федеральное государственное автономное образовательное учреждение высшего образования
Первый Московский государственный медицинский университет им. И.М. Сеченова Министерства здравоохранения Российской Федерации (Сеченовский Университет)

Институт цифрового биодизайна и искусственного интеллекта в медицине
Кафедра информационных технологий и обработки медицинских данных

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

Прикладное программирование

основная профессиональная образовательная программа высшего профессионального образования - программа бакалавриата

45.03.04 Интеллектуальные системы в гуманитарной сфере

Перечень вопросов по курсу для проведения централизованного тестирования.
Правильные варианты ответов обозначены буквой «а».

1. Что такое объектно-ориентированное программирование?
a) Парадигма программирования, основанная на объектах, которые содержат данные и методы.
b) Парадигма программирования, основанная на чисто функциональных преобразованиях.
c) Парадигма программирования, основанная на списках и циклах.
d) Парадигма программирования, основанная на процедурах и функциях без состояния.

2. Что такое класс в ООП?
a) Шаблон для создания объектов, определяющий их свойства и поведение.
b) Конкретный объект с данными.
c) Функция, которая создаёт переменные.
d) Модуль программы.

3. Что такое объект (экземпляр класса)?
a) Конкретная реализация класса с собственным состоянием.
b) Исходный код программы.
c) Глобальная переменная.
d) Функция внутри модуля.

4. Чем класс отличается от объекта?
a) Класс — это шаблон, объект — его конкретное воплощение.
b) Класс — это переменная, объект — функция.
c) Класс — это модуль, объект — класс.
d) Класс — это объект, объект — класс.

5. Какие основные принципы ООП вы знаете?
a) Инкапсуляция, наследование, полиморфизм.
b) Декораторы, генераторы, итераторы.
c) Переменные, функции, модули.
d) SQL, HTTP, JSON.

6. Что такое инкапсуляция?
a) Сокрытие внутренней реализации и предоставление интерфейса.
b) Наследование свойств одного класса другим.
c) Возможность одного метода работать с разными типами данных.
d) Разделение программы на независимые модули.

7. Что такое наследование?
a) Механизм создания нового класса на основе существующего.
b) Сокрытие данных внутри класса.
c) Перегрузка операторов.
d) Создание множества копий объекта.

8. Что такое полиморфизм?
a) Возможность объектов с одинаковым интерфейсом иметь разную реализацию.
b) Создание классов с одинаковыми именами.
c) Использование только одного типа данных.
d) Жёсткая привязка метода к конкретному классу.

9. Что такое абстракция в ООП?
a) Выделение существенных характеристик объекта и игнорирование несущественных.
b) Полное копирование всех свойств объекта.
c) Преобразование объекта в строку.
d) Создание графического интерфейса.

10. Зачем использовать ООП в программировании?
a) Для улучшения структурированности, повторного использования кода и упрощения поддержки.
b) Для ускорения выполнения программ.
c) Для уменьшения объёма кода без изменения функциональности.
d) Только для создания графических интерфейсов.

11. Что такое атрибут объекта?
a) Переменная, принадлежащая объекту и хранящая его состояние.
b) Функция внутри объекта.
c) Имя объекта.
d) Модуль, в котором определён объект.

12. Что такое метод объекта?
a) Функция, принадлежащая объекту и определяющая его поведение.
b) Глобальная переменная.
c) Атрибут класса.
d) Декоратор.

13. Чем метод отличается от обычной функции?
a) Метод связан с объектом и имеет доступ к его данным через self.
b) Метод всегда возвращает число.
c) Метод нельзя вызывать из других функций.
d) Метод существует только в глобальной области видимости.

14. Что такое состояние объекта?
a) Совокупность значений его атрибутов в данный момент времени.
b) Имя класса объекта.
c) Список методов объекта.
d) Размер объекта в памяти.

15. Что такое поведение объекта?
a) Действия, которые объект может выполнять, определяемые его методами.
b) Цвет объекта в интерфейсе.
c) Скорость выполнения методов.
d) Количество атрибутов.

16. Что означает выражение «экземпляр класса»?
a) Конкретный объект, созданный по описанию класса.
b) Копия исходного кода класса.
c) Имя класса.
d) Родительский класс.

17. Что означает «создать объект»?

a) Создать экземпляр класса с выделением памяти и инициализацией атрибутов.
b) Написать новый класс.
c) Удалить старый объект.
d) Импортировать модуль.

18. Что такое иерархия классов?
a) Система классов, организованная через отношения наследования.
b) Алфавитный список всех классов.
c) Таблица с атрибутами классов.
d) Файловая структура проекта.

19. Что такое базовый (родительский) класс?
a) Класс, от которого наследуются другие классы.
b) Класс без методов.
c) Класс, который нельзя изменить.
d) Самый первый класс в файле.

20. Что такое дочерний (производный) класс?
a) Класс, который наследует свойства и методы другого класса.
b) Класс без атрибутов.
c) Класс, который нельзя наследовать.
d) Класс, определённый внутри функции.

21. Что означает «переопределить метод»?
a) Заменить реализацию метода в дочернем классе.
b) Удалить метод из класса.
c) Создать два одинаковых метода.
d) Переименовать метод.

22. Что означает «расширить функциональность класса»?
a) Добавить новые методы или атрибуты, не меняя существующие.
b) Удалить часть методов.
c) Переименовать класс.
d) Сделать все методы статическими.

23. Что такое интерфейс класса в широком смысле?
a) Набор публичных методов и свойств, через которые можно взаимодействовать с объектом.
b) Графическое представление класса.
c) Исходный код класса.
d) Список приватных атрибутов.

24. Что такое связность (cohesion) в ООП?
a) Мера того, насколько элементы внутри класса связаны друг с другом.
b) Количество классов в программе.
c) Скорость выполнения методов.
d) Глубина наследования.

25. Что такое зацепление (coupling) в ООП?
a) Мера зависимости одного класса от другого.

b) Количество методов в классе.
c) Размер объекта в памяти.
d) Число импортов в модуле.

26. Почему важно уменьшать зацепление между классами?
a) Чтобы изменения в одном классе минимально влияли на другие, повышая сопровождаемость.
b) Чтобы увеличить скорость выполнения программы.
c) Чтобы сделать код более компактным.
d) Чтобы использовать меньше памяти.

27. Что такое композиция объектов?
a) Включение одного объекта в качестве части другого объекта.
b) Наследование свойств от нескольких классов.
c) Передача объектов по ссылке.
d) Создание копий объектов.

