1. Cloud service

As you can see, the `CloudService` class directly depends on Google-specific classes: `GoogleCredentials`, `GoogleServiceProvider`, and `GoogleStorage`. You want to remove this direct dependency. However, you can't change the original classes provided by Google. In fact, you don't even have access to the source code of those classes. How do you solve this? Refactor your code to remove the direct dependency.

Protocol classes are really useful in this case, because they allow you to define the interface of what you expect in your class, method, or function. Python's structural typing system then takes care of making sure that the types match without there being an explicit relationship.

Making this change here is relatively straightforward. You can see a solution in `solution_1.py`. I've created Protocol classes for each of the three Google-specific classes and used those in the `CloudService` class instead. You can now remove the Google-specific import.

2. Sending emails

Consider the following function that sends an email:

```
def send_email(
    message: str, to_address: str, from_address: str = DEFAULT_EMAIL
) -> None:
    server = SMTP()
    server.connect(HOST, PORT)
    server.login(LOGIN, PASSWORD)
    server.sendmail(from_address, to_address, message)
    server.quit()
```

The `send_email` function depends on the `SMTP` type from `smtplib`. Removing the dependency here is a bit more complicated because at the moment, `send_email` creates an instance of `SMTP` directly. So the first step is to change this so that `send_email` now relies on dependency injection:

```
def send_email(
    server: SMTP,
    message: str, to_address: str, from_address: str = DEFAULT_EMAIL
) -> None:
    server.connect(HOST, PORT)
    # etc
```

Now that the function no longer creates the object, we can add a Protocol class that defines the interface of an SMTP server and use that instead of the `SMTP` type. For the full solution, see `solution_2.py`.