

Сборник заданий для семинарских занятий
по курсу
«Объектно-ориентированное программирование на Python»

Содержание

1	Общие сведения	3
2	Задания	3
2.1	Семинар «Правила формирования класса для программирования в IDE PyCharm. Отработка навыков создания простых классов и объектов класса» (2 очных часа)	4
2.2	Семинар «Конструкторы, наследование и полиморфизм. 1 часть» (2 часа) . .	19

1 Общие сведения

Сборник содержит задания для семинарских занятий по курсу «Объектно-ориентированное программирование на Python» (32 часа).

Задачник находится в процессе наполнения и новые задания появляются перед проведением нового семинара.

Возможна сдача другого кода (например, выполненного в ходе проектной деятельности), если они полностью покрывают материал семинара.

2 Задания

2.1 Семинар «Правила формирования класса для программирования в IDE PyCharm. Отработка навыков создания простых классов и объектов класса» (2 очных часа)

В ходе работы создайте 5 классов с соответствующими методами, описанными в индивидуальном задании. Предполагается, что пользователь класса не имеет права обращаться к свойствам напрямую (соблюдая принцип инкапсуляции), а должен использовать методы. Важно: в задании не всегда указаны все необходимые методы и свойства, при необходимости вам надо самостоятельно их добавить. Продемонстрируйте работоспособность всех методов (из задания) посредством создания запускаемых файлов, где осуществляется вызов методов для разных ситуаций (без ручного ввода, но с выводом результатов в консоль). Каждый класс должен сохраняться в отдельном исходном файле. Необходимо соблюдать все стандартные требования к качеству кода (отступы, именования переменных, классов, методов, проверка корректности входных данных). Для каждого класса создайте отдельный запускаемый файл для проверки всех его методов (допускается использование других классов в этих тестах).

Все предлагаемые классы в заданиях упрощенные; для использования в production-окружении они требуют серьезной доработки. Суть задания — в отработке базовых навыков, а не в идеальном моделировании предложенных ситуаций.

Для сдачи работы будьте готовы пояснить или аналогично заданию модифицировать любую часть кода, а также ответить на вопросы:

1. Кратко опишите парадигму объектно-ориентированного программирования (ООП).
2. Что такое класс в парадигме ООП?
3. Что такое объект (экземпляр) в парадигме ООП?
4. Что обозначает свойство инкапсуляции в парадигме ООП?
5. Синтаксис классов в Python (в рамках выполненной работы), создание и работа с объектами в Python.

При выполнении задания предполагается самое простое базовое описание классов, соответствующее следующему примеру (вы можете использовать то, что вы ЗНАЕТЕ дополнительно, но это остается на ваше усмотрение):

Если вы нашли в задачнике ошибки, опечатки и другие недостатки, то вы можете сделать pull-request.

```
class Worker:
    def set_last_name(self, last_name):
        self.last_name = last_name

    def print_last_name(self):
        print (f"Фамилия: {self.last_name}")

    def get_last_name(self):
        return last_name

worker = Worker()
worker.set_last_name(self, "Иванов")
worker.print_last_name()
print(worker.get_last_name())
```

Срок сдачи работы (начала сдачи): следующее занятие после его выдачи. В последующие сроки оценка будет снижаться (при отсутствии оправдывающих документов).

1. **Описание ситуации:** Рассмотрим работу грузовой железнодорожной станции. На станции есть несколько путей, по которым поезда могут прибывать и отправляться. Каждый путь имеет свой номер и может вместить несколько поездов. Поезда формируются из вагонов, каждый из которых может перевозить разные грузы. Работники станции отвечают за диспетчерское управление маневровыми локомотивами, осмотр вагонов, выполнение погрузочно-разгрузочных работ, прием груза к перевозке, ремонт путей, обеспечение безопасности и т.п. Они используют радиостанции для связи друг с другом и для отслеживания положения поездов и передвижения вагонов.

Создаваемые классы: 'Путь', 'Поезд', 'Вагон', 'Станция', 'РаботникСтанции'.

Для классов реализовать следующие простые методы (ниже приведен не исчерпывающий список методов; для демонстрации работы классов вам потребуются дополнительные методы, позволяющие отследить состояние объектов), используя для хранения данных списки (['']) Python:

- (a) **Путь:** добавить поезд на путь, убрать поезд с пути, получить список поездов на конкретном пути.
- (b) **Поезд:** прицепить вагон к поезду, отцепить вагон от поезда, получить (распечатать) список вагонов в поезде, вывести информацию о грузе в поезде.
- (c) **Вагон:** добавить номер поезда, в который включался конкретный вагон, удалить номер поезда из истории, отобразить историю поездов для конкретного вагона.
- (d) **РаботникСтанции:** класс, представляющий отдельного работника на станции, имеющий идентификатор, информацию о персональной радиостанции, список закрепленных за ним поездов для осмотра, ФИО, должность.
- (e) **Станция:** добавить станционный путь, добавить поезд на станцию, нанять работника станции, вывести информацию о всех путях, поездах, работниках, удалить путь, удалить поезд, уволить работника.

2. **Описание ситуации:** Рассмотрим работу крупного логистического терминала для обработки грузовых автомобилей. На терминале есть несколько доков (рамп), куда фуры прибывают для проведения погрузочно-разгрузочных работ. Каждый док имеет свой номер и может одновременно обслуживать одну машину. Грузовики перевозят паллеты, каждая из которых содержит определенный товар. Сотрудники терминала отвечают за прием грузовиков, управление погрузочной техникой, проверку сопроводительных документов, приемку и отгрузку товара, а также техническое обслуживание доков. Они используют портативные радиы для координации действий и отслеживания статуса обработки автомобилей.

Создаваемые классы: 'Док', 'Грузовик', 'Паллета', 'Терминал', 'Сотрудник'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **Док:** занять док конкретным грузовиком, освободить док, получить информацию о грузовике, который сейчас находится на доке.
- (b) **Грузовик:** добавить паллету в грузовик, выгрузить паллету из грузовика, получить (распечатать) список паллет в грузовике, вывести информацию о товарах в грузовике.
- (c) **Паллета:** добавить номер грузовика, в который загружалась конкретная паллета, удалить номер грузовика из истории, отобразить историю перевозок (номера грузовиков) для конкретной паллеты.

- (d) **Сотрудник:** класс, представляющий отдельного сотрудника терминала, имеющий идентификатор, номер радики, список доков, за которые он отвечает, ФИО, должность.
- (e) **Терминал:** добавить новый док на терминале, зарегистрировать прибытие грузовика, нанять нового сотрудника, вывести список всех доков, грузовиков на территории, сотрудников, удалить док, удалить грузовик, уволить сотрудника.

3. **Описание ситуации:** Рассмотрим работу аэропорта. В аэропорту есть несколько взлетно-посадочных полос (ВПП), которые принимают и отправляют рейсы. Каждая ВПП имеет свой номер, длину и статус доступности. Самолеты перевозят пассажиров и их ручную кладь, размещенную в салоне. Авиадиспетчеры управляют движением самолетов, назначают полосы для взлета и посадки, следят за воздушной обстановкой и координируют действия с помощью радиосвязи.

Создаваемые классы: 'ВПП', 'Самолет', 'Пассажир', 'Аэропорт', 'Авиадиспетчер'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **ВПП:** занять полосу для взлета/посадки, освободить полосу, получить список рейсов, использовавших полосу.
- (b) **Самолет:** добавить пассажира на борт (включая вес его ручной клади), высадить пассажира, получить (распечатать) список пассажиров на борту, рассчитать общий вес ручной клади.
- (c) **Пассажир:** добавить рейс в историю перелетов пассажира, удалить рейс из истории (ошибка бронирования), отобразить всю историю перелетов.
- (d) **Авиадиспетчер:** класс, представляющий диспетчера, имеющий идентификатор, рабочую частоту, график работы (список интервалов времени в сутках), ФИО.
- (e) **Аэропорт:** добавить новую ВПП, зарегистрировать прибытие самолета, нанять диспетчера, вывести список всех ВПП, самолетов в аэропорту, диспетчеров, удалить ВПП (на ремонт), списать самолет, уволить диспетчера.

4. **Описание ситуации:** Рассмотрим работу речного порта. В порту есть несколько причалов для швартовки грузовых барж и буксиров. Каждый причал имеет уникальный номер и максимальную глубину, определяющую осадку судов, которые могут к нему подойти. Баржи перевозят контейнеры с различными грузами. Их характеризуют вес судна, максимальная грузоподъемность и осадка (как без груза, так и с максимальным грузом). Портовые рабочие отвечают за швартовку судов, управление портовыми кранами для погрузки/разгрузки контейнеров, оформление документов и поддержание порядка на территории.

Создаваемые классы: 'Причал', 'Баржа', 'Контейнер', 'Порт', 'ПортовыйРабочий'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **Причал:** пришвартовать баржу к причалу, отшвартовать баржу, получить список барж, находящихся у причала.
- (b) **Баржа:** загрузить контейнер на баржу (с указанием веса контейнера), разгрузить контейнер с баржи, получить (распечатать) список контейнеров на барже, рассчитать текущую осадку судна (предполагается линейная зависимость осадки от суммарного веса груза и баржи).

- (с) **Контейнер:** добавить номер баржи, на которую погрузили контейнер, удалить номер баржи, отобразить историю перемещений контейнера между баржами.
- (d) **ПортовыйРабочий:** класс, представляющий рабочего, имеющий идентификатор, допуск к работе с краном, список закрепленных причалов, ФИО, должность.
- (е) **Порт:** ввести новый причал в эксплуатацию, принять баржу в акваторию порта, принять на работу рабочего, вывести список причалов, барж в акватории, рабочих, списать причал, отправить баржу, уволить рабочего.

5. **Описание ситуации:** Рассмотрим работу автобусного парка. В парке есть несколько маршрутов, которые обслуживаются автобусами. Каждый маршрут имеет номер и список остановок. Автобусы имеют государственный номер, количество мест и текущий пробег. Водители закреплены за автобусами и маршрутами. Диспетчеры автопарка составляют расписание, следят за выходами автобусов на линию, учетом пробега и техническим состоянием.

Создаваемые классы: 'Маршрут', 'Автобус', 'Остановка', 'Автопарк', 'Водитель'.

Для классов реализовать следующие простые методы, используя для хранения данных списки ('[]') Python:

- (a) **Маршрут:** добавить остановку в маршрут, удалить остановку из маршрута, получить список всех остановок на маршруте.
- (b) **Автобус:** назначить автобус на маршрут, снять с маршрута, увеличить пробег на заданное значение, получить текущий пробег.
- (с) **Остановка:** добавить маршрут, проходящий через остановку, удалить маршрут, отобразить список всех маршрутов, проходящих через данную остановку.
- (d) **Водитель:** класс, представляющий водителя, имеющий идентификатор, права категории, закрепленный автобус, ФИО, график работы.
- (е) **Автопарк:** добавить новый маршрут, приобрести новый автобус, принять на работу водителя, вывести список маршрутов, автобусов (с указанием их состояния), водителей, списать автобус, уволить водителя.

6. **Описание ситуации:** Рассмотрим работу метрополитена. В метро есть линии, состоящие из станций и тоннелей между ними. Составы из вагонов перемещаются по линиям. Каждая станция имеет название и может быть точкой пересадки на другие линии. Машинисты управляют поездами. Дежурные по станции следят за порядком на платформах и работой оборудования. Управление метрополитеном координирует движение составов.

Создаваемые классы: 'ЛинияМетро', 'ПоездМетро', 'Станция', 'УправлениеМетрополитеном', 'Машинист'.

Для классов реализовать следующие простые методы, используя для хранения данных списки ('[]') Python:

- (a) **ЛинияМетро:** добавить станцию на линию, получить список станций на линии, получить список поездов на линии.
- (b) **ПоездМетро:** добавить вагон в состав, отцепить вагон, назначить машиниста на поезд.
- (с) **Станция:** добавить линию, проходящую через станцию (для моделирования пересадочных узлов), получить список линий на станции.

- (d) **Машинист:** класс, представляющий машиниста, имеющий идентификатор, допуск к управлению, закрепленный поезд, ФИО, стаж.
- (e) **Управление Метрополитеном:** открыть новую линию, ввести новый поезд в эксплуатацию, принять на работу машиниста, вывести список линий, поездов (в депо и на линиях), машинистов, закрыть линию на техобслуживание, списать поезд, вывести полную схему метро (в текстовом виде).

7. **Описание ситуации:** Рассмотрим работу службы доставки пиццы. В службе есть несколько филиалов. Каждый филиал обслуживает определенный район и имеет курьеров. Заказы формируются из позиций меню. Курьеры используют скутеры для доставки. Менеджеры филиалов принимают заказы, назначают курьеров и следят за выполнением заказов.

Создаваемые классы: 'Филиал', 'Заказ', 'Курьер', 'Скутер', 'Менеджер'.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **Филиал:** добавить курьера в филиал, уволить курьера, получить список активных заказов филиала.
- (b) **Заказ:** добавить позицию в заказ (название + цена), удалить позицию, рассчитать стоимость заказа, изменить статус заказа (принят, готовится, в пути, доставлен).
- (c) **Курьер:** назначить заказ курьеру, завершить доставку заказа, получить список доставленных заказов за смену, закрепить скутер за курьером.
- (d) **Менеджер:** класс, представляющий менеджера, имеющий идентификатор, закрепленный филиал, ФИО, смену.
- (e) **Скутер:** отправить на зарядку, вернуть в строй, увеличить пробег, получить текущий пробег.

Описание ситуации: Рассмотрим работу трамвайного депо. В депо есть несколько маршрутов, обслуживаемых трамвайными вагонами. Каждый трамвайный вагон имеет бортовой номер, вместимость и текущий пробег. Маршруты состоят из остановок и имеют определенный график движения. Водители трамваев закреплены за конкретными вагонами и маршрутами. Диспетчеры управляют выпуском трамваев на линию и ведут учет технического состояния.

Создаваемые классы: Маршрут, Трамвай, Остановка, Депо, Водитель.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **Маршрут:** добавить остановку в маршрут, удалить остановку из маршрута, получить список всех остановок на маршруте.
- (b) **Трамвай:** назначить трамвай на маршрут, снять с маршрута, увеличить пробег на заданное значение, получить текущий пробег.
- (c) **Остановка:** добавить маршрут, проходящий через остановку, удалить маршрут, отобразить список всех маршрутов, проходящих через данную остановку.
- (d) **Водитель:** класс, представляющий водителя, имеющий идентификатор, права категории, закрепленный трамвай, ФИО, график работы.

- (е) **Депо:** добавить новый маршрут, принять новый трамвай в депо, принять на работу водителя, выполнить вывод списка маршрутов, трамваев (с указанием их состояния), водителей, списать трамвай, уволить водителя.
8. **Описание ситуации:** Рассмотрим работу морского порта для приёма пассажирских паромов. В порту есть несколько причалов, каждый из которых обслуживает один паром за раз. Паромы перевозят пассажиров и автомобили. Пассажиры покупают билеты, автомобили записываются в список грузовой палубы. Сотрудники порта координируют погрузку, проверку билетов и безопасность. **Создаваемые классы:** Причал, Паром, Пассажир, Автомобиль, СотрудникПорта.
- (а) **Причал:** пришвартовать паром, освободить причал, получить информацию о пароме у причала.
- (б) **Паром:** добавить пассажира, добавить автомобиль, высадить пассажира, выгрузить автомобиль.
- (с) **Пассажир:** добавить рейс в историю поездок, удалить рейс из истории, вывести историю поездок.
- (d) **Автомобиль:** зарегистрировать номер паррома, удалить номер паррома, вывести историю перевозок.
- (е) **СотрудникПорта:** идентификатор, должность, ФИО, список закреплённых причалов.
9. **Описание ситуации:** Рассмотрим работу пригородной электрички. В системе есть станции, между которыми курсируют электрички. У каждой электрички есть номер, список вагонов и машинист. Пассажиры покупают билеты и занимают места в вагонах. Диспетчеры контролируют движение электричек. **Создаваемые классы:** Станция, Электричка, Вагон, Пассажир, Диспетчер.
- (а) **Станция:** принять электричку, отправить электричку, вывести список электричек на станции.
- (б) **Электричка:** добавить вагон, отцепить вагон, получить список вагонов.
- (с) **Вагон:** посадить пассажира, высадить пассажира, вывести список пассажиров.
- (d) **Пассажир:** добавить поездку в историю, удалить поездку, показать историю поездок.
- (е) **Диспетчер:** идентификатор, ФИО, рабочая смена, список контролируемых электричек.
10. **Описание ситуации:** Рассмотрим работу таксопарка. В таксопарке есть автомобили, водители и диспетчеры. Автомобиль закрепляется за водителем. Диспетчеры принимают заказы и назначают их водителям. Пассажиры совершают поездки. **Создаваемые классы:** Таксопарк, Автомобиль, Водитель, Заказ, Диспетчер.
- (а) **Таксопарк:** добавить автомобиль, принять водителя, вывести список машин и водителей, уволить водителя.
- (б) **Автомобиль:** назначить водителя, снять водителя, увеличить пробег, получить пробег.
- (с) **Водитель:** назначить заказ, завершить заказ, вывести список выполненных заказов.

- (d) **Заказ:** назначить пассажира, завершить поездку, вывести информацию о заказе.
 - (e) **Диспетчер:** идентификатор, ФИО, список назначенных заказов.
11. **Описание ситуации:** Рассмотрим работу грузового аэропорта. Самолёты перевозят контейнеры. В аэропорту есть ангары для хранения самолётов и площадки для погрузки. Работники аэропорта координируют загрузку и выгрузку контейнеров. **Создаваемые классы:** Самолёт, Контейнер, Ангар, РаботникАэропорта, Аэропорт.
- (a) **Самолёт:** загрузить контейнер, выгрузить контейнер, вывести список контейнеров.
 - (b) **Контейнер:** добавить номер самолёта, удалить номер самолёта, вывести историю перевозок.
 - (c) **Ангар:** принять самолёт, вывести список самолётов, освободить ангар.
 - (d) **РаботникАэропорта:** идентификатор, ФИО, должность, список самолётов в обслуживании.
 - (e) **Аэропорт:** принять самолёт, убрать самолёт, принять раотника, уволить работника, вывести список самолётов и работников.
12. **Описание ситуации:** Рассмотрим работу велопроката. В прокате есть велосипеды, станции для их хранения, клиенты и сотрудники. Клиенты арендуют велосипеды и возвращают их на станцию. **Создаваемые классы:** Велосипед, СтанцияПроката, Клиент, Сотрудник, Прокат.
- (a) **Велосипед:** выдать в аренду, вернуть на станцию, получить пробог.
 - (b) **СтанцияПроката:** добавить велосипед, убрать велосипед, вывести список велосипедов.
 - (c) **Клиент:** арендовать велосипед, вернуть велосипед, вывести историю аренд.
 - (d) **Сотрудник:** идентификатор, ФИО, должность, список закреплённых станций.
 - (e) **Прокат:** добавить станцию, демонтировать станцию, вывести список станций и велосипедов, уволить сотрудника, нанять сотрудника, вывести список сотрудников.
13. **Описание ситуации:** Рассмотрим работу речных теплоходов. У каждого теплохода есть рейсы и список пассажиров. Пассажиры покупают билеты. Работники пристани обслуживают теплоходы. **Создаваемые классы:** Теплоход, Рейс, Пассажир, Пристань, РаботникПристани.
- (a) **Теплоход:** добавить рейс, убрать рейс, вывести список рейсов.
 - (b) **Рейс:** добавить пассажира, удалить пассажира, вывести список пассажиров.
 - (c) **Пассажир:** добавить рейс в историю, удалить рейс, вывести историю.
 - (d) **Пристань:** принять теплоход, отправить теплоход, вывести список теплоходов.
 - (e) **РаботникПристани:** идентификатор, ФИО, должность, закреплённые рейсы.
14. **Описание ситуации:** Рассмотрим работу каршеринга. В системе есть автомобили, клиенты и диспетчеры. Автомобили бронируются клиентами и возвращаются после поездки. Диспетчеры контролируют состояние машин. **Создаваемые классы:** Автомобиль, Клиент, Диспетчер, Заказ, Каршеринг.

- (a) **Автомобиль:** выдать клиенту, вернуть, увеличить пробег, вывести пробег.
 - (b) **Клиент:** арендовать автомобиль, завершить аренду, вывести историю аренд.
 - (c) **Диспетчер:** идентификатор, ФИО, список автомобилей под контролем.
 - (d) **Заказ:** назначить автомобиль, завершить поездку, вывести данные заказа.
 - (e) **Каршеринг:** добавить автомобиль, списать автомобиль, добавить клиента, удалить клиента, добавить диспетчера, удалить диспетчера, вывести список клиентов, диспетчеров и машин.
15. **Описание ситуации:** Рассмотрим работу железнодорожного музея. В музее есть экспонаты (локомотивы и вагоны), экскурсии и экскурсоводы. Посетители записываются на экскурсии. **Создаваемые классы:** Экспонат, Экскурсия, Экскурсовод, Посетитель, Музей.
- (a) **Экспонат:** добавить к экскурсии, убрать, вывести список экскурсий.
 - (b) **Экскурсия:** записать посетителя, удалить, вывести список посетителей.
 - (c) **Экскурсовод:** идентификатор, ФИО, список экскурсий.
 - (d) **Посетитель:** записаться на экскурсию, отменить запись, вывести историю.
 - (e) **Музей:** добавить экспонат, списать экспонат, добавить экскурсовода, уволить экскурсовода, провести экскурсию, вывести список всех экскурсий и экскурсоводов.
16. **Описание ситуации:** Рассмотрим работу автозаправочной станции. На станции есть топливо, колонки и операторы. Автомобили приезжают заправляться. **Создаваемые классы:** Колонка, Автомобиль, Оператор, Топливо, АЗС.
- (a) **Колонка:** заправить автомобиль, освободить колонку, вывести статус.
 - (b) **Автомобиль:** получить заправку, вывести историю заправок.
 - (c) **Оператор:** идентификатор, ФИО, список закреплённых колонок.
 - (d) **Топливо:** уменьшить количество, увеличить количество, вывести остаток.
 - (e) **АЗС:** добавить колонку, нанять оператора, уволить оператора, демонтировать колонку, вывести список машин, операторов и колонок.
17. **Описание ситуации:** Рассмотрим работу сортировочного центра курьерской службы. В центре есть зоны обработки посылок, конвейерные линии и сотрудники. Каждая посылка имеет трек-номер и проходит через несколько этапов обработки. Сотрудники сканируют посылки, сортируют их по направлениям и загружают в транспортировочные контейнеры. Менеджеры контролируют процесс сортировки и работу оборудования.
- Создаваемые классы:** 'ЗонаОбработки', 'Посылка', 'Конвейер', 'СотрудникЦентра', 'СортировочныйЦентр'.
- Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:
- (a) **ЗонаОбработки:** добавить посылку в зону, удалить посылку из зоны, получить список посылок в зоне.
 - (b) **Посылка:** добавить статус обработки (принята, сортируется, отправлена), удалить ошибочный статус, отобразить историю статусов обработки.

- (с) **Конвейер:** запустить конвейерную ленту, остановить конвейер, добавить посылку на конвейер, снять посылку с конвейера.
- (d) **СотрудникЦентра:** класс, представляющий сотрудника, имеющий идентификатор, смену, список закрепленных зон обработки, ФИО, должность.
- (е) **СортировочныйЦентр:** добавить новую зону обработки, ввести в эксплуатацию конвейер, нанять сотрудника, вывести список всех зон, конвейеров, сотрудников, удалить зону, вывести из эксплуатации конвейер, уволить сотрудника.

18. **Описание ситуации:** Рассмотрим работу диспетчерской службы городского пассажирского транспорта. Диспетчеры отслеживают движение автобусов, троллейбусов и трамваев на маршрутах, регулируют интервалы движения, фиксируют отклонения от графика. Транспортные средства оснащены GPS-трекерами для передачи местоположения.

Создаваемые классы: 'Маршрут', 'ТранспортноеСредство', 'Диспетчер', 'Остановка', 'ДиспетчерскаяСлужба'.

Для классов реализовать следующие простые методы, используя для хранения данных списки ('[]') Python:

- (a) **Маршрут:** добавить транспортное средство на маршрут, снять с маршрута, получить список транспорта на маршруте.
- (b) **ТранспортноеСредство:** обновить местоположение (координаты), получить текущее местоположение, добавить информацию о задержке/опережении графика.
- (с) **Диспетчер:** класс, представляющий диспетчера, имеющий идентификатор, смену, список контролируемых маршрутов, ФИО.
- (d) **Остановка:** добавить маршрут, проходящий через остановку, удалить маршрут, получить список маршрутов на остановке.
- (е) **ДиспетчерскаяСлужба:** добавить новый маршрут, зарегистрировать транспортное средство, нанять диспетчера, вывести информацию о всех маршрутах, транспорте, диспетчерах, удалить маршрут, списать транспорт, уволить диспетчера.

