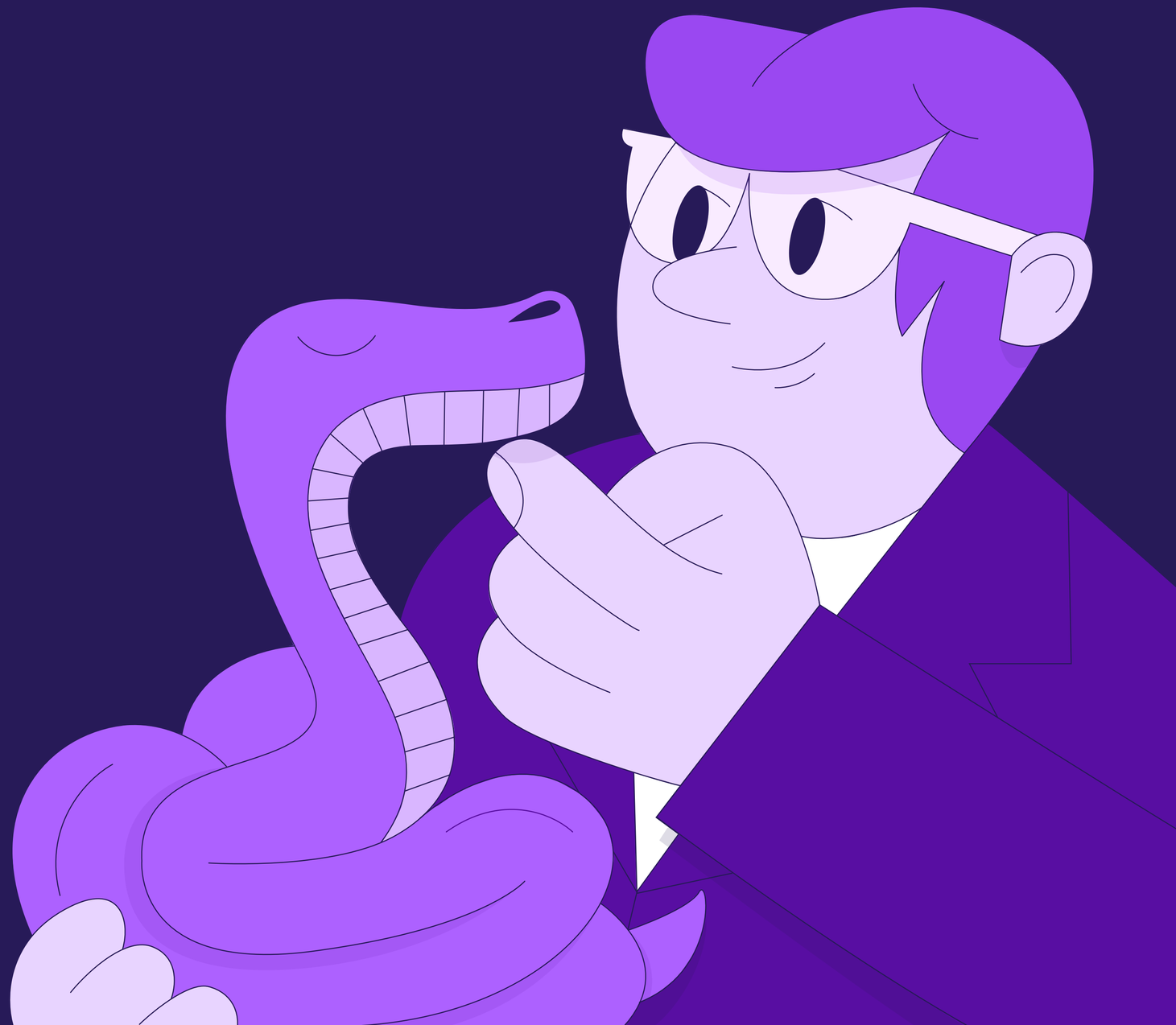
