Часть A: Вопросы с выбором ответа

1. Какой метод вызывается автоматически при создании нового экземпляра класса? a) `__new__` b) `__init__` c) `__call__` d) `__del__`

2. Чем отличается атрибут `_protected` от `__private` в Python? a) `_protected` запрещает доступ извне, а `__private` — нет. b) `__private` вызывает ошибку при обращении, а `_protected` — предупреждение. c) `_protected` — это соглашение, `__private` использует name mangling. d) Ничем, это синонимы.

3. Для чего используется функция `super()` в дочернем классе? a) Для вызова метода родительского класса. b) Для создания супер-объекта. c) Для получения списка всех родительских классов. d) Для явного указания, что метод публичный.

4. Какой магический метод вызывается функцией `print()` по умолчанию для отображения объекта? a) `__repr__` b) `__str__` c) `__info__` d) `__print__`

5. Что позволяет сделать классу реализация метода `__getitem__(self, key)`? a) Сделать объект итерируемым. b) Сделать объект вызываемым, как функцию. c) Поддержать операцию индексирования `obj[key]`. d) Поддержать оператор сложения `obj + value`.

Часть B: Открытые вопросы

6. Объясните, что такое "утиная типизация" (duck typing) в контексте полиморфизма в Python. Приведите простой пример.

7. В чем разница между `@classmethod`, `@staticmethod` и обычным методом экземпляра? Для чего используется первый параметр в каждом из них?

8. Что такое MRO (Method Resolution Order) и какую проблему он решает при множественном наследовании? Как можно посмотреть MRO для класса `C`?

9. Объясните, как работают менеджеры контекста с точки зрения магических методов. Какие два метода должны быть реализованы?

10. Каков основной принцип работы дескриптора? Какие методы протокола дескрипторов вы знаете и когда они вызываются?

Домашнее задание (5 задач)

Задача 1: Библиотека с инкапсуляцией и свойствами Создайте класс `Book`.

- В `__init__` инициализируйте "приватные" атрибуты для названия (`__title`), автора (`__author`) и года издания (`__year`).
- Реализуйте свойства (`@property`) для чтения и записи названия и автора. При установке нового названия убедитесь, что это строка и она не пустая.
- Реализуйте свойство только для чтения (`@property`) для года издания.
- Добавьте `@classmethod from_dict`, который создает и возвращает экземпляр `Book` из словаря вида `{"title": "...", "author": "...", "year": ...}`.
- Реализуйте `__repr__` и `__str__`.

Задача 2: Иерархия геометрических фигур с полиморфизмом

1. Создайте абстрактный базовый класс `Shape` с использованием `abc.ABC`.
2. Определите абстрактный метод `area()` и абстрактный метод `perimeter()`.
3. Создайте классы-наследники: `Rectangle` (прямоугольник, принимает ширину и высоту) и `Circle` (круг, принимает радиус).
4. В каждом классе реализуйте обязательные методы. Для `Circle` используйте `math.pi`.
5. Реализуйте для `Rectangle` метод `__eq__`, который сравнивает два прямоугольника по площади.
6. Создайте список `shapes = [Rectangle(4, 5), Circle(7)]`. В цикле выведите площадь и периметр для каждой фигуры, демонстрируя полиморфизм.

Задача 3: Контейнер с магическими методами Создайте класс `SmartList`, который будет вести себя как умный список.

- Реализуйте методы `__getitem__`, `__setitem__`, `__len__`.
- Реализуйте методы `__iter__` и `__next__`, чтобы по объекту можно было итерироваться.
- Реализуйте метод `__call__`, который принимает функцию и возвращает новый список, где к каждому элементу применена эта функция (аналог `map`). Пример: `smart_list(lambda x: x*2)`.
- Реализуйте метод `__enter__` и `__exit__`, чтобы класс можно было использовать как менеджер контекста. При входе он должен печатать "Начало работы с SmartList", при выходе (в т.ч. при исключении) — "Работа завершена. Длина списка: N".

Задача 4: Дескриптор для валидации Создайте класс-дескриптор `PositiveNumber`.

- В методе `__set__` дескриптор должен проверять, что устанавливаемое значение является числом (`int` или `float`) и больше нуля. Иначе выбрасывать `ValueError`.
- Используйте этот дескриптор в классе `Product` для атрибутов `price` (цена) и `quantity` (количество).
- Убедитесь, что создание `Product("Яблоко", price=-100)` вызывает ошибку, а `Product("Яблоко", price=100, quantity=50)` работает корректно.

Задача 5: Паттерн "Стратегия" для логирования Реализуйте паттерн "Стратегия" для системы логирования.

1. Создайте абстрактный класс `LoggingStrategy` с абстрактным методом `log(message)`.
2. Создайте несколько конкретных стратегий: `ConsoleLogger` (выводит сообщение в консоль), `FileLogger` (записывает в файл, имя файла передается в `__init__`), `ElasticsearchLogger` (эмулирует запись, просто печатает "`[Elasticsearch] {message}`").
3. Создайте класс `Application`, которому в конструктор передается стратегия логирования.
4. Класс `Application` должен иметь метод `do_work()`, который выполняет какую-то "работу" (например, генерирует случайное число) и логирует результат с помощью переданной стратегии.
5. Продемонстрируйте, как одно и то же приложение может использовать разные способы логирования, не меняя свой код.