Question 1: Basic Python

(a) Your friend has found some good code on the internet, but accidentally discarded the indentation when copy-pasting. Fix the problem(s) by inserting appropriate 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.

```
from collections import defaultdict
d = defaultdict(int)
dna = "gattaca"
for s in dna:
    d[s] +1 = 1
```

```
d = {}
dna = "gattaca"
for s in dna:
  if s in d:
      d[s] +1=1
  else:
      d[s] =0
```

(c) Explain duck typing in Python using an example

Duck typing is a way of programming in which the suitability of an object is determined by the presence of certain methods and properties, rather than 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")
```

```
Pl = Plane()
Bi = Bird()
Pr = Person()

for obj in [Pl, Bi, Pr]:
    try:
      obj.fly()
    else:
      print(f"The {obj} doesn't fly")
```

~~Pl = Plane()~~
~~Bi = Bird()~~
~~Pr = Person()~~

(d) Rewrite the following using itertools What is the benefit of this change?

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

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

The main benefit of this change is to improve 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 ε_{mach} such that $1 + \varepsilon_{mach} > 1$. One of the ways to find ε_{mach} in Python is:

```
L = [10**(-n) for n in range(1000)]

for n in range(len(L)):
    if(L[n+1] == 0 & 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

Memoisation 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'.

The properties that a function must have for memoisation to be useful are:

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

(b) Suppose we have a string s containing an arithmetic expression in Python syntax, including constants and variables x and y. Write code which will create a function f(x,y), as in the example below

```
s = "x**2 + y + 1"
# Your code in here
f(3,2)
```

→

```
def f(x,y):
    return eval(s)
```

(c) The following regular expression matches common email addresses. Show how we can use grouping to extract the username portion from a given address adr, i.e. the portion before the symbol @. Show both the changes in the regular expression and the necessary Python code.

```
p = r"[\w\.-]+@\w+(\.\w+)+$"
```

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

(d) Rewrite the following function as a generator (not a generation comprehension). What is the main benefit of using a generator over a function?

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

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

While the first function was returning the list with all values, a generator returns one a value at a time. Therefore it helps to save memory. On top of that, it may be more efficient if the program does not need to use the whole lists of values.

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

```
def f(a,b):
    if a and not b:
        g0()
    else:
        if a:
            g1()
        elif b:
            g2()
        else:
            g3()
```

```
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) Given a Numpy array named X with contents as below (left), use fancy indexing of X to give the array below (right)

```
[[100, 101, 102, 103],
 [110, 111, 112, 113],
 [120, 121, 122, 123],
 [130, 131, 132, 133]]
```