28. Чем композиция отличается от наследования?
a) Композиция использует включение объектов, а наследование — расширение класса.
b) Композиция позволяет переопределять методы, а наследование — нет.
c) Композиция всегда лучше наследования.
d) Композиция создаёт более сильную связь, чем наследование.

29. Что означает принцип «предпочитай композицию наследованию»?
a) Лучше включать объекты, чем наследовать, чтобы уменьшить связность.
b) Наследование всегда следует использовать вместо композиции.
c) Композиция ухудшает читаемость кода.
d) Наследование проще реализовать, чем композицию.

30. Что такое модуль в контексте ООП-системы?
a) Файл или группа файлов, содержащих связанные классы и функции.
b) Объект с одним методом.
c) Родительский класс для всех классов.
d) Специальный тип данных.

31. Что такое «контракт» класса?
a) Обещание, что класс предоставит определённые методы и свойства с заданным поведением.
b) Документация к классу.
c) Список всех атрибутов класса.
d) Лицензионное соглашение.

32. Что такое инварианты класса?
a) Условия, которые всегда истинны для объекта на протяжении его жизни.
b) Методы, которые нельзя переопределить.
c) Атрибуты, доступные только для чтения.
d) Глобальные переменные класса.

33. Зачем нужны конструкторы в ООП?
a) Для инициализации объекта, установки начальных значений атрибутов.

b) Для удаления объекта из памяти.
c) Для преобразования объекта в строку.
d) Для сравнения объектов.

34. Зачем нужны деструкторы в ООП?
a) Для освобождения ресурсов при удалении объекта.
b) Для создания нового объекта.
c) Для изменения состояния объекта.
d) Для наследования.

35. Что такое перегрузка методов?
a) Создание нескольких методов с одним именем, но разными параметрами.
b) Изменение поведения метода в дочернем классе.
c) Удаление метода из класса.
d) Замена метода свойством.

36. Что такое переопределение методов?
a) Замена реализации метода в дочернем классе на новую.
b) Создание нескольких версий метода в одном классе.
c) Скрытие метода от внешнего доступа.
d) Изменение имени метода.

37. Чем перегрузка отличается от переопределения?
a) Перегрузка — несколько методов в одном классе с разными сигнатурами, переопределение — замена метода в потомке.
b) Перегрузка — изменение имени метода, переопределение — изменение реализации.
c) Перегрузка возможна только в Python, переопределение — в любом языке.
d) Переопределение — создание нескольких методов, перегрузка — замена одного.

38. Что такое статический метод?
a) Метод, принадлежащий классу, а не экземпляру, и не получающий self.
b) Метод, который нельзя вызывать из класса.
c) Метод, который автоматически вызывается при создании объекта.
d) Метод, который меняет состояние класса.

39. Что такое метод класса?
a) Метод, который принимает класс (cls) в качестве первого параметра и может работать с атрибутами класса.
b) Метод, который принадлежит только одному экземпляру.
c) Метод, который нельзя переопределить.
d) Метод, который вызывается автоматически при удалении объекта.

40. Что такое метод экземпляра?
a) Метод, который принимает self и работает с данными конкретного объекта.
b) Метод, который не может обращаться к атрибутам объекта.
c) Метод, который принадлежит только классу.
d) Метод, который вызывается без создания объекта.

41. Что такое поле класса?
a) Атрибут, общий для всех экземпляров класса.

b) Атрибут, который уникален для каждого экземпляра.
c) Метод, который возвращает значение поля.
d) Специальный метод инициализации.

42. Что такое поле экземпляра?
a) Атрибут, принадлежащий конкретному объекту, его состояние.
b) Атрибут, общий для всех объектов класса.
c) Метод, который изменяет состояние класса.
d) Глобальная переменная.

43. Что такое UML-диаграмма классов?
a) Графическое представление классов, их атрибутов, методов и отношений между ними.
b) Диаграмма последовательности вызовов методов.
c) Схема базы данных.
d) График выполнения программы.

44. Что такое шаблон (паттерн) проектирования?
a) Типовое решение часто встречающейся проблемы проектирования.
b) Шаблон для создания UML-диаграмм.
c) Способ компиляции программы.
d) Метод тестирования кода.

45. Зачем изучать паттерны проектирования?
a) Чтобы использовать готовые, проверенные решения и улучшать архитектуру кода.
b) Чтобы быстрее писать код без планирования.
c) Чтобы избежать использования ООП.
d) Только для собеседований.

46. Что такое принципы SOLID?
a) Пять принципов проектирования классов для создания устойчивого к изменениям кода.
b) Принципы написания алгоритмов.
c) Правила именования переменных.
d) Стандарты оформления кода.

47. Для чего нужен принцип единственной ответственности (SRP)?
a) Чтобы у класса была только одна причина для изменения.
b) Чтобы класс выполнял как можно больше задач.
c) Чтобы уменьшить количество классов.
d) Чтобы запретить наследование.

48. Для чего нужен принцип открытости/закрытости (OCP)?
a) Классы должны быть открыты для расширения, но закрыты для изменений.
b) Классы должны быть открыты для изменений и расширений.
c) Классы должны быть закрыты для наследования.
d) Классы должны быть открыты только для дружественных классов.

49. Для чего нужен принцип подстановки Лисков (LSP)?
a) Объекты базового класса должны быть заменяемы объектами потомков без

нарушения работы программы.
b) Наследник должен добавлять новые обязательства.
c) Наследник может игнорировать поведение базового класса.
d) Наследник должен быть менее специфичным, чем родитель.

50. Для чего нужен принцип разделения интерфейса (ISP)?
a) Клиенты не должны зависеть от методов, которые они не используют.
b) Интерфейс должен содержать как можно больше методов.
c) Интерфейс должен быть только один на весь проект.
d) Интерфейс должен скрывать все детали реализации.

51. Для чего нужен принцип инверсии зависимостей (DIP)?
a) Модули высокого уровня не должны зависеть от модулей низкого уровня; оба должны зависеть от абстракций.
b) Зависимости должны быть жёстко закодированы.
c) Модули низкого уровня должны управлять модулями высокого уровня.
d) Абстракции должны зависеть от деталей.

52. Что значит «объект имеет ссылку на другой объект»?
a) Объект хранит идентификатор или указатель на другой объект, может взаимодействовать с ним.
b) Объект является копией другого объекта.
c) Объект наследует от другого объекта.
d) Объект не знает о существовании другого объекта.

53. Что такое «сильная связь» между объектами?
a) Когда один объект напрямую создаёт или жёстко зависит от конкретного другого объекта.
b) Когда объекты взаимодействуют через абстракции.
c) Когда объекты не взаимодействуют.
d) Когда объекты находятся в одном модуле.

