For each question, paste your solution in the text field under the description.

For programming questions:

- All answers should be provided as working Python code
- You should perform your own tests to verify that the code is correct. However test cases need not be provided with your answer. If you do provide test cases, you should identify those clearly (for example with comments).
- for maximum credits, the provided answer should be in the best possible complexity class. Less efficient answers will be given partial credits.
- It is not necessary to comment on your code, it is more important that the code is clean and easy to read.

The maximum possible total number of points is 65. 24 points correspond to grade 3, 36 points correspond to grade 4, and 48 points correspond to grade 5.

Allowed aids:

- You can use the python interpreter available on this website https://python-introduction.cse.chalmers.se/sandbox.html to test your code. From the Safe Exam Browser, you should click the "Open Python" button on the bottom left of your screen to open it. So will open the self-practice website that we used in the course in another window. On the self-practice website, it is intended for you to use the "Exam Sandbox" exercise block, which you can find at the bottom of the exercise list. Note that your code won't be saved if you switch exercise in the self-practice website, or if you close or reload its window.
- You can refer to your notes and books. **These notes or books must be on paper.**

Disallowed aids: anything else. In particular, it is not allowed to use the following electronic tools.

- AI tools (Copilot, ChatGPT, Llama, ...)
- Communication tools (chat, mail, social media, ...)
- Electronic documentation (Google, stack overflow, python guides or manuals, ...)
- Your own notes in electronic format

Question 1

The period of revolution (T) of a planet is connected with the mass of the sun M and the radius R of its orbit by the following formula, where G is the gravitational constant:

$$T^2 = (4\pi^2/GM)R^3$$

Write a Python function `planet_period(name)` which prints the period of rotation of the planet with the given `name`, if it is a valid planet name. Otherwise, print "Unknown planet name".

You can assume the following data (expressed in SI units--- meters, seconds, etc.):

```
M = 1.991e30
G = 6.67e-11
radii = {
    "mercury": 57.9e9,
    "venus": 108.2e9,
    "earth": 149.6e9,
    "mars": 228e9,
    "jupiter": 779.3e9,
    "saturn": 1427e9,
    "uranus": 2871e9,
    "neptune": 4497e9,
    "pluto": 5913e9
}
```

Hints:

- You can compute π using `math.pi`, square root using `math.sqrt`, and general exponents with the `**` operator.
- To test your program, check the period of revolution for Earth. (Remember that applying the above formula to the above inputs will give a result in number of seconds.)
- You can assume that planet names are input in lowercase.

Answer

```
import math

M = 1.991e30
G = 6.67e-11
radii = {
    "mercury": 57.9e9,
    "venus": 108.2e9,
    "earth": 149.6e9,
    "mars": 228e9,
    "jupiter": 779.3e9,
    "saturn": 1427e9,
    "uranus": 2871e9,
    "neptune": 4497e9,
    "pluto": 5913e9
}

def kepler(R):
    return math.sqrt  ((4* math.pi**2 /(G*M)) * R**3)

# test:
# seconds_per_day = 24*60*60
# print(kepler(radii["earth"]))
def planet_period(name):
  if planet in radii:
      print(kepler(radii[name]))
  else:
      print("Unknown planet")
```

Question 2

Write a function `gene_match(gene,genome)` that searches for a genetic subsequence in a genome. The gene is given as a string. The genome is given as a file name. Both the gene string and the genome file contents are comprised only of the characters `A`, `C`, `G` or `T`. The function should print all the positions in the genome where the gene can be found. Print the starting positions, indexed from zero. For example, if the gene contains `ACA` and the genome file contains `GTACACAGT`, then both 2 and 4 should be printed.

For half credits, you can assume that the genome file can fit in memory.
For full credits, you must deal with files which cannot fit in memory.
In both cases, you can assume that the gene is smaller than the genome.

