1. The code below implements a rate limiter to only allow 5 requests per second. What are the problems with the code?

```
from time import sleep, time from threading import Thread

class RateLimiter:
def init(self, requests_per_second):
self.rate = requests_per_second
self.tokens = requests_per_second
self.worker = Thread(target=self.add_tokens)
self.worker.start()

def add_tokens(self):
    while True:
        self.tokens = self.rate
        sleep(1)

def check(self):
    if self.tokens > 0:
        self.tokens -= 1
        return True

    return False

def stop(self):
    self.worker.join()
def send_requests(idx, rate_limiter):
for i in range(20):
sleep(0.05)
status = “accepted” if rate_limiter.check() else “rejected”
print(f”Request {idx+1}-{i+1} at {time()} is {status}”)

if name == “main”:
rate_limiter = RateLimiter(5)
workers = list()
for i in range(5):
workers.append(Thread(args=[i, rate_limiter], target=send_requests))
workers[-1].start()
for worker in workers:
worker.join()
rate_limiter.stop()
</code>
```

2. We wrote a Grade desciptor to reuse across our codebase. However, we've encountered repeated values when using multiple StudentGrades instances.

   Why does that happen, and how can we fix the issue?

```
class Grade: def __init__(self): self.__value = 0

def __get__(self, instance, instance_type):
    return self.__value

def __set__(self, instance, value):
    if not (0 <= value <= 100):
        raise ValueError("Grade must be between 0 and 100")
    self.__value = value
class StudentGrades:
algorithms = Grade()
data_structures = Grade()
databases = Grade()
</code>
```

3. The code below implements a Contact with a name and arbitray key-value attibutes, but it raises an exception when un.

   What are the problems, and how would you fix them??

```
class Contact: def __init__(self, name, default_fn=lambda _: None): self.name = name self.vals = dict() self.default_fn =
default_fn

def __getattribute__(self, item):
    if item == "name": return self.name
    return self.vals[item] if item in self.vals else self.default_fn()

def __setattr__(self, item, value):
    if item == "name":
        self.name = value
        return
    self.vals[item] = value

def __repr__(self):
    lines = [f"Name: {self.name}"]
    if self.vals:
        lines.append("Attributes:")
        lines.extend(f"- {k}: {v}" for k, v in self.vals.items())
    return "\n".join(lines)
john = Contact("John Smith")
john.age = 31
print(f"{john!r}")
</code>
```

4. Simple Line Wrapping

   Exercise:
   We are building a word processor and we would like to implement a “word-wrap” functionality.

   Question:
   We are building a word processor and we would like to implement a “word-wrap” functionality.
   Given a maximum number of characters in a line followed by a list of words, retun a collection of stings where each sting element represents a line that contains as many words as possible, with the words in each line being concatenated with a space.
   The length of each sting must not exceed the maximum character length per line.

   Approach:
   Loop through each word, testing if new length would exceed width before adding word. Using efficient sting building (i.e. via a buffer).

   Language: Python 3

5. Reflow and Justify

   Exercise
   We are building a word processor and we would like to implement a “word-wrap” functionality.

   Question
   Given an aray containing lines of text and a new maximum width, re-flow the text to fit the new width. Each line should have the exact specified width. If any line is too shot, inset spaces between words as equally as possible until it fits.

   Approach
   Candidate ran out of time before getting to this question

   Language: Python 3