Classes

Solve the following exercises and upload your solutions to Moodle until the specified due date.

Important Information!

Please try to *exactly match the output* given in the examples (naturally, the input can be different). We are running automated tests to aid in the correction and grading process, and deviations from the specified output lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

Make sure to use the *exact filenames* that are specified for each individual exercise.

Also, use the provided unit tests to check your scripts before submission (see the slides Handing in Assignments on Moodle). Feel free to copy the example text from the assignment sheet, and then change it according to the exercise task to match the output as best as possible.

In this assignment, it is of *particular importance* to wrap the printing that you see in the example outputs in `if __name__=='__main__:'`, as your exercises essentially only consist of a function definition. Example - let's say the task is to write a function that doubles the float value that is passed:

```python
def double(var: float) -> float:
    return var*2

if __name__ == '__main__':
    print(double(3.4))
```

Unless explicitly stated otherwise, you can assume correct user input and correct arguments.

You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

You are allowed in this assignment to implement additional or multiplication attributes and methods as long as the original interface remains unchanged.

Exercise 1 – Submission: `a8_ex1.py` 40 Points

Create a class `Voltage` that models voltage values. The class has the following instance attributes:

- `current: float`
  Represents the current value in the electrical circuits.
- `resistance: float`
  Represents the resistance value in the electrical circuits.

The class has the following instance methods:

- `__init__(self, current: float, resistance: float)`
  Sets both instance attributes.
- `print(self)`
  Prints this `Voltage` object to the console. Format: `<sign><current> amps + <resistance> ohms`, where `<current>` is the flow current in the electrical circuits, `<resistance>` is the value of the resistors in the circuit), and `<sign>` is either the plus character `+` if the current is in the orientation of the voltage source or the minus character `-` if the current is opposite of the orientation of the voltage source. The `<sign>` thus depends on the orientation of the voltage source. Example: `-2.0 amps + 20 ohms`
- `volt(self) -> float`
  Returns the voltage value of this `Voltage` object. The voltage value is defined as $current \times resistance$, where *current* and *resistance* are the flow current and resistance in electrical circuits, respectively.
- `increase_resistance(self, delta: float)`
  Increases the `resistance` value of this `Voltage` object by `delta` in-place (no return value). If `delta` is not a `float` or `int`, raise a `TypeError: Please provide float value instead of <wrong_type>`. Use `type(obj).__name__` to get the type of some object `obj`.

Moreover, add the following static method (`@staticmethod`):

- `add_all(volt_obj: "Voltage", *volt_objs: "Voltage") -> "Voltage"`
  Adds the resistance values of `volt_obj` and `*volt_objs` together and returns a new `Voltage` object containing the common current value (must be equal for all objects) and the sum of the resistance values. None of the input objects shall be changed, i.e., all attributes in `volt_obj` and `*volt_objs` must remain the same.
  - If any of `volt_obj` or `*volt_objs` are not instances of class `Voltage`, raise a `TypeError: Can only add objects of type 'Voltage'`.
  - All current values in `volt_obj` and `*volt_objs` must be the same. If they are not equal, raise a `ValueError: The current must be equal`.

Example program execution:

```
c1 = Voltage(-5.0, 20)
c1.print()
c2 = Voltage(3.0, 4.0)
c2.print()
print(c2.volt())
```

Example output:

```
-5.0 amps + 20 ohms
+3.0 amps + 4.0 ohms
12.0
```

Another example program execution:

```
c1 = Voltage(10.0, 2.0)
c1.print()
c2 = Voltage(10.0, 5.0)
c1.increase_resistance(5)
c1.print()
c_sum = Voltage.add_all(c1, c1, c2, Voltage(10.0, 5.0))
c_sum.print()
c1.print()

# will fail
Voltage.add_all(100.0)
c1.increase_resistance('abc')

# different currents, will fail
c3 = Voltage(15.0, 22.0)
print(Voltage.add_all(c1, c3).resistance)
```

