Exercises

Set 10

DM562 Scientific Programming
DM857 Introduction to Programming
DS830 Introduction to Programming

1 Time management (DM857, DS830)

A team developing a project realised that they need to be able to represent points in time.

1. Initially, the teams decides that representing points in time as hour, minute, and second is enough for the project.

   Implement a class `TimeStamp` whose objects are points in time represented by hours, minutes, and seconds. Decide which attributes this class should have and which getters and setters should be available for these attributes. The class should also provide the following definitions:

   (a) A constructor that takes three optional arguments, corresponding (respectively) to the hours, minutes, and seconds of the required timestamps (pick a default value).

   (b) A static method `is_valid(hours:int, minutes:int, seconds:int) -> bool` that checks
   whether its given arguments can be passed along to the constructor.

   (c) Methods `skip_second(self)`, `skip_minute(self)`, and `skip_hour(self)` that add one second, one minute, and one hour, respectively, to the timestamp (assume that 23:59:59 is always followed by 00:00:00).

   (d) A method `skip(self, time:TimeStamp)` that adds the amount of time described in `time` to the timestamp.

   (e) A method `copy(self) -> TimeStamp` that returns a copy of this timestamp.

   (f) A method `__eq__(self,other) -> bool`[1] that determines whether this timestamp represents the same point in time of `other`.

   (g) Methods `__lt__(self,other) -> bool` and `__le__(self,other) -> bool` that determine whether this timestamp represents a point in time before or equal to that of `other`. [2]

   (h) A method `__str__(self) -> str`[3] that returns a textual representation of the time described by this timestamp in the format HH:MM:SS (you may omit the zeros usually added for alignment, e.g., return `'0:0:0'` instead of `'00:00:00'`).

[1]Method `__eq__(self,other)` is called by Python to implement `self == other`. You can find the documentation at `https://docs.python.org/3/reference/datamodel.html#object.__eq__`

[2]Method `__lt__(self,other)` is called by Python to implement `self <= other` and `__le__(self,other)` to implement `self <= other`. The symmetric operators by default use < and <= swapping the argument. To change this behaviour, a class needs to implement `__gt__(self,other)` and `__ge__(self,other)`.

[3]Method `__str__(self)` is called by `str(object)` and the built-in functions `format()` and `print()` to compute the “informal” or nicely printable/readable string representation of an object. You can find the documentation at `https://docs.python.org/3/reference/datamodel.html#object.__str__`.

    (i) A method `__repr__(self) -> str`[4] to return a Python-like representation of this timestamp (e.g., `'TimeStamp(12,0,0)'`).

   Test your code with `doctest` and a small client program (it can be in the same module).

2. As the project grows, the team concluded that in some cases they need to enrich timestamps with information about the date, represented as a year, month, and day.

   Implement a class `Date` that represents a point in time in a particular date, including information about the year, month, day and timestamp. Decide which attributes this class should have and which getters and setters should be available for these attributes.

    (a) A constructor with four arguments for the year, month, day, time, respectively. The last argument must be optional and defaults to midnight.

    (b) A static method `is_valid(year:int, month:int, day:int) -> bool` that checks whether its given arguments can be passed along to the constructor.

    (c) Methods `skip_day(self)`, `skip_month(self)`, and `skip_year(self)` that add one day, one month, and one year, respectively, to this timestamp (remember leap years).

    (d) A method `skip_time(self, time:TimeStamp)` that adds the amount of time described in `time` to this object (adding a second to 23:59:59 carries over to the days and so on).

    (e) A method `copy(self) -> Date` that returns a copy of this timestamp.

    (f) Methods `__eq__(self,other)`, `__lt__(self,other)`, and `__le__(self,other)` (all with return type `bool`) for comparing this timestamp and `other`.

    (g) A method `__str__(self) -> str` that returns a textual representation of the date and time described by this object in the (ISO) format YYYY-MM-DD HH:MM:DD (e.g., `'2012-11-02 14:15:00'`).