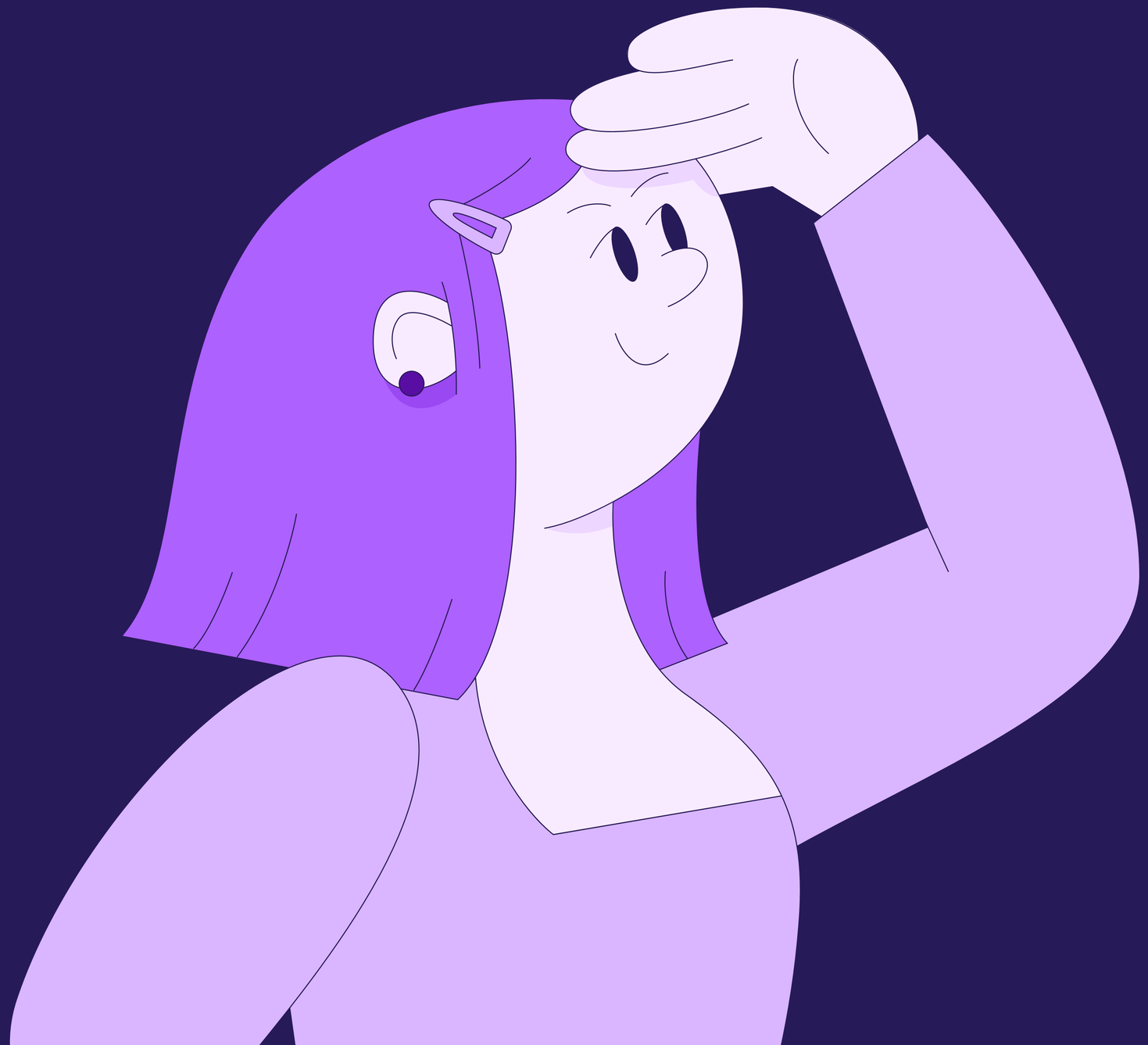
Python-разработчик