54. Что такое «слабая связь» между объектами?
a) Когда объекты взаимодействуют через интерфейсы или абстракции, уменьшая зависимость от конкретных реализаций.
b) Когда объекты жёстко связаны наследованием.
c) Когда объекты не могут взаимодействовать.
d) Когда объекты ссылаются друг на друга напрямую.

55. Что означает «жизненный цикл объекта»?
a) Время существования объекта от создания до уничтожения.
b) Количество методов у объекта.
c) Иерархия наследования объекта.
d) Место объекта в памяти.

56. Что означает «инкапсулировать детали реализации»?
a) Скрыть внутреннее устройство класса, предоставив только публичный интерфейс.
b) Вынести все детали в документацию.
c) Сделать все атрибуты публичными.
d) Использовать только глобальные переменные.

57. Почему важно скрывать внутреннее устройство класса?
a) Чтобы защитить данные от некорректного изменения и упростить изменение реализации без влияния на клиентов.
b) Чтобы сделать код более запутанным.
c) Чтобы запретить наследование.
d) Чтобы увеличить скорость работы.

58. Что такое публичный интерфейс класса?
a) Набор методов и свойств, доступных для использования внешним кодом.
b) Все атрибуты класса, включая приватные.
c) Исходный код класса.
d) Документация класса.

59. Что такое внутреннее (закрытое) устройство класса?
a) Реализация методов и приватные атрибуты, скрытые от внешнего мира.
b) Публичные методы класса.
c) Имя класса и модуль.
d) Родительские классы.

60. Что такое «расширяемость» ООП-системы?
a) Возможность добавлять новую функциональность с минимальными изменениями существующего кода.
b) Возможность удалять функциональность.
c) Неизменность системы после разработки.
d) Скорость добавления новых строк кода.

61. Как объявить класс в Python?
a) С помощью ключевого слова class.
b) С помощью ключевого слова def.
c) С помощью ключевого слова struct.
d) С помощью ключевого слова object.

62. Для чего используется ключевое слово class в Python?
a) Для определения нового класса.
b) Для создания объекта.
c) Для импорта модуля.
d) Для определения функции.

63. Как создать объект класса в Python?
a) Вызвать класс как функцию: obj = ClassName().
b) Использовать ключевое слово new.
c) Использовать ключевое слово object.
d) Скопировать существующий объект.

64. Что такое self в методах класса?
a) Ссылка на экземпляр класса, который вызывает метод.
b) Ссылка на класс.
c) Зарезервированное слово для наследования.
d) Имя метода по умолчанию.

65. Обязательно ли называть первый параметр метода self?
a) Нет, но это общепринятое соглашение; можно использовать другое имя, но self стандартно.
b) Да, иначе метод не будет работать.
c) Нет, первый параметр всегда называется cls.
d) Да, это требование синтаксиса Python.

66. Когда вызывается метод __init__?
a) Сразу после создания объекта, для его инициализации.
b) До создания объекта.
c) При удалении объекта.
d) При вызове любого метода объекта.

67. Для чего используется метод __init__ в классе?
a) Для инициализации атрибутов экземпляра.
b) Для создания самого класса.
c) Для удаления атрибутов.
d) Для преобразования объекта в строку.

68. Можно ли в Python иметь несколько конструкторов в одном классе?
a) Официально — нет, но можно имитировать с помощью @classmethod или параметров по умолчанию.
b) Да, с помощью перегрузки как в C++.
c) Нет, вообще нельзя.
d) Да, используя специальный синтаксис.

69. Как задать значение атрибута экземпляра в методе __init__?
a) Через self: self.attribute = value.
b) Через cls: cls.attribute = value.
c) Без self: attribute = value.
d) Через global.

70. Как обратиться к атрибуту экземпляра внутри метода?
a) Через self: self.attribute.
b) Через имя класса: ClassName.attribute.
c) Напрямую по имени: attribute.
d) Через global.

71. Как обратиться к атрибуту класса внутри метода?
a) Через self.__class__.attribute или имя класса ClassName.attribute.
b) Только через self.attribute.
c) Только через имя класса.
d) Через global.

72. Что такое атрибут класса в Python?
a) Переменная, определённая внутри класса, но вне методов, общая для всех экземпляров.
b) Переменная внутри метода __init__.
c) Локальная переменная метода.

d) Глобальная переменная модуля.

73. Что такое атрибут экземпляра в Python?
a) Переменная, принадлежащая конкретному объекту, обычно определяется в __init__.
b) Переменная, определённая в теле класса.
c) Переменная внутри статического метода.
d) Импортированная переменная.

74. Как определить метод экземпляра в Python?
a) Определить функцию внутри класса с первым параметром self.
b) Использовать декоратор @staticmethod.
c) Использовать декоратор @classmethod.
d) Определить функцию вне класса.

75. Как определить статический метод в классе Python?
a) Использовать декоратор @staticmethod.
b) Использовать декоратор @classmethod.
c) Не использовать self в методе экземпляра.
d) Определить функцию вне класса.

76. Как определить метод класса в Python?
a) Использовать декоратор @classmethod с первым параметром cls.
b) Использовать декоратор @staticmethod.
c) Определить функцию с первым параметром self.
d) Определить функцию в другом модуле.

77. Чем отличается @staticmethod от @classmethod?
a) @staticmethod не получает неявных аргументов, @classmethod получает класс (cls).
b) @staticmethod получает self, @classmethod получает cls.
c) @classmethod нельзя вызывать от экземпляра.
d) @staticmethod может изменять состояние класса.

78. В чём разница между методами класса и методами экземпляра?
a) Метод класса работает с классом (cls), метод экземпляра — с конкретным объектом (self).
b) Метод класса нельзя вызывать от экземпляра.
c) Метод экземпляра не может обращаться к атрибутам класса.
d) Метод класса всегда приватный.

79. Как в Python реализуется наследование между классами?
a) Указанием родительского класса в скобках после имени класса: class Child(Parent).
b) С помощью ключевого слова extends.
c) С помощью ключевого слова inherits.
d) Автоматически для всех классов.

80. Как объявить класс-потомок в Python?
a) class DerivedClass(BaseClass):
b) class DerivedClass extends BaseClass:
c) class DerivedClass inherits BaseClass:
d) class DerivedClass from BaseClass:

81. Для чего используется встроенная функция super()?
a) Для вызова метода родительского класса из дочернего.
b) Для создания суперкласса.
c) Для получения списка всех родителей.
d) Для проверки типа объекта.