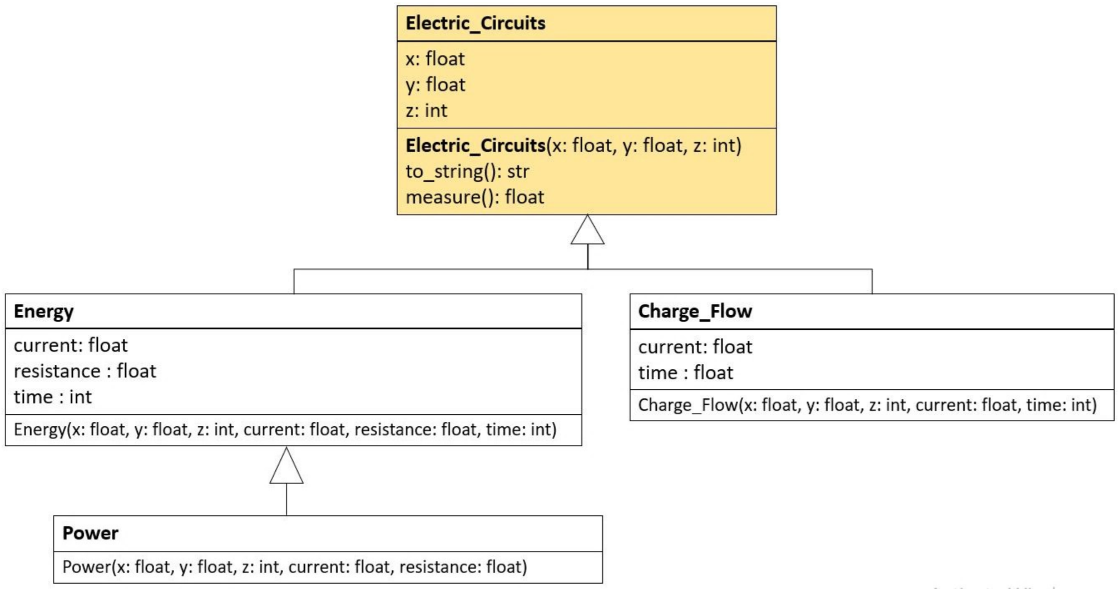
Example output (error message is up to you and may differ):

```
+10.0 amps + 2.0 ohms
+10.0 amps + 7.0 ohms
+10.0 amps + 24.0 ohms
+10.0 amps + 7.0 ohms

TypeError: Can only add objects of type 'Voltage'

TypeError: Please provide float value instead of str

ValueError: The current must be equal
```

Hint: In the `add_all()` method, you must not change the given objects. Instead, create a new `Voltage` object (choose an appropriate initialization for its properties) that you can manipulate and return.

Exercise 2 – Submission: `a8_ex2.py` 15 Points

You are given the following class hierarchy that models the flow of current in electrical circuits. You will have to implement the classes in this and the following exercises.


Electric_Circuits
x: float
y: float
z: int
Electric_Circuits(x: float, y: float, z: int)
to_string(): str
measure(): float
Energy
current: float
resistance : float
time : int
Energy(x: float, y: float, z: int, current: float, resistance: float, time: int)
Charge_Flow
current: float
time : float
Charge_Flow(x: float, y: float, z: int, current: float, time: int)
Power
Power(x: float, y: float, z: int, current: float, resistance: float)

In this exercise, you have to implement the class `Electric_Circuits`, which represents the base class for measuring three parameters. The class has the following instance attributes:

- `x: float`
  The current in the circuit.
- `y: float`
  The resistance in the circuit.
- `z: int`
  The time in the circuit needed to compute the measure.