Режимы доступа



skypro

Сегодня на уроке



План урока

1. Рассмотреть методы classmethod и staticmethod.
2. Узнать, какие есть режимы доступа к атрибутам в классе.
3. Понять, как и зачем работать с декоратором `@property`.

Типы методов

- Обычные методы автоматически получают на вход аргумент `self` — ссылку на экземпляр класса.
- Методы классов (`@classmethod`) автоматически получают на вход аргумент `cls` — ссылку на класс.
- Статические методы (`@staticmethod`) — изолированные функции, которые не получают автоматически никаких аргументов.

Лайвкодинг в PyCharm

Задачи

- Добавить класс-метод `from_string` для изменения атрибутов «имя», «фамилия» из полного имени.
- Добавить класс-метод `set_raise_amt` для изменения атрибута класса, хранящего уровень индексации ЗП.
- Добавить статический метод `is_workday` для определения рабочего дня.

Задача

Реализация методов from_string, set_raise_amt и is_workday.

Флоу решения:

1. Реализовать метод `from_string` с декоратором `@classmethod`.
2. Реализовать метод `set_raise_amt` с декоратором `@classmethod`.
3. Реализовать метод `is_workday` с декоратором `@staticmethod`.

Режимы доступа к атрибутам

- `attr` — без подчеркиваний — публичный атрибут (public).
- `_attr` — одно подчеркивание — защищенный режим доступа (protected), для обращения внутри класса и во всех дочерних классах.
- `__attr` — с двумя подчеркиваниями — приватный режим доступа (private), для обращения только внутри класса.

Лайвкодинг в PyCharm

Задача: сделать атрибуты класса недоступными извне.

Лайвкодинг в PyCharm

Задача: сделать атрибуты класса недоступными извне.

Флоу решения:

1. Добавить режим protected (_) к атрибутам класса.
2. Убедиться в доступности атрибутов извне.
3. Добавить режим private (_ _) к атрибутам класса.
4. Убедиться в недоступности атрибутов извне.

Декоратор @property

Декоратор `@property` облегчает создание свойств в классах Python.

Свойства выглядят как обычные атрибуты класса, но:

- при их чтении вызывается геттер (getter);
- при записи — сеттер (setter);
- при удалении — делитер (deleter).

Геттер

```
class Employee:

    def __init__(self, first, last):
        self.first = first
        self.last = last

    # Геттер для fullname
    @property
    def fullname(self):
        """Возвращает полное имя сотрудника."""
        """К атрибуту можно обращаться без ()."""
        return f'{self.first} {self.last}'
```

Сеттер

```
class Employee:

    def __init__(self, first, last):
        self.first = first
        self.last = last

    # Чтобы иметь возможность присваивать атрибуту fullname значения,
    # надо определить его сеттер. Это работает только для атрибутов с @property
    @fullname.setter
    def fullname(self, name):
        """Метод срабатывает при операции присваивания."""
        first, last = name.split(' ')
        self.first = first
        self.last = last
```

Делитер

```
class Employee:

    def __init__(self, first, last):
        self.first = first
        self.last = last

    @fullname.deleter
    def fullname(self):
        print('Delete Name!')
        self.first = None
        self.last = None
```

Лайвкодинг в PyCharm

Задачи

Реализовать метод fullname:

- Можно обращаться как к обычному атрибуту.
- Возвращает имя и фамилию через пробел.
- При присваивании строки с именем и фамилией обновляет соответствующие атрибуты.

Реализовать метод email:

- Возвращает email сотрудника.
- Доступ к методу как к обычному атрибуту, без вызова.

