Exercises on Python classes

Exercise 1

Write a Python program to create a `Vehicle` class with `name`, `seats` and `trunk_space` instance attributes.

Once you do, this code

```
car = Vehicle("Dodge", 2, 1)
print(f'The {car.name} has {car.seats} seats and {car.trunk_space} units of trunk space')
```

should produce this output:

```
The Dodge has 2 seats and 1 units of trunk space
```

Exercise 2

Add a `__str__` method to the class to that this code

```
car = Vehicle("Jeep", 4, 2)
print(car)
```

produces this output:

```
Jeep
```

Exercise 3

Add a method `at_capacity`, which returns `True` if the Vehicle is overfull and `False` otherwise. The method should take two arguments: `passengers` and `suitcases`. Passengers take up one seat each. Suitcases take up one seat or one trunk space unit - and you cannot put passengers in the trunk.

Once you do, this code

```
car = Vehicle("Volvo", 4, 3)
print(f'It is {car.at_capacity(5, 1)} that the {car.name} is over capacity')
```

should produce this output:

```
It is True that the Volvo is over capacity
```

Exercise 4

Create a `Bus` class that inherits from the `Vehicle` class. A `Bus` is different from a `Vehicle` in that passengers can sit with a suitcase in their lap, but cannot put any in the trunk. So add an `at_capacity` method that overrides the `Vehicle` method of the same name, so that this difference is taken into account.

Once you do, this code:

```python
school_bus = Bus("School bus", 22, 10)
print(f'The {school_bus.name} has {school_bus.seats} seats and {school_bus.trunk_space} unit
print(f'It is {school_bus.at_capacity(22, 11)} that the {school_bus.name} is over capacity'
```

should produce this output:

```
The School bus has 22 seats and 10 units of trunk space
It is False that the School bus is over capacity
```

Exercise 5

Define a property that must have the same value for every class instance (object)

Define a class attribute `fuel` to the `Vehicle` and `Bus` classes with a default values `"gasoline"` and `"diesel"`. I.e., Every Bus should use diesel and every Vehicle uses gasoline.

Once you do, this code:

```python
tour_bus = Bus("Dodge", 30, 20)
convertible =  Vehicle("Saab", 4, 1)
print(f'The {tour_bus.name} uses {tour_bus.fuel}')
print(f'The {convertible.name} uses {convertible.fuel}')
```

should produce this output:

```
The Dodge uses diesel
The Saab uses gasoline
```

Exercise 6

Look up the `super()` builtin function. What attributes does this class have?

```python
class Truck(Vehicle):

    def __init__(self, name, passengers, trunk_units, trailer_units):
        super().__init__(name, passengers, trunk_units)
        self.trailer_units = trailer_units
```

What will this print:

```python
sixteen_wheeler = Truck('Man', 2, 2, 542)
print(sixteen_wheeler)
```

Exercise 7

Add a `load` method to this function:

```python
class AutoTransport(Vehicle):

    def __init__(self, name, passengers, trunk_units):
        super().__init__(name, passengers, trunk_units)
        self.loaded_cars = []

    def __str__(self):
        return f"{self.name} with: {', '.join(map(str, self.loaded_cars))} loaded"
```

Once you do, this code:

```
auto_trans = AutoTransport('Man', 2, 2)
auto_trans.load(Vehicle('Mustang', 4, 1))
auto_trans.load(Vehicle('Charger', 4, 1))
auto_trans.load(Vehicle('Corvette', 4, 1))
auto_trans.load(Vehicle('Challenger', 4, 1))
print(auto_trans)
```

should produce this output:

```
Man with: Mustang, Charger, Corvette, Challenger loaded
```

Exercise 8

Now you have learned a little about classes, you can try to create your own classes.

Create a class called DNA that takes a string as input. The class should have the following methods:

- `__init__` that takes a string as input and stores it as an attribute
- `count_nucleotides` that counts the number of each nucleotide in the string and returns a dictionary with the counts
- `gc_content` that calculates the GC content of the string and returns the value
- `codons` that returns a list of codons in the string
- `translate` that translates the string into a protein sequence and returns the protein sequence as a string
- `reverse_complement` that returns the reverse complement of the string

A subclass called RNA that inherits from DNA. The RNA class should have the following methods:

- `__init__` that takes a either a DNA string or an RNA string as input and stores it (translated) to/as an RNA string as an attribute
- `reverse_complement` (updated to RNA) that returns the reverse complement of the string
- `codons` (updated to RNA) that returns a list of codons in the string
- `translate` (updated to RNA; RNA has U instead of T but you can just translate right back to use the old table) that translates the string into a protein sequence and returns the protein sequence as a string.