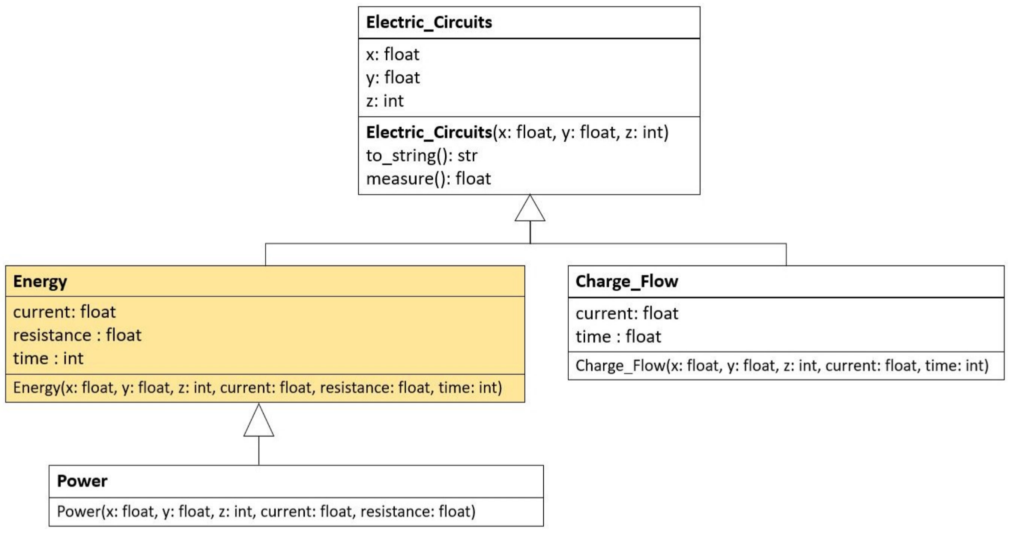
The class has the following instance methods:

- `__init__(self, x: float, y: float, z: int)`
  Sets both instance attributes.
- `to_string(self) -> str`
  Returns a string representation of the form
  `"Electric_Circuits: x=<x_value>, y=<y_value>, z=<z_value>"`, where `<?_value>` represents the value of the corresponding attribute.
- `measure(self) -> float`
  Returns the measure of the electric circuits as a float, which must be implemented by all concrete subclasses. In the `Electric_Circuits` class, a `NotImplementedError` is raised.

Hint: In the `to_string` method, you need the name of the class. While this could be hard-coded, you could also use `type(x).__name__` to get the name of the type/class of some object `x`. This has the benefit that instances of any subclasses will also return their correct names. Alternatively, you can write a helper method that returns the name, and you override it in the subclasses.

Exercise 3 – Submission: `a8_ex3.py` 15 Points

You are given the same class hierarchy as in the previous exercise that models the flow of current in an electric circuit.


Electric_Circuits
x: float
y: float
z: int
Electric_Circuits(x: float, y: float, z: int)
to_string(): str
measure(): float
Energy
current: float
resistance : float
time : int
Energy(x: float, y: float, z: int, current: float, resistance: float, time: int)
Charge_Flow
current: float
time : float
Charge_Flow(x: float, y: float, z: int, current: float, time: int)
Power
Power(x: float, y: float, z: int, current: float, resistance: float)

In this exercise, you have to implement the concrete subclass `Energy`, which represents an energy measure and extends the base class `Electric_Circuits`. The class has the following additional instance attributes:

- `current: float`
  The current in the circuit.
- `resistance: float`
  The resistance in the circuit.
- `time: int`
  The time in the circuit needed to compute energy.

The class has the following instance methods (reuse code from superclasses to avoid unnecessary code duplication):

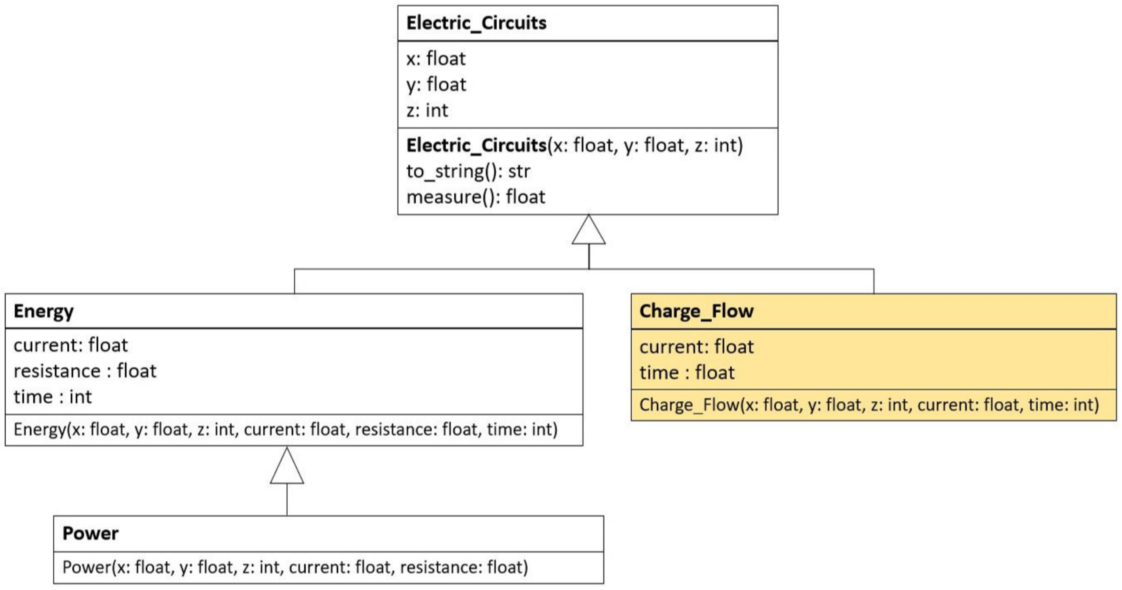
- `__init__(self, x: float, y: float, z: int, current: float,`
  `resistance: float, time: int)`
  Sets both instance attributes (in addition to the attributes of the base class `Electric_Circuits`).
- `to_string(self) -> str`
  Returns a string representation of the form `"Energy: x=<x_value>, y=<y_value>, z=<z_value>, current=<current_value>, resistance=<resistance_value>, time=<time_value>"`, where `<?_value>` represents the value of the corresponding attribute.
- `measure(self) -> float`
  Returns the measure of the energy, i.e., $\texttt{current}^2$ `* resistance * time`, as a float.

