📚 Краткая справка: Что понадобится в задаче

🔹 Наследование

Позволяет одному классу (дочернему) унаследовать атрибуты и методы другого (родительского).
Пример:

```
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    pass
```

🔹 Полиморфизм

Разные объекты могут реагировать на один и тот же метод по-разному.
Пример:

```
animals = [Dog("Бобик"), Cat("Мурка")]
for a in animals:
    a.make_sound()  # каждый издаёт свой звук
```

🔹 Магические методы (dunder methods)

Специальные методы, которые вызываются автоматически при использовании встроенных операций.

Метод	Когда вызывается	Пример использования
`__str__`	`print(obj)`, `str(obj)`	Человекочитаемое представление
`__repr__`	`repr(obj)`, отладка	Точное представление (для разработчика)
`__eq__`	`obj1 == obj2`	Сравнение на равенство
`__lt__`	`obj1 < obj2`	Сравнение "меньше" (для сортировки)

> 💡 Чтобы объекты можно было **сортировать**, реализуйте `__lt__`.
> Чтобы можно было **сравнивать**, реализуйте `__eq__`.

🧪 Практическая задача (60 минут): Система учёта сотрудников IT-компании

📌 Сценарий

Вы разрабатываете систему HR для IT-компании. В компании работают **разные типы сотрудников**:

- **Разработчики** — получают оклад + бонус за каждую задачу.
- **Дизайнеры** — получают оклад + бонус за каждый завершённый проект.
- **Менеджеры** — получают фиксированный оклад (без бонусов).

Все сотрудники имеют:

- Имя
- Должность
- Оклад (base_salary)
- Общий доход (оклад + бонусы)

Ваша задача — создать иерархию классов, в которой:

1. Все сотрудники **наследуются от базового класса**.
2. Каждый тип **по-своему рассчитывает доход** (полиморфизм).
3. Объекты можно **сравнивать по доходу** (`==`, `<`).
4. Объекты имеют **красивый и информативный вывод** (`print`, отладка).

🎯 Требования к реализации

1. Базовый класс `Employee`

- **Атрибуты**:
  - `name` (str)
  - `position` (str)
  - `base_salary` (float)
- **Методы**:
  - `__init__(self, name, base_salary)`
  - `calculate_total_earnings()` — **абстрактный по смыслу**, должен быть переопределён в потомках. Возвращает `float`.
  - **Магические методы**:
    - `__str__()` → `"Имя (Должность): доход XXX руб."`
    - `__repr__()` → `"Employee('Имя', 'Должность', base_salary)"`
    - `__eq__(self, other)` → `True`, если `total_earnings` равны
    - `__lt__(self, other)` → `True`, если `self.total_earnings < other.total_earnings`

> 💡 Чтобы не пересчитывать доход каждый раз, можно сохранить его в атрибут `self._total` при первом вызове, но **в рамках задачи разрешено пересчитывать каждый раз** через `calculate_total_earnings()`.

---

2. Дочерние классы

а) `Developer(Employee)`

- В `__init__` принимает `name`, `base_salary`, **`tasks_completed`** (целое число).
- `position = "Разработчик"`
- `calculate_total_earnings()` → `base_salary + tasks_completed * 1000`

b) **`Designer(Employee)`**

- В `__init__` принимает `name`, `base_salary`, **`projects_done`** (целое число).
- `position = "Дизайнер"`
- `calculate_total_earnings()` → `base_salary + projects_done * 1500`

c) **`Manager(Employee)`**

- В `__init__` принимает `name`, `base_salary`.
- `position = "Менеджер"`
- `calculate_total_earnings()` → `base_salary` (без бонусов)

> 💡 Во всех дочерних классах вызывайте `super().__init__(name, base_salary)` и устанавливайте `self.position`.

---

3. Функция анализа команды

Напишите функцию:

```
def analyze_team(employees):
    """
    Принимает список сотрудников.
    1. Выводит каждого через print() (используется __str__).
    2. Сортирует список по доходу (от низкого к высокому) — использует __lt__.
    3. Находит самого высокооплачиваемого сотрудника.
    4. Проверяет, есть ли два сотрудника с одинаковым доходом (__eq__).
    """
```

Выведите:

- Список до сортировки
- Список после сортировки
- Самый высокооплачиваемый
- Есть ли сотрудники с одинаковым доходом?

---

4. Демонстрация

Создайте 5 сотрудников:

```
team = [
    Developer("Анна", 80000, tasks_completed=12),
    Designer("Борис", 70000, projects_done=5),
```

```
    Manager("Виктор", 120000),
    Developer("Глеб", 80000, tasks_completed=12),  # такой же доход, как у Анны
    Designer("Дина", 70000, projects_done=3)
]
```

Вызовите `analyze_team(team)`.

📥 Пример ожидаемого вывода

```
=== До сортировки ===
Анна (Разработчик): доход 92000.0 руб.
Борис (Дизайнер): доход 77500.0 руб.
Виктор (Менеджер): доход 120000.0 руб.
Глеб (Разработчик): доход 92000.0 руб.
Дина (Дизайнер): доход 74500.0 руб.

=== После сортировки ===
Дина (Дизайнер): доход 74500.0 руб.
Борис (Дизайнер): доход 77500.0 руб.
Анна (Разработчик): доход 92000.0 руб.
Глеб (Разработчик): доход 92000.0 руб.
Виктор (Менеджер): доход 120000.0 руб.

Самый высокооплачиваемый: Виктор (Менеджер): доход 120000.0 руб.
Есть сотрудники с одинаковым доходом: True
```

📝 Подсказки

- В магических методах вызывайте `self.calculate_total_earnings()`.
- Для сортировки используйте `sorted(employees)`.
- Для проверки одинаковых доходов:

```
has_duplicates = any(emp1 == emp2 for i, emp1 in enumerate(employees)
                                  for emp2 in employees[i+1:])
```

✅ Что сдать

- Один файл `.py` с классами и функцией.
- Демонстрационный код с 5 сотрудниками.
- Без `input()`, без `_`/`__` для атрибутов (кроме временных в магических методах), без внешних библиотек.