1. Output of cProfile, showing the top 10 most time-consuming functions:

```
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ python -m pstats matmult.prof
Welcome to the profile statistics browser.
matmult.prof% sort tottime
matmult.prof% stats 10
Tue Mar  4 15:31:08 2025    matmult.prof

         1101421 function calls (1101394 primitive calls) in 2.657 seconds

   Ordered by: internal time
   List reduced from 118 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    2.476    2.476    2.657    2.657 matmult.py:1(<module>)
   125250    0.061    0.000    0.119    0.000 /home/linnekst/mambaforge/lib/python3.10/random.py:292(randrange)
   125250    0.035    0.000    0.047    0.000 /home/linnekst/mambaforge/lib/python3.10/random.py:239(_randbelow_with_getrandbits)
   125250    0.020    0.000    0.139    0.000 /home/linnekst/mambaforge/lib/python3.10/random.py:366(randint)
      250    0.015    0.000    0.015    0.000 {built-in method builtins.print}
   375750    0.012    0.000    0.012    0.000 {built-in method _operator.index}
      250    0.009    0.000    0.079    0.000 matmult.py:9(<listcomp>)
      250    0.009    0.000    0.078    0.000 matmult.py:14(<listcomp>)
   158751    0.007    0.000    0.007    0.000 {method 'getrandbits' of '_random.Random' objects}
   125250    0.005    0.000    0.005    0.000 {method 'bit_length' of 'int' objects}


matmult.prof%
```

- Line 1 in matmult.py takes the most time (2.476s total) and should hence be where we start the speed optimization. matmult.py runs all executions at the top level (there is no function defined in the script) which is the cause for the cProfile output attributing the time to Line 1.

Other observations:

- random.randint(), randrange(), and _randbelow_with_getrandbits() each appear 125,250 times, indicating the random number generation also is time consuming.
- List comprehension (line 9 and 14) also takes some time

To analyze the matmult.py with line_profiler we need an actual function in the script to be analyzed. The output shows the time allocated to each of the lines in this function. The results show that the most time-consuming line is line 19, which is the matrix multiplication.

```
Total time: 7.57198 s
File: matmult_modified.py
Function: matrix_multiplication at line 14

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    14                                           @profile
    15                                           def matrix_multiplication():
    16       251         37.9      0.2      0.0      for i in range(len(X)):  # Iterate through rows of X
    17     63000       6990.3      0.1      0.1          for j in range(len(Y[0])):  # Iterate through columns of Y
    18  15750250    1772190.3      0.1     23.4              for k in range(len(Y)):  # Iterate through rows of Y
    19  15687500    5792762.8      0.4     76.5                  result[i][j] += X[i][k] * Y[k][j]
```

2. Investigate the performance of the eulers72.py script. In which line(s) of the script would you start optimizing for speed?

Output from cProfile:

```
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ python -m pstats euler72.prof
Welcome to the profile statistics browser.
euler72.prof% sort tottime
euler72.prof% stats 10
Tue Mar  4 16:31:42 2025    euler72.prof

         161691 function calls in 0.066 seconds

   Ordered by: internal time
   List reduced from 77 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     9999    0.034    0.000    0.044    0.000 euler72.py:22(factorize)
     9999    0.013    0.000    0.058    0.000 euler72.py:51(fast_phi)
    99314    0.008    0.000    0.008    0.000 {built-in method math.sqrt}
        1    0.004    0.004    0.066    0.066 euler72.py:1(<module>)
    32153    0.003    0.000    0.003    0.000 {method 'append' of 'list' objects}
        1    0.002    0.002    0.002    0.002 euler72.py:5(gen_primes)
        5    0.001    0.000    0.001    0.000 {built-in method posix.stat}
    10000    0.001    0.000    0.001    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {built-in method _imp.create_dynamic}
        4    0.000    0.000    0.002    0.000 <frozen importlib._bootstrap_external>:1536(find_spec)
```

- The factorization-function at line 22 and the fast_phi function at line 51 are the most time-consuming functions and where I should start optimizing the script for speed.

