Практика 2 (углублённая, 60 минут): Создание класса `SmartDevice` — полное погружение в ООП

> **Цель**: Закрепить понимание классов, объектов, атрибутов, методов, конструктора и `self` через создание сложного, многофункционального класса с валидацией, форматированным выводом, пользовательским вводом и дополнительной логикой.

Время выполнения: 60 минут
Сложность: высокая — для углублённого понимания основ ООП
Включает конспект лекции 1 в начале

Конспект Лекции 1: Классы и объекты — краткая шпаргалка

Что такое ООП?

Объектно-ориентированное программирование — это парадигма, где программа строится вокруг **объектов**, объединяющих **данные (атрибуты)** и **поведение (методы)**.

Класс vs Объект

- **Класс** — шаблон (чертёж). Определяет, какие атрибуты и методы будут у объектов.
- **Объект (экземпляр)** — конкретная реализация класса в памяти.

```
class Dog:
    def __init__(self, name):
        self.name = name   # атрибут экземпляра

my_dog = Dog("Buddy")      # объект
```

`__init__` — конструктор

- Вызывается автоматически при создании объекта.
- Используется для инициализации атрибутов.
- Первый параметр — всегда `self`.

`self`

- Ссылка на текущий экземпляр класса.
- Через `self` мы обращаемся к атрибутам и методам объекта внутри класса.
- **Обязателен** в определении всех методов экземпляра.

Атрибуты и методы

- **Атрибуты** — переменные объекта (`self.color`, `self.model`).
- **Методы** — функции внутри класса. Всегда принимают `self` первым параметром.

```
    def turn_on(self):
        print(f"{self.name} is now ON")
```

🔹 Создание объекта

```
device = SmartDevice("Lamp", "Philips", 100)
```

🔹 Вызов методов и доступ к атрибутам

```
print(device.name)
device.turn_on()
```

🧪 Практика 1: Создание класса `SmartDevice`

> Вы создадите класс для умного устройства (лампа, чайник, розетка и т.д.), который умеет включаться/выключаться, менять уровень энергопотребления, следить за временем работы и выводить статистику.

🎯 Часть 1: Создание класса `SmartDevice` (20 минут)

✅ Требования к классу:

1. **Конструктор** `__init__` принимает:
   - `name` (str) — название устройства (например, "Лампа в гостиной").
   - `brand` (str) — производитель.
   - `max_power` (int) — максимальная мощность в ваттах (например, 100 Вт).
2. **Атрибуты экземпляра** (все устанавливаются в `__init__`):
   - `name`, `brand`, `max_power` — как передано.
   - `is_on` — булево, изначально `False` (устройство выключено).
   - `current_power` — текущее энергопотребление (изначально `0`).
   - `total_runtime_minutes` — сколько минут устройство работало за всё время (изначально `0`).
3. **Методы**:

Метод	Описание
`turn_on()`	Включает устройство (`is_on = True`), устанавливает `current_power = max_power`.

Метод	Описание
`turn_off()`	Выключает устройство (`is_on = False`), `current_power = 0`.
`set_power(level)`	Устанавливает `current_power` вручную (от 0 до `max_power`). Если вне диапазона — `ValueError`.
`update_runtime(minutes)`	Добавляет минуты к `total_runtime_minutes` (только если устройство включено!).
`get_status()`	Возвращает строку: `"[ON/OFF] {name} ({brand}), потребление: {current_power}/{max_power} Вт, всего работы: {total_runtime_minutes} мин"`
`is_working_efficiently()`	Возвращает `True`, если `current_power >= 0.8 * max_power` (работает на 80%+ мощности), иначе `False`. Только если включено. Если выключено — `False`.