19. **Описание ситуации:** Рассмотрим работу центра технического обслуживания городского транспорта. В центре есть ремонтные зоны для разных видов транспорта, запасы запчастей и бригады механиков. Транспортные средства проходят плановое ТО и внеплановый ремонт.

Создаваемые классы: 'РемонтнаяЗона', 'ТранспортноеСредство', 'Запчасть', 'Механик', 'ЦентрТехОбслуживания'.

Для классов реализовать следующие простые методы, используя для хранения данных списки ('[]') Python:

- (a) **РемонтнаяЗона:** поставить транспорт на ремонт, завершить ремонт, получить список транспорта в ремонте.
- (b) **ТранспортноеСредство:** добавить запись о ремонте (дата, вид работ), удалить ошибочную запись, отобразить историю ремонтов.
- (с) **Запчасть:** уменьшить количество на складе, увеличить количество, получить текущий остаток.

- (d) **Механик:** класс, представляющий механика, имеющий идентификатор, квалификацию, список закрепленных ремонтных зон, ФИО.
- (e) **ЦентрТехОбслуживания:** добавить ремонтную зону, закупить запчасти, нанять механика, вывести информацию о зонах, запчастях, механиках, удалить зону, уволить механика.

20. **Описание ситуации:** Рассмотрим работу логистического центра междугородных автобусных перевозок. Автобусы совершают рейсы между городами по определенным маршрутам, перевозят пассажиров и их багаж. Диспетчеры формируют расписание, продают билеты и контролируют отправку автобусов.

Создаваемые классы: 'Автобус', 'Маршрут', 'Пассажир', 'Диспетчер', 'ЛогистическийЦентр'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **Автобус:** назначить на маршрут, снять с маршрута, добавить пассажира, высадить пассажира, получить список пассажиров.
- (b) **Маршрут:** добавить город в маршрут, удалить город, получить список всех городов на маршруте.
- (c) **Пассажир:** купить билет (добавить маршрут в историю), сдать билет (удалить маршрут), показать историю поездок.
- (d) **Диспетчер:** класс, представляющий диспетчера, имеющий идентификатор, список закрепленных маршрутов, ФИО, график работы.
- (e) **ЛогистическийЦентр:** добавить автобус в парк, добавить маршрут, нанять диспетчера, вывести список автобусов, маршрутов и диспетчеров, списать автобус, уволить диспетчера.

21. **Описание ситуации:** Рассмотрим работу центра управления интеллектуальной транспортной системой города. Система включает в себя управление светофорами, камеры видеонаблюдения, датчики транспортного потока. Операторы следят за дорожной ситуацией и оперативно реагируют на инциденты.

Создаваемые классы: 'Перекресток', 'Светофор', 'КамераНаблюдения', 'ОператорИТС', 'ЦентрУправления'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **Перекресток:** добавить светофор к перекрестку, удалить светофор, получить список светофоров на перекрестке.
- (b) **Светофор:** изменить режим работы (красный/желтый/зеленый), получить текущий режим, добавить информацию о неисправности, вывести список неисправностей.
- (c) **КамераНаблюдения:** включить запись, выключить запись, получить статус работы, зафиксировать нарушение ПДД, вывести список нарушений.
- (d) **ОператорИТС:** класс, представляющий оператора, имеющий идентификатор, смену, список контролируемых перекрестков, ФИО.

- (е) **ЦентрУправления:** добавить новый перекресток в систему, установить светофор, установить камеру, нанять оператора, вывести информацию о перекрестках, светофорах, камерах, операторах, удалить перекресток, уволить оператора, снять камеру, снять светофор.

22. **Описание ситуации:** Рассмотрим работу службы эвакуации аварийных транспортных средств. Эвакуаторы дежурят на специальных парковках и выезжают по вызову на места ДТП или поломок. Диспетчеры принимают вызовы и направляют ближайший свободный эвакуатор.

Создаваемые классы: 'Эвакуатор', 'Вызов', 'ПарковкаЭвакуаторов', 'ДиспетчерЭвакуации', 'СлужбаЭвакуации'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (а) **Эвакуатор:** принять вызов, завершить вызов, получить текущий статус (свободен/занят), обновить местоположение.
- (б) **Вызов:** зафиксировать время принятия, время выполнения, получить статус выполнения.
- (с) **ПарковкаЭвакуаторов:** принять эвакуатор на парковку, выпустить эвакуатор с парковки, получить список эвакуаторов на парковке.
- (d) **ДиспетчерЭвакуации:** класс, представляющий диспетчера, имеющий идентификатор, смену, список обработанных вызовов, ФИО.
- (е) **СлужбаЭвакуации:** добавить эвакуатор в парк, списать эвакуатор, нанять диспетчера, вывести информацию о эвакуаторах, вызовах, диспетчерах, уволить диспетчера.

23. **Описание ситуации:** Рассмотрим работу центра контроля коммерческих грузоперевозок. Система отслеживает движение грузовых автомобилей, контролирует соблюдение маршрутов, норм труда водителей и расход топлива. Менеджеры по логистике планируют маршруты и анализируют отчеты.

Создаваемые классы: 'ГрузовойАвтомобиль', 'МаршрутПеревозки', 'Водитель', 'Рейс', 'МенеджерЛогистики'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (а) **ГрузовойАвтомобиль:** начать рейс, завершить рейс, получить текущий статус, зафиксировать расход топлива.
- (б) **МаршрутПеревозки:** добавить точку маршрута (город, склад), удалить точку, получить полный маршрут.
- (с) **Водитель:** класс, представляющий водителя, имеющий идентификатор, права, график работы, ФИО, стаж.
- (d) **Рейс:** закрепить автомобиль за рейсом, закрепить водителя за рейсом, открепить автомобиль, снять водителя, получить информацию о рейсе.
- (е) **МенеджерЛогистики:** класс, представляющий менеджера, имеющий идентификатор, список контролируемых маршрутов, ФИО.

24. **Описание ситуации:** Рассмотрим работу службы парковки аэропорта. На территории аэропорта есть несколько парковочных зон для разных типов транспорта (краткосрочная, долгосрочная, VIP). Операторы контролируют занятость мест, прием оплаты и работу шлагбаумов.

Создаваемые классы: 'ПарковочнаяЗона', 'ПарковочноеМесто', 'Автомобиль', 'ОператорПарковки', 'СлужбаПарковки'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **ПарковочнаяЗона:** добавить парковочное место, удалить место, получить список мест в зоне, получить список всех автомобилей. Так же парковочной зоне соответствует стоимость часа стоянки.
- (b) **ПарковочноеМесто:** занять место автомобилем, освободить место, получить текущий статус (свободно/занято).
- (c) **Автомобиль:** зафиксировать время въезда, время выезда + рассчитать стоимость парковки (с учетом стоимости часа), получить историю.
- (d) **ОператорПарковки:** класс, представляющий оператора, имеющий идентификатор, смену, список контролируемых зон, ФИО.
- (e) **СлужбаПарковки:** добавить новую парковочную зону, нанять оператора, вывести информацию о зонах, местах, операторах, удалить зону, уволить оператора.

25. **Описание ситуации:** Рассмотрим работу центра управления речным судоходством. Диспетчеры следят за движением судов по фарватеру, распределяют шлюзы, контролируют соблюдение графика движения и обеспечивают безопасность судоходства.

Создаваемые классы: 'Судно', 'Шлюз', 'Фарватер', 'ДиспетчерСудоходства', 'ЦентрУправления'.

Для классов реализовать следующие простые методы, используя для хранения данных списки (['']) Python:

- (a) **Судно:** начать движение по фарватеру, завершить движение, получить текущее местоположение, зафиксировать прохождение шлюза.
- (b) **Шлюз:** принять судно для шлюзования, завершить шлюзование, получить текущий статус (свободен/занят).
- (c) **Фарватер:** добавить участок фарватера, удалить участок, получить список судов на фарватере.
- (d) **ДиспетчерСудоходства:** класс, представляющий диспетчера, имеющий идентификатор, смену, список контролируемых шлюзов, ФИО.
- (e) **ЦентрУправления:** добавить шлюз в систему, зарегистрировать судно, нанять диспетчера, вывести информацию о шлюзах, фарватерах, судах, диспетчерах, удалить шлюз, уволить диспетчера.

26. **Описание ситуации:** Рассмотрим работу службы технического контроля метрополитена. Инспекторы проверяют состояние путей, тоннелей, подвижного состава и оборудования станций. Дефекты фиксируются в системе для оперативного устранения ремонтными бригадами.

Создаваемые классы: 'УчастокПути', 'ПодвижнойСостав', 'Инспектор', 'Дефект', 'СлужбаКонтроля'.

Для классов реализовать следующие простые методы, использующие для хранения данных списки ([]) Python:

- (a) **УчастокПути:** добавить информацию о дефекте, получить список неустраненных дефектов на участке.
- (b) **ПодвижнойСостав:** добавить запись о техническом осмотре, удалить ошибочную запись, отобразить историю осмотров.
- (c) **Инспектор:** класс, представляющий инспектора, имеющий идентификатор, квалификацию, список закрепленных участков, ФИО.
- (d) **Дефект:** зафиксировать время обнаружения, время устранения, получить статус устранения.
- (e) **СлужбаКонтроля:** добавить участок пути в систему, зарегистрировать подвижной состав, нанять инспектора, вывести информацию об участках, составе, инспекторах, дефектах, удалить участок, уволить инспектора, снять с эксплуатации подвижной состав.

27. **Описание ситуации:** Рассмотрим работу центра управления умными светофорами на перекрестках. Умные светофоры адаптивно меняют режим работы в зависимости от транспортного потока, приоритизируя общественный транспорт и спецтранспорт. Система анализирует данные с датчиков и камер, оптимизируя пропускную способность перекрестков.

Создаваемые классы: УмныйСветофор, Перекресток, ДатчикТранспортногоПотока, ИнженерАТС, ЦентрУправленияСветофорами.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **УмныйСветофор:** изменить длительность фаз (красный/зеленый), перейти в аварийный режим, получить текущий режим работы.
- (b) **Перекресток:** добавить светофор к перекрестку, удалить светофор, получить список всех светофоров перекрестка.
- (c) **ДатчикТранспортногоПотока:** установить текущие данные о интенсивности движения, получить текущие показания, получить историю показаний.
- (d) **ИнженерАТС:** класс, представляющий инженера автоматизированной транспортной системы, имеющий идентификатор, квалификацию, список закрепленных перекрестков, ФИО.
- (e) **ЦентрУправленияСветофорами:** добавить новый перекресток в систему, установить умный светофор, нанять инженера, вывести информацию о перекрестках, светофорах, инженерах, удалить перекресток, уволить инженера, снять умный светофор.

28. **Описание ситуации:** Рассмотрим работу монорельсовой транспортной системы. Монорельс движется по эстакаде, состоящей из станций и перегонов. Составы имеют фиксированное количество вагонов. Операторы управляют движением составов, следят за соблюдением графика и безопасностью пассажиров.

Создаваемые классы: СтанцияМонорельса, СоставМонорельса, ВагонМонорельса, ОператорСистемы, УправлениеМонорельсом.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **СтанцияМонорельса:** принять состав, отправить состав, получить список составов на станции.
- (b) **СоставМонорельса:** добавить вагон в состав (при техническом обслуживании), удалить вагон, получить список вагонов.
- (c) **ВагонМонорельса:** зафиксировать текущий пробег, провести техническое обслуживание, получить историю обслуживаний.
- (d) **ОператорСистемы:** класс, представляющий оператора, имеющий идентификатор, смену, список закрепленных станций, ФИО.
- (e) **УправлениеМонорельсом:** добавить новую станцию, ввести состав в эксплуатацию, нанять оператора, вывести информацию о станциях, составах, операторах, закрыть станцию на ремонт, списать состав, уволить оператора.

29. **Описание ситуации:** Рассмотрим работу канатной дороги. Канатная дорога состоит из линий с опорами и кабинок, перемещающихся между станциями. Кабинки имеют ограниченную вместимость. Техники обслуживают механизмы и следят за безопасностью.

Создаваемые классы: ЛинияКанатнойДороги, Кабинка, СтанцияКанатнойДороги, Техник, УправлениеКанатнойДорогой.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **ЛинияКанатнойДороги:** добавить кабинку на линию, снять кабинку, получить список кабинок на линии.
- (b) **Кабинка:** запустить в движение, остановить для посадки/высадки, получить текущий статус (движется/стоит).
- (c) **СтанцияКанатнойДороги:** принять кабинку, отправить кабинку, получить список кабинок на станции.
- (d) **Техник:** класс, представляющий техника, имеющий идентификатор, квалификацию, список закрепленных линий, ФИО.
- (e) **УправлениеКанатнойДорогой:** добавить новую линию, ввести кабинку в эксплуатацию, нанять техника, вывести информацию о линиях, кабинках, техниках, закрыть линию на обслуживание, списать кабинку, уволить техника.

30. **Описание ситуации:** Рассмотрим работу службы доставки с использованием дронов. Дроны осуществляют доставку небольших грузов между пунктами выдачи. Каждый дрон имеет грузоподъемность и дальность полета. Операторы управляют полетами дронов и обслуживают пункты выдачи.

Создаваемые классы: ПунктВыдачи, Дрон, Груз, ОператорДронов, СлужбаДоставки.

Для классов реализовать следующие простые методы, используя для хранения данных списки ([]) Python:

- (a) **ПунктВыдачи:** принять дрон с грузом, отправить дрон, получить список дронов в пункте.
- (b) **Дрон:** загрузить груз, выгрузить груз, начать полет, завершить полет, получить текущий статус (в полете/на земле).

- (с) **Груз:** зарегистрировать отправку, зарегистрировать доставку, получить историю перемещений.
- (d) **ОператорДронов:** класс, представляющий оператора, имеющий идентификатор, смену, список закрепленных пунктов выдачи, ФИО.
- (е) **СлужбаДоставки:** добавить новый пункт выдачи, ввести дрон в эксплуатацию, нанять оператора, вывести информацию о пунктах, дронах, операторах, закрыть пункт, списать дрон, уволить оператора.

2.2 Семинар «Конструкторы, наследование и полиморфизм. 1 часть» (2 часа)

В ходе работы решите 4 задачи. Предполагается, что пользователь класса не имеет права обращаться к свойствам напрямую (соблюдая принцип инкапсуляции), а должен использовать методы.

Продемонстрируйте работоспособность всех методов (из задания) посредством создания запускаемых файлов, где осуществляется вызов методов для разных ситуаций (без ручного ввода, но с выводом результатов в консоль).

Каждый класс должен сохраняться в отдельном исходном файле. Необходимо соблюдать все стандартные требования к качеству кода (отступы, именования переменных, классов, методов, проверка корректности входных данных). Для каждого класса создайте отдельный запускаемый файл для проверки всех его методов (допускается использование других классов в этих тестах).

Все предлагаемые классы в заданиях упрощенные; для использования в production-окружении они требуют серьезной доработки. Суть задания — в отработке базовых навыков, а не в идеальном моделировании предложенных ситуаций.

Для сдачи работы будьте готовы пояснить или аналогично заданию модифицировать любую часть кода, а также ответить на вопросы:

1. Что обозначает свойство наследования в парадигме ООП?
2. Что обозначает свойство полиморфизма в парадигме ООП?
3. Опишите реализацию наследования в Python
4. Как создать конструктор в Python
5. Как реализовать абстрактный класс в Python (и что это значит)
6. Как реализовать абстрактные методы в Python (и что это значит)

Если вы нашли в задачнике ошибки, опечатки и другие недостатки, то вы можете сделать pull-request.

Срок сдачи работы (начала сдачи): через одно занятие после его выдачи. В последующие сроки оценка будет снижаться (при отсутствии оправдывающих документов).

Задача 1

1. Напишите программу, которая создаёт класс `Circle` с методами для вычисления площади и длины окружности (периметра). Программа должна запрашивать у пользователя радиус и выводить вычисленные площадь и длину окружности.

Инструкции:

- (a) Создайте класс `Circle` с методом `__init__`, который принимает радиус окружности в качестве аргумента и сохраняет его в атрибуте `self.__radius`.
- (b) Создайте метод `calculate_circle_area`, без аргументов, который вычисляет площадь круга по формуле:

$$\pi \cdot \text{__radius}^2$$

- (c) Создайте метод `calculate_circle_perimeter` без аргументов, который вычисляет длину окружности по формуле:

$$2 \cdot \pi \cdot \text{__radius}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя радиус окружности,
 - создавать экземпляр класса `Circle` с этим радиусом,
 - вычислять площадь и длину окружности с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
radius = 3
circle = Circle(radius)
area = circle.calculate_circle_area()
perimeter = circle.calculate_circle_perimeter()
print(f"Площадь окружности равна: {area}")
print(f"Периметр окружности равен: {perimeter}")
```

Вывод:

```
Площадь окружности равна: 28.274333882308138
Периметр окружности равен: 18.84955592153876
```

2. Напишите программу, которая создаёт класс `Square` с методами для вычисления площади и периметра. Программа должна запрашивать у пользователя длину стороны и выводить вычисленные площадь и периметр.

Инструкции:

- (a) Создайте класс `Square` с методом `__init__`, который принимает длину стороны квадрата в качестве аргумента и сохраняет её в атрибуте `self.__side`.
- (b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь квадрата по формуле:

$$\text{__side}^2$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет периметр квадрата по формуле:

$$4 \cdot \text{__side}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя длину стороны квадрата,
 - создавать экземпляр класса `Square` с этой длиной,
 - вычислять площадь и периметр с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
side = 5
square = Square(side)
area = square.calculate_area()
perimeter = square.calculate_perimeter()
print(f"Площадь квадрата равна: {area}")
print(f"Периметр квадрата равен: {perimeter}")
```

Вывод:

Площадь квадрата равна: 25
Периметр квадрата равен: 20

3. Напишите программу, которая создаёт класс **Rectangle** с методами для вычисления площади и периметра. Программа должна запрашивать у пользователя ширину прямоугольника (при соотношении сторон 1:2) и выводить вычисленные площадь и периметр.

Инструкции:

- (a) Создайте класс **Rectangle** с методом `__init__`, который принимает ширину прямоугольника в качестве аргумента и сохраняет её в атрибуте `self.__width`. Высота прямоугольника должна быть в два раза больше ширины.
- (b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь прямоугольника по формуле:

$$\text{__width} \cdot (2 \cdot \text{__width})$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет периметр прямоугольника по формуле:

$$2 \cdot (\text{__width} + 2 \cdot \text{__width})$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя ширину прямоугольника,
 - ii. создавать экземпляр класса **Rectangle** с этой шириной,
 - iii. вычислять площадь и периметр с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
width = 3
rectangle = Rectangle(width)
area = rectangle.calculate_area()
perimeter = rectangle.calculate_perimeter()
print(f"Площадь прямоугольника равна: {area}")
print(f"Периметр прямоугольника равен: {perimeter}")
```

Вывод:

Площадь прямоугольника равна: 18
Периметр прямоугольника равен: 18

4. Напишите программу, которая создаёт класс **Triangle** с методами для вычисления площади и периметра. Программа должна запрашивать у пользователя длину стороны равностороннего треугольника и выводить вычисленные площадь и периметр.

Инструкции:

- (a) Создайте класс **Triangle** с методом `__init__`, который принимает длину стороны треугольника в качестве аргумента и сохраняет её в атрибуте `self.__side`.
(b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь равностороннего треугольника по формуле:

$$\frac{\sqrt{3}}{4} \cdot \text{__side}^2$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет периметр треугольника по формуле:

$$3 \cdot \text{__side}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя длину стороны треугольника,
 - создавать экземпляр класса **Triangle** с этой длиной,
 - вычислять площадь и периметр с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
side = 4
triangle = Triangle(side)
area = triangle.calculate_area()
perimeter = triangle.calculate_perimeter()
print(f"Площадь треугольника равна: {area}")
print(f"Периметр треугольника равен: {perimeter}")
```

Вывод:

Площадь треугольника равна: 6.928203230275509
Периметр треугольника равен: 12

5. Напишите программу, которая создаёт класс **Sphere** с методами для вычисления площади поверхности и объёма. Программа должна запрашивать у пользователя радиус сферы и выводить вычисленные площадь поверхности и объём.

Инструкции:

- (a) Создайте класс `Sphere` с методом `__init__`, который принимает радиус сферы в качестве аргумента и сохраняет его в атрибуте `self.__radius`.
- (b) Создайте метод `calculate_surface_area`, без аргументов, который вычисляет площадь поверхности сферы по формуле:

$$4 \cdot \pi \cdot \text{__radius}^2$$

- (c) Создайте метод `calculate_volume` без аргументов, который вычисляет объём сферы по формуле:

$$\frac{4}{3} \cdot \pi \cdot \text{__radius}^3$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя радиус сферы,
 - ii. создавать экземпляр класса `Sphere` с этим радиусом,
 - iii. вычислять площадь поверхности и объём с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
radius = 2
sphere = Sphere(radius)
surface_area = sphere.calculate_surface_area()
volume = sphere.calculate_volume()
print(f"Площадь поверхности сферы равна: {surface_area}")
print(f"Объём сферы равен: {volume}")
```

Вывод:

```
Площадь поверхности сферы равна: 50.26548245743669
Объём сферы равен: 33.510321638291124
```

- 6. Напишите программу, которая создаёт класс `Cylinder` с методами для вычисления объёма и площади боковой поверхности. Программа должна запрашивать у пользователя радиус основания и выводить вычисленные объём и площадь боковой поверхности (высота цилиндра фиксирована и равна 5).

Инструкции:

- (a) Создайте класс `Cylinder` с методом `__init__`, который принимает радиус основания цилиндра в качестве аргумента и сохраняет его в атрибуте `self.__radius`. Высота цилиндра фиксирована и равна 5.
- (b) Создайте метод `calculate_volume`, без аргументов, который вычисляет объём цилиндра по формуле:

$$\pi \cdot \text{__radius}^2 \cdot 5$$

- (c) Создайте метод `calculate_lateral_area` без аргументов, который вычисляет площадь боковой поверхности цилиндра по формуле:

$$2 \cdot \pi \cdot \text{__radius} \cdot 5$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя радиус основания цилиндра,
 - создавать экземпляр класса `Cylinder` с этим радиусом,
 - вычислять объём и площадь боковой поверхности с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
radius = 3
cylinder = Cylinder(radius)
volume = cylinder.calculate_volume()
lateral_area = cylinder.calculate_lateral_area()
print(f"Объём цилиндра равен: {volume}")
print(f"Площадь боковой поверхности равна: {lateral_area}")
```

Вывод:

```
Объём цилиндра равен: 141.3716694115407
Площадь боковой поверхности равна: 94.24777960769379
```

7. Напишите программу, которая создаёт класс `Cone` с методами для вычисления объёма и площади боковой поверхности. Программа должна запрашивать у пользователя радиус основания и выводить вычисленные объём и площадь боковой поверхности (высота конуса фиксирована и равна 10).

Инструкции:

- (a) Создайте класс `Cone` с методом `__init__`, который принимает радиус основания конуса в качестве аргумента и сохраняет его в атрибуте `self.__radius`. Высота конуса фиксирована и равна 10.
- (b) Создайте метод `calculate_volume`, без аргументов, который вычисляет объём конуса по формуле:

$$\frac{1}{3} \cdot \pi \cdot \text{__radius}^2 \cdot 10$$

- (c) Создайте метод `calculate_lateral_area` без аргументов, который вычисляет площадь боковой поверхности конуса по формуле:

$$\pi \cdot \text{__radius} \cdot \sqrt{\text{__radius}^2 + 10^2}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:

- i. запрашивать у пользователя радиус основания конуса,
- ii. создавать экземпляр класса `Cone` с этим радиусом,
- iii. вычислять объём и площадь боковой поверхности с помощью соответствующих методов,
- iv. выводить результаты на экран.

Пример использования:

```
radius = 3
cone = Cone(radius)
volume = cone.calculate_volume()
lateral_area = cone.calculate_lateral_area()
print(f"Объём конуса равен: {volume}")
print(f"Площадь боковой поверхности равна: {lateral_area}")
```

Вывод:

```
Объём конуса равен: 94.24777960769379
Площадь боковой поверхности равна: 94.86832980505137
```

8. Напишите программу, которая создаёт класс `Cube` с методами для вычисления объёма и площади полной поверхности. Программа должна запрашивать у пользователя длину ребра куба и выводить вычисленные объём и площадь.