```
[[111, 112, 113],
 [121, 122, 123]]
```

`X[1:3, 1:]`

(b) Describe the rules for broadcasting compatibility in Numpy. Using these rules, say whether this code will work, and if so what is the result, and if not why not.

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

The rules for broadcasting compatibility in Numpy are:

1. If two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side
2. If the shape of two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape
3. If any dim disagrees and neither is 1 raised an error.

As in this example, a.shape = (2,3), b.shape = (2,1), broadcasting will use rule 2.

In this way:

$$a+b = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + \begin{pmatrix} 10 \\ 100 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + \begin{pmatrix} 10 & 10 & 10 \\ 100 & 100 & 100 \end{pmatrix} = \begin{pmatrix} 11 & 12 & 13 \\ 104 & 105 & 106 \end{pmatrix}$$

(c) Given a Pandas Dataframe as shown, named d, what is the result when we call d.groupby("y").mean()?

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

y	z
a	15
b	55

x will be ignored as it is not a numerical value. The values would be grouped by y and z would be the mean of the grouped values

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

The shape of the image with 3 colour channels is 100 x 100 x 3.
In tidy format, the image would be represented as:

ith	jth	R	G	B
0	0	.	.	.
⋮	⋮			
0	99			
⋮	⋮			
99	0			
⋮	⋮			
99	99			

therefore, the shape of the array would be 10000 x 5

(e) Describe Scikit-Learn's GridSearchCV: what inputs does it require, what does it do, what is the result?

GridSearchCV has as input an untrained model, a training set and a dictionary with parameters for the model and possible values for each param.

GridSearchCV trains the model using the train set for each possible combination of parameters and returns the trained model with the highest score.

Question 4: Tools and Applications

(a) Suppose we have an electrocardiogram (ECG) time-series signal of length 20 seconds, stored in a Numpy array. Describe (no need to provide code or calculations) how we can use basic Numpy operations to estimate the heart rate in bpm.

An ECG is a devices that meaoures the electrical activity near the heart at a rate of 360 Hz. This is 360 times per second. In this case we could use to variables to defined the frequency and length of signal in seconds:

$fs = 360$, $L = 10$. Using np.linspace$(0, L, fs*L)$ we can have a list ranging from 0 up to 10 seconds. As in each second we capture the heart beat a total of 360 times. The total signal should have a total of $fs*L$ captures.

Finally, we should find the peaks, one per heart-beat. This can be done by applying a threshold to make signal binary, setting $x_t = 1$ for any very high value and 0 otherwise. Then, we can check the differences, change from point to point. The points where the first differences is positive will give us the peaks. We are searching the amount of peaks per unit of time.

Note that there are several consecutives samples in each unit of time that will have a value of 1 after applying the threshold.

Using np.diff$(x_t, \text{prepend} = 0)$ we can calculate the differences between points. We just one the positive differences.

Now we can calculate the heart rate in bpm by summing all the 1's after applying np.diff and dividing by L and multiplying by 60.

(b) In an adjacency matrix representation for graphs, give an interpretation of the following properties: the row-sum for a given row, asymmetry in the matrix, non-zero values on the diagonal.

Given an adjacency matrix $A_G = (a_{ij})_{i,j=1,\dots,N}$ then:

- The row-sum for a given row is $\sum_{j=1}^{N} a_{ij}$. As $a_{ij} \in \{0, 1\}$ where $a_{ij} = 1$ if and only if there is an edge connecting node i with node j. This sum gives us the number of nodes connected to node i. This is the degree of node i.

$$\text{degree}(i) = \sum_{j=1}^{N} a_{ij}$$

- Asymmetry: If $a_{ij} = a_{ji}$, this is that if node i has an edge to j, then node j has an edge to node i. In case of not having this property it means that there will be at least a j_1 and j_2 such that j_1 has an edge to j_2 but j_2 does not have an edge to j_1. This is just possible in case that G is a directed graph in which at least two nodes have this property.

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

Topological sorting is an ordering of the nodes of the graphs based on the relationships between them. It helps to identify dependencies between them. In project planning, many tasks may be dependent on others, therefore using topological sorting can give a way for prioritizing these tasks.

(d) Given the finite state machine defined below, write out the sequence of states and outputs which will occur when it is executed with input 0010011110

#(state, input) ⟶ (state, action) mapping

```
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 down")}

start_state = "W"
end_state = {"F"}
```

Starting at "W" with input 0 ⟶ ("W": "waiting for a task")

("W", 0) ⟶ ("W": " " " " " ")
("W", 1) ⟶ ("A": "acting on a task")
("A", 0) ⟶ ("W" : "finished a task")
("W", 0) ⟶ ("W" : "waiting for a task")
("W", 1) ⟶ ("A" : "Acting on a task")
("A", 1) ⟶ ("F", "shutting down")

(e) Consider the grammar below, where <expr> is the start symbol. Show how the sentence not (x[0] or not x[1]) can be derived from it. What is the difference between a terminal and a non-terminal?

Terminal symbols are elementary in the way that no production rule will derive any other terminal or non-terminal from it. On the other hand, non-terminal symbols have production rules that derives to other terminal or non-terminal.

<expr> ::= (<expr> <biop> <expr>) | <uop> <expr> | <var> | <const>
<biop> ::= and | or
<uop> ::= not
<var> ::= x[0] | x[1] | x[2]
<const> ::= True | False

starting at <expr> <biop> <expr>
(1) (2) (3)

(1) <expr> → <var> → x[0]
(2) <biop> → OR
(3) <expr> → <uop> <expr> → NOT <var> → NOT x[1]