'
myDictionary['type'] = 'analog'
print(myDictionary)
```

```
{'color': 'green', 'type': 'analog'}
```

```
del myDictionary['type']
myDictionary['color'] = 'blue'
```

Alternatively, you can define a dictionary by using the function dict( ). You can print all the keys and values of a dictionary by invoking the keys( ) and values( ) method on the dictionary.

```
my_dictionary = dict([('key1', 'value1'), ('key2':'value2')])
print(my_dictionary.keys())
print(my_dictionary.values())
```

```
dict_keys(['key1', 'key2'])
dict_values(['value1', 'value2'])
```

You can loop through a dictionary and perform any operation with the key or values, lets just print them:

```
for k, v in atomic_number.items():
    print(f"\nKey: {k}")
    print(f"Value: {k}")
```

```
Key: hydrogen
Value: 1

Key: carbon
Value: 6

Key: nitrogen
Value: 7

Key: silicon
Value: 14
```

****Dictionaries can get more complex if you have a list instead of a single value for any key or even a dictionary inside of a dictionary. Also you can have a a list of dictionaries all separated by commas as items in a list.

Pandas DataFrame and Numpy

A DataFrame is a two-dimensional size-mutable tabular data structure with labeled axes (rows and columns). Each row of a dataframe is defined by the index number which are usually numbers starting at 0 but can also be timestamps if handling time-series data. *Numpy* is another library that is quite useful when it comes to performing operations on top of dataframes.

You can create a DataFrame in several ways:

```
import pandas as pd
```

```
data = [['tom', 10], ['nick', 15], ['juli', 14]]
df = pd.DataFrame(data, columns = ['Name', 'Age'])
```

	Name	Age
0	tom	10
1	nick	15
2	juli	14

```
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}
df = pd.DataFrame(data)
```

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

```
from datetime import datetime
import numpy as np


data = {'Temperature':[112, 98, 120, 96]}
df = pd.DataFrame(data, index =['2020-01-01 5:00', '2020-02-01 3:00',
                                '2020-03-01 2:00', '2020-04-01 21:00'])
indexListDatetime = [datetime.strptime(x,'%Y-%m-%d %H:%M') for x in
                     list(df.index)]
df = pd.DataFrame(data, index = indexListDatetime)

df.loc[df['Temperature'] >= 100, 'Over100'] = 1
df['_Over100_'] = np.where(df['Temperature'] >= 100, 1, 0)
df2 = df.loc[df['Over100'] == 1]
```

	Temperature	Over100	_Over100_
2020-01-01 05:00:00	112	1.0	1
2020-02-01 03:00:00	98	NaN	0
2020-03-01 02:00:00	120	1.0	1
2020-04-01 21:00:00	96	NaN	0

	Temperature	Over100	_Over100_
2020-01-01 05:00:00	112	1.0	1
2020-03-01 02:00:00	120	1.0	1

APIs

TrendMiner has developed a list of API calls that one can utilize to exchange information with TrendMiner through the various requests methods available: GET, PUT, POST, etc. The following packages are needed and recommended when performing API requests:

```
import requests
import json
import datetime
import pytz
```

Converting a string to a dictionary (i.e. parsing text to json format):

```
payload_json = json.loads(payload_string)
```

Convert a dictionary to string format:

```
payload_string = json.dumps(payload_json)
```

```
import json
payload_string = '''{"startDate":"2020-01-01", "endDate":"2020-
                    02-03"}'''
payload_json = json.loads(payload_string)
print(type(payload_json))
print(payload_json)
```

```
<class 'dict'>
{'startDate': '2020-01-01', 'endDate': '2020-02-03'}
```

The following is an example of a typical GET request requiring a payload. Use the json( ) method on the reponse to convert the response to json format (dictionary).

```
url = <tm_url> + "/end/point"
payload_json = payload in json format
auth_header = headers in json format
```

```
r = request.get(url, data=payload_json, headers=auth_header)
r_jsonFormat = r.json()
r_textFormat = r.text()
r_statusCode = r.status_code()
```

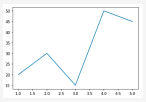
Find the complete documentation of all TrendMiner APIs at developers.trendminer.com

Data Visualization with *Matplotlib*

Matplotlib is a user friendly package that allows you to create line plots, bar graphs and other graphs to visualize your time series data from TrendMiner. Modules and their respective functions can be uploaded using:
import *module_name.function_name* as *alias* OR
from *module_name* import *function_name* as alias

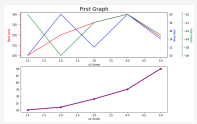
```
import matplotlib.pyplot as plt
```

```
x_val = [1, 2, 3, 4, 5]
y_val = [20, 30, 15, 50, 45]

fig, ax = plt.subplots()
ax.plot(x_val, y_val, <**key_arguments>)
plt.show()
```



```
<**key_arguments> = {linewidth = n, linestyle = '--',
marker = '.', etc.}
```

```
    x1 = [1, 2, 3, 4, 5]
    x2 = [6, 7, 8, 9, 10]

y1a = [100, 120, 132, 140, 118]
y1b = [10, 20, 12, 20, 14]
y1c = [-10, -20, -12, -10, -15]

y2 = [20, 22, 28, 35, 50]

fig, (ax1, ax2) = plt.subplots(2, figsize=(10, 7))
ax1.plot(x1, y1a, color = 'red')
ax1.spines['left'].set_color('red')
ax1.spines['left'].set_position(("axes", -0.005))
ax1.tick_params(axis='y', color='red', size=8)
ax1.set_ylabel('Red Axis', color='red', size=12)
ax1.set_xlabel('x1 (time)')
ax1.set_title("First Graph")

ax1b = ax1.twinx()
ax1b.plot(x1, y1b, color='blue')
ax1b.spines['right'].set_color('blue')
ax1b.tick_params(axis='y', color='blue')
ax1b.set_ylabel('Blue Axis')
ax1b.yaxis.label.set_color('blue')

ax1c = ax1.twinx()
ax1c.plot(x1, y1c, color='green')
ax1c.spines['right'].set_position(("axes", 1.10))
ax1c.spines['right'].set_color('green')
ax1c.tick_params(axis='y', color='green')
ax1c.set_ylabel('Green Axis')
ax1c.yaxis.label.set_color('green')

 ax2.plot(x1, y2, color='purple', linewidth=3, marker='X')
   ax2.set_xlabel('x2 (time)')

       plt.show ()
```



OTHER USEFUL FUNCTIONS...

```
round(x, n)          ->  round a number to n decimal point
str(x)               ->  convert float type to string type
print(x)             ->  print the object x
print(type(x))       ->  print the object type of x
print(len(x))        ->  get length of a list, string, df, etc.
print(list[0:3])     ->  prints first three elements of a list
print(list[-n:])     ->  prints last n elemnets of a list
print('\n')          ->  insert a break (empty line)
```