82. Как вызвать метод родительского класса из дочернего?
a) Использовать super().method_name(args).
b) Использовать ParentClass.method_name(self, args).
c) Использовать self.method_name(args).
d) Использовать classmethod.

83. Что такое порядок разрешения методов (MRO) в Python?
a) Порядок, в котором Python ищет метод в иерархии классов (C3 linearization).
b) Порядок создания объектов.
c) Порядок вызова конструкторов.
d) Порядок импорта модулей.

84. Как узнать порядок MRO для класса?
a) Использовать атрибут class.__mro__ или функцию help(class).
b) Использовать dir(class).
c) Использовать type(class).
d) Использовать inspect.getmro.

85. Что такое множественное наследование в Python?
a) Когда класс наследует от нескольких родительских классов.
b) Когда класс имеет много потомков.
c) Когда объект принадлежит нескольким классам.
d) Когда метод наследуется от многих классов.

86. Какие проблемы может вызывать множественное наследование?
a) Конфликты имён методов, сложность понимания MRO.
b) Увеличение скорости выполнения.
c) Упрощение кода.
d) Автоматическое разрешение всех конфликтов.

87. Что такое псевдоприватные атрибуты (name mangling) в Python?
a) Механизм, при котором имена атрибутов, начинающиеся с __ (но не заканчивающиеся ими), искажаются для избежания конфликтов.
b) Сокрытие атрибутов от любого доступа.
c) Автоматическое создание геттеров и сеттеров.
d) Шифрование имён атрибутов.

88. Как объявить «приватный» атрибут в Python?
a) Использовать двойное подчёркивание в начале имени: __private_attr.
b) Использовать одинарное подчёркивание: _private_attr.
c) Использовать ключевое слово private.
d) Нельзя объявить полностью приватный атрибут.

89. Как объявить «защищённый» атрибут в Python?
a) Использовать одно подчёркивание в начале имени: _protected_attr.
b) Использовать двойное подчёркивание: __protected_attr.
c) Использовать ключевое слово protected.
d) Использовать аннотации типов.

90. Что такое свойство (property) в Python?
a) Механизм для управления доступом к атрибуту через геттеры, сеттеры и делетеры.
b) Атрибут класса.
c) Метод, который возвращает значение.
d) Декоратор для функций.

91. Как создать свойство с помощью декоратора @property?
a) Определить метод с декоратором @property для геттера, затем @prop_name.setter для сеттера.
b) Использовать только @property.
c) Использовать @attribute.
d) Использовать @getter.

92. Зачем нужны геттеры и сеттеры в Python, если есть прямой доступ к атрибутам?
a) Для добавления валидации, вычисляемых значений, логирования при доступе к атрибуту.
b) Они не нужны, прямой доступ всегда лучше.
c) Только для совместимости с другими языками.
d) Чтобы сделать код длиннее.

93. Что такое метод __str__ в классе Python?
a) Метод, возвращающий строковое представление объекта для пользователя (str(obj), print(obj)).
b) Метод, возвращающий официальное строковое представление для разработчика.
c) Метод для сравнения объектов.
d) Метод для сериализации объекта.

94. Что такое метод __repr__ в классе Python?
a) Метод, возвращающий официальное строковое представление объекта, желательно позволяющее воссоздать объект (repr(obj)).
b) Метод для красивого вывода на печать.
c) Метод для преобразования в байты.
d) Метод для хеширования.

95. Чем __str__ отличается от __repr__?
a) __str__ — для пользователей, __repr__ — для разработчиков; repr часто используется как fallback для str.
b) __str__ — для разработчиков, __repr__ — для пользователей.
c) __str__ возвращает байты, __repr__ — строку.
d) Отличий нет, это синонимы.

96. Как сделать объект итерируемым в Python?
a) Реализовать метод __iter__, возвращающий итератор.
b) Реализовать только метод __next__.

c) Унаследовать от класса Iterable.
d) Использовать декоратор @iterable.

97. Для чего реализуют метод __iter__?
a) Чтобы объект можно было использовать в цикле for (возвращает итератор).
b) Чтобы объект можно было вызвать как функцию.
c) Чтобы объект можно было сложить с другим.
d) Чтобы объект можно было хешировать.

98. Для чего реализуют метод __next__?
a) Чтобы итератор возвращал следующее значение; вызывается функцией next().
b) Чтобы объект можно было итерировать без __iter__.
c) Чтобы определить длину объекта.
d) Чтобы сравнить объекты.

99. Как сделать объект индексируемым (поддерживающим obj[index])?
a) Реализовать метод __getitem__.
b) Реализовать метод __setitem__.
c) Реализовать метод __iter__.
d) Реализовать метод __call__.

100. За что отвечает метод __getitem__?
a) За получение значения по индексу или ключу (obj[key]).
b) За установку значения по индексу.
c) За удаление элемента.
d) За итерацию по объекту.

101. За что отвечает метод __setitem__?
a) За установку значения по индексу или ключу (obj[key] = value).
b) За получение значения.
c) За удаление элемента.
d) За создание среза.

102. За что отвечает метод __delitem__?
a) За удаление элемента по индексу или ключу (del obj[key]).
b) За установку значения.
c) За получение значения.
d) За очистку всего объекта.

103. Что такое протокол итератора в Python?
a) Протокол, состоящий из методов __iter__ (возвращает сам объект) и __next__ (возвращает следующий элемент).
b) Протокол, состоящий только из __next__.
c) Протокол для контекстных менеджеров.
d) Протокол для сравнения объектов.

104. Что такое контекстный менеджер в Python?
a) Объект, который определяет методы __enter__ и __exit__ для управления контекстом (with).
b) Объект, который управляет памятью.

c) Объект, который создаёт потоки.
d) Объект, который сериализует данные.

105. Какие методы должен реализовать контекстный менеджер?
a) __enter__ и __exit__.
b) __init__ и __del__.
c) __call__ и __iter__.
d) __getitem__ и __setitem__.

106. Для чего используют конструкцию with?
a) Для гарантированного выполнения кода входа и выхода из контекста (например, закрытие файла).
b) Для создания циклов.
c) Для условных операторов.
d) Для определения функций.

107. Что такое протокол дескриптора в Python?
a) Протокол, позволяющий объекту управлять доступом к атрибуту другого объекта через методы __get__, __set__, __delete__.
b) Протокол для создания контекстных менеджеров.
c) Протокол для итерации.
d) Протокол для сериализации.

108. Какие методы реализует дескриптор?
a) __get__, __set__, __delete__ (хотя бы один).
b) __enter__, __exit__.
c) __iter__, __next__.
d) __call__.

