PTAI: 1MA02, 1MAI1

- Question 1: Basic Python

(a) Fix the problem(s) by inserting appropiate indentation:

```
def int_sqrt(x):
    """Approximate integer square root"""
    if x < 0:
        raise ValueError("Bad input")
    else:
        guess = 0
        while (guess + 1) ** 2 <= x
            guess += 1
        return guess
```

(b) Re-implement the following code in pure Python without collections

```
dna = "gattaca"
d = {x:0 for x in list(set(dna))}
for s in dna:
    d[s] += 1
```

(c) Explain duck typing in Python using an example:

In duck typing, an object suitability is determined by the presence of certain methods and properties, rather then the object itself.

```
class Plane:
    def fly(self):
        print("Plane flying")

class Bird:
    def fly(self)
        print("Bird flying")
```

```
class Person:
    def run(self):
        print("Person running")

P = Plane()
B = Bird()
Pn = Person()
for n in [P, B, Pn]:
    try:
        n.fly()
    except:
        print(str(n) + "doesn't fly")
```

(d) Rewrite the code using itertools.

```
xs = [0,1,2,3]
ys = [False, True]
for (x,y) in product(xs,ys):
    print(x, y, f(x,y))
```

It improves readability of the code

(e) Describe the concept of machine epsilon. How could we go about finding the value of machine epsilon in Python

Machine epsilon is defined as the smallest number such that $1 + \varepsilon_{mach} > 1$.

In order to find ε_{mach} in python we could create a list `[10**(-n) for n in range(0,1000)] = L` and find the last non zero element.

```
for n in len(L):
    if L[n+1] = 0 and L[n] != 0:
        return n.
```

- Question 2: Advanced Python

(a) Define memoisation and describe the properties a function must have for memoisation to be useful

It is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same input occurs again.

Function should:

→ may be called often with the same arguments, and
→ is deterministic, and
→ has no side effects, and
→ this is enough to make our program slow

(b) Write code which will create a function f(x,y) as in the example below

```
s = "x**2 +y+1"
def f(x,y):
    return eval(s)
f(3,2)
```

(c) Show how can we use grouping to extract user name portion from a given email address

```
p = r"([\w\.-]+@\w+(\.\w+)+)$"
re.findall(p, adr)[0][0]
```

(d) Rewrite using generator

```
def f(filename):
    result = []
    for line in open(filename):
        x = int(line)
        yield x**2
```

(e) Rewrite function using a dictionary for dispatch and without if statements

a	b
0	0
0	1
1	0
1	1

```
dt = {
    (0,0): g3()
    (0,1): g2()
    (1,0): g0()
    (1,1): g1() }

def f(a,b):
    dt[(a,b)]()
```

- Question 3: Data Science

(a) use string to get correct input

```
X[:3,1:]
```

(b) Describe rules of broadcasting and use them to see if the following code will work

Rules:

1. If two arrays differ in their num of dimensions, the shape of the one with fewer dim is padded with ones on its leading (left) side.

2. If shape of two arrays does not match in any dimension, the array with shape equal to 1 in that dim is streched to match the other shape.

