Software Training

Task 2

Subtask 1

You are required to implement a few basic methods for an employee class.

Take a quick look at an example of `Employee` below:

Employee

```
from random import randint
class Employee:
  def __init__(self, name, family, manager=None):
    self._name = name
    self._id = randint(1000,9999)
    self._family = family
    self._manager = manager
    self.salary = 2500
```

Instantiating Employee

```
  name = 'John Smith'
  family = {
    'Son': {
      'Insured': True,
      'Age': 16
    },
    'Wife': {
      'Insured': False,
      'Age': 32
    }
  }
  boss = Employee('AUR', {})
  my_employee = Employee(name, family, boss)
```

Requirements

Required methods, setters & getters:

- `id` – getter-only property
- `family` – getter-only property (hint: `dicts` are mutable, make sure someone can't modify the private variable by using the property from the getter)
- `apply_raise(self, managed_employee: Employee, raise_percent: int)` – function run by the manager, taking a managed employee and an `int` representing a raise percentage. Check if the employee is actually managed by the instance running the method (`self`) before applying the raise. After successfully applying the raise, print the new salary within the method.

Required Code

```python
from random import randint
class Employee:
  def __init__(self, name, family, manager=None):
    self._name = name
    self._id = randint(1000,9999)
    self._family = family.copy()
    self._manager = manager
    self.salary = 2500

  @property
  def id(self) -> int:
    ... # erase this line and implement method

  @property
  def family(self) -> dict:
    ... # erase this line and implement method

  def apply_raise(self, managed_employee: 'Employee', raise_percent: int):
    ... # erase this line and implement method
    # you must handle error where managed_employee._manager isn't self
    # choose to return success status or raise exceptions



##### Test code: #####
if __name__ == '__main__':
  boss = Employee('Jane Redmond', {})
  name = 'John Smith'
  family = {
    'Son': {
      'Insured': True,
      'Age': 16
    },
    'Wife': {
      'Insured': False,
      'Age': 32
    }
  }
  my_employee = Employee(name, family, boss)
  not_boss = Employee('Adam Cater', {})
  # do not change:
  print(id(my_employee.family))
  print(id(my_employee._family)) # should be different
  boss.apply_raise(my_employee, 25)
  print(not_boss.apply_raise(my_employee, 25))
```

Subtask 2

You are going to enhance printing to the terminal by making your own function called `send_Msg`. The function expects an object that is an instance of `BaseMsg`, and uses `rich` – an external library – to add information and styles (colors) when necessary.

Requirements

- Make a virtual environment to install `rich` to [return to session 1 slides]
- `send_Msg` is given as:

send_Msg

```
from rich.console import Console
console = Console()
def send_Msg(msg):
  if isinstance(msg, BaseMsg):
    console.print(msg, style=msg.style)
  else:
    print(msg)

```

- Implement a `BaseMsg` class that holds data to be printed on the terminal
- `BaseMsg` has properties:
  - `style`: `str` – refer to https://rich.readthedocs.io/en/latest/console.html#printing and https://rich.readthedocs.io/en/latest/appendix/colors.html#appendix-colors
  - `data`: `str` – read-only value of the raw message data
  - `__str__()`: `str` – representation of the text our function will print (including potential extra information)
  - `__len__()`: `int` – length of our printable string
  - `__eq__()`: `bool` – truth value of whether two message objects have the same printable string
  - `__add__()`: `BaseMsg` or derivative – takes and adds two message objects or one message object and a string. If two different message types are added, i.e `a + b`, then `a`'s add method is called and the returned message is of a similar type to `a`
- Implement a `LogMsg` that inherits `BaseMsg`, with:
  - a re-implemented `__str__()` method which prepends '[*time in seconds*]' to the data before printing
  - its `style` – when used with `send_Msg` – will set the background color of the text to yellow
- Implement a `WarnMsg` that inherits `LogMsg`, with:
  - a re-implemented `__str__()` method which prepends '[!WARN][*time in seconds*]' to the data before printing
  - its `style` – when used with `send_Msg` – will set the background color of the text to red, and the color of the letters to white
- Optionally try not to re-implement `__add__()` for every subclass to hard-code its instantiation. See if you can get the class of the left-hand message in addition from the `self` argument's attributes

Required Code

```python
from time import time
... # get_Msg code goes here


class BaseMsg:
  def __init__(self, data: str):
    self._data = data

  @property
  def style(self):
    return '' # BaseMsg-specific

  @property
  def data(self):
    return self._data

  def __str__(self):
    return self._data # BaseMsg-specific

  def __len__(self):
    ... # erase this line and implement method

  def __eq__(self, other):
    ... # erase this line and implement method

  def __add__(self, other):
    ... # erase this line and implement method


class LogMsg(BaseMsg):
  def __init__(self, data):
    super().__init__(data)
    self._timestamp: int = ... # erase dots and assign value to use it in __str__()


class WarnMsg(LogMsg):
  ... # erase this line and reimplement specified methods


if __name__ == '__main__':
  m1 = BaseMsg('Normal message')
  m2 = LogMsg('Log')
  m3 = WarnMsg('Warning')
  send_Msg(m1)
  send_Msg(m2)
  send_Msg(m3)
```

Bonus

You will find an option to submit the tasks via your Github profile in the submission form. If you had time to watch the upcoming Git session, please try submitting by uploading the code under a Git repo with the following structure:

```
AUR-Training-25 (repo name)
└─ Phase 2
   ├─ Subtask 1.py
   └─ Subtask 2.py
```

Extra Preparation

To prepare for the next session, you may install the package `opencv-python`. Contact your head for help if you are struggling to follow the instructions given in the first slide for using `pip`.

Please note that OpenCV is a large, resource-intensive package and may not work properly on your system. In the workshop, you may team up with other trainees to work hands-on on a common laptop.

END OF DOCUMENT