109. В чём отличие дескриптора от свойства property?
a) property — это частный случай дескриптора, упрощённый для использования внутри одного класса.
b) Дескриптор — это подкласс property.
c) property реализует больше методов.
d) Отличий нет, это одно и то же.

110. Что такое __slots__ и для чего он используется?
a) Специальный атрибут класса, ограничивающий набор допустимых атрибутов экземпляров для экономии памяти.
b) Механизм для создания слотовых функций.
c) Способ определить статические методы.
d) Способ сделать все атрибуты приватными.

111. Как __slots__ влияет на потребление памяти объектами?
a) Уменьшает память, т.к. экземпляры хранят атрибуты в фиксированном массиве, а не в словаре __dict__.
b) Увеличивает память за счёт накладных расходов.
c) Не влияет на память.
d) Влияет только на скорость.

112. Какие ограничения накладывает использование __slots__?
a) Нельзя динамически добавлять новые атрибуты, не указанные в __slots__.
b) Нельзя использовать наследование.
c) Нельзя определять методы.
d) Нельзя создавать экземпляры.

113. Что такое dataclass в Python?
a) Декоратор/класс из модуля dataclasses для автоматического создания методов __init__, __repr__, __eq__ и других.
b) Класс для работы с базами данных.
c) Структура данных из collections.
d) Абстрактный базовый класс.

114. Какие преимущества дают dataclasses при описании классов?
a) Уменьшение шаблонного кода для инициализации, сравнения, представления.
b) Увеличение скорости выполнения.
c) Автоматическое создание графического интерфейса.
d) Шифрование данных.

115. Как объявить класс с помощью декоратора @dataclass?
a) @dataclass class MyClass: поля с аннотациями типов.
b) class MyClass(dataclass):
c) class MyClass: @dataclass
d) dataclass(MyClass)

116. Какие специальные методы автоматически создаёт @dataclass?
a) __init__, __repr__, __eq__, возможно __hash__, __str__ и другие в зависимости от параметров.
b) __iter__, __next__.
c) __enter__, __exit__.
d) __getitem__, __setitem__.

117. Как в dataclass задать значения по умолчанию для полей?
a) Использовать default или default_factory в field().
b) Присвоить значение прямо в аннотации: field: int = 0.
c) В методе __init__.
d) Использовать @default декоратор.

118. Как запретить изменение поля после создания dataclass-объекта?
a) Использовать field(init=False) или сделать поле свойством.
b) Использовать ключевое слово const.
c) Использовать __slots__.
d) Нельзя запретить.

119. Что такое замороженный (frozen) dataclass?
a) dataclass с параметром frozen=True, делающий экземпляры неизменяемыми (immutable).
b) Класс, который нельзя наследовать.
c) Класс, который нельзя инстанцировать.
d) Класс для работы с замороженными данными.

120. Можно ли комбинировать @dataclass и наследование?
a) Да, dataclass-классы могут наследоваться друг от друга.
b) Нет, это запрещено.
c) Только если родительский класс тоже dataclass.
d) Только при использовании множественного наследования.

121. Как перегрузить оператор сложения для класса?
a) Реализовать метод __add__.
b) Реализовать метод __sum__.
c) Реализовать метод __plus__.
d) Использовать декоратор @add.

122. За какой метод отвечает перегрузка оператора +?
a) __add__ для obj + other, __radd__ для other + obj.
b) __plus__.
c) __sum__.
d) __iadd__.

123. Как перегрузить оператор сравнения ==?
a) Реализовать метод __eq__.
b) Реализовать метод __cmp__.
c) Реализовать метод __equal__.
d) Использовать декоратор @eq.

124. Как сделать объекты сравниваемыми по выбранному полю?
a) В методе __eq__ сравнивать это поле (и возможно __hash__).
b) Использовать @total_ordering.
c) Определить метод compare.
d) Унаследовать от Comparable.

125. За какой метод отвечает функция len() для пользовательского класса?
a) __len__.
b) __length__.
c) __size__.
d) __len__ или __bool__.

126. Как реализовать логическое значение объекта (bool(obj))?
a) Реализовать метод __bool__, возвращающий True или False.
b) Реализовать метод __len__ (если нет __bool__, используется он).
c) Использовать декоратор @bool.
d) Определить атрибут bool = True.

127. За какой метод отвечает вызов bool(obj)?
a) __bool__, а если его нет, то __len__ (ноль -> False).
b) __true__.
c) __false__.
d) __bool__ только для чисел.

128. Как сделать объект вызываемым как функция?

a) Реализовать метод __call__.
b) Реализовать метод __invoke__.
c) Использовать декоратор @callable.
d) Сделать его функцией.

129. За какой метод отвечает запись obj()?
a) __call__.
b) __invoke__.
c) __run__.
d) __execute__.

130. Как работает метод __enter__ в контекстном менеджере?
a) Вызывается при входе в блок with, может возвращать объект (например, открытый файл).
b) Вызывается при выходе из блока with.
c) Вызывается при создании объекта.
d) Вызывается при удалении объекта.

131. Как работает метод __exit__ в контекстном менеджере?
a) Вызывается при выходе из блока with, обрабатывает исключения, освобождает ресурсы.
b) Вызывается при входе в блок with.
c) Вызывается при создании объекта.
d) Вызывается при вызове метода.

132. Что произойдёт, если метод __exit__ вернёт True?
a) Исключение, возникшее в блоке with, будет подавлено (не propagated).
b) Исключение будет перевыброшено.
c) Блок with завершится с ошибкой.
d) Ресурсы не будут освобождены.

133. Как правильно организовать импорт классов из других модулей?
a) Импортировать модуль и обращаться к классу через точку, или from module import class.
b) Всегда использовать import *.
c) Копировать код класса в текущий модуль.
d) Использовать глобальные переменные.

134. Что такое переменная __all__ в модуле и как она связана с классами?
a) Список имён, которые будут импортированы при from module import *; может включать имена классов.
b) Список всех классов в модуле.
c) Специальный атрибут каждого класса.
d) Переменная для хранения всех объектов.

135. Как организовать пакет, содержащий несколько связанных классов?
a) Создать папку с __init__.py, разложить классы по модулям внутри, импортировать их в __init__.py.
b) Положить все классы в один файл.
c) Использовать только один класс на пакет.

d) Использовать подклассы вместо пакетов.

136. Какова роль магического метода __init__ в классе Python?
a) Конструктор экземпляра, инициализирует атрибуты объекта.
b) Создаёт сам класс.
c) Удаляет объект.
d) Преобразует объект в строку.

137. Какова роль магического метода __new__ в классе Python?
a) Создаёт и возвращает новый экземпляр класса (вызывается до __init__).
b) Инициализирует атрибуты.
c) Удаляет объект.
d) Клонирует объект.