Инструкции:

- (a) Создайте класс `Cube` с методом `__init__`, который принимает длину ребра куба в качестве аргумента и сохраняет её в атрибуте `self.__side`.
- (b) Создайте метод `calculate_volume`, без аргументов, который вычисляет объём куба по формуле:

$$\text{__side}^3$$

- (c) Создайте метод `calculate_surface_area` без аргументов, который вычисляет площадь полной поверхности куба по формуле:

$$6 \cdot \text{__side}^2$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя длину ребра куба,
 - ii. создавать экземпляр класса `Cube` с этой длиной,
 - iii. вычислять объём и площадь полной поверхности с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
side = 4
cube = Cube(side)
volume = cube.calculate_volume()
surface_area = cube.calculate_surface_area()
print(f"Объём куба равен: {volume}")
print(f"Площадь полной поверхности равна: {surface_area}")
```

Вывод:

Объём куба равен: 64
Площадь полной поверхности равна: 96

9. Напишите программу, которая создаёт класс **Parallelogram** с методами для вычисления площади и периметра. Программа должна запрашивать у пользователя длину основания параллелограмма и выводить вычисленные площадь и периметр (высота параллелограмма фиксирована и равна 8, а боковая сторона равна 6).

Инструкции:

- (a) Создайте класс **Parallelogram** с методом **__init__**, который принимает длину основания параллелограмма в качестве аргумента и сохраняет её в атрибуте **self.__base**. Высота параллелограмма фиксирована и равна 8, а боковая сторона равна 6.
- (b) Создайте метод **calculate_area**, без аргументов, который вычисляет площадь параллелограмма по формуле:

$$\text{__base} \cdot 8$$

- (c) Создайте метод **calculate_perimeter** без аргументов, который вычисляет периметр параллелограмма по формуле:

$$2 \cdot (\text{__base} + 6)$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя длину основания параллелограмма,
 - ii. создавать экземпляр класса **Parallelogram** с этой длиной,
 - iii. вычислять площадь и периметр с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
base = 5
parallelogram = Parallelogram(base)
area = parallelogram.calculate_area()
perimeter = parallelogram.calculate_perimeter()
print(f"Площадь параллелограмма равна: {area}")
print(f"Периметр параллелограмма равен: {perimeter}")
```

Вывод:

Площадь параллелограмма равна: 40

Периметр параллелограмма равен: 22

10. Напишите программу, которая создаёт класс `Ellipse` с методами для вычисления площади и приближённого значения периметра. Программа должна запрашивать у пользователя длину большой полуоси и выводить вычисленные площадь и периметр (длина малой полуоси фиксирована и равна 3).

Инструкции:

- (a) Создайте класс `Ellipse` с методом `__init__`, который принимает длину большой полуоси эллипса в качестве аргумента и сохраняет её в атрибуте `self.__major_axis`. Длина малой полуоси фиксирована и равна 3.
- (b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь эллипса по формуле:

$$\pi \cdot \text{__major_axis} \cdot 3$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет приближённое значение периметра эллипса по формуле Рамануджана:

$$\pi \cdot \left(3(\text{__major_axis} + 3) - \sqrt{(3\text{__major_axis} + 3)(\text{__major_axis} + 9)} \right)$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя длину большой полуоси эллипса,
 - создавать экземпляр класса `Ellipse` с этой длиной,
 - вычислять площадь и периметр с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
major_axis = 5
ellipse = Ellipse(major_axis)
area = ellipse.calculate_area()
perimeter = ellipse.calculate_perimeter()
print(f"Площадь эллипса равна: {area}")
print(f"Периметр эллипса равен: {perimeter}")
```

Вывод:

Площадь эллипса равна: 47.12388980384689

Периметр эллипса равен: 25.74488980384689

11. Напишите программу, которая создаёт класс `BankAccount` с методами для вычисления начисленных процентов и суммы налога на доход. Программа должна запрашивать у пользователя начальный баланс счёта и выводить вычисленные проценты и налог (процентная ставка фиксирована и равна 5%, налоговая ставка на доход фиксирована и равна 13%).

Инструкции:

- (a) Создайте класс `BankAccount` с методом `__init__`, который принимает начальный баланс счёта в качестве аргумента и сохраняет его в атрибуте `self.__balance`.
- (b) Создайте метод `calculate_interest`, без аргументов, который вычисляет начисленные проценты по формуле:

$$\text{__balance} \cdot 0.05$$

- (c) Создайте метод `calculate_tax` без аргументов, который вычисляет сумму налога на полученный доход (проценты) по формуле:

$$(\text{__balance} \cdot 0.05) \cdot 0.13$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя начальный баланс счёта,
 - ii. создавать экземпляр класса `BankAccount` с этим балансом,
 - iii. вычислять начисленные проценты и сумму налога с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
balance = 1000
account = BankAccount(balance)
interest = account.calculate_interest()
tax = account.calculate_tax()
print(f"Начисленные проценты: {interest}")
print(f"Сумма налога на доход: {tax}")
```

Вывод:

```
Начисленные проценты: 50.0
Сумма налога на доход: 6.5
```

- 12. Напишите программу, которая создаёт класс `TemperatureConverter` с методами для преобразования температуры из градусов Цельсия в Фаренгейты и Кельвины. Программа должна запрашивать у пользователя температуру в Цельсиях и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `TemperatureConverter` с методом `__init__`, который принимает температуру в градусах Цельсия в качестве аргумента и сохраняет её в атрибуте `self.__celsius`.

- (b) Создайте метод `to_fahrenheit`, без аргументов, который преобразует температуру в Фаренгейты по формуле:

$$(__celsius \times \frac{9}{5}) + 32$$

- (c) Создайте метод `to_kelvin` без аргументов, который преобразует температуру в Кельвины по формуле:

$$__celsius + 273.15$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя температуру в градусах Цельсия,
 - создавать экземпляр класса `TemperatureConverter` с этим значением,
 - вычислять температуру в Фаренгейтах и Кельвинах с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
celsius = 25
converter = TemperatureConverter(celsius)
fahrenheit = converter.to_fahrenheit()
kelvin = converter.to_kelvin()
print(f"Температура в Фаренгейтах: {fahrenheit}")
print(f"Температура в Кельвинах: {kelvin}")
```

Вывод:

```
Температура в Фаренгейтах: 77.0
Температура в Кельвинах: 298.15
```

13. Напишите программу, которая создаёт класс `DistanceConverter` с методами для преобразования расстояния из метров в километры и мили. Программа должна запрашивать у пользователя расстояние в метрах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `DistanceConverter` с методом `__init__`, который принимает расстояние в метрах в качестве аргумента и сохраняет его в атрибуте `self.__meters`.
- (b) Создайте метод `to_kilometers`, без аргументов, который преобразует расстояние в километры по формуле:

$$__meters \div 1000$$

- (c) Создайте метод `to_miles` без аргументов, который преобразует расстояние в мили по формуле:

$$__meters \div 1609.344$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:

- i. запрашивать у пользователя расстояние в метрах,
- ii. создавать экземпляр класса `DistanceConverter` с этим значением,
- iii. вычислять расстояние в километрах и милях с помощью соответствующих методов,
- iv. выводить результаты на экран.

Пример использования:

```
meters = 1609.344
converter = DistanceConverter(meters)
kilometers = converter.to_kilometers()
miles = converter.to_miles()
print(f"Расстояние в километрах: {kilometers}")
print(f"Расстояние в милях: {miles}")
```

Вывод:

```
Расстояние в километрах: 1.609344
Расстояние в милях: 1.0
```

14. Напишите программу, которая создаёт класс `WeightConverter` с методами для преобразования массы из килограммов в граммы и фунты. Программа должна запрашивать у пользователя массу в килограммах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `WeightConverter` с методом `__init__`, который принимает массу в килограммах в качестве аргумента и сохраняет её в атрибуте `self.__kg`.
- (b) Создайте метод `to_grams`, без аргументов, который преобразует массу в граммы по формуле:

$$\text{__kg} \times 1000$$

- (c) Создайте метод `to_pounds` без аргументов, который преобразует массу в фунты по формуле:

$$\text{__kg} \times 2.20462$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя массу в килограммах,
 - ii. создавать экземпляр класса `WeightConverter` с этим значением,
 - iii. вычислять массу в граммах и фунтах с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
kg = 2.5
converter = WeightConverter(kg)
grams = converter.to_grams()
pounds = converter.to_pounds()
print(f"Масса в граммах: {grams}")
print(f"Масса в фунтах: {pounds}")
```

Вывод:

```
Масса в граммах: 2500.0
Масса в фунтах: 5.51155
```

15. Напишите программу, которая создаёт класс `TimeConverter` с методами для преобразования времени из секунд в минуты и часы. Программа должна запрашивать у пользователя время в секундах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `TimeConverter` с методом `__init__`, который принимает время в секундах в качестве аргумента и сохраняет его в атрибуте `self.__seconds`.
- (b) Создайте метод `to_minutes`, без аргументов, который преобразует время в минуты по формуле:

$$\text{__seconds} \div 60$$

- (c) Создайте метод `to_hours` без аргументов, который преобразует время в часы по формуле:

$$\text{__seconds} \div 3600$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя время в секундах,
 - ii. создавать экземпляр класса `TimeConverter` с этим значением,
 - iii. вычислять время в минутах и часах с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
seconds = 7200
converter = TimeConverter(seconds)
minutes = converter.to_minutes()
hours = converter.to_hours()
print(f"Время в минутах: {minutes}")
print(f"Время в часах: {hours}")
```

Вывод:

Время в минутах: 120.0

Время в часах: 2.0

16. Напишите программу, которая создаёт класс **SpeedConverter** с методами для преобразования скорости из километров в час в метры в секунду и мили в час. Программа должна запрашивать у пользователя скорость в км/ч и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс **SpeedConverter** с методом `__init__`, который принимает скорость в км/ч в качестве аргумента и сохраняет её в атрибуте `self.__kmh`.
- (b) Создайте метод `to_ms`, без аргументов, который преобразует скорость в м/с по формуле:

$$\text{__kmh} \times \frac{1000}{3600}$$

- (c) Создайте метод `to_mph` без аргументов, который преобразует скорость в мили/ч по формуле:

$$\text{__kmh} \div 1.60934$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя скорость в км/ч,
 - создавать экземпляр класса **SpeedConverter** с этим значением,
 - вычислять скорость в м/с и милях/ч с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
kmh = 100
converter = SpeedConverter(kmh)
ms = converter.to_ms()
mph = converter.to_mph()
print(f"Скорость в м/с: {ms}")
print(f"Скорость в милях/ч: {mph}")
```

Вывод:

Скорость в м/с: 27.77777777777778

Скорость в милях/ч: 62.13727366498068

17. Напишите программу, которая создаёт класс **AreaConverter** с методами для преобразования площади из квадратных метров в гектары и акры. Программа должна запрашивать у пользователя площадь в м² и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс **AreaConverter** с методом `__init__`, который принимает площадь в м² в качестве аргумента и сохраняет её в атрибуте `self.__sq_meters`.
- (b) Создайте метод `to_hectares`, без аргументов, который преобразует площадь в гектары по формуле:

$$\text{__sq_meters} \div 10000$$

- (c) Создайте метод `to_acres` без аргументов, который преобразует площадь в акры по формуле:

$$\text{__sq_meters} \div 4046.86$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя площадь в м²,
 - ii. создавать экземпляр класса **AreaConverter** с этим значением,
 - iii. вычислять площадь в гектарах и акрах с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
sq_meters = 10000
converter = AreaConverter(sq_meters)
hectares = converter.to_hectares()
acres = converter.to_acres()
print(f"Площадь в гектарах: {hectares}")
print(f"Площадь в акрах: {acres}")
```

Вывод:

```
Площадь в гектары: 1.0
Площадь в акрах: 2.4710514233241505
```

18. Напишите программу, которая создаёт класс **VolumeConverter** с методами для преобразования объёма из литров в галлоны и кубические метры. Программа должна запрашивать у пользователя объём в литрах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс **VolumeConverter** с методом `__init__`, который принимает объём в литрах в качестве аргумента и сохраняет его в атрибуте `self.__liters`.
- (b) Создайте метод `to_gallons`, без аргументов, который преобразует объём в галлоны по формуле:

$$\text{__liters} \div 3.78541$$

- (c) Создайте метод `to_cubic_meters` без аргументов, который преобразует объём в кубические метры по формуле:

$$\text{__liters} \div 1000$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя объём в литрах,
 - создавать экземпляр класса **VolumeConverter** с этим значением,
 - вычислять объём в галлонах и кубических метрах с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
liters = 10
converter = VolumeConverter(liters)
gallons = converter.to_gallons()
cubic_meters = converter.to_cubic_meters()
print(f"Объём в галлонах: {gallons}")
print(f"Объём в кубических метрах: {cubic_meters}")
```

Вывод:

```
Объём в галлонах: 2.641720523581484
Объём в кубических метрах: 0.01
```

19. Напишите программу, которая создаёт класс **EnergyConverter** с методами для преобразования энергии из джоулей в калории и киловатт-часы. Программа должна запрашивать у пользователя энергию в джоулях и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс **EnergyConverter** с методом `__init__`, который принимает энергию в джоулях в качестве аргумента и сохраняет её в атрибуте `self.__joules`.
- (b) Создайте метод `to_calories`, без аргументов, который преобразует энергию в калории по формуле:

$$\text{__joules} \div 4.184$$

- (c) Создайте метод `to_kwh` без аргументов, который преобразует энергию в киловатт-часы по формуле:

$$\text{__joules} \div 3.6 \times 10^6$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя энергию в джоулях,
 - создавать экземпляр класса **EnergyConverter** с этим значением,
 - вычислять энергию в калориях и киловатт-часах с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
joules = 10000
converter = EnergyConverter(joules)
calories = converter.to_calories()
kwh = converter.to_kwh()
print(f"Энергия в калориях: {calories}")
print(f"Энергия в киловатт-часах: {kwh}")
```

Вывод:

```
Энергия в калориях: 2390.057361376673
Энергия в киловатт-часах: 0.002777777777777778
```

20. Напишите программу, которая создаёт класс **PowerConverter** с методами для преобразования мощности из ватт в лошадиные силы и киловатты. Программа должна запрашивать у пользователя мощность в ваттах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс **PowerConverter** с методом `__init__`, который принимает мощность в ваттах в качестве аргумента и сохраняет её в атрибуте `self.__watts`.
- (b) Создайте метод `to_horsepower`, без аргументов, который преобразует мощность в лошадиные силы по формуле:

$$\text{__watts} \div 745.7$$

- (c) Создайте метод `to_kilowatts` без аргументов, который преобразует мощность в киловатты по формуле:

$$\text{__watts} \div 1000$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя мощность в ваттах,
 - ii. создавать экземпляр класса **PowerConverter** с этим значением,
 - iii. вычислять мощность в л.с. и киловаттах с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
watts = 1000
converter = PowerConverter(watts)
horsepower = converter.to_horsepower()
kilowatts = converter.to_kilowatts()
print(f"Мощность в л.с.: {horsepower}")
print(f"Мощность в киловаттах: {kilowatts}")
```

Вывод:

Мощность в л.с.: 1.3410220903956017

Мощность в киловаттах: 1.0

21. Напишите программу, которая создаёт класс `PressureConverter` с методами для преобразования давления из паскалей в атмосферы и бары. Программа должна запрашивать у пользователя давление в паскалях и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `PressureConverter` с методом `__init__`, который принимает давление в паскалях в качестве аргумента и сохраняет его в атрибуте `self.__pascals`.
- (b) Создайте метод `to_atm`, без аргументов, который преобразует давление в атмосферы по формуле:

$$\text{__pascals} \div 101325$$

- (c) Создайте метод `to_bar` без аргументов, который преобразует давление в бары по формуле:

$$\text{__pascals} \div 100000$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя давление в паскалях,
 - создавать экземпляр класса `PressureConverter` с этим значением,
 - вычислять давление в атмосферах и барах с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
pascals = 101325
converter = PressureConverter(pascals)
atm = converter.to_atm()
bar = converter.to_bar()
print(f"Давление в атмосферах: {atm}")
print(f"Давление в барах: {bar}")
```

Вывод:

Давление в атмосферах: 1.0

Давление в барах: 1.01325

22. Напишите программу, которая создаёт класс `ForceConverter` с методами для преобразования силы из ньютонов в дины и фунты-силы. Программа должна запрашивать у пользователя силу в ньютонах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `ForceConverter` с методом `__init__`, который принимает силу в ньютонах в качестве аргумента и сохраняет её в атрибуте `self.__newtons`.
- (b) Создайте метод `to_dyne`, без аргументов, который преобразует силу в дины по формуле:

$$__newtons \times 100000$$

- (c) Создайте метод `to_pound_force` без аргументов, который преобразует силу в фунты-силы по формуле:

$$__newtons \div 4.44822$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя силу в ньютонах,
 - ii. создавать экземпляр класса `ForceConverter` с этим значением,
 - iii. вычислять силу в динах и фунтах-силы с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
newtons = 10
converter = ForceConverter(newtons)
dyne = converter.to_dyne()
pound_force = converter.to_pound_force()
print(f"Сила в динах: {dyne}")
print(f"Сила в фунтах-силы: {pound_force}")
```

Вывод:

```
Сила в динах: 1000000.0
Сила в фунтах-силы: 2.248089430997145
```

23. Задание: Конвертер силы

Напишите программу, которая создаёт класс `ForceConverter` с методами для преобразования силы из ньютонов в дины и фунты-силы. Программа должна запрашивать у пользователя силу в ньютонах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `ForceConverter` с методом `__init__`, который принимает силу в ньютонах в качестве аргумента и сохраняет её в атрибуте `self.__newtons`.
- (b) Создайте метод `to_dyne`, без аргументов, который преобразует силу в дины по формуле:

$$__newtons \times 100000$$

- (c) Создайте метод `to_pound_force` без аргументов, который преобразует силу в фунты-силы по формуле:

$$\text{__newtons} \div 4.44822$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя силу в ньютонах,
 - создавать экземпляр класса `ForceConverter` с этим значением,
 - вычислять силу в динах и фунтах-силы с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
newtons = 10
converter = ForceConverter(newtons)
dyne = converter.to_dyne()
pound_force = converter.to_pound_force()
print(f"Сила в динах: {dyne}")
print(f"Сила в фунтах-силы: {pound_force}")
```

Вывод:

```
Сила в динах: 1000000.0
Сила в фунтах-силы: 2.248089430997145
```

24. Напишите программу, которая создаёт класс `ResistanceConverter` с методами для преобразования электрического сопротивления из омов в килоомы и мегаомы. Программа должна запрашивать у пользователя сопротивление в омах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `ResistanceConverter` с методом `__init__`, который принимает сопротивление в омах в качестве аргумента и сохраняет его в атрибуте `self.__ohms`.
- (b) Создайте метод `to_kiloohms`, без аргументов, который преобразует сопротивление в килоомы по формуле:

$$\text{__ohms} \div 1000$$

- (c) Создайте метод `to_megaohms` без аргументов, который преобразует сопротивление в мегаомы по формуле:

$$\text{__ohms} \div 1000000$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя сопротивление в омах,
 - создавать экземпляр класса `ResistanceConverter` с этим значением,
 - вычислять сопротивление в килоомах и мегаомах с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
ohms = 10000
converter = ResistanceConverter(ohms)
kiloohms = converter.to_kiloohms()
megaohms = converter.to_megaohms()
print(f"Сопротивление в килоомах: {kiloohms}")
print(f"Сопротивление в мегаомах: {megaohms}")
```

Вывод:

```
Сопротивление в килоомах: 10.0
Сопротивление в мегаомах: 0.01
```

25. Дополнительные задания

26. Напишите программу, которая создаёт класс `Pentagon` с методами для вычисления площади и периметра правильного пятиугольника. Программа должна запрашивать у пользователя длину стороны и выводить вычисленные площадь и периметр.

Инструкции:

- (a) Создайте класс `Pentagon` с методом `__init__`, который принимает длину стороны пятиугольника в качестве аргумента и сохраняет её в атрибуте `self.__side`.
- (b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь правильного пятиугольника по формуле:

$$\frac{1}{4} \sqrt{5(5 + 2\sqrt{5})} \cdot \text{__side}^2$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет периметр пятиугольника по формуле:

$$5 \cdot \text{__side}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя длину стороны пятиугольника,
 - ii. создавать экземпляр класса `Pentagon` с этой длиной,
 - iii. вычислять площадь и периметр с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
side = 5
pentagon = Pentagon(side)
area = pentagon.calculate_area()
perimeter = pentagon.calculate_perimeter()
print(f"Площадь пятиугольника: {area}")
print(f"Периметр пятиугольника: {perimeter}")
```

Вывод:

Площадь пятиугольника: 43.01193501472417

Периметр пятиугольника: 25

27. Напишите программу, которая создаёт класс `Hexagon` с методами для вычисления площади и периметра правильного шестиугольника. Программа должна запрашивать у пользователя длину стороны и выводить вычисленные площадь и периметр.

Инструкции:

- (a) Создайте класс `Hexagon` с методом `__init__`, который принимает длину стороны шестиугольника в качестве аргумента и сохраняет её в атрибуте `self.__side`.
- (b) Создайте метод `calculate_area`, без аргументов, который вычисляет площадь правильного шестиугольника по формуле:

$$\frac{3\sqrt{3}}{2} \cdot \text{__side}^2$$

- (c) Создайте метод `calculate_perimeter` без аргументов, который вычисляет периметр шестиугольника по формуле:

$$6 \cdot \text{__side}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя длину стороны шестиугольника,
 - ii. создавать экземпляр класса `Hexagon` с этой длиной,
 - iii. вычислять площадь и периметр с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
side = 4
hexagon = Hexagon(side)
area = hexagon.calculate_area()
perimeter = hexagon.calculate_perimeter()
print(f"Площадь шестиугольника: {area}")
print(f"Периметр шестиугольника: {perimeter}")
```

Вывод:

Площадь шестиугольника: 41.569219381653056

Периметр шестиугольника: 24

28. Напишите программу, которая создаёт класс `AngleConverter` с методами для преобразования углов из градусов в радианы и градусы. Программа должна запрашивать у пользователя угол в градусах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `AngleConverter` с методом `__init__`, который принимает угол в градусах в качестве аргумента и сохраняет его в атрибуте `self.__degrees`.
- (b) Создайте метод `to_radians`, без аргументов, который преобразует угол в радианы по формуле:

$$\text{__degrees} \times \frac{\pi}{180}$$

- (c) Создайте метод `to_gradians` без аргументов, который преобразует угол в грады по формуле:

$$\text{__degrees} \times \frac{10}{9}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя угол в градусах,
 - ii. создавать экземпляр класса `AngleConverter` с этим значением,
 - iii. вычислять угол в радианах и градах с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
degrees = 90
converter = AngleConverter(degrees)
radians = converter.to_radians()
gradians = converter.to_gradians()
print(f"Угол в радианах: {radians}")
print(f"Угол в градах: {gradians}")
```

Вывод:

```
Угол в радианах: 1.5707963267948966
Угол в градах: 100.0
```

29. Напишите программу, которая создаёт класс `Tetrahedron` с методами для вычисления объёма и площади поверхности правильного тетраэдра. Программа должна запрашивать у пользователя длину ребра и выводить вычисленные объём и площадь поверхности.

Инструкции:

- (a) Создайте класс `Tetrahedron` с методом `__init__`, который принимает длину ребра тетраэдра в качестве аргумента и сохраняет её в атрибуте `self.__edge`.
- (b) Создайте метод `calculate_volume`, без аргументов, который вычисляет объём тетраэдра по формуле:

$$\frac{\text{__edge}^3}{6\sqrt{2}}$$

- (c) Создайте метод `calculate_surface_area` без аргументов, который вычисляет площадь поверхности тетраэдра по формуле:

$$\sqrt{3} \cdot \text{__edge}^2$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
- запрашивать у пользователя длину ребра тетраэдра,
 - создавать экземпляр класса `Tetrahedron` с этой длиной,
 - вычислять объём и площадь поверхности с помощью соответствующих методов,
 - выводить результаты на экран.

Пример использования:

```
edge = 3
tetrahedron = Tetrahedron(edge)
volume = tetrahedron.calculate_volume()
surface_area = tetrahedron.calculate_surface_area()
print(f"Объём тетраэдра: {volume}")
print(f"Площадь поверхности: {surface_area}")
```

Вывод:

```
Объём тетраэдра: 3.181980515339464
Площадь поверхности: 15.588457268119896
```

30. Напишите программу, которая создаёт класс `CubicMeterConverter` с методами для преобразования объёма из кубических метров в литры и кубические футы. Программа должна запрашивать у пользователя объём в кубометрах и выводить преобразованные значения.

Инструкции:

- (a) Создайте класс `CubicMeterConverter` с методом `__init__`, который принимает объём в кубических метрах в качестве аргумента и сохраняет его в атрибуте `self.__cubic_meters`.
- (b) Создайте метод `to_liters`, без аргументов, который преобразует объём в литры по формуле:

$$\text{__cubic_meters} \times 1000$$

- (c) Создайте метод `__cubic_feet` без аргументов, который преобразует объём в кубические футы по формуле:

$$\text{__cubic_meters} \times 35.3147$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:

- i. запрашивать у пользователя объём в кубических метрах,
- ii. создавать экземпляр класса `CubicMeterConverter` с этим значением,
- iii. вычислять объём в литрах и кубических футах с помощью соответствующих методов,
- iv. выводить результаты на экран.

Пример использования:

```
cubic_meters = 2
converter = CubicMeterConverter(cubic_meters)
liters = converter.to_liters()
cubic_feet = converter.to_cubic_feet()
print(f"Объём в литрах: {liters}")
print(f"Объём в кубических футах: {cubic_feet}")
```

Вывод:

```
Объём в литрах: 2000.0
Объём в кубических футах: 70.6294
```

31. Напишите программу, которая создаёт класс `RightTriangle` с методами для вычисления гипотенузы и площади прямоугольного треугольника. Программа должна запрашивать у пользователя длину одного катета (второй катет фиксирован и равен 4) и выводить вычисленные гипотенузу и площадь.

Инструкции:

- (a) Создайте класс `RightTriangle` с методом `__init__`, который принимает длину первого катета в качестве аргумента и сохраняет его в атрибуте `self.__cathetus`. Второй катет фиксирован и равен 4.
- (b) Создайте метод `calculate_hypotenuse`, без аргументов, который вычисляет гипотенузу по формуле:

$$\sqrt{\text{__cathetus}^2 + 4^2}$$

- (c) Создайте метод `calculate_area` без аргументов, который вычисляет площадь треугольника по формуле:

$$\frac{\text{__cathetus} \times 4}{2}$$

- (d) Напишите цикл, который повторяется 10 раз. В каждой итерации программа должна:
 - i. запрашивать у пользователя длину катета,
 - ii. создавать экземпляр класса `RightTriangle` с этой длиной,
 - iii. вычислять гипотенузу и площадь с помощью соответствующих методов,
 - iv. выводить результаты на экран.

Пример использования:

```
cathetus = 3
triangle = RightTriangle(cathetus)
hypotenuse = triangle.calculate_hypotenuse()
area = triangle.calculate_area()
print(f"Гипотенуза: {hypotenuse}")
print(f"Площадь: {area}")
```

Вывод:

```
Гипотенуза: 5.0
Площадь: 6.0
```

Задача 2

1. Написать программу, которая создаёт класс `LeapYearChecker` для определения високосного года. В классе должен быть статический метод `is_leap_year` и возвращать `True`, если год високосный, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого года от 2000 до 2099 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `LeapYearChecker`.
- (b) Создайте **статический** метод `is_leap_year`, который принимает год в качестве аргумента и проверяет, является ли год високосным. Если год делится на 4 без остатка и не делится на 100 без остатка, или делится на 400 без остатка, то возвращает `True`. В противном случае возвращает `False`.
- (c) Используйте цикл для проверки каждого года от 2000 до 2099 (включительно), вызывая статический метод `is_leap_year` и вывода результат на экран.

Пример использования:

```
v = LeapYearChecker.is_leap_year(1999)
```

Вывод (первые и последние строки):

```
2000 True
2001 False
...
2098 False
2099 False
```

2. Написать программу, которая создаёт класс `PrimeChecker` для определения простого числа. В классе должен быть статический метод `is_prime` и возвращать `True`, если число простое, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PrimeChecker`.
- (b) Создайте **статический** метод `is_prime`, который принимает число в качестве аргумента и проверяет, является ли число простым. Простое число делится только на 1 и на само себя.
- (c) Используйте цикл для проверки каждого числа от 1 до 100 (включительно), вызывая статический метод `is_prime` и выводя результат на экран.

Пример использования:

```
v = PrimeChecker.is_prime(17)
```

Вывод (первые и последние строки):

```
1 False
2 True
3 True
...
98 False
99 False
100 False
```

- 3. Написать программу, которая создаёт класс `EvenChecker` для определения чётности числа. В классе должен быть статический метод `is_even` и возвращать `True`, если число чётное, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 50 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `EvenChecker`.
- (b) Создайте **статический** метод `is_even`, который принимает число в качестве аргумента и проверяет, является ли число чётным.
- (c) Используйте цикл для проверки каждого числа от 1 до 50 (включительно), вызывая статический метод `is_even` и выводя результат на экран.

Пример использования:

```
v = EvenChecker.is_even(25)
```

Вывод (первые и последние строки):

```
1 False
2 True
3 False
...
48 True
49 False
50 True
```

4. Написать программу, которая создаёт класс **SquareChecker** для определения квадратного числа. В классе должен быть статический метод **is_square** и возвращать **True**, если число является квадратом целого числа, и **False** в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс **SquareChecker**.
- (b) Создайте **статический** метод **is_square**, который принимает число в качестве аргумента и проверяет, является ли число квадратом целого числа.
- (c) Используйте цикл для проверки каждого числа от 1 до 100 (включительно), вызывая статический метод **is_square** и выводя результат на экран.

Пример использования:

```
v = SquareChecker.is_square(36)
```

Вывод (первые и последние строки):

```
1 True
2 False
3 False
...
99 False
100 True
```

5. Написать программу, которая создаёт класс **FactorialCalculator** для вычисления факториала числа. В классе должен быть статический метод **factorial** и возвращать факториал числа. Программа также должна использовать цикл для вычисления факториала каждого числа от 1 до 10 и вывода результата на экран.

Инструкции:

- (a) Создайте класс **FactorialCalculator**.
- (b) Создайте **статический** метод **factorial**, который принимает число в качестве аргумента и возвращает его факториал.
- (c) Используйте цикл для вычисления факториала каждого числа от 1 до 10 (включительно), вызывая статический метод **factorial** и выводя результат на экран.

Пример использования:

```
v = FactorialCalculator.factorial(5)
```

Вывод (первые и последние строки):

```

1 1
2 2
3 6
...
9 362880
10 3628800

```

6. Написать программу, которая создаёт класс `PalindromeChecker` для определения палиндрома числа. В классе должен быть статический метод `is_palindrome` и возвращать `True`, если число является палиндромом, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 100 до 200 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PalindromeChecker`.
- (b) Создайте **статический** метод `is_palindrome`, который принимает число в качестве аргумента и проверяет, является ли число палиндромом (читается одинаково слева направо и справа налево).
- (c) Используйте цикл для проверки каждого числа от 100 до 200 (включительно), вызывая статический метод `is_palindrome` и выводя результат на экран.

Пример использования:

```
v = PalindromeChecker.is_palindrome(121)
```

Вывод (первые и последние строки):

```

100 False
101 True
102 False
...
199 False
200 False

```

7. Написать программу, которая создаёт класс `ArmstrongChecker` для определения числа Армстронга. В классе должен быть статический метод `is_armstrong` и возвращать `True`, если число является числом Армстронга, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 100 до 500 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `ArmstrongChecker`.
- (b) Создайте **статический** метод `is_armstrong`, который принимает число в качестве аргумента и проверяет, является ли число числом Армстронга (сумма цифр в степени, равной количеству цифр, равна самому числу).
- (c) Используйте цикл для проверки каждого числа от 100 до 500 (включительно), вызывая статический метод `is_armstrong` и выводя результат на экран.

Пример использования:

```
v = ArmstrongChecker.is_armstrong(153)
```

Вывод (первые и последние строки):

```
100 False
101 False
102 False
...
499 False
500 False
```

8. Написать программу, которая создаёт класс `PerfectNumberChecker` для определения совершенного числа. В классе должен быть статический метод `is_perfect` и возвращать `True`, если число является совершенным, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 1000 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PerfectNumberChecker`.
- (b) Создайте **статический** метод `is_perfect`, который принимает число в качестве аргумента и проверяет, является ли число совершенным (сумма делителей равна числу).
- (c) Используйте цикл для проверки каждого числа от 1 до 1000 (включительно), вызывая статический метод `is_perfect` и выводя результат на экран.

Пример использования:

```
v = PerfectNumberChecker.is_perfect(28)
```

Вывод (первые и последние строки):

```
1 False
2 False
3 False
...
998 False
999 False
1000 False
```

9. Написать программу, которая создаёт класс `FibonacciChecker` для проверки числа Фибоначчи. В классе должен быть статический метод `is_fibonacci` и возвращать `True`, если число является числом Фибоначчи, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `FibonacciChecker`.
- (b) Создайте **статический** метод `is_fibonacci`, который принимает число в качестве аргумента и проверяет, является ли число числом Фибоначчи.
- (c) Используйте цикл для проверки каждого числа от 1 до 100 (включительно), вызывая статический метод `is_fibonacci` и выводя результат на экран.

Пример использования:

```
v = FibonacciChecker.is_fibonacci(21)
```

Вывод (первые и последние строки):

```
1 True
2 True
3 True
...
98 False
99 False
100 False
```

10. Написать программу, которая создаёт класс `PowerOfTwoChecker` для проверки степени двойки. В классе должен быть статический метод `is_power_of_two` и возвращать `True`, если число является степенью двойки, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 128 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PowerOfTwoChecker`.
- (b) Создайте **статический** метод `is_power_of_two`, который принимает число в качестве аргумента и проверяет, является ли число степенью двойки.
- (c) Используйте цикл для проверки каждого числа от 1 до 128 (включительно), вызывая статический метод `is_power_of_two` и выводя результат на экран.

Пример использования:

```
v = PowerOfTwoChecker.is_power_of_two(64)
```

Вывод (первые и последние строки):

```
1 True
2 True
3 False
...
127 False
128 True
```

11. Написать программу, которая создаёт класс `SumOfDigitsCalculator` для вычисления суммы цифр числа. В классе должен быть статический метод `sum_of_digits` и возвращать сумму цифр. Программа также должна использовать цикл для вычисления суммы цифр каждого числа от 1 до 50 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `SumOfDigitsCalculator`.
- (b) Создайте **статический** метод `sum_of_digits`, который принимает число в качестве аргумента и возвращает сумму его цифр.
- (c) Используйте цикл для вычисления суммы цифр каждого числа от 1 до 50 (включительно), вызывая статический метод `sum_of_digits` и выводя результат на экран.

Пример использования:

```
v = SumOfDigitsCalculator.sum_of_digits(123)
```

Вывод (первые и последние строки):

```
1 1
2 2
3 3
...
49 13
50 5
```

12. Написать программу, которая создаёт класс `PrimeSumCalculator` для вычисления суммы простых чисел в диапазоне. В классе должен быть статический метод `sum_of_primes` и возвращать сумму простых чисел в заданном диапазоне. Программа также должна использовать цикл для вычисления суммы простых чисел от 1 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PrimeSumCalculator`.
- (b) Создайте **статический** метод `sum_of_primes`, который принимает два аргумента (начало и конец диапазона) и возвращает сумму простых чисел в этом диапазоне.
- (c) Используйте метод для вычисления суммы простых чисел от 1 до 100 и выведите результат.

Пример использования:

```
v = PrimeSumCalculator.sum_of_primes(1, 10)
```

Вывод:

Сумма простых чисел от 1 до 100: 1060

13. Написать программу, которая создаёт класс `DigitCountCalculator` для подсчёта количества цифр в числе. В классе должен быть статический метод `digit_count` и возвращать количество цифр. Программа также должна использовать цикл для подсчёта цифр каждого числа от 1 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `DigitCountCalculator`.
- (b) Создайте **статический** метод `digit_count`, который принимает число в качестве аргумента и возвращает количество его цифр.
- (c) Используйте цикл для подсчёта цифр каждого числа от 1 до 100 (включительно), вызывая статический метод `digit_count` и выводя результат на экран.

Пример использования:

```
v = DigitCountCalculator.digit_count(12345)
```

Вывод (первые и последние строки):

```
1 1
2 1
3 1
...
99 2
100 3
```

14. Написать программу, которая создаёт класс `BinaryConverter` для преобразования числа в двоичное представление. В классе должен быть статический метод `to_binary` и возвращать строку с двоичным представлением числа. Программа также должна использовать цикл для преобразования каждого числа от 1 до 16 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `BinaryConverter`.
- (b) Создайте **статический** метод `to_binary`, который принимает число в качестве аргумента и возвращает его двоичное представление в виде строки.
- (c) Используйте цикл для преобразования каждого числа от 1 до 16 (включительно), вызывая статический метод `to_binary` и выводя результат на экран.

Пример использования:

```
v = BinaryConverter.to_binary(10)
```

Вывод (первые и последние строки):

```

1 1
2 10
3 11
...
15 1111
16 10000

```

15. Написать программу, которая создаёт класс `HexConverter` для преобразования числа в шестнадцатеричное представление. В классе должен быть статический метод `to_hex` и возвращать строку с шестнадцатеричным представлением числа. Программа также должна использовать цикл для преобразования каждого числа от 1 до 20 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `HexConverter`.
- (b) Создайте **статический** метод `to_hex`, который принимает число в качестве аргумента и возвращает его шестнадцатеричное представление в виде строки.
- (c) Используйте цикл для преобразования каждого числа от 1 до 20 (включительно), вызывая статический метод `to_hex` и выводя результат на экран.

Пример использования:

```
v = HexConverter.to_hex(255)
```

Вывод (первые и последние строки):

```

1 1
2 2
3 3
...
19 13
20 14

```

16. Написать программу, которая создаёт класс `DivisorChecker` для проверки делителей числа. В классе должен быть статический метод `get_divisors` и возвращать список делителей числа. Программа также должна использовать цикл для вывода делителей каждого числа от 1 до 20 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `DivisorChecker`.
- (b) Создайте **статический** метод `get_divisors`, который принимает число в качестве аргумента и возвращает список его делителей.
- (c) Используйте цикл для вывода делителей каждого числа от 1 до 20 (включительно), вызывая статический метод `get_divisors` и выводя результат на экран.

Пример использования:

```
v = DivisorChecker.get_divisors(12)
```

Вывод (первые и последние строки):

```
1 [1]
2 [1, 2]
3 [1, 3]
...
19 [1, 19]
20 [1, 2, 4, 5, 10, 20]
```

17. Написать программу, которая создаёт класс `Multiplier` для создания таблицы умножения. В классе должен быть статический метод `multiply_table` и выводить таблицу умножения для заданного числа. Программа также должна использовать цикл для вывода таблицы умножения для чисел от 1 до 10 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `Multiplier`.
- (b) Создайте **статический** метод `multiply_table`, который принимает число в качестве аргумента и выводит таблицу умножения для этого числа от 1 до 10.
- (c) Используйте цикл для вывода таблицы умножения для чисел от 1 до 10 (включительно), вызывая статический метод `multiply_table` и выводя результат на экран.

Пример использования:

```
Multiplier.multiply_table(5)
```

Вывод (для числа 5):

```
5 * 1 = 5
5 * 2 = 10
...
5 * 10 = 50
```

18. Написать программу, которая создаёт класс `GCDCalculator` для вычисления НОД двух чисел. В классе должен быть статический метод `gcd` и возвращать наибольший общий делитель. Программа также должна использовать цикл для вычисления НОД чисел (1,1), (2,4), (3,9), ..., (10,100) и вывода результата на экран.

Инструкции:

- (a) Создайте класс `GCDCalculator`.
- (b) Создайте **статический** метод `gcd`, который принимает два числа в качестве аргументов и возвращает их наибольший общий делитель.
- (c) Используйте цикл для вычисления НОД пар чисел (1,1), (2,4), (3,9), ..., (10,100), вызывая статический метод `gcd` и выводя результат на экран.

Пример использования:

```
v = GCDCalculator.gcd(48, 18)
```

Вывод:

```
НОД(1, 1) = 1
НОД(2, 4) = 2
НОД(3, 9) = 3
...
НОД(10, 100) = 10
```

19. Написать программу, которая создаёт класс `LCMCalculator` для вычисления НОК двух чисел. В классе должен быть статический метод `lcm` и возвращать наименьшее общее кратное. Программа также должна использовать цикл для вычисления НОК чисел (1,1), (2,3), (3,5), ..., (10,11) и вывода результата на экран.

Инструкции:

- (a) Создайте класс `LCMCalculator`.
- (b) Создайте **статический** метод `lcm`, который принимает два числа в качестве аргументов и возвращает их наименьшее общее кратное.
- (c) Используйте цикл для вычисления НОК пар чисел (1,1), (2,3), (3,5), ..., (10,11), вызывая статический метод `lcm` и выводя результат на экран.

Пример использования:

```
v = LCMCalculator.lcm(4, 6)
```

Вывод:

```
НОК(1, 1) = 1
НОК(2, 3) = 6
НОК(3, 5) = 15
...
НОК(10, 11) = 110
```

20. Написать программу, которая создаёт класс `DigitReverse` для разворота цифр числа. В классе должен быть статический метод `reverse_digits` и возвращать число с обратным порядком цифр. Программа также должна использовать цикл для разворота каждого числа от 10 до 20 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `DigitReverse`.
- (b) Создайте **статический** метод `reverse_digits`, который принимает число в качестве аргумента и возвращает число с обратным порядком цифр.
- (c) Используйте цикл для разворота каждого числа от 10 до 20 (включительно), вызывая статический метод `reverse_digits` и выводя результат на экран.

Пример использования:

```
v = DigitReverse.reverse_digits(123)
```

Вывод:

```
10 1
11 11
12 21
13 31
...
19 91
20 2
```

21. Написать программу, которая создаёт класс `NumberTypeChecker` для определения типа числа (положительное/отрицательное/ноль). В классе должен быть статический метод `check_number_type` и возвращать строку с типом числа. Программа также должна использовать цикл для проверки чисел `[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]` и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberTypeChecker`.
- (b) Создайте **статический** метод `check_number_type`, который принимает число в качестве аргумента и возвращает строку "positive" "negative" или "zero".
- (c) Используйте цикл для проверки чисел `[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]`, вызывая статический метод `check_number_type` и выводя результат на экран.

Пример использования:

```
v = NumberTypeChecker.check_number_type(-7)
```

Вывод:

```
-5 negative
-4 negative
-3 negative
-2 negative
-1 negative
0 zero
1 positive
2 positive
3 positive
4 positive
5 positive
```

22. Написать программу, которая создаёт класс `FactorialChecker` для проверки факториала числа. В классе должен быть статический метод `is_factorial` и возвращать `True`, если число является факториалом какого-либо числа, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 120 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `FactorialChecker`.
- (b) Создайте **статический** метод `is_factorial`, который принимает число в качестве аргумента и проверяет, является ли число факториалом какого-либо числа.
- (c) Используйте цикл для проверки каждого числа от 1 до 120 (включительно), вызывая статический метод `is_factorial` и выводя результат на экран.

Пример использования:

```
v = FactorialChecker.is_factorial(24)
```

Вывод (первые и последние строки):

```
1 True
2 True
3 False
...
119 False
120 True
```

23. Написать программу, которая создаёт класс `PowerChecker` для проверки степени числа. В классе должен быть статический метод `is_power` и возвращать `True`, если число является степенью заданного основания, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 1 до 100 относительно основания 3 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `PowerChecker`.
- (b) Создайте **статический** метод `is_power`, который принимает число и основание в качестве аргументов и проверяет, является ли число степенью основания.
- (c) Используйте цикл для проверки каждого числа от 1 до 100 (включительно) относительно основания 3, вызывая статический метод `is_power` и выводя результат на экран.

Пример использования:

```
v = PowerChecker.is_power(81, 3)
```

Вывод (первые и последние строки):

```
1 True
2 False
3 True
...
99 False
100 False
```


24. Написать программу, которая создаёт класс `DigitProductCalculator` для вычисления произведения цифр числа. В классе должен быть статический метод `digit_product` и возвращать произведение цифр. Программа также должна использовать цикл для вычисления произведения цифр каждого числа от 1 до 50 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `DigitProductCalculator`.
- (b) Создайте **статический** метод `digit_product`, который принимает число в качестве аргумента и возвращает произведение его цифр.
- (c) Используйте цикл для вычисления произведения цифр каждого числа от 1 до 50 (включительно), вызывая статический метод `digit_product` и выводя результат на экран.

Пример использования:

```
v = DigitProductCalculator.digit_product(123)
```

Вывод (первые и последние строки):

```
1 1
2 2
3 3
...
49 36
50 0
```

25. Написать программу, которая создаёт класс `NumberLengthChecker` для проверки длины числа. В классе должен быть статический метод `get_length` и возвращать количество цифр в числе. Программа также должна использовать цикл для проверки длины каждого числа от 1 до 1000 с шагом 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberLengthChecker`.
- (b) Создайте **статический** метод `get_length`, который принимает число в качестве аргумента и возвращает количество его цифр.
- (c) Используйте цикл для проверки длины чисел 1, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, вызывая статический метод `get_length` и выводя результат на экран.

Пример использования:

```
v = NumberLengthChecker.get_length(12345)
```

Вывод:

```
1 1
100 3
200 3
300 3
400 3
500 3
600 3
700 3
800 3
900 3
1000 4
```

26. Написать программу, которая создаёт класс `NumberSquareSumCalculator` для вычисления суммы квадратов чисел. В классе должен быть статический метод `square_sum` и возвращать сумму квадратов чисел в диапазоне. Программа также должна использовать метод для вычисления суммы квадратов чисел от 1 до 10 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberSquareSumCalculator`.
- (b) Создайте **статический** метод `square_sum`, который принимает два аргумента (начало и конец диапазона) и возвращает сумму квадратов чисел в этом диапазоне.
- (c) Используйте метод для вычисления суммы квадратов чисел от 1 до 10 и выведите результат.

Пример использования:

```
v = NumberSquareSumCalculator.square_sum(1, 3)
```

Вывод:

Сумма квадратов чисел от 1 до 10: 385

27. Написать программу, которая создаёт класс `NumberCubeSumCalculator` для вычисления суммы кубов чисел. В классе должен быть статический метод `cube_sum` и возвращать сумму кубов чисел в диапазоне. Программа также должна использовать метод для вычисления суммы кубов чисел от 1 до 10 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberCubeSumCalculator`.
- (b) Создайте **статический** метод `cube_sum`, который принимает два аргумента (начало и конец диапазона) и возвращает сумму кубов чисел в этом диапазоне.
- (c) Используйте метод для вычисления суммы кубов чисел от 1 до 10 и выведите результат.

Пример использования:

```
v = NumberCubeSumCalculator.cube_sum(1, 3)
```

Вывод:

Сумма кубов чисел от 1 до 10: 3025

28. Написать программу, которая создаёт класс `NumberRangeChecker` для проверки числа на принадлежность диапазону. В классе должен быть статический метод `in_range` и возвращать `True`, если число находится в заданном диапазоне, и `False` в противном случае. Программа также должна использовать цикл для проверки чисел от -5 до 5 на принадлежность диапазону `[0, 10]` и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberRangeChecker`.
- (b) Создайте **статический** метод `in_range`, который принимает число, начало и конец диапазона и проверяет, находится ли число в этом диапазоне.
- (c) Используйте цикл для проверки чисел от -5 до 5 (включительно) на принадлежность диапазону `[0, 10]`, вызывая статический метод `in_range` и выводя результат на экран.

Пример использования:

```
v = NumberRangeChecker.in_range(5, 0, 10)
```

Вывод:

```
-5 False
-4 False
-3 False
-2 False
-1 False
0 True
1 True
2 True
3 True
4 True
5 True
```

29. Написать программу, которая создаёт класс `NumberSignChecker` для проверки знака числа. В классе должен быть статический метод `get_sign` и возвращать строку с знаком числа (+, - или 0). Программа также должна использовать цикл для проверки чисел `[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]` и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberSignChecker`.
- (b) Создайте **статический** метод `get_sign`, который принимает число в качестве аргумента и возвращает строку с его знаком (+, - или 0).
- (c) Используйте цикл для проверки чисел [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], вызывая статический метод `get_sign` и выводя результат на экран.

Пример использования:

```
v = NumberSignChecker.get_sign(-7)
```

Вывод:

```
-5 -  
-4 -  
-3 -  
-2 -  
-1 -  
0 0  
1 +  
2 +  
3 +  
4 +  
5 +
```

30. Написать программу, которая создаёт класс `NumberPalindromeChecker` для проверки палиндрома числа. В классе должен быть статический метод `is_palindrome` и возвращать `True`, если число является палиндромом, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 100 до 150 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberPalindromeChecker`.
- (b) Создайте **статический** метод `is_palindrome`, который принимает число в качестве аргумента и проверяет, является ли число палиндромом.
- (c) Используйте цикл для проверки каждого числа от 100 до 150 (включительно), вызывая статический метод `is_palindrome` и выводя результат на экран.

Пример использования:

```
v = NumberPalindromeChecker.is_palindrome(121)
```

Вывод (первые и последние строки):

```
100 False
101 True
102 False
...
149 False
150 False
```

31. Написать программу, которая создаёт класс `NumberAscendingChecker` для проверки, что цифры числа идут в порядке возрастания. В классе должен быть статический метод `is_ascending` и возвращать `True`, если цифры числа идут в порядке возрастания, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 10 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberAscendingChecker`.
- (b) Создайте **статический** метод `is_ascending`, который принимает число в качестве аргумента и проверяет, идут ли его цифры в порядке возрастания.
- (c) Используйте цикл для проверки каждого числа от 10 до 100 (включительно), вызывая статический метод `is_ascending` и выводя результат на экран.

Пример использования:

```
v = NumberAscendingChecker.is_ascending(123)
```

Вывод (первые и последние строки):

```
10 False
11 False
12 True
13 True
...
98 False
99 False
100 False
```

32. Написать программу, которая создаёт класс `NumberDescendingChecker` для проверки, что цифры числа идут в порядке убывания. В классе должен быть статический метод `is_descending` и возвращать `True`, если цифры числа идут в порядке убывания, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 10 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberDescendingChecker`.
- (b) Создайте **статический** метод `is_descending`, который принимает число в качестве аргумента и проверяет, идут ли его цифры в порядке убывания.
- (c) Используйте цикл для проверки каждого числа от 10 до 100 (включительно), вызывая статический метод `is_descending` и выводя результат на экран.

Пример использования:

```
v = NumberDescendingChecker.is_descending(321)
```

Вывод (первые и последние строки):

```
10 False
11 False
12 False
13 False
...
98 True
99 True
100 False
```

33. Написать программу, которая создаёт класс `NumberPrimeDigitChecker` для проверки, что все цифры числа простые. В классе должен быть статический метод `all_digits_prime` и возвращать `True`, если все цифры числа простые, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 10 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberPrimeDigitChecker`.
- (b) Создайте **статический** метод `all_digits_prime`, который принимает число в качестве аргумента и проверяет, являются ли все его цифры простыми числами.
- (c) Используйте цикл для проверки каждого числа от 10 до 100 (включительно), вызывая статический метод `all_digits_prime` и выводя результат на экран.

Пример использования:

```
v = NumberPrimeDigitChecker.all_digits_prime(23)
```

Вывод (первые и последние строки):

```
10 False
11 False
12 False
13 False
...
98 False
99 False
100 False
```

34. Написать программу, которая создаёт класс `NumberEvenDigitChecker` для проверки, что все цифры числа чётные. В классе должен быть статический метод `all_digits_even` и возвращать `True`, если все цифры числа чётные, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 10 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberEvenDigitChecker`.
- (b) Создайте **статический** метод `all_digits_even`, который принимает число в качестве аргумента и проверяет, являются ли все его цифры чётными.
- (c) Используйте цикл для проверки каждого числа от 10 до 100 (включительно), вызывая статический метод `all_digits_even` и выводя результат на экран.

Пример использования:

```
v = NumberEvenDigitChecker.all_digits_even(24)
```

Вывод (первые и последние строки):

```
10 False
11 False
12 False
13 False
...
98 False
99 False
100 False
```

35. Написать программу, которая создаёт класс `NumberOddDigitChecker` для проверки, что все цифры числа нечётные. В классе должен быть статический метод `all_digits_odd` и возвращать `True`, если все цифры числа нечётные, и `False` в противном случае. Программа также должна использовать цикл для проверки каждого числа от 10 до 100 и вывода результата на экран.

Инструкции:

- (a) Создайте класс `NumberOddDigitChecker`.
- (b) Создайте **статический** метод `all_digits_odd`, который принимает число в качестве аргумента и проверяет, являются ли все его цифры нечётными.
- (c) Используйте цикл для проверки каждого числа от 10 до 100 (включительно), вызывая статический метод `all_digits_odd` и выводя результат на экран.

Пример использования:

```
v = NumberOddDigitChecker.all_digits_odd(135)
```

Вывод (первые и последние строки):

```
10 False
11 True
12 False
13 True
```

```
...
98 False
99 True
100 False
```

Задача 3

1. Написать программу на Python, которая создает класс **Person** для представления сотрудника персонала. Класс должен содержать закрытые атрибуты `__name`, `__country`, `__date_of_birth` и метод `calculate_age`. Доступ к атрибутам только через методы-геттеры. Создать экземпляры и вывести информацию о каждом человеке.

Инструкции:

- (a) Создайте класс **Person** с методом `__init__`, который принимает имя, страну и дату рождения.
- (b) Создайте методы-геттеры: `get_name()`, `get_country()`, `get_date_of_birth()`.
- (c) Создайте метод `calculate_age()` для вычисления возраста.
- (d) Создайте несколько экземпляров класса **Person**.
- (e) Выведите данные каждого человека через методы класса.

Пример использования:

Листинг 1: Пример кода

```
from datetime import date

person1 = Person("Иванов Иван Иванович", "Россия", date(1946, 8, 15))
person2 = Person("Петров Сергей Александрович", "Белоруссия", date(1982, 10,
    22))

print("Персона 1:")
print("Имя: ", person1.get_name())
print("Страна: ", person1.get_country())
print("Дата рождения: ", person1.get_date_of_birth())
print("Возраст: ", person1.calculate_age())

print("Персона 2:")
print("Имя: ", person2.get_name())
print("Страна: ", person2.get_country())
print("Дата рождения: ", person2.get_date_of_birth())
print("Возраст: ", person2.calculate_age())
```

Вывод:

Листинг 2: Ожидаемый вывод

```
Персона 1:
Имя:  Иванов Иван Иванович
Страна:  Россия
Дата рождения:  1946-08-15
Возраст:  77
```


Персона 2:
Имя: Петров Сергей Александрович
Страна: Белоруссия
Дата рождения: 1982-10-22
Возраст: 41

2. Создайте класс `Student` с закрытыми атрибутами `__full_name`, `__enrollment_date`, `__major`. Реализуйте методы-геттеры и метод `study_duration()` для вычисления количества лет с момента зачисления.

Инструкции:

- (a) Создайте класс `Student` с методом `__init__`.
- (b) Методы-геттеры: `get_full_name()`, `get_enrollment_date()`, `get_major()`.
- (c) Метод `study_duration()` вычисляет количество лет с зачисления.
- (d) Создайте несколько экземпляров класса.
- (e) Выведите данные каждого студента.

Пример использования:

Листинг 3: Пример кода

```
from datetime import date

student1 = Student("Сидоров Алексей", date(2018, 9, 1), "Математика")
student2 = Student("Иванова Мария", date(2021, 9, 1), "Физика")

print("Студент 1:")
print("Имя: ", student1.get_full_name())
print("Направление: ", student1.get_major())
print("Дата зачисления: ", student1.get_enrollment_date())
print("Стаж учёбы: ", student1.study_duration())

print("Студент 2:")
print("Имя: ", student2.get_full_name())
print("Направление: ", student2.get_major())
print("Дата зачисления: ", student2.get_enrollment_date())
print("Стаж учёбы: ", student2.study_duration())
```

Вывод:

Листинг 4: Ожидаемый вывод

```
Студент 1:
Имя: Сидоров Алексей
Направление: Математика
Дата зачисления: 2018-09-01
Стаж учёбы: 5
Студент 2:
Имя: Иванова Мария
Направление: Физика
Дата зачисления: 2021-09-01
Стаж учёбы: 2
```

3. Создайте класс `Employee` с закрытыми атрибутами `__name`, `__position`, `__hire_date`. Реализуйте методы-геттеры и метод `work_experience()` для вычисления количества лет работы.

Инструкции:

- (a) Создайте класс `Employee` с методом `__init__`.
- (b) Методы-геттеры: `get_name()`, `get_position()`, `get_hire_date()`.
- (c) Метод `work_experience()` вычисляет стаж в годах.
- (d) Создайте несколько экземпляров класса.
- (e) Выведите данные каждого сотрудника.

Пример использования:

Листинг 5: Пример кода

```
from datetime import date

emp1 = Employee("Кузнецов Дмитрий", "Инженер", date(2010, 5, 10))
emp2 = Employee("Смирнова Ольга", "Менеджер", date(2015, 8, 1))

print("Сотрудник 1:")
print("Имя: ", emp1.get_name())
print("Должность: ", emp1.get_position())
print("Дата приёма: ", emp1.get_hire_date())
print("Стаж: ", emp1.work_experience())

print("Сотрудник 2:")
print("Имя: ", emp2.get_name())
print("Должность: ", emp2.get_position())
print("Дата приёма: ", emp2.get_hire_date())
print("Стаж: ", emp2.work_experience())
```

Вывод:

Листинг 6: Ожидаемый вывод

```
Сотрудник 1:
Имя: Кузнецов Дмитрий
Должность: Инженер
Дата приёма: 2010-05-10
Стаж: 17
Сотрудник 2:
Имя: Смирнова Ольга
Должность: Менеджер
Дата приёма: 2015-08-01
Стаж: 8
```

4. Создайте класс `Book` с закрытыми атрибутами `__title`, `__author`, `__publish_date`. Реализуйте геттеры и метод `book_age()` для вычисления возраста книги.

Инструкции:

- (a) Создайте класс `Book`.
- (b) Методы-геттеры: `get_title()`, `get_author()`, `get_publish_date()`.
- (c) Метод `book_age()` вычисляет возраст книги.
- (d) Создайте экземпляры класса.
- (e) Выведите данные каждой книги.

Пример использования:

Листинг 7: Пример кода

```
from datetime import date

book1 = Book("Программирование на Python", "Иванов И.И.", date(2015, 3, 10))
book2 = Book("Алгебра", "Петров П.П.", date(2000, 9, 1))

print("Книга 1:")
print("Название: ", book1.get_title())
print("Автор: ", book1.get_author())
print("Дата публикации: ", book1.get_publish_date())
print("Возраст книги: ", book1.book_age())

print("Книга 2:")
print("Название: ", book2.get_title())
print("Автор: ", book2.get_author())
print("Дата публикации: ", book2.get_publish_date())
print("Возраст книги: ", book2.book_age())
```

Вывод:

Листинг 8: Ожидаемый вывод

```
Книга 1:
Название: Программирование на Python
Автор: Иванов И.И.
Дата публикации: 2015-03-10
Возраст книги: 8
Книга 2:
Название: Алгебра
Автор: Петров П.П.
Дата публикации: 2000-09-01
Возраст книги: 23
```

5. Создайте класс `Car` с закрытыми атрибутами `__model`, `__manufacturer`, `__production_date`. Геттеры и метод `car_age()` для вычисления возраста автомобиля.

Инструкции:

- (a) Создайте класс `Car`.
- (b) Методы-геттеры: `get_model()`, `get_manufacturer()`, `get_production_date()`.
- (c) Метод `car_age()` вычисляет возраст автомобиля.
- (d) Создайте экземпляры класса.
- (e) Выведите данные каждого автомобиля.

Пример использования:

Листинг 9: Пример кода

```
from datetime import date

car1 = Car("Camry", "Toyota", date(2012, 6, 15))
car2 = Car("Focus", "Ford", date(2018, 4, 20))

print("Автомобиль 1:")
print("Модель: ", car1.get_model())
print("Производитель: ", car1.get_manufacturer())
print("Дата выпуска: ", car1.get_production_date())
print("Возраст авто: ", car1.car_age())

print("Автомобиль 2:")
print("Модель: ", car2.get_model())
print("Производитель: ", car2.get_manufacturer())
print("Дата выпуска: ", car2.get_production_date())
print("Возраст авто: ", car2.car_age())
```

Вывод:

Листинг 10: Ожидаемый вывод

```
Автомобиль 1:
Модель: Camry
Производитель: Toyota
Дата выпуска: 2012-06-15
Возраст авто: 11
Автомобиль 2:
Модель: Focus
Производитель: Ford
Дата выпуска: 2018-04-20
Возраст авто: 5
```

6. Создайте класс `Pet` с закрытыми атрибутами `__name`, `__species`, `__birth_date`. Реализуйте методы-геттеры и метод `pet_age()` для вычисления возраста питомца. Создайте несколько экземпляров и выведите их данные.

Инструкции:

- (a) Создайте класс `Pet` с методом `__init__`.
- (b) Методы-геттеры: `get_name()`, `get_species()`, `get_birth_date()`.
- (c) Метод `pet_age()` вычисляет возраст питомца в годах.
- (d) Создайте несколько экземпляров класса.
- (e) Выведите данные каждого питомца через методы класса.

Пример использования:

Листинг 11: Пример кода

```
from datetime import date

pet1 = Pet("Барсик", "Кошка", date(2018, 5, 12))
pet2 = Pet("Рекс", "Собака", date(2015, 8, 1))

print("Питомец 1:")
print("Имя: ", pet1.get_name())
print("Вид: ", pet1.get_species())
print("Дата рождения: ", pet1.get_birth_date())
print("Возраст: ", pet1.pet_age())

print("Питомец 2:")
print("Имя: ", pet2.get_name())
print("Вид: ", pet2.get_species())
print("Дата рождения: ", pet2.get_birth_date())
print("Возраст: ", pet2.pet_age())
```

Вывод:

Листинг 12: Ожидаемый вывод

```
Питомец 1:
Имя: Барсик
Вид: Кошка
Дата рождения: 2018-05-12
Возраст: 7
Питомец 2:
Имя: Рекс
Вид: Собака
Дата рождения: 2015-08-01
Возраст: 10
```

7. Создайте класс `Membership` с закрытыми атрибутами `__member_name`, `__membership_type`, `__join_date`. Реализуйте методы-геттеры и метод `membership_duration()` для вычисления длительности членства в годах.

Инструкции:

- (a) Создайте класс `Membership`.
- (b) Методы-геттеры: `get_member_name()`, `get_membership_type()`, `get_join_date()`.
- (c) Метод `membership_duration()` вычисляет длительность членства в годах.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого участника.

Пример использования:

Листинг 13: Пример кода

```
from datetime import date

member1 = Membership("Иванов Иван", "Золотой", date(2018, 3, 15))
member2 = Membership("Петров Петр", "Серебряный", date(2020, 6, 1))
```

```

print("Член 1:")
print("Имя: ", member1.get_member_name())
print("Тип членства: ", member1.get_membership_type())
print("Дата вступления: ", member1.get_join_date())
print("Длительность членства: ", member1.membership_duration())

print("Член 2:")
print("Имя: ", member2.get_member_name())
print("Тип членства: ", member2.get_membership_type())
print("Дата вступления: ", member2.get_join_date())
print("Длительность членства: ", member2.membership_duration())

```

Вывод:

Листинг 14: Ожидаемый вывод

```

Член 1:
Имя:  Иванов Иван
Тип членства:  Золотой
Дата вступления:  2018-03-15
Длительность членства:  5
Член 2:
Имя:  Петров Петр
Тип членства:  Серебряный
Дата вступления:  2020-06-01
Длительность членства:  3

```

8. Создайте класс `Event` с закрытыми атрибутами `__event_name`, `__location`, `__event_date`. Реализуйте методы-геттеры и метод `days_until_event()` для вычисления количества дней до события.

Инструкции:

- (a) Создайте класс `Event`.
- (b) Методы-геттеры: `get_event_name()`, `get_location()`, `get_event_date()`.
- (c) Метод `days_until_event()` вычисляет дни до события.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого события.

Пример использования:

Листинг 15: Пример кода

```

from datetime import date

event1 = Event("Концерт", "Стадион", date(2025, 12, 1))
event2 = Event("Выставка", "Музей", date(2025, 11, 20))

print("Событие 1:")
print("Название: ", event1.get_event_name())
print("Место: ", event1.get_location())
print("Дата: ", event1.get_event_date())

```

```

print("Дней до события: ", event1.days_until_event())

print("Событие 2:")
print("Название: ", event2.get_event_name())
print("Место: ", event2.get_location())
print("Дата: ", event2.get_event_date())
print("Дней до события: ", event2.days_until_event())

```

Вывод:

Листинг 16: Ожидаемый вывод

```

Событие 1:
Название: Концерт
Место: Стадион
Дата: 2025-12-01
Дней до события: 112
Событие 2:
Название: Выставка
Место: Музей
Дата: 2025-11-20
Дней до события: 101

```

9. Создайте класс `Course` с закрытыми атрибутами `__course_name`, `__start_date`, `__duration_weeks`. Реализуйте методы-геттеры и метод `weeks_elapsed()` для вычисления прошедших недель с начала курса.

Инструкции:

- (a) Создайте класс `Course`.
- (b) Методы-геттеры: `get_course_name()`, `get_start_date()`, `get_duration_weeks()`.
- (c) Метод `weeks_elapsed()` вычисляет количество прошедших недель.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого курса.

Пример использования:

Листинг 17: Пример кода

```

from datetime import date

course1 = Course("Python", date(2025, 1, 1), 12)
course2 = Course("Алгебра", date(2025, 2, 1), 10)

print("Курс 1:")
print("Название: ", course1.get_course_name())
print("Дата начала: ", course1.get_start_date())
print("Продолжительность (недель): ", course1.get_duration_weeks())
print("Прошло недель: ", course1.weeks_elapsed())

print("Курс 2:")
print("Название: ", course2.get_course_name())
print("Дата начала: ", course2.get_start_date())
print("Продолжительность (недель): ", course2.get_duration_weeks())
print("Прошло недель: ", course2.weeks_elapsed())

```

Вывод:

Листинг 18: Ожидаемый вывод

```
Курс 1:
Название: Python
Дата начала: 2025-01-01
Продолжительность (недель): 12
Прошло недель: 36
Курс 2:
Название: Алгебра
Дата начала: 2025-02-01
Продолжительность (недель): 10
Прошло недель: 31
```

10. Создайте класс `Subscription` с закрытыми атрибутами `__user`, `__plan`, `__start_date`. Реализуйте методы-геттеры и метод `subscription_age()` для вычисления возраста подписки в годах.

Инструкции:

- (a) Создайте класс `Subscription`.
- (b) Методы-геттеры: `get_user()`, `get_plan()`, `get_start_date()`.
- (c) Метод `subscription_age()` вычисляет возраст подписки.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой подписки.

Пример использования:

Листинг 19: Пример кода

```
from datetime import date

sub1 = Subscription("Иванов И.", "Premium", date(2021, 3, 1))
sub2 = Subscription("Петров П.", "Basic", date(2022, 7, 15))

print("Подписка 1:")
print("Пользователь: ", sub1.get_user())
print("План: ", sub1.get_plan())
print("Дата начала: ", sub1.get_start_date())
print("Возраст подписки: ", sub1.subscription_age())

print("Подписка 2:")
print("Пользователь: ", sub2.get_user())
print("План: ", sub2.get_plan())
print("Дата начала: ", sub2.get_start_date())
print("Возраст подписки: ", sub2.subscription_age())
```

Вывод:

Листинг 20: Ожидаемый вывод

```
Подписка 1:
Пользователь:  Иванов И.
План:  Premium
Дата начала:  2021-03-01
Возраст подписки:  4
Подписка 2:
Пользователь:  Петров П.
План:  Basic
Дата начала:  2022-07-15
Возраст подписки:  3
```

11. Создайте класс `Flight` с закрытыми атрибутами `__flight_number`, `__departure_date`, `__destination`. Реализуйте методы-геттеры и метод `days_until_departure()` для вычисления количества дней до вылета.

Инструкции:

- (a) Создайте класс `Flight`.
- (b) Методы-геттеры: `get_flight_number()`, `get_departure_date()`, `get_destination()`.
- (c) Метод `days_until_departure()` вычисляет количество дней до вылета.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого рейса.

Пример использования:

Листинг 21: Пример кода

```
from datetime import date

flight1 = Flight("SU123", date(2025, 10, 15), "Москва")
flight2 = Flight("AF456", date(2025, 11, 1), "Париж")

print("Рейс 1:")
print("Номер: ", flight1.get_flight_number())
print("Дата вылета: ", flight1.get_departure_date())
print("Пункт назначения: ", flight1.get_destination())
print("Дней до вылета: ", flight1.days_until_departure())

print("Рейс 2:")
print("Номер: ", flight2.get_flight_number())
print("Дата вылета: ", flight2.get_departure_date())
print("Пункт назначения: ", flight2.get_destination())
print("Дней до вылета: ", flight2.days_until_departure())
```

Вывод:

Листинг 22: Ожидаемый вывод

```
Рейс 1:
Номер:  SU123
Дата вылета:  2025-10-15
Пункт назначения:  Москва
```

Дней до вылета: 54
Рейс 2:
Номер: AF456
Дата вылета: 2025-11-01
Пункт назначения: Париж
Дней до вылета: 71

12. Создайте класс `Project` с закрытыми атрибутами `__project_name`, `__start_date`, `__deadline`. Реализуйте методы-геттеры и метод `days_remaining()` для вычисления количества дней до завершения проекта.

Инструкции:

- (a) Создайте класс `Project`.
- (b) Методы-геттеры: `get_project_name()`, `get_start_date()`, `get_deadline()`.
- (c) Метод `days_remaining()` вычисляет дни до дедлайна.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого проекта.

Пример использования:

Листинг 23: Пример кода

```
from datetime import date

project1 = Project("Разработка сайта", date(2025, 9, 1), date(2025, 12, 1))
project2 = Project("Анализ данных", date(2025, 10, 1), date(2025, 11, 30))

print("Проект 1:")
print("Название: ", project1.get_project_name())
print("Дата начала: ", project1.get_start_date())
print("Дедлайн: ", project1.get_deadline())
print("Дней до завершения: ", project1.days_remaining())

print("Проект 2:")
print("Название: ", project2.get_project_name())
print("Дата начала: ", project2.get_start_date())
print("Дедлайн: ", project2.get_deadline())
print("Дней до завершения: ", project2.days_remaining())
```

Вывод:

Листинг 24: Ожидаемый вывод

```
Проект 1:
Название:  Разработка сайта
Дата начала:  2025-09-01
Дедлайн:  2025-12-01
Дней до завершения:  101
Проект 2:
Название:  Анализ данных
Дата начала:  2025-10-01
Дедлайн:  2025-11-30
Дней до завершения:  91
```

13. Создайте класс `Doctor` с закрытыми атрибутами `__full_name`, `__specialty`, `__birth_date`. Реализуйте методы-геттеры и метод `calculate_age()` для вычисления возраста врача.

Инструкции:

- (a) Создайте класс `Doctor`.
- (b) Методы-геттеры: `get_full_name()`, `get_specialty()`, `get_birth_date()`.
- (c) Метод `calculate_age()` вычисляет возраст.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого врача.

Пример использования:

Листинг 25: Пример кода

```
from datetime import date

doc1 = Doctor("Иванов И.И.", "Терапевт", date(1980, 5, 12))
doc2 = Doctor("Петров П.П.", "Хирург", date(1975, 8, 1))

print("Врач 1:")
print("Имя: ", doc1.get_full_name())
print("Специальность: ", doc1.get_specialty())
print("Дата рождения: ", doc1.get_birth_date())
print("Возраст: ", doc1.calculate_age())

print("Врач 2:")
print("Имя: ", doc2.get_full_name())
print("Специальность: ", doc2.get_specialty())
print("Дата рождения: ", doc2.get_birth_date())
print("Возраст: ", doc2.calculate_age())
```

Вывод:

Листинг 26: Ожидаемый вывод

```
Врач 1:
Имя:  Иванов И.И.
Специальность:  Терапевт
Дата рождения:  1980-05-12
Возраст:  45
Врач 2:
Имя:  Петров П.П.
Специальность:  Хирург
Дата рождения:  1975-08-01
Возраст:  50
```

14. Создайте класс `Patient` с закрытыми атрибутами `__full_name`, `__admission_date`, `__diagnosis`. Реализуйте методы-геттеры и метод `hospital_stay()` для вычисления количества дней пребывания в больнице.

Инструкции:

- (a) Создайте класс `Patient`.
- (b) Методы-геттеры: `get_full_name()`, `get_admission_date()`, `get_diagnosis()`.
- (c) Метод `hospital_stay()` вычисляет дни пребывания.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого пациента.

Пример использования:

Листинг 27: Пример кода

```
from datetime import date

patient1 = Patient("Сидоров С.С.", date(2025, 9, 1), "ОРВИ")
patient2 = Patient("Кузнецов К.К.", date(2025, 8, 28), "Грипп")

print("Пациент 1:")
print("Имя: ", patient1.get_full_name())
print("Дата госпитализации: ", patient1.get_admission_date())
print("Диагноз: ", patient1.get_diagnosis())
print("Дней в больнице: ", patient1.hospital_stay())

print("Пациент 2:")
print("Имя: ", patient2.get_full_name())
print("Дата госпитализации: ", patient2.get_admission_date())
print("Диагноз: ", patient2.get_diagnosis())
print("Дней в больнице: ", patient2.hospital_stay())
```

Вывод:

Листинг 28: Ожидаемый вывод

```
Пациент 1:
Имя: Сидоров С.С.
Дата госпитализации: 2025-09-01
Диагноз: ОРВИ
Дней в больнице: 15
Пациент 2:
Имя: Кузнецов К.К.
Дата госпитализации: 2025-08-28
Диагноз: Грипп
Дней в больнице: 19
```

15. Создайте класс `Concert` с закрытыми атрибутами `__artist`, `__venue`, `__concert_date`. Реализуйте методы-геттеры и метод `days_until_concert()`.

Инструкции:

- (a) Создайте класс `Concert`.
- (b) Методы-геттеры: `get_artist()`, `get_venue()`, `get_concert_date()`.
- (c) Метод `days_until_concert()` вычисляет дни до концерта.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого концерта.

Пример использования:

Листинг 29: Пример кода