    (h) A method `__repr__(self) -> str` that returns a Python-like representation of this date and time.

   Test your code with `doctest` and a small client program (it can be in the same module).

2 A class for matrices (DM562)

Implement a class `Matrix` whose objects represents matrices of numbers (`typing.Number`). Decide which attributes this class should have and which getters and setters should be available for these attributes. The class should also provide the following methods:

1. A constructor with two arguments `m` and `n` for creating a new zero matrix of size `m` by `n` (`m` rows and `n` columns).

2. Static methods

   - `row_vector(v:List[Number]) -> Matrix` that returns a `len(v)` by `1` matrix filled with the elements of `v`;
   - `column_vector(v:List[Number]) -> Matrix` that returns a `1` by `len(v)` matrix filled with the elements of `v`;

[4] Method `__repr__(self)` is called by the `repr()` built-in function to compute the "official" string representation of an object. If at all possible, this should look like a valid Python expression that could be used to recreate an object with the same value (given an appropriate environment). You can find the documentation at `https://docs.python.org/3/reference/datamodel.html#object.__repr__`.

- `diagonal(v:List[Number]) -> Matrix` that returns a `len(v)` by `len(v)` matrix with the elements of `v` on its diagonal and `0` everywhere else.

3. A method `__eq__(self, other) -> bool`[5] that determines whether this timestamp represents the same point in time of `other`.
4. Methods
   - `is_row_vector(self) -> bool` that returns whether this matrix is a row vector;
   - `is_column_vector(self) -> bool` that returns whether this matrix is a column vector;
   - `is_diagonal(self) -> bool` that returns whether this matrix is diagonal;
   - `is_upper_triangular(self)->bool` that returns whether this matrix is upper triangular;
   - `is_lower_triangular(self)->bool` that returns whether this matrix is lower triangular.
5. A method `copy(self) -> Matrix` that returns a new matrix with the same entries of this one.
6. A method `transpose(self) -> Matrix` that returns the transpose of this matrix.
7. A method `add_scalar(self, s:Number)` that increments every element of this matrix by `s`.
8. A method `multiply_scalar(self, s:Number)` that multiplies every element of this matrix by `s`.
9. A method `add_matrix(self, m:Matrix) -> Matrix` that returns the (entrywise) sum of this matrix and `m` in case their dimensions are compatible.
10. A method `multiply_matrix(self, m:Matrix) -> Matrix` that returns the product of this matrix and `m` in case their dimensions are compatible. Use this method to define a method `__matmul__(self, other) -> Matrix` which is called by Python to implement the binary operator for matrix multiplication `@`.
11. A method `determinant(self) -> Number` that returns the determinant of this matrix.
12. A method `gaussian_elim(v:List[Number]) -> List[Number]` that returns a vector `w` such that this matrix multiplied by `w` yields `v`, computed by Gaussian elimination (this method should raise an error if there is not exactly one solution).
13. Methods
    - `__getitem__(self, key:Tuple[int,int]) -> Number`[6] that takes a pair `row,column` as `key` and returns the entry at `row` and `column` of this matrix.
    - `__setitem__(self,key:Tuple[int,int],value:Number)` that takes a pair `row,column` as `key` and replaces the entry at `row` and `column` of this matrix with `value`;

    to support the square bracket notation (e.g., `m[1,2] = 0`).
14. A method `__str__(self) -> str` that returns a textual representation of this matrix.

Test your code with `doctest` and a small client program (it can be in the same module).

[5]Method `__eq__(self, other)` is called by Python to implement `self == other`. You can find the documentation at `https://docs.python.org/3/reference/datamodel.html#object.__eq__`

[6]Method `__getitem__` is called to implement the evaluation of `self[key]`. If key is of an inappropriate type, `TypeError` may be raised; if of a value outside the set of indexes for the sequence (after any special interpretation of negative values), `IndexError` should be raised. You can find the documentation at `https://docs.python.org/3/reference/datamodel.html#object.__getitem__`.