3. If any dims disagree and neither is 1
raised an Error

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[10],[100]])
a+b.
```

shape = (2,3)

shape = (2,1)

$$a+b = \begin{pmatrix}[1,2,3]\\[4,5,6]\end{pmatrix} + \begin{pmatrix}[10,10,10]\\[100,100,100]\end{pmatrix}$$

(c) Result of doing d.groupby("y").mean()?

	x	y	z
0	a	a	10
1	a	b	50
2	b	a	20
3	b	b	60

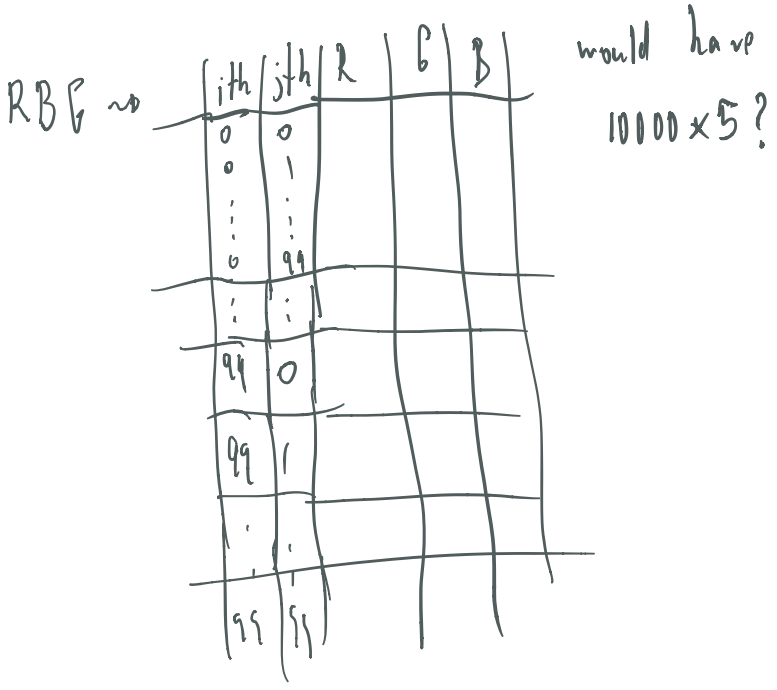
d.groupby("y").mean()

y	x	z
a	Nan	15
b	Nan	55

(d) Consider an image 100 x 100 pixels with 3 colour channels. What shape does it have? What shape would it have converted to a tidy format?

It has $100 \times 100 \times 3$.

If converted to tidy format



would have 10000×5?

(e) Describe GridSearchCV
- inputs required
- what it does
- Result

Given a model / train set and a dictionary with parameters (keys) and possible values for that parameter, GridSearchCV

returns the train model that has the best score from all possible combinations of parameters. Additionally, the score is calculated performing cross-validation.

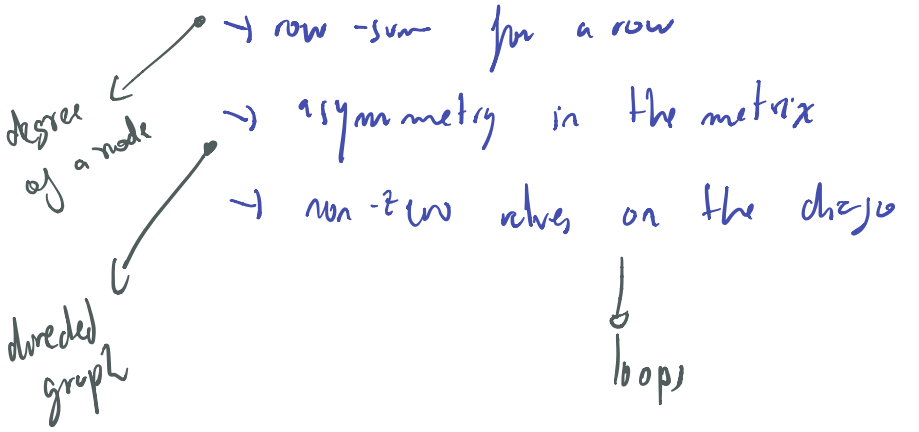
- Question 4: Tools and Applications

(a) How can we use basic Numpy oper to estimate the heart rate in bpm.

```
x = sio.loadmat("_____")['val']
fs = 360 # sampling frequency in Hz
L = 10 # Length of signal in seconds
t = np.linspace(0, L, fs*L)
plt.plot(t, x)

xt = x > 1050
xtd = np.diff(xt, prepend=0)
xtdp = xtd > 0
np.sum(xtdp)/L * 60 # heart rate in Bpm
```

(b) From Adjacency matrix for graphs give an interpretation of:



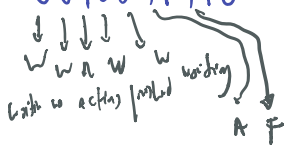
(c) How topological sorting of a graph can be applied in project planning.

If each task is represented by a node, a topological sorting helps to identify dependencies between them. You can't start task jth until rth has done first for a given $i > j$.

(d)

Given this (state, input) → (state, action)

which will occur given input 00100 11 110



start state = "W"

end state = {"F"}

SisAs = {

("W", 0) : ("W", "waiting for a task"),

("W", 1) : ("A" : "acting on a task"),

("A", 0) : ("W", "finished a task"),

("A", 1) : ("F" : "shutting now") }

("W", 0) → ("W", "waiting ...")

("W", 0) → ("W", "waiting...")

("W", 1) → ("A", "acting ...")

("A", 0) → ("W", "finished ...")

("W", 0) → ("W", "waiting ...")

("W", 1) → ("A", "acting")

("A", 1) → ("F", "shutting down")

(e)

Show how the sentence "not (x[0] or not x[1])" can be derived from this grammar.

Difference between terminal and a not-terminal

Terminal symbols are elementary in the way that a production rule will not be able to derivate other symbols from it. Non-terminal symbols can derive to non-terminal & terminal symbols using prod rules.