138. Какова роль магического метода __del__ в классе Python?
a) Деструктор, вызывается при удалении объекта (но время вызова не гарантировано).
b) Конструктор.
c) Метод для сравнения.
d) Метод для сериализации.

139. Какова роль магического метода __str__ в классе Python?
a) Возвращает строковое представление для пользователя.
b) Возвращает официальное представление для разработчика.
c) Преобразует объект в байты.
d) Сравнивает объекты.

140. Какова роль магического метода __repr__ в классе Python?
a) Возвращает официальное строковое представление, часто используемое для отладки.
b) Возвращает представление для пользователя.
c) Преобразует в байты.
d) Вызывается при вызове repr().

141. Какова роль магического метода __bytes__ в классе Python?
a) Возвращает байтовое представление объекта (bytes(obj)).
b) Возвращает строковое представление.
c) Возвращает целое число.
d) Сериализует объект.

142. Какова роль магического метода __format__ в классе Python?
a) Определяет поведение при форматировании (format(obj, spec), f-string).
b) Форматирует строку.
c) Преобразует объект в число.
d) Вызывается при выводе на печать.

143. Какова роль магического метода __len__ в классе Python?
a) Возвращает длину объекта (len(obj)).
b) Возвращает логическое значение.
c) Возвращает размер в байтах.
d) Сравнивает объекты.

144. Какова роль магического метода __bool__ в классе Python?
a) Возвращает логическое значение объекта (bool(obj)).
b) Возвращает длину.
c) Преобразует в строку.
d) Вызывается при сравнении.

145. Какова роль магического метода __call__ в классе Python?
a) Позволяет экземпляру класса вызываться как функция (obj()).
b) Позволяет классу вызываться.
c) Позволяет создать объект.
d) Позволяет удалить объект.

146. Какова роль магического метода __iter__ в классе Python?
a) Возвращает итератор для объекта (используется в for).
b) Возвращает следующий элемент.
c) Проверяет вхождение элемента.
d) Переворачивает объект.

147. Какова роль магического метода __next__ в классе Python?
a) Возвращает следующий элемент итератора (вызывается next()).
b) Возвращает итератор.
c) Проверяет конец итерации.
d) Создаёт итератор.

148. Какова роль магического метода __contains__ в классе Python?
a) Определяет поведение оператора in (x in obj).
b) Определяет поведение оператора not in.
c) Проверяет тип объекта.
d) Ищет подстроку.

149. Какова роль магического метода __getitem__ в классе Python?
a) Позволяет получать элемент по индексу/ключу (obj[key]).
b) Позволяет устанавливать элемент.
c) Позволяет удалять элемент.
d) Позволяет итерировать.

150. Какова роль магического метода __setitem__ в классе Python?
a) Позволяет устанавливать элемент по индексу/ключу (obj[key] = value).
b) Позволяет получать элемент.
c) Позволяет удалять элемент.
d) Позволяет создавать срезы.

151. Какова роль магического метода __delitem__ в классе Python?
a) Позволяет удалять элемент по индексу/ключу (del obj[key]).
b) Позволяет получать элемент.
c) Позволяет устанавливать элемент.
d) Позволяет очищать объект.

152. Какова роль магического метода __reversed__ в классе Python?
a) Возвращает обратный итератор (reversed(obj)).

b) Переворачивает объект на месте.
c) Сортирует объект.
d) Возвращает строку в обратном порядке.

153. Какова роль магического метода __enter__ в классе Python?
a) Вызывается при входе в контекст with, возвращает объект для использования.
b) Вызывается при выходе из контекста.
c) Вызывается при создании объекта.
d) Вызывается при удалении объекта.

154. Какова роль магического метода __exit__ в классе Python?
a) Вызывается при выходе из контекста with, обрабатывает исключения, освобождает ресурсы.
b) Вызывается при входе в контекст.
c) Вызывается при создании объекта.
d) Вызывается при вызове метода.

155. Какова роль магического метода __add__ в классе Python?
a) Определяет поведение оператора сложения (obj + other).
b) Определяет поведение оператора вычитания.
c) Определяет поведение оператора умножения.
d) Определяет поведение оператора деления.

156. Какова роль магического метода __radd__ в классе Python?
a) Определяет поведение оператора сложения, когда объект находится справа (other + obj).
b) Определяет правое вычитание.
c) Определяет правое умножение.
d) Определяет правое деление.

157. Какова роль магического метода __iadd__ в классе Python?
a) Определяет поведение оператора += (сложение на месте).
b) Определяет поведение +.
c) Определяет поведение =+.
d) Определяет поведение ++.

158. Какова роль магического метода __sub__ в классе Python?
a) Определяет поведение оператора вычитания (obj - other).
b) Определяет поведение сложения.
c) Определяет поведение умножения.
d) Определяет поведение деления.

159. Какова роль магического метода __mul__ в классе Python?
a) Определяет поведение оператора умножения (obj * other).
b) Определяет поведение сложения.
c) Определяет поведение вычитания.
d) Определяет поведение деления.

160. Какова роль магического метода __truediv__ в классе Python?
a) Определяет поведение оператора деления (obj / other).

b) Определяет поведение целочисленного деления.
c) Определяет поведение умножения.
d) Определяет поведение сложения.

161. Какова роль магического метода __floordiv__ в классе Python?
a) Определяет поведение оператора целочисленного деления (obj // other).
b) Определяет поведение обычного деления.
c) Определяет поведение умножения.
d) Определяет поведение вычитания.

162. Какова роль магического метода __eq__ в классе Python?
a) Определяет поведение оператора равенства (obj == other).
b) Определяет поведение неравенства.
c) Определяет поведение меньше.
d) Определяет поведение больше.

163. Какова роль магического метода __lt__ в классе Python?
a) Определяет поведение оператора меньше (obj < other).
b) Определяет поведение равенства.
c) Определяет поведение больше.
d) Определяет поведение не равно.

164. Какова роль магического метода __hash__ в классе Python?
a) Возвращает хеш-значение объекта (используется в словарях, множествах).
b) Возвращает строку.
c) Возвращает длину.
d) Сравнивает объекты.

165. Какова роль магического метода __copy__ в классе Python?
a) Определяет поведение поверхностного копирования (copy.copy(obj)).
b) Определяет поведение глубокого копирования.
c) Определяет поведение клонирования.
d) Определяет поведение сериализации.

166. В чём суть паттерна проектирования Singleton?
a) Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.
b) Гарантирует, что класс имеет много экземпляров.
c) Гарантирует, что класс нельзя инстанцировать.
d) Гарантирует, что экземпляры не разделяют состояние.