Output from line_profiler with one @profile decorator at each function

```
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ kernprof -l -v euler72.py
30397485.0
Wrote profile results to euler72.py.lprof
Timer unit: 1e-06 s

Total time: 0.0061851 s
File: euler72.py
Function: gen_primes at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def gen_primes(n):
     6         1          5.4      5.4      0.1      l = range(2,n)
     7         1          0.5      0.5      0.0      primes = []
     8       999        349.4      0.3      5.6      for j in range(0,len(l)):
     9       998        249.8      0.3      4.0          p = True
    10      2968        851.7      0.3     13.8          for d in primes:
    11      2967       1978.0      0.7     32.0              if(d > sqrt(l[j])):
    12       167         53.3      0.3      0.9                  break
    13      2800       1720.5      0.6     27.8              if(l[j] % d == 0):
    14       830        201.1      0.2      3.3                  p = False
    15       830        222.2      0.3      3.6                  break;
    16       998        413.2      0.4      6.7          if(p):
    17       168        139.3      0.8      2.3              primes.append(l[j])
    18
    19         1          0.5      0.5      0.0      return primes

Total time: 0.0843567 s
File: euler72.py
Function: factorize at line 21

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    21                                           @profile
    22                                           def factorize(n,primes):
    23      9999       1368.0      0.1      1.6      factors = []
    24      9999       1286.4      0.1      1.5      init_n = n
    25     96347      13360.3      0.1     15.8      for p in primes:
    26    118736      27531.0      0.2     32.6          while(n%p == 0):
    27     22389       3923.1      0.2      4.7              n = n/p
    28     22389       5269.9      0.2      6.2              factors.append(p)
    29     96347      24758.1      0.3     29.3          if(p > sqrt(n)):
    30      9999       1458.7      0.1      1.7              break
    31      9999       1718.5      0.2      2.0      if(n > 1):
    32      9596       2236.0      0.2      2.7          factors.append(n)
    33      9999       1446.8      0.1      1.7      return factors

Total time: 0.194258 s
File: euler72.py
Function: fast_phi at line 50

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    50                                           @profile
    51                                           def fast_phi(n,primes):
    52      9999     169224.5     16.9     87.1      factors = factorize(n,primes)
    53      9999       2223.0      0.2      1.1      phi = factors[0]-1
    54     31985       7963.3      0.2      4.1      for i in range(1,len(factors)):
    55     21986       5908.1      0.3      3.0          if(factors[i] == factors[i-1]):
    56      7685       3574.9      0.5      1.8              phi *= (factors[i]-1)*(factors[i])/(factors[i]-1)
    57                                               else:
    58     14301       4122.8      0.3      2.1              phi *= (factors[i]-1)
    59      9999       1241.5      0.1      0.6      return phi
```

- The most time consuming row in the factorize function is row 26, while(n%p == 0) was executed 118736 times.
- The most time consuming row in the fast_phi function is row 52, factors = factorize(n,primes), which used 87% of the time spent by this function.

3. Using NumPy for the matrix multiplication reduces the running time.

```
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ python -m pstats npmatmult.prof
Welcome to the profile statistics browser.
npmatmult.prof% sort tottime
npmatmult.prof% stats 10
Tue Mar  4 16:54:53 2025    npmatmult.prof

         99120 function calls (96458 primitive calls) in 0.187 seconds

   Ordered by: internal time
   List reduced from 972 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      645    0.040    0.000    0.040    0.000 {built-in method posix.stat}
    28/27    0.039    0.001    0.056    0.002 {built-in method _imp.create_dynamic}
        1    0.012    0.012    0.012    0.012 npmatmult.py:13(matrix_multiplication)
      117    0.010    0.000    0.010    0.000 {built-in method marshal.loads}
    28/21    0.004    0.000    0.016    0.001 {built-in method _imp.exec_dynamic}
      313    0.003    0.000    0.049    0.000 <frozen importlib._bootstrap_external>:1536(find_spec)
  239/238    0.003    0.000    0.006    0.000 {built-in method builtins.__build_class__}
      117    0.002    0.000    0.002    0.000 {method 'read' of '_io.BufferedReader' objects}
    11958    0.002    0.000    0.002    0.000 {built-in method builtins.isinstance}
     1401    0.002    0.000    0.003    0.000 /home/linnekst/mambaforge/lib/python3.10/typing.py:986(__setattr__)


npmatmult.prof% quit
Goodbye.
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ nano npmatmult.py
(base) linnekst@linnekst:/mnt/c/Users/linne/advanced_programming/day2-bestpractices-1$ kernprof -l -v npmatmult.py
Wrote profile results to npmatmult.py.lprof
Timer unit: 1e-06 s

Total time: 0.0123502 s
File: npmatmult.py
Function: matrix_multiplication at line 13

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    13                                           @profile
    14                                           def matrix_multiplication():
    15         1      12349.8  12349.8    100.0      result = np.dot(X, Y)
    16         1          0.4      0.4      0.0      return result
```