```
from datetime import date

concert1 = Concert("Imagine Dragons", "Лужники", date(2025, 10, 10))
concert2 = Concert("Coldplay", "O2 Arena", date(2025, 11, 5))

print("Концерт 1:")
print("Исполнитель: ", concert1.get_artist())
print("Место: ", concert1.get_venue())
print("Дата: ", concert1.get_concert_date())
print("Дней до концерта: ", concert1.days_until_concert())

print("Концерт 2:")
print("Исполнитель: ", concert2.get_artist())
print("Место: ", concert2.get_venue())
print("Дата: ", concert2.get_concert_date())
print("Дней до концерта: ", concert2.days_until_concert())
```

Вывод:

Листинг 30: Ожидаемый вывод

```
Концерт 1:
Исполнитель: Imagine Dragons
Место: Лужники
Дата: 2025-10-10
Дней до концерта: 49
Концерт 2:
Исполнитель: Coldplay
Место: O2 Arena
Дата: 2025-11-05
Дней до концерта: 75
```

16. Создайте класс `Holiday` с закрытыми атрибутами `__name`, `__country`, `__holiday_date`. Реализуйте методы-геттеры и метод `days_until_holiday()`.

Инструкции:

- (a) Создайте класс `Holiday`.
- (b) Методы-геттеры: `get_name()`, `get_country()`, `get_holiday_date()`.
- (c) Метод `days_until_holiday()` вычисляет дни до праздника.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого праздника.

Пример использования:

Листинг 31: Пример кода

```
from datetime import date
```

```

holiday1 = Holiday("Новый Год", "Россия", date(2026, 1, 1))
holiday2 = Holiday("Рождество", "Германия", date(2025, 12, 25))

print("Праздник 1:")
print("Название: ", holiday1.get_name())
print("Страна: ", holiday1.get_country())
print("Дата: ", holiday1.get_holiday_date())
print("Дней до праздника: ", holiday1.days_until_holiday())

print("Праздник 2:")
print("Название: ", holiday2.get_name())
print("Страна: ", holiday2.get_country())
print("Дата: ", holiday2.get_holiday_date())
print("Дней до праздника: ", holiday2.days_until_holiday())

```

Вывод:

Листинг 32: Ожидаемый вывод

```

Праздник 1:
Название:  Новый Год
Страна:   Россия
Дата:     2026-01-01
Дней до праздника:  83
Праздник 2:
Название:  Рождество
Страна:   Германия
Дата:     2025-12-25
Дней до праздника:  67

```

17. Создайте класс `Employee` с закрытыми атрибутами `__full_name`, `__position`, `__hire_date`. Реализуйте методы-геттеры и метод `years_worked()` для вычисления стажа работы в годах.

Инструкции:

- (a) Создайте класс `Employee`.
- (b) Методы-геттеры: `get_full_name()`, `get_position()`, `get_hire_date()`.
- (c) Метод `years_worked()` вычисляет стаж в годах.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого сотрудника.

Пример использования:

Листинг 33: Пример кода

```

from datetime import date

emp1 = Employee("Иванов И.И.", "Менеджер", date(2015, 4, 1))
emp2 = Employee("Петров П.П.", "Разработчик", date(2018, 7, 15))

print("Сотрудник 1:")
print("Имя: ", emp1.get_full_name())

```

```

print("Должность: ", emp1.get_position())
print("Дата приема: ", emp1.get_hire_date())
print("Стаж: ", emp1.years_worked())

print("Сотрудник 2:")
print("Имя: ", emp2.get_full_name())
print("Должность: ", emp2.get_position())
print("Дата приема: ", emp2.get_hire_date())
print("Стаж: ", emp2.years_worked())

```

Вывод:

Листинг 34: Ожидаемый вывод

```

Сотрудник 1:
Имя:  Иванов И.И.
Должность:  Менеджер
Дата приема:  2015-04-01
Стаж:  10
Сотрудник 2:
Имя:  Петров П.П.
Должность:  Разработчик
Дата приема:  2018-07-15
Стаж:  7

```

18. Создайте класс `LibraryBook` с закрытыми атрибутами `__title`, `__author`, `__publication_date`. Реализуйте методы-геттеры и метод `book_age()` для вычисления возраста книги.

Инструкции:

- (a) Создайте класс `LibraryBook`.
- (b) Методы-геттеры: `get_title()`, `get_author()`, `get_publication_date()`.
- (c) Метод `book_age()` вычисляет возраст книги в годах.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой книги.

Пример использования:

Листинг 35: Пример кода

```

from datetime import date

book1 = LibraryBook("Война и мир", "Толстой", date(1869, 1, 1))
book2 = LibraryBook("Мастер и Маргарита", "Булгаков", date(1967, 5, 1))

print("Книга 1:")
print("Название: ", book1.get_title())
print("Автор: ", book1.get_author())
print("Дата публикации: ", book1.get_publication_date())
print("Возраст книги: ", book1.book_age())

print("Книга 2:")
print("Название: ", book2.get_title())

```

```
print("Автор: ", book2.get_author())
print("Дата публикации: ", book2.get_publication_date())
print("Возраст книги: ", book2.book_age())
```

Вывод:

Листинг 36: Ожидаемый вывод

```
Книга 1:
Название:  Война и мир
Автор:    Толстой
Дата публикации: 1869-01-01
Возраст книги: 156
Книга 2:
Название:  Мастер и Маргарита
Автор:    Булгаков
Дата публикации: 1967-05-01
Возраст книги: 59
```

19. Создайте класс `Vehicle` с закрытыми атрибутами `__brand`, `__model`, `__manufacture_date`. Реализуйте методы-геттеры и метод `vehicle_age()`.

Инструкции:

- (a) Создайте класс `Vehicle`.
- (b) Методы-геттеры: `get_brand()`, `get_model()`, `get_manufacture_date()`.
- (c) Метод `vehicle_age()` вычисляет возраст транспортного средства.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого транспортного средства.

Пример использования:

Листинг 37: Пример кода

```
from datetime import date

vehicle1 = Vehicle("Toyota", "Camry", date(2015, 5, 1))
vehicle2 = Vehicle("BMW", "X5", date(2018, 3, 10))

print("Транспорт 1:")
print("Марка: ", vehicle1.get_brand())
print("Модель: ", vehicle1.get_model())
print("Дата производства: ", vehicle1.get_manufacture_date())
print("Возраст: ", vehicle1.vehicle_age())

print("Транспорт 2:")
print("Марка: ", vehicle2.get_brand())
print("Модель: ", vehicle2.get_model())
print("Дата производства: ", vehicle2.get_manufacture_date())
print("Возраст: ", vehicle2.vehicle_age())
```


Вывод:

Листинг 38: Ожидаемый вывод

```
Транспорт 1:
Марка: Toyota
Модель: Camry
Дата производства: 2015-05-01
Возраст: 10
Транспорт 2:
Марка: BMW
Модель: X5
Дата производства: 2018-03-10
Возраст: 7
```

20. Создайте класс `Student` с закрытыми атрибутами `__full_name`, `__enrollment_date`, `__major`. Реализуйте методы-геттеры и метод `study_years()`.

Инструкции:

- (a) Создайте класс `Student`.
- (b) Методы-геттеры: `get_full_name()`, `get_enrollment_date()`, `get_major()`.
- (c) Метод `study_years()` вычисляет количество лет учебы.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого студента.

Пример использования:

Листинг 39: Пример кода

```
from datetime import date

student1 = Student("Иванов И.И.", date(2020, 9, 1), "Математика")
student2 = Student("Петров П.П.", date(2021, 9, 1), "Физика")

print("Студент 1:")
print("Имя: ", student1.get_full_name())
print("Дата зачисления: ", student1.get_enrollment_date())
print("Специальность: ", student1.get_major())
print("Лет учебы: ", student1.study_years())

print("Студент 2:")
print("Имя: ", student2.get_full_name())
print("Дата зачисления: ", student2.get_enrollment_date())
print("Специальность: ", student2.get_major())
print("Лет учебы: ", student2.study_years())
```

Вывод:

Листинг 40: Ожидаемый вывод

```
Студент 1:
Имя: Иванов И.И.
```

Дата зачисления: 2020-09-01
Специальность: Математика
Лет учебы: 5
Студент 2:
Имя: Петров П.П.
Дата зачисления: 2021-09-01
Специальность: Физика
Лет учебы: 4

21. Создайте класс `Ticket` с закрытыми атрибутами `__ticket_number`, `__issue_date`, `__valid_until`. Реализуйте методы-геттеры и метод `days_valid()`.

Инструкции:

- (a) Создайте класс `Ticket`.
- (b) Методы-геттеры: `get_ticket_number()`, `get_issue_date()`, `get_valid_until()`.
- (c) Метод `days_valid()` вычисляет дни до окончания действия билета.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого билета.

Пример использования:

Листинг 41: Пример кода

```
from datetime import date

ticket1 = Ticket("A123", date(2025, 9, 1), date(2025, 12, 1))
ticket2 = Ticket("B456", date(2025, 10, 1), date(2026, 1, 1))

print("Билет 1:")
print("Номер: ", ticket1.get_ticket_number())
print("Дата выдачи: ", ticket1.get_issue_date())
print("Действителен до: ", ticket1.get_valid_until())
print("Дней до окончания: ", ticket1.days_valid())

print("Билет 2:")
print("Номер: ", ticket2.get_ticket_number())
print("Дата выдачи: ", ticket2.get_issue_date())
print("Действителен до: ", ticket2.get_valid_until())
print("Дней до окончания: ", ticket2.days_valid())
```

Вывод:

Листинг 42: Ожидаемый вывод

```
Билет 1:
Номер: A123
Дата выдачи: 2025-09-01
Действителен до: 2025-12-01
Дней до окончания: 91
Билет 2:
Номер: B456
Дата выдачи: 2025-10-01
Действителен до: 2026-01-01
Дней до окончания: 92
```

22. Создайте класс `Appointment` с закрытыми атрибутами `__client`, `__service`, `__appointment_date`. Реализуйте методы-геттеры и метод `days_until_appointment()`.

Инструкции:

- (a) Создайте класс `Appointment`.
- (b) Методы-геттеры: `get_client()`, `get_service()`, `get_appointment_date()`.
- (c) Метод `days_until_appointment()` вычисляет дни до приёма.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого приёма.

Пример использования:

Листинг 43: Пример кода

```
from datetime import date

app1 = Appointment("Иванов И.", "Массаж", date(2025, 10, 5))
app2 = Appointment("Петров П.", "Стрижка", date(2025, 10, 15))

print("Приём 1:")
print("Клиент: ", app1.get_client())
print("Услуга: ", app1.get_service())
print("Дата: ", app1.get_appointment_date())
print("Дней до приёма: ", app1.days_until_appointment())

print("Приём 2:")
print("Клиент: ", app2.get_client())
print("Услуга: ", app2.get_service())
print("Дата: ", app2.get_appointment_date())
print("Дней до приёма: ", app2.days_until_appointment())
```

Вывод:

Листинг 44: Ожидаемый вывод

```
Приём 1:
Клиент:  Иванов И.
Услуга:  Массаж
Дата:    2025-10-05
Дней до приёма:  44
Приём 2:
Клиент:  Петров П.
Услуга:  Стрижка
Дата:    2025-10-15
Дней до приёма:  54
```

23. Создайте класс `Subscription` с закрытыми атрибутами `__subscriber`, `__start_date`, `__end_date`. Реализуйте методы-геттеры и метод `days_remaining()`.

Инструкции:

- (a) Создайте класс `Subscription`.
- (b) Методы-геттеры: `get_subscriber()`, `get_start_date()`, `get_end_date()`.
- (c) Метод `days_remaining()` вычисляет дни до окончания подписки.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой подписки.

Пример использования:

Листинг 45: Пример кода

```
from datetime import date

sub1 = Subscription("Иванов И.", date(2025, 1, 1), date(2025, 12, 31))
sub2 = Subscription("Петров П.", date(2025, 6, 1), date(2026, 5, 31))

print("Подписка 1:")
print("Абонент: ", sub1.get_subscriber())
print("Дата начала: ", sub1.get_start_date())
print("Дата окончания: ", sub1.get_end_date())
print("Дней до окончания: ", sub1.days_remaining())

print("Подписка 2:")
print("Абонент: ", sub2.get_subscriber())
print("Дата начала: ", sub2.get_start_date())
print("Дата окончания: ", sub2.get_end_date())
print("Дней до окончания: ", sub2.days_remaining())
```

Вывод:

Листинг 46: Ожидаемый вывод

```
Подписка 1:
Абонент:  Иванов И.
Дата начала:  2025-01-01
Дата окончания:  2025-12-31
Дней до окончания:  113
Подписка 2:
Абонент:  Петров П.
Дата начала:  2025-06-01
Дата окончания:  2026-05-31
Дней до окончания:  245
```

24. Создайте класс `MembershipCard` с закрытыми атрибутами `__owner`, `__issue_date`, `__expiry_date`. Реализуйте методы-геттеры и метод `days_until_expiry()`.

Инструкции:

- (a) Создайте класс `MembershipCard`.
- (b) Методы-геттеры: `get_owner()`, `get_issue_date()`, `get_expiry_date()`.
- (c) Метод `days_until_expiry()` вычисляет дни до истечения действия карты.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой карты.

Пример использования:

Листинг 47: Пример кода

```
from datetime import date

card1 = MembershipCard("Иванов И.", date(2025, 1, 1), date(2026, 1, 1))
card2 = MembershipCard("Петров П.", date(2025, 5, 1), date(2026, 5, 1))

print("Карта 1:")
print("Владелец: ", card1.get_owner())
print("Дата выдачи: ", card1.get_issue_date())
print("Срок действия: ", card1.get_expiry_date())
print("Дней до окончания: ", card1.days_until_expiry())

print("Карта 2:")
print("Владелец: ", card2.get_owner())
print("Дата выдачи: ", card2.get_issue_date())
print("Срок действия: ", card2.get_expiry_date())
print("Дней до окончания: ", card2.days_until_expiry())
```

Вывод:

Листинг 48: Ожидаемый вывод

```
Карта 1:
Владелец:  Иванов И.
Дата выдачи:  2025-01-01
Срок действия:  2026-01-01
Дней до окончания:  113
Карта 2:
Владелец:  Петров П.
Дата выдачи:  2025-05-01
Срок действия:  2026-05-01
Дней до окончания:  204
```

25. Создайте класс `Event` с закрытыми атрибутами `__title`, `__location`, `__event_date`. Реализуйте методы-геттеры и метод `days_until_event()`.

Инструкции:

- (a) Создайте класс `Event`.
- (b) Методы-геттеры: `get_title()`, `get_location()`, `get_event_date()`.
- (c) Метод `days_until_event()` вычисляет дни до события.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого события.

Пример использования:

Листинг 49: Пример кода

```
from datetime import date
```

```

event1 = Event("Фестиваль науки", "Москва", date(2025, 10, 20))
event2 = Event("Конференция IT", "Санкт-Петербург", date(2025, 11, 10))

print("Событие 1:")
print("Название: ", event1.get_title())
print("Место: ", event1.get_location())
print("Дата: ", event1.get_event_date())
print("Дней до события: ", event1.days_until_event())

print("Событие 2:")
print("Название: ", event2.get_title())
print("Место: ", event2.get_location())
print("Дата: ", event2.get_event_date())
print("Дней до события: ", event2.days_until_event())

```

Вывод:

Листинг 50: Ожидаемый вывод

```

Событие 1:
Название: Фестиваль науки
Место: Москва
Дата: 2025-10-20
Дней до события: 59
Событие 2:
Название: Конференция IT
Место: Санкт-Петербург
Дата: 2025-11-10
Дней до события: 80

```

26. Создайте класс `CarRental` с закрытыми атрибутами `__client`, `__rental_date`, `__return_date`. Реализуйте методы-геттеры и метод `rental_duration()`.

Инструкции:

- (a) Создайте класс `CarRental`.
- (b) Методы-геттеры: `get_client()`, `get_rental_date()`, `get_return_date()`.
- (c) Метод `rental_duration()` вычисляет длительность аренды в днях.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой аренды.

Пример использования:

Листинг 51: Пример кода

```

from datetime import date

rental1 = CarRental("Иванов И.", date(2025, 10, 1), date(2025, 10, 10))
rental2 = CarRental("Петров П.", date(2025, 11, 1), date(2025, 11, 5))

print("Аренда 1:")
print("Клиент: ", rental1.get_client())
print("Дата аренды: ", rental1.get_rental_date())

```

```

print("Дата возврата: ", rental1.get_return_date())
print("Длительность аренды: ", rental1.rental_duration())

print("Аренда 2:")
print("Клиент: ", rental2.get_client())
print("Дата аренды: ", rental2.get_rental_date())
print("Дата возврата: ", rental2.get_return_date())
print("Длительность аренды: ", rental2.rental_duration())

```

Вывод:

Листинг 52: Ожидаемый вывод

```

Аренда 1:
Клиент:  Иванов И.
Дата аренды:  2025-10-01
Дата возврата:  2025-10-10
Длительность аренды:  9
Аренда 2:
Клиент:  Петров П.
Дата аренды:  2025-11-01
Дата возврата:  2025-11-05
Длительность аренды:  4

```

27. Создайте класс `Visa` с закрытыми атрибутами `__holder`, `__issue_date`, `__expiry_date`. Реализуйте методы-геттеры и метод `days_until_expiry()`.

Инструкции:

- Создайте класс `Visa`.
- Методы-геттеры: `get_holder()`, `get_issue_date()`, `get_expiry_date()`.
- Метод `days_until_expiry()` вычисляет дни до окончания визы.
- Создайте несколько экземпляров.
- Выведите данные каждой визы.

Пример использования:

Листинг 53: Пример кода

```

from datetime import date

visa1 = Visa("Иванов И.", date(2025, 1, 1), date(2026, 1, 1))
visa2 = Visa("Петров П.", date(2025, 6, 1), date(2026, 6, 1))

print("Виза 1:")
print("Держатель: ", visa1.get_holder())
print("Дата выдачи: ", visa1.get_issue_date())
print("Дата окончания: ", visa1.get_expiry_date())
print("Дней до окончания: ", visa1.days_until_expiry())

print("Виза 2:")
print("Держатель: ", visa2.get_holder())
print("Дата выдачи: ", visa2.get_issue_date())
print("Дата окончания: ", visa2.get_expiry_date())
print("Дней до окончания: ", visa2.days_until_expiry())

```

Вывод:

Листинг 54: Ожидаемый вывод

```
Виза 1:
Держатель:  Иванов И.
Дата выдачи:  2025-01-01
Дата окончания:  2026-01-01
Дней до окончания:  113
Виза 2:
Держатель:  Петров П.
Дата выдачи:  2025-06-01
Дата окончания:  2026-06-01
Дней до окончания:  204
```

28. Создайте класс `Reservation` с закрытыми атрибутами `__guest`, `__checkin_date`, `__checkout_date`. Реализуйте методы-геттеры и метод `stay_duration()`.

Инструкции:

- (a) Создайте класс `Reservation`.
- (b) Методы-геттеры: `get_guest()`, `get_checkin_date()`, `get_checkout_date()`.
- (c) Метод `stay_duration()` вычисляет продолжительность пребывания в днях.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой брони.

Пример использования:

Листинг 55: Пример кода

```
from datetime import date

res1 = Reservation("Иванов И.", date(2025, 10, 1), date(2025, 10, 7))
res2 = Reservation("Петров П.", date(2025, 11, 5), date(2025, 11, 12))

print("Бронь 1:")
print("Гость: ", res1.get_guest())
print("Дата заезда: ", res1.get_checkin_date())
print("Дата выезда: ", res1.get_checkout_date())
print("Продолжительность пребывания: ", res1.stay_duration())

print("Бронь 2:")
print("Гость: ", res2.get_guest())
print("Дата заезда: ", res2.get_checkin_date())
print("Дата выезда: ", res2.get_checkout_date())
print("Продолжительность пребывания: ", res2.stay_duration())
```

Вывод:

Листинг 56: Ожидаемый вывод

```
Бронь 1:
Гость:  Иванов И.
```


Дата заезда: 2025-10-01
Дата выезда: 2025-10-07
Продолжительность пребывания: 6
Бронь 2:
Гость: Петров П.
Дата заезда: 2025-11-05
Дата выезда: 2025-11-12
Продолжительность пребывания: 7

29. Создайте класс `Conference` с закрытыми атрибутами `__name`, `__city`, `__start_date`. Реализуйте методы-геттеры и метод `days_until_start()`.

Инструкции:

- (a) Создайте класс `Conference`.
- (b) Методы-геттеры: `get_name()`, `get_city()`, `get_start_date()`.
- (c) Метод `days_until_start()` вычисляет дни до начала конференции.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждой конференции.

Пример использования:

Листинг 57: Пример кода

```
from datetime import date

conf1 = Conference("PythonConf", "Москва", date(2025, 10, 20))
conf2 = Conference("DataScience Summit", "Санкт-Петербург", date(2025, 11,
15))

print("Конференция 1:")
print("Название: ", conf1.get_name())
print("Город: ", conf1.get_city())
print("Дата начала: ", conf1.get_start_date())
print("Дней до начала: ", conf1.days_until_start())

print("Конференция 2:")
print("Название: ", conf2.get_name())
print("Город: ", conf2.get_city())
print("Дата начала: ", conf2.get_start_date())
print("Дней до начала: ", conf2.days_until_start())
```

Вывод:

Листинг 58: Ожидаемый вывод

```
Конференция 1:
Название: PythonConf
Город: Москва
Дата начала: 2025-10-20
Дней до начала: 59
Конференция 2:
Название: DataScience Summit
```

Город: Санкт-Петербург
Дата начала: 2025-11-15
Дней до начала: 85

30. Создайте класс `Medication` с закрытыми атрибутами `__name`, `__manufacturer`, `__expiry_date`. Реализуйте методы-геттеры и метод `days_until_expiry()`.

Инструкции:

- (a) Создайте класс `Medication`.
- (b) Методы-геттеры: `get_name()`, `get_manufacturer()`, `get_expiry_date()`.
- (c) Метод `days_until_expiry()` вычисляет дни до окончания срока годности.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого лекарства.

Пример использования:

Листинг 59: Пример кода

```
from datetime import date

med1 = Medication("Парацетамол", "Фармком", date(2026, 1, 1))
med2 = Medication("Ибупрофен", "БиоФарм", date(2025, 12, 1))

print("Лекарство 1:")
print("Название: ", med1.get_name())
print("Производитель: ", med1.get_manufacturer())
print("Срок годности: ", med1.get_expiry_date())
print("Дней до окончания: ", med1.days_until_expiry())

print("Лекарство 2:")
print("Название: ", med2.get_name())
print("Производитель: ", med2.get_manufacturer())
print("Срок годности: ", med2.get_expiry_date())
print("Дней до окончания: ", med2.days_until_expiry())
```

Вывод:

Листинг 60: Ожидаемый вывод

```
Лекарство 1:
Название: Парацетамол
Производитель: Фармком
Срок годности: 2026-01-01
Дней до окончания: 113
Лекарство 2:
Название: Ибупрофен
Производитель: БиоФарм
Срок годности: 2025-12-01
Дней до окончания: 92
```

31. Создайте класс `Project` с закрытыми атрибутами `__title`, `__start_date`, `__deadline`. Реализуйте методы-геттеры и метод `days_until_deadline()`.

Инструкции:

- (a) Создайте класс `Project`.
- (b) Методы-геттеры: `get_title()`, `get_start_date()`, `get_deadline()`.
- (c) Метод `days_until_deadline()` вычисляет дни до дедлайна.
- (d) Создайте несколько экземпляров.
- (e) Выведите данные каждого проекта.

Пример использования:

Листинг 61: Пример кода

```
from datetime import date

proj1 = Project("Разработка сайта", date(2025, 9, 1), date(2025, 12, 1))
proj2 = Project("Мобильное приложение", date(2025, 10, 1), date(2026, 1, 15))

print("Проект 1:")
print("Название: ", proj1.get_title())
print("Дата начала: ", proj1.get_start_date())
print("Дедлайн: ", proj1.get_deadline())
print("Дней до дедлайна: ", proj1.days_until_deadline())

print("Проект 2:")
print("Название: ", proj2.get_title())
print("Дата начала: ", proj2.get_start_date())
print("Дедлайн: ", proj2.get_deadline())
print("Дней до дедлайна: ", proj2.days_until_deadline())
```

Вывод:

Листинг 62: Ожидаемый вывод

