🧪 Практика 1 (расширенная): Создание класса `Rectangle` — углублённая работа с ООП

> **Цель**: Закрепить понимание классов, объектов, атрибутов, методов и взаимодействия с пользователем через создание полноценного класса `Rectangle` с валидацией, интерактивным вводом и расширенной логикой.

⏱️ **Время выполнения**: 45 минут
🎯 **Сложность**: от базовой к средней — с элементами углубления

📝 Раздаточный материал для студентов

Создайте файл `practice1_extended.py`.

📌 Часть 1: Теоретическая справка (5 минут)

Перед началом — кратко вспомните:

- `class` — определяет новый тип объекта.
- `__init__` — конструктор, вызывается при создании объекта.
- `self` — ссылка на текущий экземпляр класса.
- Методы — функции внутри класса. Всегда принимают `self` первым параметром.
- Атрибуты — данные, привязанные к объекту (например, `self.width`).

🎯 Основное задание (30 минут)

✅ Шаг 1: Создайте класс `Rectangle`

Класс должен:

- Принимать в конструкторе `width` и `height`.
- **Проверять**, что оба значения — положительные числа (`> 0`). Если нет — выбрасывать `ValueError` с сообщением `"Width and height must be positive numbers"`.
- Иметь следующие методы:

Метод	Описание
`area()`	Возвращает площадь (`width * height`)
`perimeter()`	Возвращает периметр (`2 * (width + height)`)
`is_square()`	Возвращает `True`, если `width == height`, иначе `False`
`describe()`	Возвращает **форматированную строку** с описанием: `"Rectangle(width=X, height=Y), площадь=S, периметр=P, это {'квадрат' если is_square, иначе 'прямоугольник'}"`

📌 Пример использования `describe()`:

```
rect = Rectangle(5, 5)
print(rect.describe())
# Вывод: Rectangle(width=5, height=5), площадь=25, периметр=20, это квадрат
```

✅ Шаг 2: Интерактивный ввод от пользователя

Напишите код, который:

1. Запрашивает у пользователя ввод ширины и высоты **в цикле**, пока не будут введены корректные значения (числа > 0).
2. При некорректном вводе — выводит сообщение об ошибке и просит ввести снова.
3. Создаёт объект `Rectangle` с введёнными значениями.
4. Выводит результат метода `describe()`.

📌 Пример работы программы:

```
Введите ширину прямоугольника: -5
Ошибка: Width and height must be positive numbers
Введите ширину прямоугольника: 0
Ошибка: Width and height must be positive numbers
Введите ширину прямоугольника: 7
Введите высоту прямоугольника: abc
Ошибка: пожалуйста, введите число
Введите высоту прямоугольника: 3.5
Rectangle(width=7, height=3.5), площадь=24.5, периметр=21.0, это прямоугольник
```

> 💡 Подсказка: используйте `try-except` для обработки `float(input(...))` и собственного `ValueError`.

✅ Шаг 3: Создайте 3 объекта и сравните их

Создайте три прямоугольника:

- `rect1` — 10 x 5
- `rect2` — 7 x 7
- `rect3` — 4 x 8

Для каждого выведите:

- Результат `describe()`
- Отдельно — является ли квадратом (`"Это квадрат!"` или `"Это не квадрат"`)

📌 Пример вывода:

```
[Объект 1]
Rectangle(width=10, height=5), площадь=50, периметр=30, это прямоугольник
Это не квадрат

[Объект 2]
Rectangle(width=7, height=7), площадь=49, периметр=28, это квадрат
Это квадрат!

[Объект 3]
Rectangle(width=4, height=8), площадь=32, периметр=24, это прямоугольник
Это не квадрат
```

🌟 Бонусное задание (10–15 минут, для быстрых студентов)

✅ Шаг 4: Добавьте метод `compare_area(other)`

- Метод принимает другой объект `Rectangle` (или совместимый объект с атрибутом `area()`).
- Возвращает:
  - `1`, если площадь текущего объекта **больше**
  - `-1`, если **меньше**
  - `0`, если **равны**

📌 Пример:

```
rect1 = Rectangle(10, 5)  # площадь 50
rect2 = Rectangle(7, 7)   # площадь 49
print(rect1.compare_area(rect2))  # 1
```

✅ Шаг 5: Найдите “самый большой” прямоугольник

Используя метод `compare_area`, определите, какой из трёх созданных прямоугольников имеет наибольшую площадь, и выведите сообщение:

```
Самый большой прямоугольник: Rectangle(width=10, height=5), площадь=50
```

> 💡 Подсказка: можно использовать простое сравнение вручную или цикл с запоминанием “чемпиона”.

🧠 Вопросы для самопроверки и обсуждения

1. Почему важно проверять входные данные в `__init__`?
2. Что произойдёт, если не обработать `ValueError` при вводе?
3. Можно ли вызвать `describe()` до инициализации объекта? Почему?
4. Как бы вы изменили класс, чтобы поддерживать целочисленные размеры только?

5. Зачем нужен `self` в каждом методе?

📚 Дополнительно (если закончили раньше)

- Добавьте метод `scale(factor)`, который умножает ширину и высоту на число `factor` (например, 2 — удвоить размер).
- Добавьте статический метод `create_square(side)`, который создаёт квадрат по одной стороне.
- Реализуйте `__str__` для красивого вывода объекта через `print(rect)`.

✅ **Практика завершена!**

Теперь студенты не просто создали класс — они поработали с валидацией, исключениями, пользовательским вводом, сравнением объектов и форматированным выводом. Это полноценное погружение в ООП с первого занятия.