Hints:

- You can import the previous exercise as module to avoid having to copy the entire class hierarchy. For example, you can write `from ex8_ex3 import Electric_Circuits`.
- Use `super().some_method()` to access the `some_method` implementation of the superclass.

Exercise 4 – Submission: `a8_ex4.py` 15 Points

You are given the same class hierarchy as in the previous exercise that models the flow of current in electrical circuits.


Electric_Circuits
x: float
y: float
z: int
Electric_Circuits(x: float, y: float, z: int)
to_string(): str
measure(): float
Energy
current: float
resistance : float
time : int
Energy(x: float, y: float, z: int, current: float, resistance: float, time: int)
Charge_Flow
current: float
time : float
Charge_Flow(x: float, y: float, z: int, current: float, time: int)
Power
Power(x: float, y: float, z: int, current: float, resistance: float)

In this exercise, you have to implement the concrete subclass `Charge_Flow`, which represents a charge flow measure and extends the base class `Electric_Circuits`. The class has the following additional instance attributes:

- `current: float`
  The current in the circuit.
- `time: int`
  The time in the circuit needed to compute charge flow.

The class has the following instance methods (reuse code from superclasses to avoid unnecessary code duplication):

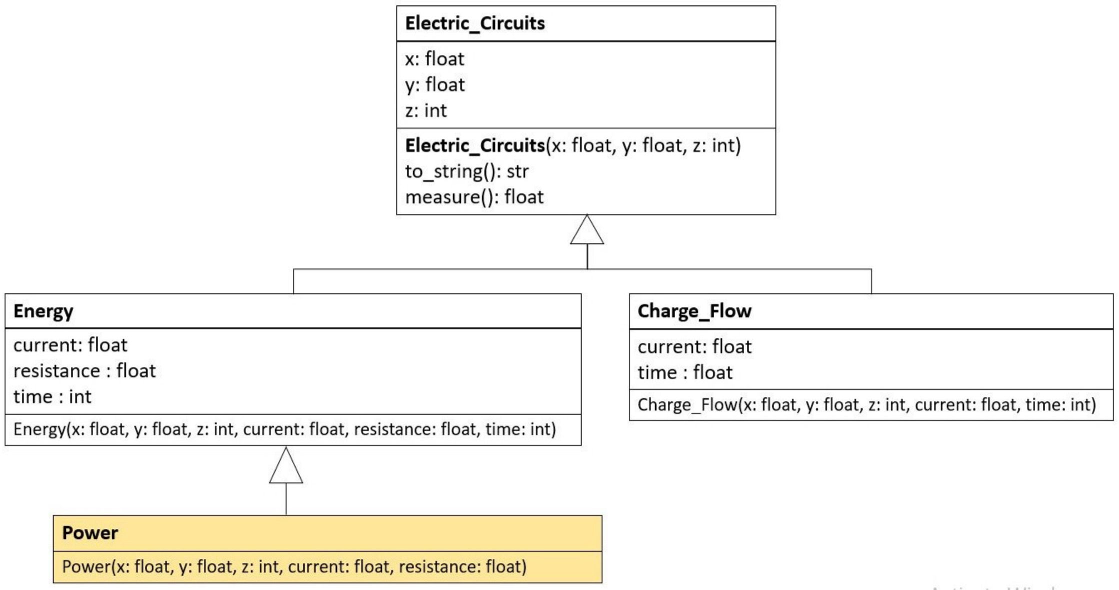
- `__init__(self, x: float, y: float, z: int, current: float, time: int)`
  Sets the instance attribute (in addition to the attributes of the base class `Electric_Circuits`).
- `to_string(self) -> str`
  Returns a string representation of the form `"Charge_Flow: x=<x_value>, y=<y_value>, z=<z_value>, current=<current_value>, time=<time_value>"`, where `<?_value>` represents the value of the corresponding attribute.
- `measure(self) -> float`
  Returns the measure of the charge flow, i.e., `current * time`.