```
Проект 1:
Название:  Разработка сайта
Дата начала:  2025-09-01
Дедлайн:  2025-12-01
Дней до дедлайна:  91
Проект 2:
Название:  Мобильное приложение
Дата начала:  2025-10-01
Дедлайн:  2026-01-15
Дней до дедлайна:  106
```

Задача 4

1. Написать программу на Python, которая создает абстрактный класс `Shape` для представления геометрической фигуры. Класс должен содержать абстрактные методы `calculate_area` и `calculate_perimeter`, которые вычисляют площадь и периметр фигуры соответственно. Программа также должна создавать дочерние классы `Circle`, `Rectangle` и `Triangle`, которые наследуют от класса `Shape` и реализуют специфические для каждого класса методы вычисления площади и периметра.

Инструкции:

- (a) Создайте абстрактный класс `Shape` (с использованием модуля `abc`) с абстрактными методами `calculate_area()` и `calculate_perimeter()`.
- (b) Создайте класс `Circle` с конструктором `__init__(self, radius)`, который принимает радиус окружности в качестве аргумента и сохраняет его в приватном атрибуте `__radius`. Добавьте `@property`-геттер `radius` для получения значения радиуса. Реализуйте методы `calculate_area()` и `calculate_perimeter()` для вычисления площади и периметра окружности.
- (c) Создайте класс `Rectangle` с конструктором `__init__(self, length, width)`, который принимает длину и ширину прямоугольника в качестве аргументов и сохраняет их в приватных атрибутах `_length` и `_width`. Добавьте `@property`-геттеры `length` и `width` для получения значений атрибутов. Реализуйте методы `calculate_area()` и `calculate_perimeter()` для вычисления площади и периметра прямоугольника.
- (d) Создайте класс `Triangle` с конструктором `__init__(self, base, height, side1, side2, side3)`, который принимает основание, высоту и три стороны треугольника в качестве аргументов и сохраняет их в приватных атрибутах `_base`, `_height`, `_side1`, `_side2` и `_side3`. Добавьте `@property`-геттеры `base`, `height`, `side1`, `side2`, `side3`. Реализуйте методы `calculate_area()` и `calculate_perimeter()` для вычисления площади и периметра треугольника.
- (e) Создайте экземпляр каждого класса и вызовите методы `calculate_area()` и `calculate_perimeter()` для вычисления площади и периметра фигуры. Выведите результаты на экран, используя геттеры для доступа к атрибутам.

Пример использования:

```
# Вычисление параметров окружности.
r = 7
circle = Circle(r)
print("Радиус окружности:", circle.radius)
print("Площадь окружности:", circle.calculate_area())
print("Периметр окружности:", circle.calculate_perimeter())
```

Примечание: В этом примере используется библиотека `math` для вычисления числа π и квадратного корня.

Вывод:

```
Радиус окружности: 7
Площадь окружности: 153.93804002589985
Периметр окружности: 43.982297150257104
```

Далее вывод для прямоугольника и треугольника.

2. Написать программу на Python, которая создает абстрактный класс `ElectricalComponent` (с использованием модуля `abc`) для представления электрических элементов. Класс должен содержать абстрактные методы `calculate_power()` и `calculate_energy()`. Программа также должна создавать дочерние классы `Resistor`, `Capacitor` и `Inductor`, которые наследуют от класса `ElectricalComponent` и реализуют специфические для каждого класса методы вычисления мощности и энергии.

Подсказка по формулам:

- Resistor: $P = U^2/R$, $E = P \cdot t$
- Capacitor: $P = V \cdot I$, $E = 0.5 \cdot C \cdot V^2$
- Inductor: $P = L \cdot I^2$, $E = 0.5 \cdot L \cdot I^2$

Инструкции:

- Создайте абстрактный класс `ElectricalComponent` с методами `calculate_power()` и `calculate_energy()`, используя модуль `abc`.
- Создайте класс `Resistor` с конструктором `__init__(self, voltage, resistance, time)`, который сохраняет приватные атрибуты `__voltage`, `__resistance`, `__time`. Добавьте `@property`-геттеры для всех атрибутов. Реализуйте методы вычисления мощности и энергии.
- Создайте класс `Capacitor` с конструктором `__init__(self, voltage, current, capacitance)`, приватными атрибутами `__voltage`, `__current`, `__capacitance` и геттерами. Реализуйте методы.
- Создайте класс `Inductor` с конструктором `__init__(self, inductance, current)`, приватными атрибутами `__inductance`, `__current` и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы `calculate_power()` и `calculate_energy()`, используя геттеры для доступа к атрибутам. Выведите результаты на экран.

Пример использования:

```
r = Resistor(10, 5, 10)
print("Сопротивление резистора:", r.resistance)
print("Мощность резистора:", r.calculate_power())
print("Энергия резистора:", r.calculate_energy())
```

Вывод:

```
Сопротивление резистора: 5
Мощность резистора: 20
Энергия резистора: 200
```

Далее вывод для конденсатора и катушки индуктивности.

- Написать программу на Python, которая создает абстрактный класс `MotionObject` (с использованием модуля `abc`) для представления движущихся тел. Класс должен содержать абстрактные методы `calculate_kinetic_energy()` и `calculate_momentum()`. Программа также должна создавать дочерние классы `LinearBody`, `RotatingBody` и `FallingBody`, которые наследуют от класса `MotionObject` и реализуют специфические для каждого класса методы вычисления кинетической энергии и импульса.

Подсказка по формулам:

- `LinearBody`: $KE = 0.5 \cdot m \cdot v^2$, $p = m \cdot v$
- `RotatingBody`: $KE = 0.5 \cdot I \cdot \omega^2$, $p = I \cdot \omega$
- `FallingBody`: $KE = m \cdot g \cdot h$, $p = m \cdot v$

Инструкции:

- Создайте абстрактный класс `MotionObject` с методами `calculate_kinetic_energy()` и `calculate_momentum()`, используя модуль `abc`.
- Создайте класс `LinearBody` с конструктором `__init__(self, mass, velocity)`, приватными атрибутами `__mass`, `__velocity` и геттерами. Реализуйте методы.
- Создайте класс `RotatingBody` с конструктором `__init__(self, moment_of_inertia, angular_velocity)`, приватными атрибутами `__moment_of_inertia`, `__angular_velocity` и геттерами. Реализуйте методы.
- Создайте класс `FallingBody` с конструктором `__init__(self, mass, height, velocity)`, приватными атрибутами `__mass`, `__height`, `__velocity` и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы `calculate_kinetic_energy()` и `calculate_momentum()`, используя геттеры для доступа к атрибутам. Выведите результаты на экран.

Пример использования:

```
body = LinearBody(2, 3)
print("Масса тела:", body.mass)
print("Кинетическая энергия:", body.calculate_kinetic_energy())
print("Импульс:", body.calculate_momentum())
```

Вывод:

Масса тела: 2
Кинетическая энергия: 6
Импульс: 6

Далее вывод для вращающегося тела и падающего тела.

- Написать программу на Python, которая создает абстрактный класс `Investment` (с использованием модуля `abc`) для финансовых вложений. Класс должен содержать абстрактные методы `calculate_simple_interest()` и `calculate_total_value()`. Программа также должна создавать дочерние классы `ShortTerm`, `LongTerm` и `CompoundInvestment`, которые наследуют от класса `Investment` и реализуют специфические методы вычисления процентов и итоговой суммы.

Подсказка по формулам:

- `ShortTerm`: $SI = P \cdot R \cdot T / 100$, $Total = P + SI$
- `LongTerm`: $SI = P \cdot R \cdot T / 100 + 50$, $Total = P + SI$
- `CompoundInvestment`: $Total = P \cdot (1 + R/100)^T$, $SI = Total - P$

Инструкции:

- Создайте абстрактный класс `Investment` с методами `calculate_simple_interest()` и `calculate_total_value()`, используя модуль `abc`.
- Создайте класс `ShortTerm` с конструктором `__init__(self, principal, rate, time)`, приватными атрибутами `__principal`, `__rate`, `__time` и геттерами. Реализуйте методы.

- (c) Создайте класс `LongTerm` с конструктором `__init__(self, principal, rate, time)`, приватными атрибутами `__principal`, `__rate`, `__time` и геттерами. Реализуйте методы.
- (d) Создайте класс `CompoundInvestment` с конструктором `__init__(self, principal, rate, time)`, приватными атрибутами `__principal`, `__rate`, `__time` и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы `calculate_simple_interest()` и `calculate_total_value()`, используя геттеры для доступа к атрибутам. Выведите результаты на экран.

Пример использования:

```
inv = ShortTerm(1000, 5, 2)
print("Начальная сумма:", inv.principal)
print("Простой процент:", inv.calculate_simple_interest())
print("Итоговая сумма:", inv.calculate_total_value())
```

Вывод:

```
Начальная сумма: 1000
Простой процент: 100
Итоговая сумма: 1100
```

Далее вывод для долгосрочного и сложного вложения.

5. Написать программу на Python, которая создает абстрактный класс `Solid` (с использованием модуля `abc`) для твердого тела. Класс должен содержать абстрактные методы `calculate_volume()` и `calculate_surface_area()`. Программа также должна создавать дочерние классы `Cube`, `RectangularPrism` и `Cylinder`, которые наследуют от класса `Solid` и реализуют специфические методы вычисления объема и площади поверхности.

Подсказка по формулам:

- `Cube`: $V = a^3$, $S = 6 \cdot a^2$
- `RectangularPrism`: $V = l \cdot w \cdot h$, $S = 2(lw + lh + wh)$
- `Cylinder`: $V = \pi r^2 h$, $S = 2\pi r(r + h)$

Инструкции:

- (a) Создайте абстрактный класс `Solid` с методами `calculate_volume()` и `calculate_surface_area()`, используя модуль `abc`.
- (b) Создайте класс `Cube` с конструктором `__init__(self, side)`, приватным атрибутом `__side` и геттером. Реализуйте методы.
- (c) Создайте класс `RectangularPrism` с конструктором `__init__(self, length, width, height)`, приватными атрибутами `__length`, `__width`, `__height` и геттерами. Реализуйте методы.
- (d) Создайте класс `Cylinder` с конструктором `__init__(self, radius, height)`, приватными атрибутами `__radius`, `__height` и геттерами. Реализуйте методы.

- (e) Создайте экземпляр каждого класса и вызовите методы `calculate_volume()` и `calculate_surface_area()`, используя геттеры. Выведите результаты на экран.

Пример использования:

```
cube = Cube(3)
print("Сторона куба:", cube.side)
print("Объем куба:", cube.calculate_volume())
print("Площадь поверхности куба:", cube.calculate_surface_area())
```

Вывод:

```
Сторона куба: 3
Объем куба: 27
Площадь поверхности куба: 54
```

Далее вывод для прямоугольного параллелепипеда и цилиндра.

6. Написать программу на Python, которая создает абстрактный класс `ChemicalSubstance` (с использованием модуля `abc`) для химических веществ. Класс должен содержать абстрактные методы `calculate_molar_mass()` и `calculate_density()`. Программа также должна создавать дочерние классы `Element`, `Compound` и `Mixture`, которые наследуют от класса `ChemicalSubstance` и реализуют специфические методы вычисления молярной массы и плотности.

Подсказка по формулам:

- `Element`: $M = atomic_mass$, $\rho = mass/volume$
- `Compound`: $M = \sum(fraction \cdot atomic_mass)$, $\rho = mass/volume$
- `Mixture`: $M = \sum(fraction \cdot molar_mass)$, $\rho = \sum(fraction \cdot density)$

Инструкции:

- (a) Создайте абстрактный класс `ChemicalSubstance` с методами `calculate_molar_mass()` и `calculate_density()`, используя модуль `abc`.
- (b) Создайте класс `Element` с конструктором `__init__(self, atomic_mass, mass, volume)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Compound` с конструктором `__init__(self, fractions, atomic_masses, mass, volume)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Mixture` с конструктором `__init__(self, fractions, molar_masses, densities)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
el = Element(12, 24, 2)
print("Атомная масса элемента:", el.atomic_mass)
print("Молярная масса:", el.calculate_molar_mass())
print("Плотность:", el.calculate_density())
```


Вывод:

Атомная масса элемента: 12
Молярная масса: 12
Плотность: 12

Далее вывод для соединения и смеси.

7. Написать программу на Python, которая создает абстрактный класс `BankAccount` (с использованием модуля `abc`) для банковских счетов. Класс должен содержать абстрактные методы `calculate_interest()` и `calculate_balance()`. Программа также должна создавать дочерние классы `Savings`, `Checking` и `FixedDeposit`, которые наследуют от класса `BankAccount` и реализуют специфические методы вычисления процентов и баланса.

Подсказка по формулам:

- **Savings:** $Interest = balance \cdot rate \cdot time / 100$, $Balance = balance + Interest$
- **Checking:** $Interest = balance \cdot rate \cdot time / 100 - fee$, $Balance = balance + Interest$
- **FixedDeposit:** $Balance = principal \cdot (1 + rate/100)^{time}$, $Interest = Balance - principal$

Инструкции:

- Создайте абстрактный класс `BankAccount` с методами `calculate_interest()` и `calculate_balance()`, используя модуль `abc`.
- Создайте класс `Savings` с конструктором `__init__(self, balance, rate, time)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Checking` с конструктором `__init__(self, balance, rate, time, fee)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `FixedDeposit` с конструктором `__init__(self, principal, rate, time)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
s = Savings(1000, 5, 2)
print("Баланс на сберегательном счете:", s.balance)
print("Проценты:", s.calculate_interest())
print("Итоговый баланс:", s.calculate_balance())
```

Вывод:

Баланс на сберегательном счете: 1000
Проценты: 100
Итоговый баланс: 1100

Далее вывод для расчетного счета и срочного депозита.

8. Написать программу на Python, которая создает абстрактный класс `Shape3D` (с использованием модуля `abc`) для трехмерных фигур. Класс должен содержать абстрактные методы `calculate_volume()` и `calculate_surface_area()`. Программа также должна создавать дочерние классы `Sphere`, `Cone` и `Pyramid`, которые наследуют от класса `Shape3D` и реализуют специфические методы вычисления объема и площади поверхности.

Подсказка по формулам:

- **Sphere:** $V = 4/3 \cdot \pi r^3$, $S = 4 \cdot \pi r^2$
- **Cone:** $V = 1/3 \cdot \pi r^2 h$, $S = \pi r(r + \sqrt{r^2 + h^2})$
- **Pyramid:** $V = 1/3 \cdot base_area \cdot height$, $S = base_area + lateral_area$

Инструкции:

- Создайте абстрактный класс `Shape3D` с методами `calculate_volume()` и `calculate_surface_area()`, используя модуль `abc`.
- Создайте класс `Sphere` с конструктором `__init__(self, radius)`, приватным атрибутом и геттером. Реализуйте методы.
- Создайте класс `Cone` с конструктором `__init__(self, radius, height)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Pyramid` с конструктором `__init__(self, base_area, lateral_area, height)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
s = Sphere(3)
print("Радиус сферы:", s.radius)
print("Объем сферы:", s.calculate_volume())
print("Площадь поверхности сферы:", s.calculate_surface_area())
```

Вывод:

```
Радиус сферы: 3
Объем сферы: 113.097
Площадь поверхности сферы: 113.097
```

Далее вывод для конуса и пирамиды.

9. Написать программу на Python, которая создает абстрактный класс `Vehicle` (с использованием модуля `abc`) для транспортных средств. Класс должен содержать абстрактные методы `calculate_fuel_consumption()` и `calculate_range()`. Программа также должна создавать дочерние классы `Car`, `Truck` и `Motorcycle`, которые наследуют от класса `Vehicle` и реализуют специфические методы вычисления расхода топлива и запаса хода.

Подсказка по формулам:

- **Car:** $fuel = distance/efficiency$, $range = tank_capacity \cdot efficiency$

- Truck: $fuel = (distance/efficiency) \cdot load_factor$, $range = tank_capacity \cdot efficiency/load_factor$
- Motorcycle: $fuel = distance/efficiency \cdot 0.8$, $range = tank_capacity \cdot efficiency \cdot 1.2$

Инструкции:

- Создайте абстрактный класс `Vehicle` с методами `calculate_fuel_consumption()` и `calculate_range()`, используя модуль `abc`.
- Создайте класс `Car` с конструктором `__init__(self, efficiency, distance, tank_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Truck` с конструктором `__init__(self, efficiency, distance, tank_capacity, load_factor)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Motorcycle` с конструктором `__init__(self, efficiency, distance, tank_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
car = Car(15, 150, 50)
print("Эффективность автомобиля:", car.efficiency)
print("Расход топлива:", car.calculate_fuel_consumption())
print("Запас хода:", car.calculate_range())
```

Вывод:

```
Эффективность автомобиля: 15
Расход топлива: 10
Запас хода: 750
```

Далее вывод для грузовика и мотоцикла.

- Написать программу на Python, которая создает абстрактный класс `Plant` (с использованием модуля `abc`) для растений. Класс должен содержать абстрактные методы `calculate_growth()` и `calculate_water_needs()`. Программа также должна создавать дочерние классы `Tree`, `Flower` и `Shrub`, которые наследуют от класса `Plant` и реализуют специфические методы вычисления роста и потребности в воде.

Подсказка по формулам:

- Tree: $growth = height_rate \cdot time$, $water = area \cdot water_rate$
- Flower: $growth = height_rate \cdot time \cdot 0.5$, $water = area \cdot water_rate \cdot 0.3$
- Shrub: $growth = height_rate \cdot time \cdot 0.8$, $water = area \cdot water_rate \cdot 0.6$

Инструкции:

- Создайте абстрактный класс `Plant` с методами `calculate_growth()` и `calculate_water_needs()`, используя модуль `abc`.

- (b) Создайте класс `Tree` с конструктором `__init__(self, height_rate, time, area, water_rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Flower` с конструктором `__init__(self, height_rate, time, area, water_rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Shrub` с конструктором `__init__(self, height_rate, time, area, water_rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
tree = Tree(2, 5, 10, 3)
print("Скорость роста дерева:", tree.height_rate)
print("Рост:", tree.calculate_growth())
print("Потребность в воде:", tree.calculate_water_needs())
```

Вывод:

```
Скорость роста дерева: 2
Рост: 10
Потребность в воде: 30
```

Далее вывод для цветка и кустарника.

11. Написать программу на Python, которая создает абстрактный класс `Sensor` (с использованием модуля `abc`) для измерительных датчиков. Класс должен содержать абстрактные методы `calculate_signal()` и `calculate_accuracy()`. Программа также должна создавать дочерние классы `TemperatureSensor`, `PressureSensor` и `LightSensor`, которые наследуют от класса `Sensor` и реализуют специфические методы вычисления сигнала и точности.

Подсказка по формулам:

- `TemperatureSensor`: $signal = voltage \cdot sensitivity$, $accuracy = tolerance$
- `PressureSensor`: $signal = pressure \cdot sensitivity$, $accuracy = tolerance \cdot 1.1$
- `LightSensor`: $signal = intensity \cdot sensitivity$, $accuracy = tolerance \cdot 0.9$

Инструкции:

- (a) Создайте абстрактный класс `Sensor` с методами `calculate_signal()` и `calculate_accuracy()`, используя модуль `abc`.
- (b) Создайте класс `TemperatureSensor` с конструктором `__init__(self, voltage, sensitivity, tolerance)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `PressureSensor` с конструктором `__init__(self, pressure, sensitivity, tolerance)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `LightSensor` с конструктором `__init__(self, intensity, sensitivity, tolerance)`, приватными атрибутами и геттерами. Реализуйте методы.

- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
temp_sensor = TemperatureSensor(5, 2, 0.1)
print("Напряжение:", temp_sensor.voltage)
print("Сигнал:", temp_sensor.calculate_signal())
print("Точность:", temp_sensor.calculate_accuracy())
```

Вывод:

Напряжение: 5
Сигнал: 10
Точность: 0.1

Далее вывод для датчиков давления и света.

12. Написать программу на Python, которая создает абстрактный класс `CookingIngredient` (с использованием модуля `abc`) для ингредиентов. Класс должен содержать абстрактные методы `calculate_calories()` и `calculate_mass()`. Программа также должна создавать дочерние классы `Vegetable`, `Meat` и `Grain`, которые наследуют от класса `CookingIngredient` и реализуют специфические методы вычисления калорий и массы.

Подсказка по формулам:

- **Vegetable:** $calories = weight \cdot cal_per_100g / 100$, $mass = weight$
- **Meat:** $calories = weight \cdot cal_per_100g / 100 \cdot 1.2$, $mass = weight$
- **Grain:** $calories = weight \cdot cal_per_100g / 100 \cdot 1.1$, $mass = weight$

Инструкции:

- (a) Создайте абстрактный класс `CookingIngredient` с методами `calculate_calories()` и `calculate_mass()`, используя модуль `abc`.
- (b) Создайте класс `Vegetable` с конструктором `__init__(self, weight, cal_per_100g)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Meat` с конструктором `__init__(self, weight, cal_per_100g)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Grain` с конструктором `__init__(self, weight, cal_per_100g)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
veg = Vegetable(200, 30)
print("Вес овоща:", veg.weight)
print("Калории:", veg.calculate_calories())
print("Масса:", veg.calculate_mass())
```

Вывод:

Вес овоща: 200
Калории: 60
Масса: 200

Далее вывод для мяса и зерна.

13. Написать программу на Python, которая создает абстрактный класс `ElectronicDevice` (с использованием модуля `abc`) для электронных устройств. Класс должен содержать абстрактные методы `calculate_power()` и `calculate_efficiency()`. Программа также должна создавать дочерние классы `Laptop`, `Smartphone` и `Tablet`, которые наследуют от класса `ElectronicDevice` и реализуют специфические методы вычисления мощности и эффективности.

Подсказка по формулам:

- Laptop: $power = voltage \cdot current$, $efficiency = useful_power / power$
- Smartphone: $power = voltage \cdot current \cdot 0.8$, $efficiency = useful_power / power$
- Tablet: $power = voltage \cdot current \cdot 0.9$, $efficiency = useful_power / power$

Инструкции:

- Создайте абстрактный класс `ElectronicDevice` с методами `calculate_power()` и `calculate_efficiency()`, используя модуль `abc`.
- Создайте класс `Laptop` с конструктором `__init__(self, voltage, current, useful_power)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Smartphone` с конструктором `__init__(self, voltage, current, useful_power)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Tablet` с конструктором `__init__(self, voltage, current, useful_power)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
laptop = Laptop(19, 3, 50)
print("Напряжение ноутбука:", laptop.voltage)
print("Мощность:", laptop.calculate_power())
print("Эффективность:", laptop.calculate_efficiency())
```

Вывод:

Напряжение ноутбука: 19
Мощность: 57
Эффективность: 0.877

Далее вывод для смартфона и планшета.

14. Написать программу на Python, которая создает абстрактный класс `MusicalInstrument` (с использованием модуля `abc`) для музыкальных инструментов. Класс должен содержать абстрактные методы `calculate_sound_level()` и `calculate_frequency()`. Программа также должна создавать дочерние классы `Piano`, `Guitar` и `Flute`, которые наследуют от класса `MusicalInstrument` и реализуют специфические методы вычисления уровня звука и частоты.

Подсказка по формулам:

- Piano: $sound_level = keys \cdot intensity$, $frequency = 440 \cdot 2^{(note-49)/12}$
- Guitar: $sound_level = strings \cdot intensity \cdot 0.8$, $frequency = 440 \cdot 2^{(note-49)/12}$
- Flute: $sound_level = holes \cdot intensity \cdot 0.9$, $frequency = 440 \cdot 2^{(note-49)/12}$

Инструкции:

- Создайте абстрактный класс `MusicalInstrument` с методами `calculate_sound_level()` и `calculate_frequency()`, используя модуль `abc`.
- Создайте класс `Piano` с конструктором `__init__(self, keys, intensity, note)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Guitar` с конструктором `__init__(self, strings, intensity, note)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Flute` с конструктором `__init__(self, holes, intensity, note)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
piano = Piano(88, 5, 49)
print("Клавиши:", piano.keys)
print("Уровень звука:", piano.calculate_sound_level())
print("Частота:", piano.calculate_frequency())
```

Вывод:

```
Клавиши: 88
Уровень звука: 440
Частота: 440
```

Далее вывод для гитары и флейты.

15. Написать программу на Python, которая создает абстрактный класс `Workout` (с использованием модуля `abc`) для физических упражнений. Класс должен содержать абстрактные методы `calculate_calories_burned()` и `calculate_duration()`. Программа также должна создавать дочерние классы `Cardio`, `Strength` и `Flexibility`, которые наследуют от класса `Workout` и реализуют специфические методы вычисления сожженных калорий и длительности тренировки.

