

# Ikkoe programming test

Laurent Siksous

October 27, 2020

All tests have to be performed in Python 3. Use of Numpy & Pandas is allowed. You also may use language & API documentation, but not require any help of anyone. You should at all times regularly push onto a Git repository your coding attempts. You have one hour to complete the three exercises, until Thursday October 27th 2020

```
# import needed libraries
import numpy as np
import pandas as pd
```

## 1 Question 1:

Sum all the elements in a square matrix of 1000x1000 dimension.

```
# create a square matrix of 1000x1000 dimension with random elements
A = np.random.randint(10, size=(1000,1000))
```

```
# print the sum of all elements
print(np.sum(A))
```

4498892

## 2 Question 2:

You are given a data frame containing N lines and n columns. Columns i and j are qualitative and should be binary encoded by spawning categories into new columns bearing the labels and 0 or 1. Write the procedure for this one-hot encoding.

```
df = pd.read_csv("./cars.csv",sep=',',index_col=0)
df = pd.get_dummies(df, columns=['brand', 'color'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 2499 entries, 0 to 2498
```

```
Data columns (total 87 columns):
```

#	Column	Non-Null Count	Dtype
0	price	2499 non-null	int64
1	model	2499 non-null	object
2	year	2499 non-null	int64
3	title_status	2499 non-null	object
4	mileage	2499 non-null	float64
5	vin	2499 non-null	object
6	lot	2499 non-null	int64
7	state	2499 non-null	object
8	country	2499 non-null	object
9	condition	2499 non-null	object
10	brand_acura	2499 non-null	uint8
11	brand_audi	2499 non-null	uint8
12	brand_bmw	2499 non-null	uint8
13	brand_buick	2499 non-null	uint8
14	brand_cadillac	2499 non-null	uint8
15	brand_chevrolet	2499 non-null	uint8
16	brand_chrysler	2499 non-null	uint8
17	brand_dodge	2499 non-null	uint8
18	brand_ford	2499 non-null	uint8
19	brand_gmc	2499 non-null	uint8
20	brand_harley-davidson	2499 non-null	uint8
21	brand_heartland	2499 non-null	uint8
22	brand_honda	2499 non-null	uint8
23	brand_hyundai	2499 non-null	uint8
24	brand_infiniti	2499 non-null	uint8
25	brand_jaguar	2499 non-null	uint8
26	brand_jEEP	2499 non-null	uint8
27	brand_kia	2499 non-null	uint8
28	brand_land	2499 non-null	uint8
29	brand_lexus	2499 non-null	uint8

30	brand_lincoln	2499	non-null	uint8
31	brand_maserati	2499	non-null	uint8
32	brand_mazda	2499	non-null	uint8
33	brand_mercedes-benz	2499	non-null	uint8
34	brand_nissan	2499	non-null	uint8
35	brand_peterbilt	2499	non-null	uint8
36	brand_ram	2499	non-null	uint8
37	brand_toyota	2499	non-null	uint8
38	color_beige	2499	non-null	uint8
39	color_billet silver metallic clearcoat	2499	non-null	uint8
40	color_black	2499	non-null	uint8
41	color_black clearcoat	2499	non-null	uint8
42	color_blue	2499	non-null	uint8
43	color_bright white clearcoat	2499	non-null	uint8
44	color_brown	2499	non-null	uint8
45	color_burgundy	2499	non-null	uint8
46	color_cayenne red	2499	non-null	uint8
47	color_charcoal	2499	non-null	uint8
48	color_color:	2499	non-null	uint8
49	color_competition orange	2499	non-null	uint8
50	color_dark blue	2499	non-null	uint8
51	color_glacier white	2499	non-null	uint8
52	color_gold	2499	non-null	uint8
53	color_gray	2499	non-null	uint8
54	color_green	2499	non-null	uint8
55	color_guard	2499	non-null	uint8
56	color_ingot silver	2499	non-null	uint8
57	color_ingot silver metallic	2499	non-null	uint8
58	color_jazz blue pearlcoat	2499	non-null	uint8
59	color_kona blue metallic	2499	non-null	uint8
60	color_light blue	2499	non-null	uint8
61	color_lightning blue	2499	non-null	uint8
62	color_magnetic metallic	2499	non-null	uint8
63	color_maroon	2499	non-null	uint8
64	color_morningsky blue	2499	non-null	uint8
65	color_no_color	2499	non-null	uint8
66	color_off-white	2499	non-null	uint8
67	color_orange	2499	non-null	uint8
68	color_oxford white	2499	non-null	uint8
69	color_pearl white	2499	non-null	uint8