Hints:

- You can import the previous exercise as module to avoid having to copy the entire class hierarchy. For example, you can write `from a8_ex3 import Electric_Circuits`.
- Use `super().some_method()` to access the `some_method` implementation of the superclass.

Exercise 5 – Submission: `a8_ex5.py` 15 Points

You are given the same class hierarchy as in the previous exercise that models the flow of current in electrical circuits.


Electric_Circuits
x: float
y: float
z: int
Electric_Circuits(x: float, y: float, z: int)
to_string(): str
measure(): float
Energy
current: float
resistance : float
time : int
Energy(x: float, y: float, z: int, current: float, resistance: float, time: int)
Charge_Flow
current: float
time : float
Charge_Flow(x: float, y: float, z: int, current: float, time: int)
Power
Power(x: float, y: float, z: int, current: float, resistance: float)

In this exercise, you have to implement the concrete subclass `Power`, which represents a power measure and extends the base class `Energy`. The class has *no* additional instance attributes.

The class has the following instance methods (reuse code from superclasses to avoid unnecessary code duplication):

- `__init__(self, x: float, y: float, z: int, current: float, resistance: float)`
  Internally, the attribute `time` of the base class `Energy` is used (this attribute should be set to a value of 1).
- `to_string(self) -> str`
  Returns a string representation of the form `"Power: x=<x_value>, y=<y_value>, z=<z_value>, current=<current_value>, resistance=<resistance_value>, time=<time_value>"`, where `<?_value>` represents the value of the corresponding attribute.
- `measure(self) -> float`
  Returns the measure of the power, i.e., `current`^{2} `* resistance`, as a float.

Hints:

- You can import the previous exercise as module to avoid having to copy the entire class hierarchy. For example, you can write `from a8_ex4 import Energy`.
- The above requirement "reuse code from superclasses to avoid unnecessary code duplication" is especially relevant in this exercise. Depending on your implementation, it might be that you do not need to override any methods of the superclass `Energy`.

Combined Examples for Exercises a8_ex2, a8_ex3, a8_ex4 and a8_ex5

Example program execution:

```python
s = Electric_Circuits(4, 9, 1)
print(s.to_string())
print("Electric Circuits measurement:", s.measure())

e = Energy(1, 2, 3, 0.5, 20, 10)
print(e.to_string())
print("Energy measurement:", e.measure(), "joules")

c = Charge_Flow(5, 2, 2, 0.5, 10)
print(c.to_string())
print("Charge Flow measurement:", c.measure(), "coulombs")

p = Power(5, 2, 2, 0.5, 10)
print(p.to_string())
print("Power measurement:", p.measure(), "watts")
```

Example output:

```
Electric_Circuits: x=4, y=9, z=1
raise NotImplementedError

Energy: x=1, y=2, z=3, current=0.5, resistance=20, time=10
Energy measurement: 50.0 joules

Charge_Flow: x=5, y=2, z=2, current=0.5, time=10
Charge Flow measurement: 5.0 coulombs

Power: x=5, y=2, z=2, current=0.5, resistance=10, time=1
Power measurement: 2.5 watts
```