Question 1: Basic Python

(a)
```
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
```

Grading: 1 for an effort, 3 for almost perfect.

(b)
```
d = {}
dna = "gattaca"
for s in dna:
    try:
        d[s] += 1
    except KeyError:
        d[s] = 1
```

Answer: +2 for creating a dict. +2 for concept of dealing with missing values. +1 for correct code.

(c) Duck typing means the same code can run with different types, as long as they have the right behaviour, which is determined at runtime.

```
def hist(s):
    d = collections.defaultdict(int)
    for x in s:
        d[x] += 1
    return d
```

Here, it doesn't matter whether `s` is a list, set, tuple, string, or something else, as long as we can iterate over it. Grading: +3 for explanation, +2 for code.

(d) Answer: with itertools the code structure is constant – a single for-loop with a single indentation.

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

Grading: +2 for benefit. +3 for code.

(e) Machine epsilon is the smallest number which can be represented in Python, or the smallest difference between successive numbers. We could try e.g. `10**-1000` (and see 0.0) and then `10**-10` (and see 0.0000000001), and then use binary search between these, interactively, to see what value of x in `10**-x` is the largest, before it becomes 0.

Grading: +3 for either way of characterising it. +2 for an approach. Or +2 for `np.finfo(float).eps`.

Question 2: Advanced Python

(a) Memoisation means (1) storing the result of every call to a function so that if it is ever called again with the same arguments, the result can be looked-up and returned without re-calculating it, (1) to save time. In order to be useful, (1) the function must be called repeatedly; there must be the possibility that it will be called repeatedly with the same arguments; (1) it must be relatively time-consuming; (1) it must be deterministic.

(b) `f = eval("lambda x, y:  " + s)`. Grading: +2 for lambda; +2 for eval; +1 for +s.

(c)
```
p = r"([\w\.-]+)@\w+(\.\w+)+$"
re.match(p, adr).group(1)
```

Grading: +1 for any Python re code. +1 for something good with re.match. +2 for parentheses in the re.

(d) It can save memory when there are many items. E.g. `filename` could be larger than RAM. We could read one line at a time, calculate a value and yield it, without running out of memory.

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

Grading: +1 for yield. +2 for correct rewrite. +2 for benefit.

(e)
```
def f(a, b):
    {
        (0, 0): g3,
        (0, 1): g2,
        (1, 0): g0,
        (1, 1): g1
        }[(a, b)]()
```

Grading: +2 for main concept, a dispatch dictionary. +2 for correct logic. +1 for syntax and calling the function.

Question 3: Data Science

(a) `X[1:3, 1:4]`. Grading: +2 for any effort. No extra marks for hacks like writing out constants. 5 for working and general code.

(b) We right-align the shapes, and working from the right each dimension is compatible if (a) equal, or (b) one equals 1 or (c) one is missing. Here, we have (2, 3) and (2, 1). So compatible.

```
[[ 11  12  13]
 [104 105 106]]
```

Grading: +1 for right-align. +1 for three rules. +1 for shapes of example. +1 for yes. +1 for result.

(c)

```
    z
y
a  15
b  55
```

Grading: +2 for main concept, a table in y and z. +2 for it being shortened/grouped. +1 for correct grouping/calculation.

(d) 100×100×3. Tidy format: 10000×5 (the columns are x, y, r, g, b). Grading: 2 + 3.

(e) GridSearchCV requires a (1) dictionary input (1) mapping a hyperparameter name to several possible values for that hyperparameter. It then tries (1) all possible combinations of the hyperparameter values. For each it runs a (1) cross-validation to estimate performance. The result is accessed as (1) `m.best_params_`, a new dictionary on the model.

Question 4: Tools and Applications

(a) 1. By visual inspection, choose a suitable threshold t which divides the signal into "peak" and "non-peak" values. 1. Calculate a binary signal using $x > t$. 1. Calculate the first differencse fd, and make a new binary signal using $fd > 0$. 1. This will have a 1 if and only if it is the very start of a peak. Each 1 corresponds to one heartbeat. 1. Take the sum of this array as the number of peaks per 20 seconds. 1. Multiply by 3 to get heart rate in bpm.

(b) The row-sum is the degree of the node corresponding to the node; asymmetry means it is a directed graph; non-zero on the diagonal means self-loops are present. Grading: 2 + 1 for any two answers, +2 for last.

(c) (2) We can represent a project as a directed graph, where a directed edge ij indicates a dependency between tasks i and j (i must be completed before j can begin). (2) Topological sorting will give us a feasible ordering for the tasks, ie a chronological ordering in which we can carry them out, without ever attempting a task whose dependencies have not yet finished. (1) If a project is feasible, then the graph must be acyclic. Therefore, topological labelling is possible.

(d)
```
W waiting for a task
W waiting for a task
A acting on a task
W finished a task
W waiting for a task
A acting on a task
F shutting down
```

Grading: 3 for basics. 2 for recognising that the FSM will stop before end of input.

(e) A non-terminal is a symbol which can map to some production, ie appears on LHS of some production rule. A derivation is not complete if it contains non-terminals.

```
<expr>
not <expr>
not (<expr> or <expr>)
not (<var> or <expr>)
not (x[0] or <expr>)
not (x[0] or not <expr>)
not (x[0] or not <var>)
not (x[0] or not x[1])
```

Grading: +2 for knowing what a derivation is; +1 for correct derivation. T v NT: 1 mark for mentioning angle brackets/similar, but +2 requires LHS of mapping and must not appear in final derivation.