70	color_phantom black	2499 non-null	uint8
71	color_purple	2499 non-null	uint8
72	color_red	2499 non-null	uint8
73	color_royal crimson metallic tinted clearcoat	2499 non-null	uint8
74	color_ruby red	2499 non-null	uint8
75	color_ruby red metallic tinted clearcoat	2499 non-null	uint8
76	color_shadow black	2499 non-null	uint8
77	color_silver	2499 non-null	uint8
78	color_super black	2499 non-null	uint8
79	color_tan	2499 non-null	uint8
80	color_toreador red	2499 non-null	uint8
81	color_triple yellow tri-coat	2499 non-null	uint8
82	color_turquoise	2499 non-null	uint8
83	color_tuxedo black metallic	2499 non-null	uint8
84	color_white	2499 non-null	uint8
85	color_white platinum tri-coat metallic	2499 non-null	uint8
86	color_yellow	2499 non-null	uint8

dtypes: float64(1), int64(3), object(6), uint8(77)  
memory usage: 402.7+ KB

### 3 Question 3:

Find an algorithm for determining whether subsets of connected directed vertices (i.e. edges), will spawn a circular graph.

- First, let's implement a Graph class as proposed in cite:parkBasicGraphAlgorithms Park, J., Basic Graph Algorithms, , (), 38 ().

```
class Graph():

    # Defining two arrays : E of size m and LE of size n
    def __init__(self, i):
        self.E = pd.DataFrame(columns=['to', 'nextID'])
        self.LE = [-1] * i
        self.size = i

    # Adding a new edge from u to v with ID k
    def add_edge(self, u, v):
        k = len(self.E) + 1
        self.E.loc[k] = {'to': v, 'nextID': self.LE[u-1]}
```

```

        self.LE[u-1] = k

def adjacencyMatrix(self):
    return self.E

def adjacencyList(self):
    return self.LE

def lastEdgeID(self, u):
    return self.LE[u-1]

def nextEdgeID(self, ID):
    return self.E.loc[ID].nextID

def nextHOP(self, ID):
    return self.E.loc[ID].to

def neighbors(self, u):
    neighbors = []
    ID = self.LE[u-1]
    while ID != -1:
        neighbors.append(self.E.loc[ID].to)
        ID = self.E.loc[ID].nextID
    return neighbors

def has_cycles(self):
    path = []

    def has_cycle(v):
        path.append(v)
        for n in self.neighbors(v):
            if n in path or has_cycle(n):
                return True
        path.remove(v)
        return False

    for v in self.LE:
        if has_cycle(v):
            return True
    return False

```

- Now, let's instantiate and populate a new graph of 5 nodes. This graph contains no cycles.

```
g = Graph(5)
g.add_edge(1, 2)
g.add_edge(1, 3)
g.add_edge(1, 5)
g.add_edge(2, 3)
g.add_edge(2, 5)
g.add_edge(4, 2)
g.add_edge(4, 3)
```

```
g.adjacencyMatrix()
```

```
to nextID
1 2      -1
2 3       1
3 5       2
4 3      -1
5 5       4
6 2      -1
7 3       6
```

- Implementing a Depth-First Search (DFS) algorithm in class to discover all the paths reachable from each node and if one or more is cyclic. Method `has_cycles` added to the class `Graph`. The graph created earlier had no cycle. So first call should return `False`.

```
print(g.has_cycles())
```

```
False
```

- Let's add a loop !

```
g.add_edge(3, 2)
```

```
print(g.has_cycles())
```

```
True
```

- CQFD !