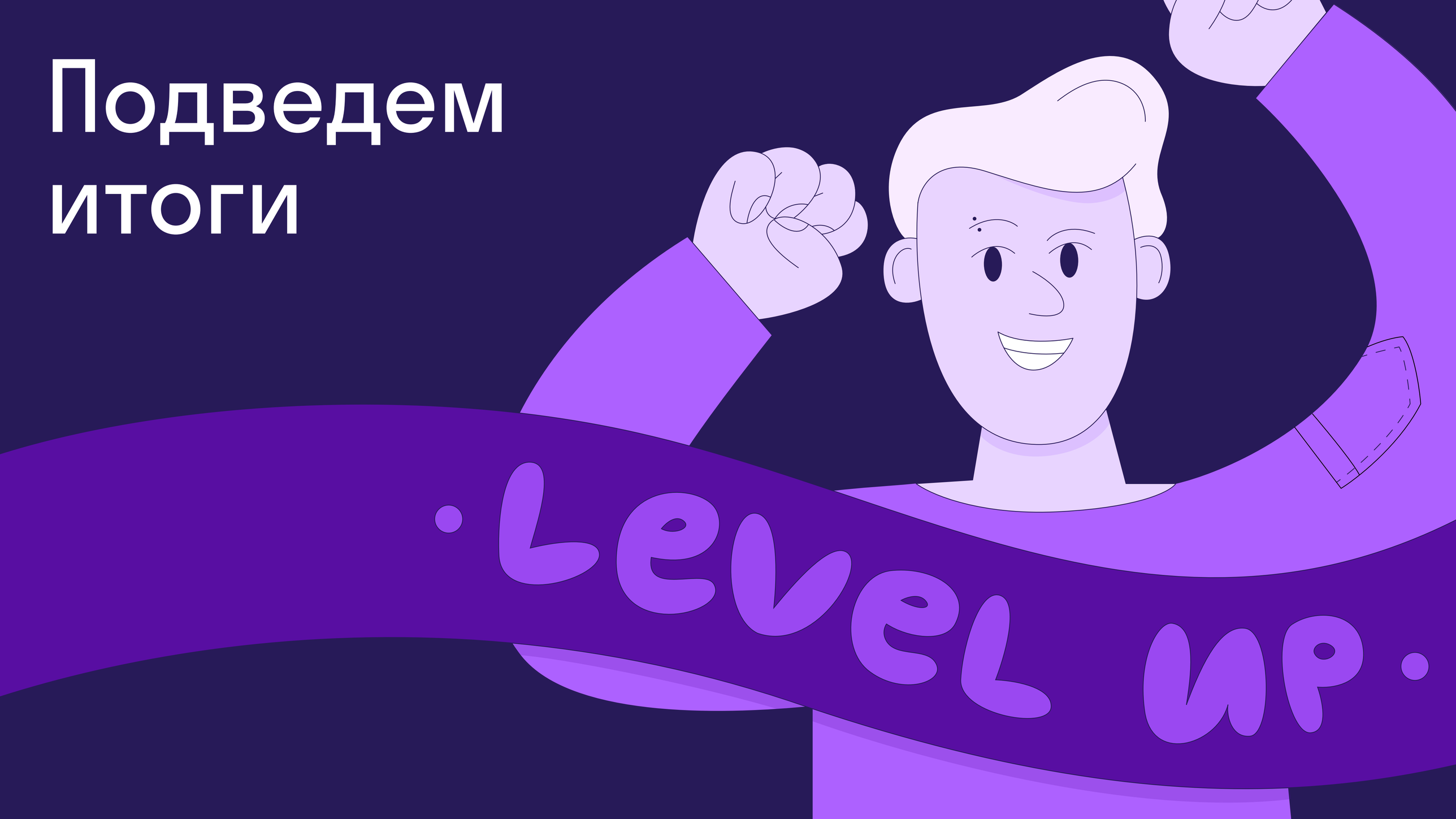
Задача

Реализация методов fullname и email.

Флоу решения:

1. Реализовать метод email.
2. Добавить методу email декоратор `@property`.
3. Реализовать метод fullname, возвращающий полное имя.
4. Добавить методу fullname декоратор `@property`.
5. Реализовать метод fullname, присваивающий атрибутам «имя» и «фамилия» значения из строки полного имени.
6. Добавить методу fullname декоратор `@fullname.setter`.

Подведем итоги



Рекап

1. Декораторы — это инструмент для изменения поведения функции/метода.

Рекап

1. Декораторы — это инструмент для изменения поведения функции/метода.
2. `classmethod` автоматически получает на вход ссылку на класс через переменную `cls`.

Рекап

1. Декораторы — это инструмент для изменения поведения функции/метода.
2. `classmethod` автоматически получает на вход ссылку на класс через переменную `cls`.
3. `staticmethod` не получает никаких специальных переменных для ссылок на объект или класс.