📌 Пример использования:

```
lamp = SmartDevice("Гостиная лампа", "Philips", 100)
lamp.turn_on()
print(lamp.get_status())
# [ON] Гостиная лампа (Philips), потребление: 100/100 Вт, всего работы: 0 мин
lamp.set_power(70)
lamp.update_runtime(30)  # добавляем 30 минут работы
print(lamp.is_working_efficiently())  # False (70 < 80)
```

💻 Часть 2: Интерактивный симулятор устройства (20 минут)

Напишите программу, которая:

1. Запрашивает у пользователя:
   - Название устройства
   - Бренд
   - Максимальную мощность (должна быть > 0, иначе — повторный ввод)
2. Создаёт объект `SmartDevice`.
3. Выводит меню управления:

```
=== Управление устройством ===
1. Включить
2. Выключить
3. Установить мощность
4. Добавить время работы (мин)
5. Показать статус
```

```
6. Проверить эффективность
0. Выход
```

4. В цикле обрабатывает выбор пользователя:
    - При выборе 3 — запрашивает уровень мощности и вызывает `set_power()`.
    - При выборе 4 — запрашивает минуты и вызывает `update_runtime()` (даже если выключено — метод сам проверит).
    - При выборе 5 — выводит `get_status()`.
    - При выборе 6 — выводит `"Работает эффективно"` или `"Неэффективно"` на основе `is_working_efficiently()`.
5. Обрабатывает ошибки (например, попытка установить мощность 150 при максимуме 100) — выводит сообщение, но не прерывает программу.

📌 Пример сессии:

```
Введите название устройства: Кухонный свет
Введите бренд: IKEA
Введите макс. мощность (Вт): 60

=== Управление устройством ===
1. Включить
2. Выключить
3. Установить мощность
4. Добавить время работы (мин)
5. Показать статус
6. Проверить эффективность
0. Выход
> 1
Устройство включено.

> 3
Введите уровень мощности: 50
Мощность установлена.

> 4
Сколько минут добавить? 15
Время обновлено.

> 5
[ON] Кухонный свет (IKEA), потребление: 50/60 Вт, всего работы: 15 мин

> 6
Неэффективно

> 0
До свидания!
```

> 💡 Подсказка: используйте `try-except` при вызове `set_power()`.

🌟 Часть 3: Расширенная логика и бонусы (20 минут)

✅ Бонус 1: Статистика по нескольким устройствам

Создайте список из 3 устройств:

- `device1` — "Настольная лампа", "Xiaomi", 40
- `device2` — "Чайник", "Bosch", 1800
- `device3` — "Телевизор", "Samsung", 120

Для каждого:

- Включите.
- Установите мощность: 35, 1500, 100 соответственно.
- Добавьте время работы: 120, 5, 90 минут.
- Выведите статус и эффективность.

Затем найдите и выведите:

- Общую мощность всех включённых устройств.
- Общее время работы всех устройств.
- Количество устройств, работающих эффективно.

📌 Пример вывода:

```
=== Общая статистика ===
Общая мощность: 1635 Вт
Общее время работы: 215 мин
Эффективно работают: 1 устройств(а)
```

✅ Бонус 2: Метод `reset_stats()`

Добавьте метод `reset_stats()`, который:

- Сбрасывает `total_runtime_minutes` в 0.
- Выключает устройство (`is_on = False`, `current_power = 0`).

Протестируйте: после сбора статистики — вызовите `reset_stats()` для одного устройства и снова выведите статус.

🧠 Вопросы для самопроверки и обсуждения

1. Почему `self` обязателен в каждом методе?
2. Что произойдёт, если вызвать `update_runtime()` для выключенного устройства?
3. Как работает валидация в `set_power()`? Почему она важна?
4. Можно ли обратиться к `total_runtime_minutes` напрямую извне? Должны ли мы это разрешать?
5. Как бы вы добавили логирование каждого действия (например, “Устройство включено в 14:30”)?

📚 Дополнительно (если закончили раньше)

- Добавьте метод `rename(new_name)` — позволяет изменить название устройства.
- Реализуйте `__str__` для красивого вывода объекта через `print(device)`.
- Добавьте атрибут `created_at` (дата создания объекта) с помощью `datetime.datetime.now()`.
- Реализуйте метод `get_power_usage_kwh(hours)` — возвращает потребление в кВт·ч за указанное количество часов.