Hints:

- if `f` is a file, then `f.read(n)` reads `n` characters from `f` and return them as a string. If the file has `m` characters remaining to read and `m<n`, then only `m` characters will be read. In

particular, at the end of the file, `f.read` always returns the string.

- Within the **Exam Sandbox, Question 2**, you can open the file "genome.txt". It has the contents GTACACAGT. You can use this file for testing.

Answer

```
def gene_match(gene,genome_file):
  i = 0
  l = len(gene)
  with open(genome_file) as f:
    ref = f.read(l)
    while True:
      if ref == gene:
        print(i)
      c = f.read(1)
      if c:
        ref = ref[1:]+c
        i=i+1
      else:
        return

gene_match("ACA","genome.txt")
```

Question 3

Write a class called `Kepler`. Its goal is to record planet observations and estimate the radius of their orbits.

The class should have the following methods:

- `__init__(self)` which initializes the attributes
- `observe(self, planet_name, angle, time)`
  This method records an observation of a given planet at a given time, at a given angle in its orbit. You can assume that the angle is given in radians.
  Note: assuming two observations, respectively at times t_1, t_2 and angles a_1, a_2 the one can estimate the period of revolution of the planet to be $2\pi \times (t_1 - t_2)/(a_1 - a_2)$. You can further assume that $t_2 > t_1$ implies $a_2 > a_1$ and that observations are reported in chronological order.
- `get_period(self,planet_name)`
  This method will return an estimation of the planet's period of revolution. You should compute this estimation averaging the estimated period of revolution for all *consecutive* observations of the given planet. (Use the above formula for each consecutive pair of observations and take the average.) If there are not enough observations, this method should return `None`.
- `get_radius(self,planet_name)`
  This method will return an estimation of the planet's radius using the Kepler formula from Question 1 and the period estimated by `get_period`.
  If there are not enough observations, this method should return `None`.
- `print_planets(self)`
  This method will print a table with every planet's name, estimated period and estimated radius.
  If there are not enough observations for any given planet, print `None` for period and radius.

Answer

```
import math

class Kepler():
    # 0 points
    def __init__(self):
        self.observations = {}
    # 5 points
    def observe(self, planet_name, position, time):
        planet_obs = self.observations.setdefault(planet_name,[])
        planet_obs.append((position,time))
    # 10 points
    def get_period(self,planet_name):
        planet_obs = self.observations.get(planet_name, [])
        if len(planet_obs) < 2:  # 2 points for the test
            return None
        else:
            ts = []
            for i in range(len(planet_obs) - 1): #
                (a1,t1) = planet_obs[i]
                (a2,t2) = planet_obs[i+1]
                a = a2-a1
                ts.append(2*math.pi*(t2-t1)/a)
            return sum(ts) / len(ts)
    # 5 points
    def get_radius(self,planet_name):
        t = self.get_period(planet_name)
        if not t: return None  # 2 points for the test
        return (t**2 * G * M / (4 * math.pi ** 2)) ** (1/3) # 3 points for the formula
    # 5 points (for doing the loop nicely)
    def print_planets(self):
        for p in self.observations:
            print(p, self.get_period(p), self.get_radius(p))

M = 1.991e30
G = 6.67e-11
k = Kepler()

k.observe("earth", 0, 0)
k.observe("earth", 2*math.pi, 31548499)
k.observe("x1", 0, 0)
k.observe("x1", 1, 100)
k.observe("x1", 2, 150)
print(k.get_period("x1"))
print(k.get_radius("x1"))
k.print_planets()
```

Question 4

Assume that a function $P(x)$ processes an input x of size n in kn^2 seconds, where k represents the speed of the computer. (That is P implements an algorithm of quadratic complexity.)

With your current hardware, you process an input size one million in one day. You consider procuring new hardware which is 10 times faster. How big of an input would you be able to process in a day with this new hardware?

(Just write your answer, not the reasoning behind it.)

Answer

```
sqrt(10)*1e6
```