167. Когда имеет смысл применять паттерн Singleton в Python-коде?
a) Когда нужен единственный экземпляр для управления ресурсами (например, подключение к БД, логгер).
b) Когда нужно создавать много объектов.
c) Когда нужно запретить создание объектов.
d) Когда нужно наследовать от нескольких классов.

168. В чём суть паттерна проектирования Factory Method?
a) Определяет интерфейс для создания объекта, но оставляет подклассам решение о

том, какой класс инстанцировать.
b) Создаёт объекты без указания конкретного класса.
c) Клонирует существующие объекты.
d) Уничтожает объекты.

169. Когда имеет смысл применять паттерн Factory Method в Python-коде?
a) Когда класс заранее не знает, объекты каких классов ему нужно создавать.
b) Когда нужно всегда создавать один и тот же объект.
c) Когда нужно скрыть конструктор класса.
d) Когда нужно создать объект без использования new.

170. В чём суть паттерна проектирования Abstract Factory?
a) Предоставляет интерфейс для создания семейств связанных или зависимых объектов без указания их конкретных классов.
b) Создаёт один объект.
c) Создаёт копии объектов.
d) Уничтожает семейства объектов.

171. Когда имеет смысл применять паттерн Abstract Factory в Python-коде?
a) Когда система должна быть независимой от того, как создаются, компонуются и представляются её продукты.
b) Когда нужно создать только один продукт.
c) Когда продукты не связаны между собой.
d) Когда нужно использовать только конкретные классы.

172. В чём суть паттерна проектирования Builder?
a) Разделяет конструирование сложного объекта от его представления, позволяя использовать один и тот же процесс конструирования для разных представлений.
b) Создаёт объект одним вызовом.
c) Клонирует объект.
d) Разрушает объект по частям.

173. Когда имеет смысл применять паттерн Builder в Python-коде?
a) Когда объект имеет много опциональных параметров или сложный процесс конструирования.
b) Когда объект имеет только обязательные параметры.
c) Когда объект нужно создавать быстро.
d) Когда нужно избежать использования конструктора.

174. В чём суть паттерна проектирования Prototype?
a) Позволяет создавать новые объекты путём копирования существующего объекта-прототипа.
b) Создаёт объекты с нуля.
c) Уничтожает прототипы.
d) Создаёт классы на лету.

175. Когда имеет смысл применять паттерн Prototype в Python-коде?
a) Когда создание объекта дороже, чем копирование существующего.
b) Когда нужно всегда создавать уникальные объекты.
c) Когда нельзя использовать наследование.

d) Когда объекты не имеют состояния.

176. В чём суть паттерна проектирования Adapter?
a) Преобразует интерфейс одного класса в интерфейс, ожидаемый клиентом.
b) Изменяет интерфейс класса без адаптации.
c) Скрывает интерфейс класса.
d) Упрощает интерфейс класса.

177. Когда имеет смысл применять паттерн Adapter в Python-коде?
a) Когда нужно использовать существующий класс, но его интерфейс не совместим с остальным кодом.
b) Когда интерфейс класса идеально подходит.
c) Когда нужно создать новый класс с нуля.
d) Когда нужно удалить интерфейс класса.

178. В чём суть паттерна проектирования Facade?
a) Предоставляет унифицированный интерфейс к набору интерфейсов в подсистеме.
b) Разделяет интерфейсы на более мелкие.
c) Скрывает все интерфейсы.
d) Усложняет интерфейс подсистемы.

179. Когда имеет смысл применять паттерн Facade в Python-коде?
a) Когда нужно упростить взаимодействие со сложной подсистемой, скрыв её детали.
b) Когда нужно предоставить все детали подсистемы.
c) Когда подсистема проста.
d) Когда нужно увеличить связанность.

180. В чём суть паттерна проектирования Decorator?
a) Динамически добавляет объекту новые обязанности, являясь гибкой альтернативой наследованию.
b) Удаляет обязанности у объекта.
c) Заменяет объект другим.
d) Создаёт копии объекта.

181. Когда имеет смысл применять паттерн Decorator в Python-коде?
a) Когда нужно добавлять обязанности объектам на лету, не изменяя их класс.
b) Когда обязанности фиксированы и не меняются.
c) Когда нужно уменьшить функциональность.
d) Когда нужно запретить наследование.

182. В чём суть паттерна проектирования Strategy?
a) Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.
b) Жёстко фиксирует алгоритм.
c) Смешивает алгоритмы.
d) Удаляет алгоритмы.

183. Когда имеет смысл применять паттерн Strategy в Python-коде?
a) Когда есть несколько вариантов алгоритма и нужно выбирать один во время выполнения.

b) Когда алгоритм только один.
c) Когда алгоритмы не могут быть заменены.
d) Когда нужно ускорить выполнение алгоритма.

184. В чём суть паттерна проектирования Observer?
a) Определяет зависимость «один ко многим» между объектами, чтобы при изменении состояния одного объекта все зависящие от него оповещались.
b) Определяет зависимость «много к одному».
c) Скрывает изменения состояния.
d) Игнорирует изменения состояния.

185. Когда имеет смысл применять паттерн Observer в Python-коде?
a) Когда изменение состояния одного объекта должно приводить к автоматическому изменению других объектов, причём их число неизвестно заранее.
b) Когда объекты не зависят друг от друга.
c) Когда нужно жёстко связать объекты.
d) Когда нужно избежать оповещений.

186. В чём суть паттерна проектирования Command?
a) Инкапсулирует запрос как объект, позволяя параметризовать клиентов с разными запросами, ставить запросы в очередь или логировать их.
b) Выполняет запрос напрямую.
c) Удаляет запросы.
d) Создаёт запросы без инкапсуляции.

187. Когда имеет смысл применять паттерн Command в Python-коде?
a) Когда нужно параметризовать объекты выполняемым действием, поддерживать отмену операций или очередь команд.
b) Когда действия фиксированы и не меняются.
c) Когда нужно выполнять действия немедленно.
d) Когда не нужно логировать действия.

188. В чём суть паттерна проектирования State?
a) Позволяет объекту изменять своё поведение при изменении внутреннего состояния.
b) Фиксирует поведение объекта.
c) Скрывает состояние объекта.
d) Удаляет состояние объекта.

189. Когда имеет смысл применять паттерн State в Python-коде?
a) Когда поведение объекта зависит от его состояния и должно изменяться во время выполнения.
b) Когда состояние объекта постоянно.
c) Когда объект не имеет состояния.
d) Когда нужно упростить код, жёстко закодировав условия.