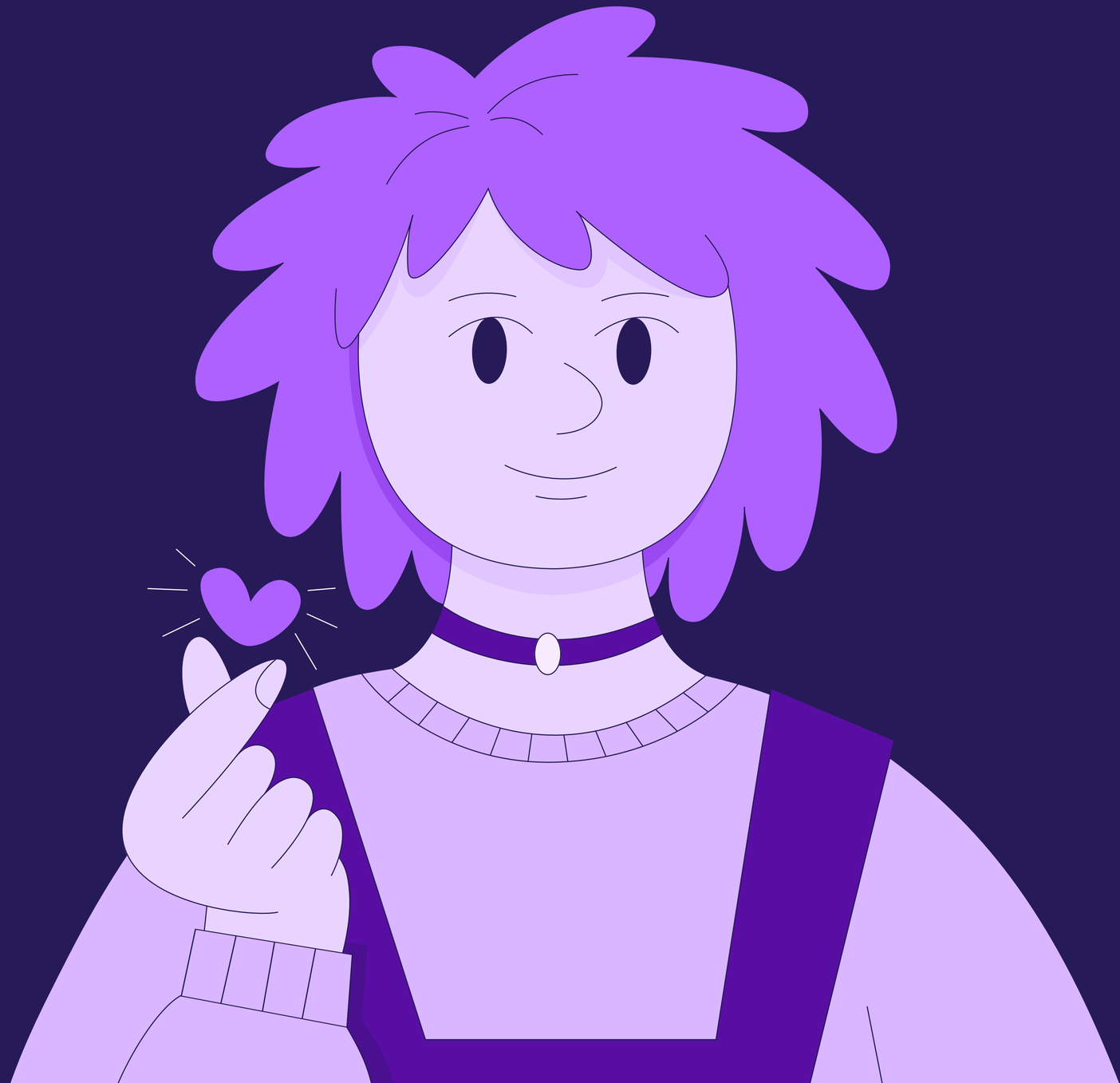
Рекап

1. Декораторы — это инструмент для изменения поведения функции/метода.
2. `classmethod` автоматически получает на вход ссылку на класс через переменную `cls`.
3. `staticmethod` не получает никаких специальных переменных для ссылок на объект или класс.
4. У атрибутов есть три режима доступа: публичные, защищенные, приватные.

Рекап

1. Декораторы — это инструмент для изменения поведения функции/метода.
2. `classmethod` автоматически получает на вход ссылку на класс через переменную `cls`.
3. `staticmethod` не получает никаких специальных переменных для ссылок на объект или класс.
4. У атрибутов есть три режима доступа: публичные, защищенные, приватные.
5. Декоратор `@property` позволяет работать с методами как с переменными.

Спасибо!



skypro