Подсказка по формулам:

- Cardio: $calories = weight \cdot time \cdot 0.1$, $duration = time$

- **Strength:** $calories = weight \cdot time \cdot 0.08$, $duration = time$
- **Flexibility:** $calories = weight \cdot time \cdot 0.05$, $duration = time$

Инструкции:

- Создайте абстрактный класс `Workout` с методами `calculate_calories_burned()` и `calculate_duration()`, используя модуль `abc`.
- Создайте класс `Cardio` с конструктором `__init__(self, weight, time)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Strength` с конструктором `__init__(self, weight, time)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Flexibility` с конструктором `__init__(self, weight, time)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
cardio = Cardio(70, 30)
print("Вес:", cardio.weight)
print("Сожженные калории:", cardio.calculate_calories_burned())
print("Длительность:", cardio.calculate_duration())
```

Вывод:

```
Вес: 70
Сожженные калории: 210
Длительность: 30
```

Далее вывод для силовой и растяжки.

- Написать программу на Python, которая создает абстрактный класс `ComputerComponent` (с использованием модуля `abc`) для компонентов компьютера. Класс должен содержать абстрактные методы `calculate_power_consumption()` и `calculate_cost()`. Программа также должна создавать дочерние классы `CPU`, `GPU` и `RAM`, которые наследуют от класса `ComputerComponent` и реализуют специфические методы вычисления энергопотребления и стоимости.

Подсказка по формулам:

- CPU: $power = cores \cdot frequency \cdot 10$, $cost = cores \cdot 50$
- GPU: $power = cores \cdot frequency \cdot 12$, $cost = cores \cdot 80$
- RAM: $power = size \cdot 3$, $cost = size \cdot 20$

Инструкции:

- Создайте абстрактный класс `ComputerComponent` с методами `calculate_power_consumption()` и `calculate_cost()`, используя модуль `abc`.
- Создайте класс `CPU` с конструктором `__init__(self, cores, frequency)`, приватными атрибутами и геттерами. Реализуйте методы.

- (c) Создайте класс `GPU` с конструктором `__init__(self, cores, frequency)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `RAM` с конструктором `__init__(self, size)`, приватным атрибутом и геттером. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
cpu = CPU(4, 3.5)
print("Ядра CPU:", cpu.cores)
print("Энергопотребление:", cpu.calculate_power_consumption())
print("Стоимость:", cpu.calculate_cost())
```

Вывод:

```
Ядра CPU: 4
Энергопотребление: 140
Стоимость: 200
```

Далее вывод для `GPU` и `RAM`.

17. Написать программу на Python, которая создает абстрактный класс `Building` (с использованием модуля `abc`) для зданий. Класс должен содержать абстрактные методы `calculate_volume()` и `calculate_floor_area()`. Программа также должна создавать дочерние классы `House`, `Office` и `Warehouse`, которые наследуют от класса `Building` и реализуют специфические методы вычисления объема и площади.

Подсказка по формулам:

- `House`: $volume = length \cdot width \cdot height$, $floor_area = length \cdot width$
- `Office`: $volume = length \cdot width \cdot height \cdot 1.2$, $floor_area = length \cdot width \cdot 1.1$
- `Warehouse`: $volume = length \cdot width \cdot height \cdot 1.5$, $floor_area = length \cdot width \cdot 1.3$

Инструкции:

- (a) Создайте абстрактный класс `Building` с методами `calculate_volume()` и `calculate_floor_area()`, используя модуль `abc`.
- (b) Создайте класс `House` с конструктором `__init__(self, length, width, height)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Office` с конструктором `__init__(self, length, width, height)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Warehouse` с конструктором `__init__(self, length, width, height)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```

house = House(10, 8, 3)
print("Длина дома:", house.length)
print("Объем:", house.calculate_volume())
print("Площадь пола:", house.calculate_floor_area())

```

Вывод:

```

Длина дома: 10
Объем: 240
Площадь пола: 80

```

Далее вывод для офиса и склада.

18. Написать программу на Python, которая создает абстрактный класс `Vehicle` (с использованием модуля `abc`) для транспортных средств. Класс должен содержать абстрактные методы `calculate_max_speed()` и `calculate_range()`. Программа также должна создавать дочерние классы `Car`, `Motorcycle` и `Bicycle`, которые наследуют от класса `Vehicle` и реализуют специфические методы вычисления максимальной скорости и дальности.

Подсказка по формулам:

- `Car`: $max_speed = engine_power \cdot 2$, $range = fuel_capacity \cdot 10$
- `Motorcycle`: $max_speed = engine_power \cdot 2.5$, $range = fuel_capacity \cdot 8$
- `Bicycle`: $max_speed = pedaling_power \cdot 3$, $range = stamina \cdot 5$

Инструкции:

- (a) Создайте абстрактный класс `Vehicle` с методами `calculate_max_speed()` и `calculate_range()`, используя модуль `abc`.
- (b) Создайте класс `Car` с конструктором `__init__(self, engine_power, fuel_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Motorcycle` с конструктором `__init__(self, engine_power, fuel_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Bicycle` с конструктором `__init__(self, pedaling_power, stamina)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```

car = Car(150, 50)
print("Мощность двигателя автомобиля:", car.engine_power)
print("Максимальная скорость:", car.calculate_max_speed())
print("Дальность:", car.calculate_range())

```

Вывод:

```

Мощность двигателя автомобиля: 150
Максимальная скорость: 300
Дальность: 500

```

Далее вывод для мотоцикла и велосипеда.

19. Написать программу на Python, которая создает абстрактный класс `BankAccount` (с использованием модуля `abc`) для банковских счетов. Класс должен содержать абстрактные методы `calculate_interest()` и `calculate_fees()`. Программа также должна создавать дочерние классы `SavingsAccount`, `CheckingAccount` и `InvestmentAccount`, которые наследуют от класса `BankAccount` и реализуют специфические методы вычисления процентов и комиссий.

Подсказка по формулам:

- `SavingsAccount`: $interest = balance \cdot 0.03$, $fees = 5$
- `CheckingAccount`: $interest = balance \cdot 0.01$, $fees = 2$
- `InvestmentAccount`: $interest = balance \cdot 0.05$, $fees = 10$

Инструкции:

- Создайте абстрактный класс `BankAccount` с методами `calculate_interest()` и `calculate_fees()`, используя модуль `abc`.
- Создайте класс `SavingsAccount` с конструктором `__init__(self, balance)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `CheckingAccount` с конструктором `__init__(self, balance)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `InvestmentAccount` с конструктором `__init__(self, balance)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
savings = SavingsAccount(1000)
print("Баланс сберегательного счета:", savings.balance)
print("Проценты:", savings.calculate_interest())
print("Комиссии:", savings.calculate_fees())
```

Вывод:

```
Баланс сберегательного счета: 1000
Проценты: 30.0
Комиссии: 5
```

Далее вывод для расчетного и инвестиционного счета.

20. Написать программу на Python, которая создает абстрактный класс `Appliance` (с использованием модуля `abc`) для бытовой техники. Класс должен содержать абстрактные методы `calculate_energy_usage()` и `calculate_operating_cost()`. Программа также должна создавать дочерние классы `Refrigerator`, `WashingMachine` и `Microwave`, которые наследуют от класса `Appliance` и реализуют специфические методы вычисления энергопотребления и стоимости эксплуатации.

Подсказка по формулам:

- Refrigerator: $energy = power \cdot hours$, $cost = energy \cdot 0.12$
- WashingMachine: $energy = power \cdot hours \cdot 1.1$, $cost = energy \cdot 0.12$
- Microwave: $energy = power \cdot hours \cdot 0.8$, $cost = energy \cdot 0.12$

Инструкции:

- Создайте абстрактный класс `Appliance` с методами `calculate_energy_usage()` и `calculate_operating_cost()`, используя модуль `abc`.
- Создайте класс `Refrigerator` с конструктором `__init__(self, power, hours)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `WashingMachine` с конструктором `__init__(self, power, hours)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Microwave` с конструктором `__init__(self, power, hours)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
fridge = Refrigerator(150, 24)
print("Мощность холодильника:", fridge.power)
print("Энергопотребление:", fridge.calculate_energy_usage())
print("Стоимость эксплуатации:", fridge.calculate_operating_cost())
```

Вывод:

```
Мощность холодильника: 150
Энергопотребление: 3600
Стоимость эксплуатации: 432.0
```

Далее вывод для стиральной машины и микроволновки.

21. Написать программу на Python, которая создает абстрактный класс `Planet` (с использованием модуля `abc`) для планет. Класс должен содержать абстрактные методы `calculate_surface_area()` и `calculate_gravity()`. Программа также должна создавать дочерние классы `Earth`, `Mars` и `Jupiter`, которые наследуют от класса `Planet` и реализуют специфические методы вычисления площади поверхности и силы гравитации.

Подсказка по формулам:

- Earth: $surface_area = 4 \cdot \pi \cdot radius^2$, $gravity = G \cdot mass / radius^2$
- Mars: $surface_area = 4 \cdot \pi \cdot radius^2 \cdot 0.95$, $gravity = G \cdot mass / radius^2 \cdot 0.38$
- Jupiter: $surface_area = 4 \cdot \pi \cdot radius^2 \cdot 11.2$, $gravity = G \cdot mass / radius^2 \cdot 2.5$

Инструкции:

- Создайте абстрактный класс `Planet` с методами `calculate_surface_area()` и `calculate_gravity()`, используя модуль `abc`.

- (b) Создайте класс `Earth` с конструктором `__init__(self, radius, mass)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Mars` с конструктором `__init__(self, radius, mass)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Jupiter` с конструктором `__init__(self, radius, mass)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
earth = Earth(6371, 5.97e24)
print("Радиус Земли:", earth.radius)
print("Площадь поверхности:", earth.calculate_surface_area())
print("Сила гравитации:", earth.calculate_gravity())
```

Вывод:

```
Радиус Земли: 6371
Площадь поверхности: 510064471
Сила гравитации: 9.8
```

Далее вывод для Марса и Юпитера.

22. Написать программу на Python, которая создает абстрактный класс `FoodItem` (с использованием модуля `abc`) для пищевых продуктов. Класс должен содержать абстрактные методы `calculate_calories()` и `calculate_price()`. Программа также должна создавать дочерние классы `Fruit`, `Vegetable` и `Meat`, которые наследуют от класса `FoodItem` и реализуют специфические методы вычисления калорийности и стоимости.

Подсказка по формулам:

- **Fruit:** $calories = weight \cdot 0.52$, $price = weight \cdot 3$
- **Vegetable:** $calories = weight \cdot 0.3$, $price = weight \cdot 2$
- **Meat:** $calories = weight \cdot 2.5$, $price = weight \cdot 10$

Инструкции:

- (a) Создайте абстрактный класс `FoodItem` с методами `calculate_calories()` и `calculate_price()`, используя модуль `abc`.
- (b) Создайте класс `Fruit` с конструктором `__init__(self, weight)`, приватным атрибутом и геттером. Реализуйте методы.
- (c) Создайте класс `Vegetable` с конструктором `__init__(self, weight)`, приватным атрибутом и геттером. Реализуйте методы.
- (d) Создайте класс `Meat` с конструктором `__init__(self, weight)`, приватным атрибутом и геттером. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
apple = Fruit(150)
print("Вес фрукта:", apple.weight)
print("Калории:", apple.calculate_calories())
print("Стоимость:", apple.calculate_price())
```

Вывод:

```
Вес фрукта: 150
Калории: 78.0
Стоимость: 450
```

Далее вывод для овощей и мяса.

23. Написать программу на Python, которая создает абстрактный класс `Tool` (с использованием модуля `abc`) для инструментов. Класс должен содержать абстрактные методы `calculate_efficiency()` и `calculate_durability()`. Программа также должна создавать дочерние классы `Hammer`, `Screwdriver` и `Wrench`, которые наследуют от класса `Tool` и реализуют специфические методы вычисления эффективности и прочности.

Подсказка по формулам:

- **Hammer:** $efficiency = weight \cdot swing_speed$, $durability = material_hardness \cdot 10$
- **Screwdriver:** $efficiency = length \cdot torque$, $durability = material_hardness \cdot 8$
- **Wrench:** $efficiency = size \cdot torque$, $durability = material_hardness \cdot 12$

Инструкции:

- Создайте абстрактный класс `Tool` с методами `calculate_efficiency()` и `calculate_durability()`, используя модуль `abc`.
- Создайте класс `Hammer` с конструктором `__init__(self, weight, swing_speed, material_hardness)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Screwdriver` с конструктором `__init__(self, length, torque, material_hardness)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Wrench` с конструктором `__init__(self, size, torque, material_hardness)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
hammer = Hammer(2, 5, 7)
print("Вес молотка:", hammer.weight)
print("Эффективность:", hammer.calculate_efficiency())
print("Прочность:", hammer.calculate_durability())
```

Вывод:

Вес молотка: 2
Эффективность: 10
Прочность: 70

Далее вывод для отвертки и ключа.

24. Написать программу на Python, которая создает абстрактный класс `Book` (с использованием модуля `abc`) для книг. Класс должен содержать абстрактные методы `calculate_reading_time()` и `calculate_cost()`. Программа также должна создавать дочерние классы `Fiction`, `NonFiction` и `Comics`, которые наследуют от класса `Book` и реализуют специфические методы вычисления времени чтения и стоимости.

Подсказка по формулам:

- Fiction: $reading_time = pages \cdot 2$, $cost = pages \cdot 1.5$
- NonFiction: $reading_time = pages \cdot 2.5$, $cost = pages \cdot 2$
- Comics: $reading_time = pages \cdot 1$, $cost = pages \cdot 1$

Инструкции:

- Создайте абстрактный класс `Book` с методами `calculate_reading_time()` и `calculate_cost()`, используя модуль `abc`.
- Создайте класс `Fiction` с конструктором `__init__(self, pages)`, приватным атрибутом и геттером. Реализуйте методы.
- Создайте класс `NonFiction` с конструктором `__init__(self, pages)`, приватным атрибутом и геттером. Реализуйте методы.
- Создайте класс `Comics` с конструктором `__init__(self, pages)`, приватным атрибутом и геттером. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
novel = Fiction(300)
print("Количество страниц:", novel.pages)
print("Время чтения:", novel.calculate_reading_time())
print("Стоимость:", novel.calculate_cost())
```

Вывод:

Количество страниц: 300
Время чтения: 600
Стоимость: 450.0

Далее вывод для научной литературы и комиксов.

25. Написать программу на Python, которая создает абстрактный класс `ElectronicDevice` (с использованием модуля `abc`) для электронных устройств. Класс должен содержать абстрактные методы `calculate_power_consumption()` и `calculate_battery_life()`. Программа также должна создавать дочерние классы `Smartphone`, `Laptop` и `Tablet`, которые наследуют от класса `ElectronicDevice` и реализуют специфические методы вычисления потребляемой мощности и времени работы от батареи.

Подсказка по формулам:

- Smartphone: $power = voltage \cdot current \cdot hours$, $battery_life = battery_capacity / current$
- Laptop: $power = voltage \cdot current \cdot hours \cdot 1.5$, $battery_life = battery_capacity / (current \cdot 1.5)$
- Tablet: $power = voltage \cdot current \cdot hours \cdot 1.2$, $battery_life = battery_capacity / (current \cdot 1.2)$

Инструкции:

- Создайте абстрактный класс `ElectronicDevice` с методами `calculate_power_consumption()` и `calculate_battery_life()`, используя модуль `abc`.
- Создайте класс `Smartphone` с конструктором `__init__(self, voltage, current, hours, battery_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Laptop` с конструктором `__init__(self, voltage, current, hours, battery_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Tablet` с конструктором `__init__(self, voltage, current, hours, battery_capacity)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
phone = Smartphone(5, 1, 10, 5000)
print("Напряжение смартфона:", phone.voltage)
print("Потребляемая мощность:", phone.calculate_power_consumption())
print("Время работы от батареи:", phone.calculate_battery_life())
```

Вывод:

```
Напряжение смартфона: 5
Потребляемая мощность: 50
Время работы от батареи: 5000.0
```

Далее вывод для ноутбука и планшета.

- Написать программу на Python, которая создает абстрактный класс `MusicalInstrument` (с использованием модуля `abc`) для музыкальных инструментов. Класс должен содержать абстрактные методы `calculate_sound_volume()` и `calculate_weight()`. Программа также должна создавать дочерние классы `Piano`, `Guitar` и `Drum`, которые наследуют от класса `MusicalInstrument` и реализуют специфические методы вычисления громкости и веса.

Подсказка по формулам:

- Piano: $volume = keys \cdot 2$, $weight = base_weight \cdot 3$
- Guitar: $volume = strings \cdot 3$, $weight = base_weight \cdot 1.5$
- Drum: $volume = diameter \cdot 4$, $weight = base_weight \cdot 2$

Инструкции:

- (a) Создайте абстрактный класс `MusicalInstrument` с методами `calculate_sound_volume()` и `calculate_weight()`, используя модуль `abc`.
- (b) Создайте класс `Piano` с конструктором `__init__(self, keys, base_weight)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Guitar` с конструктором `__init__(self, strings, base_weight)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Drum` с конструктором `__init__(self, diameter, base_weight)`, приватными атрибутами и геттерами. Реализуйте методы.
- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
piano = Piano(88, 200)
print("Количество клавиш:", piano.keys)
print("Громкость:", piano.calculate_sound_volume())
print("Вес:", piano.calculate_weight())
```

Вывод:

```
Количество клавиш: 88
Громкость: 176
Вес: 600
```

Далее вывод для гитары и барабана.

27. Написать программу на Python, которая создает абстрактный класс `VehiclePart` (с использованием модуля `abc`) для частей транспортного средства. Класс должен содержать абстрактные методы `calculate_durability()` и `calculate_maintenance_cost()`. Программа также должна создавать дочерние классы `Engine`, `Wheel` и `Brake`, которые наследуют от класса `VehiclePart` и реализуют специфические методы вычисления долговечности и стоимости обслуживания.

Подсказка по формулам:

- **Engine:** $durability = hours_run \cdot 1.2$, $maintenance = base_cost \cdot 5$
- **Wheel:** $durability = rotation_count \cdot 0.8$, $maintenance = base_cost \cdot 2$
- **Brake:** $durability = pressure_applied \cdot 0.5$, $maintenance = base_cost \cdot 3$

Инструкции:

- (a) Создайте абстрактный класс `VehiclePart` с методами `calculate_durability()` и `calculate_maintenance_cost()`, используя модуль `abc`.
- (b) Создайте класс `Engine` с конструктором `__init__(self, hours_run, base_cost)`, приватными атрибутами и геттерами. Реализуйте методы.
- (c) Создайте класс `Wheel` с конструктором `__init__(self, rotation_count, base_cost)`, приватными атрибутами и геттерами. Реализуйте методы.
- (d) Создайте класс `Brake` с конструктором `__init__(self, pressure_applied, base_cost)`, приватными атрибутами и геттерами. Реализуйте методы.

- (e) Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
engine = Engine(1000, 200)
print("Наработка двигателя:", engine.hours_run)
print("Долговечность:", engine.calculate_durability())
print("Стоимость обслуживания:", engine.calculate_maintenance_cost())
```

Вывод:

Наработка двигателя: 1000
Долговечность: 1200.0
Стоимость обслуживания: 1000

Далее вывод для колес и тормозов.

28. Написать программу на Python, которая создает абстрактный класс `Appliance` (с использованием модуля `abc`) для бытовых приборов. Класс должен содержать абстрактные методы `calculate_energy_consumption()` и `calculate_cost()`. Программа также должна создавать дочерние классы `Refrigerator`, `WashingMachine` и `Microwave`, которые наследуют от класса `Appliance` и реализуют специфические методы вычисления потребляемой энергии и стоимости эксплуатации.

Подсказка по формулам:

- `Refrigerator`: $energy = power \cdot hours \cdot 30$, $cost = energy \cdot rate$
- `WashingMachine`: $energy = power \cdot hours \cdot 1.5$, $cost = energy \cdot rate$
- `Microwave`: $energy = power \cdot hours \cdot 0.8$, $cost = energy \cdot rate$

Инструкции:

- Создайте абстрактный класс `Appliance` с методами `calculate_energy_consumption()` и `calculate_cost()`, используя модуль `abc`.
- Создайте класс `Refrigerator` с конструктором `__init__(self, power, hours, rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `WashingMachine` с конструктором `__init__(self, power, hours, rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Microwave` с конструктором `__init__(self, power, hours, rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
fridge = Refrigerator(150, 24, 0.1)
print("Мощность холодильника:", fridge.power)
print("Энергопотребление:", fridge.calculate_energy_consumption())
print("Стоимость эксплуатации:", fridge.calculate_cost())
```

Вывод:

Мощность холодильника: 150
Энергопотребление: 108000
Стоимость эксплуатации: 10800.0

Далее вывод для стиральной машины и микроволновки.

29. Написать программу на Python, которая создает абстрактный класс `SportActivity` (с использованием модуля `abc`) для спортивных занятий. Класс должен содержать абстрактные методы `calculate_calories_burned()` и `calculate_duration()`. Программа также должна создавать дочерние классы `Running`, `Swimming` и `Cycling`, которые наследуют от класса `SportActivity` и реализуют специфические методы вычисления сожженных калорий и продолжительности.

Подсказка по формулам:

- `Running`: $calories = weight \cdot distance \cdot 1.036$, $duration = distance / speed$
- `Swimming`: $calories = weight \cdot distance \cdot 1.5$, $duration = distance / speed$
- `Cycling`: $calories = weight \cdot distance \cdot 0.8$, $duration = distance / speed$

Инструкции:

- Создайте абстрактный класс `SportActivity` с методами `calculate_calories_burned()` и `calculate_duration()`, используя модуль `abc`.
- Создайте класс `Running` с конструктором `__init__(self, weight, distance, speed)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Swimming` с конструктором `__init__(self, weight, distance, speed)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Cycling` с конструктором `__init__(self, weight, distance, speed)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
run = Running(70, 5, 10)
print("Вес бегуна:", run.weight)
print("Сожженные калории:", run.calculate_calories_burned())
print("Продолжительность:", run.calculate_duration())
```

Вывод:

Вес бегуна: 70
Сожженные калории: 362.6
Продолжительность: 0.5

Далее вывод для плавания и езды на велосипеде.

30. Написать программу на Python, которая создает абстрактный класс `BuildingMaterial` (с использованием модуля `abc`) для строительных материалов. Класс должен содержать абстрактные методы `calculate_strength()` и `calculate_cost()`. Программа также должна создавать дочерние классы `Concrete`, `Wood`, `Steel`, которые наследуют от класса `BuildingMaterial` и реализуют специфические методы вычисления прочности и стоимости.

Подсказка по формулам:

- **Concrete:** $strength = density \cdot compressive_factor$, $cost = volume \cdot price_per_m3$
- **Wood:** $strength = density \cdot elastic_factor$, $cost = volume \cdot price_per_m3$
- **Steel:** $strength = density \cdot tensile_factor$, $cost = volume \cdot price_per_m3$

Инструкции:

- Создайте абстрактный класс `BuildingMaterial` с методами `calculate_strength()` и `calculate_cost()`, используя модуль `abc`.
- Создайте класс `Concrete` с конструктором `__init__(self, density, compressive_factor, volume, price_per_m3)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Wood` с конструктором `__init__(self, density, elastic_factor, volume, price_per_m3)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Steel` с конструктором `__init__(self, density, tensile_factor, volume, price_per_m3)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
concrete = Concrete(2400, 30, 2, 100)
print("Плотность бетона:", concrete.density)
print("Прочность:", concrete.calculate_strength())
print("Стоимость:", concrete.calculate_cost())
```

Вывод:

```
Плотность бетона: 2400
Прочность: 72000
Стоимость: 200
```

Далее вывод для древесины и стали.

31. Написать программу на Python, которая создает абстрактный класс `TransportVehicle` (с использованием модуля `abc`) для транспортных средств. Класс должен содержать абстрактные методы `calculate_range()` и `calculate_fuel_cost()`. Программа также должна создавать дочерние классы `Car`, `Motorcycle` и `ElectricScooter`, которые наследуют от класса `TransportVehicle` и реализуют специфические методы вычисления дальности хода и стоимости топлива/энергии.

Подсказка по формулам:

- **Car:** $range = tank_capacity / consumption \cdot 100$, $fuel_cost = tank_capacity \cdot fuel_price$
- **Motorcycle:** $range = tank_capacity / consumption \cdot 120$, $fuel_cost = tank_capacity \cdot fuel_price$
- **ElectricScooter:** $range = battery_capacity / consumption \cdot 100$, $fuel_cost = battery_capacity \cdot electricity_rate$

Инструкции:

- Создайте абстрактный класс `TransportVehicle` с методами `calculate_range()` и `calculate_fuel_cost()`, используя модуль `abc`.
- Создайте класс `Car` с конструктором `__init__(self, tank_capacity, consumption, fuel_price)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `Motorcycle` с конструктором `__init__(self, tank_capacity, consumption, fuel_price)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте класс `ElectricScooter` с конструктором `__init__(self, battery_capacity, consumption, electricity_rate)`, приватными атрибутами и геттерами. Реализуйте методы.
- Создайте экземпляр каждого класса и вызовите методы, используя геттеры, и выведите результаты.

Пример использования:

```
car = Car(50, 8, 1.5)
print("Ёмкость бака автомобиля:", car.tank_capacity)
print("Дальность хода:", car.calculate_range())
print("Стоимость топлива:", car.calculate_fuel_cost())
```

Вывод:

```
Ёмкость бака автомобиля: 50
Дальность хода: 625.0
Стоимость топлива: 75.0
```

Далее вывод для `Motorcycle` и `ElectricScooter`.