190. В чём суть паттерна проектирования Template Method?
a) Определяет скелет алгоритма в базовом классе, позволяя подклассам переопределять некоторые шаги без изменения структуры алгоритма.
b) Определяет полный алгоритм, который нельзя менять.
c) Позволяет менять структуру алгоритма.

d) Скрывает алгоритм.

191. Когда имеет смысл применять паттерн Template Method в Python-коде?
a) Когда нужно реализовать неизменную часть алгоритма один раз и позволить подклассам реализовать изменяемые части.
b) Когда алгоритм полностью изменяем.
c) Когда алгоритм не имеет структуры.
d) Когда нужно избежать наследования.

192. В чём суть паттерна проектирования Iterator?
a) Предоставляет способ последовательного доступа к элементам агрегированного объекта без раскрытия его внутреннего представления.
b) Раскрывает внутреннее представление.
c) Позволяет доступ в произвольном порядке.
d) Удаляет элементы.

193. Когда имеет смысл применять паттерн Iterator в Python-коде?
a) Когда нужно обеспечить единый интерфейс для обхода разных агрегированных структур.
b) Когда структура проста и обход не нужен.
c) Когда нужно изменить структуру во время обхода.
d) Когда нужно скрыть возможность обхода.

194. В чём суть паттерна проектирования Composite?
a) Позволяет сгруппировать объекты в древовидные структуры для представления иерархий «часть-целое».
b) Разделяет объекты на независимые.
c) Удаляет иерархии.
d) Создаёт линейные структуры.

195. Когда имеет смысл применять паттерн Composite в Python-коде?
a) Когда нужно работать с иерархией объектов, где и отдельные объекты, и их композиции обрабатываются единообразно.
b) Когда объекты не образуют иерархии.
c) Когда нужно различать листья и композиции.
d) Когда нужно избежать рекурсии.

196. Сформулируйте принцип единственной ответственности (SRP) в ООП.
a) Класс должен иметь только одну причину для изменения.
b) Класс должен выполнять как можно больше функций.
c) Класс должен отвечать за всё.
d) Класс не должен меняться никогда.

197. Приведите пример нарушения принципа единственной ответственности в коде на Python.
a) Класс, который и сохраняет данные в файл, и выполняет сложные вычисления, и отправляет email.
b) Класс, который только сохраняет данные в файл.
c) Класс, который только выполняет вычисления.
d) Класс, который только отправляет email.

198. Как соблюдение принципа единственной ответственности влияет на проектирование классов?
a) Приводит к созданию более мелких, сфокусированных классов, которые легче поддерживать и тестировать.
b) Приводит к созданию больших универсальных классов.
c) Усложняет тестирование.
d) Увеличивает связанность.

199. Сформулируйте принцип открытости/закрытости (OCP) в ООП.
a) Классы должны быть открыты для расширения, но закрыты для изменений.
b) Классы должны быть закрыты для расширения и изменений.
c) Классы должны быть открыты для изменений и расширений.
d) Классы должны быть закрыты для расширения, но открыты для изменений.

200. Приведите пример нарушения принципа открытости/закрытости в коде на Python.
a) Класс, в котором для добавления нового типа нужно изменять существующий код (например, через if-elif цепочки).
b) Класс, который использует наследование для добавления нового поведения.
c) Класс, который использует композицию для расширения.
d) Класс, который не меняется при добавлении новых типов.

201. Как соблюдение принципа открытости/закрытости влияет на проектирование классов?
a) Подталкивает к использованию абстракций (интерфейсов, абстрактных классов) и полиморфизма для расширения функциональности.
b) Подталкивает к жёсткому кодированию логики.
c) Увеличивает количество изменений в существующем коде.
d) Делает классы менее гибкими.

202. Сформулируйте принцип подстановки Лисков (LSP) в ООП.
a) Объекты базового класса должны быть заменяемы объектами подклассов без нарушения корректности программы.
b) Подклассы могут игнорировать поведение базового класса.
c) Базовый класс должен знать о всех подклассах.
d) Замена базового класса подклассом всегда приводит к ошибкам.

203. Приведите пример нарушения принципа подстановки Лисков в коде на Python.
a) Квадрат, наследующий от прямоугольника, но меняющий поведение сеттеров ширины и высоты, что ломает ожидаемое поведение прямоугольника.
b) Квадрат, который не наследует от прямоугольника.
c) Прямоугольник, который наследует от квадрата.
d) Классы, которые полностью следуют контракту родителя.

204. Как соблюдение принципа подстановки Лисков влияет на проектирование классов?
a) Заставляет тщательно проектировать иерархию наследования, обеспечивая совместимость поведения подклассов.
b) Позволяет произвольно менять поведение в подклассах.

c) Упрощает наследование без ограничений.
d) Требует, чтобы подклассы были идентичны родителю.

205. Сформулируйте принцип разделения интерфейса (ISP) в ООП.
a) Клиенты не должны зависеть от методов, которые они не используют.
b) Интерфейс должен содержать как можно больше методов.
c) Интерфейс должен быть один на всю систему.
d) Клиенты должны зависеть от всех методов интерфейса.

206. Приведите пример нарушения принципа разделения интерфейса в коде на Python.
a) Интерфейс «Работник» с методами work(), eat(), sleep(), когда некоторые классы (например, Robot) не должны реализовывать eat() и sleep().
b) Интерфейс «Работник» только с методом work().
c) Отдельные интерфейсы для work, eat, sleep.
d) Класс, реализующий только нужные ему методы.

207. Как соблюдение принципа разделения интерфейса влияет на проектирование классов?
a) Ведёт к созданию небольших, специфичных интерфейсов вместо одного громоздкого.
b) Ведёт к созданию одного универсального интерфейса.
c) Увеличивает количество зависимостей.
d) Заставляет классы реализовывать ненужные методы.

208. Сформулируйте принцип инверсии зависимостей (DIP) в ООП.
a) Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракций. Абстракции не должны зависеть от деталей, детали должны зависеть от абстракций.
b) Модули нижнего уровня должны управлять модулями верхнего уровня.
c) Зависимости должны быть жёстко закодированы.
d) Абстракции должны зависеть от деталей.

209. Приведите пример нарушения принципа инверсии зависимостей в коде на Python.
a) Класс ReportGenerator напрямую создаёт экземпляр DatabaseConnection вместо получения абстракции (например, интерфейса DataSource) через конструктор.
b) ReportGenerator принимает абстракцию DataSource в конструкторе.
c) DatabaseConnection реализует интерфейс DataSource.
d) Зависимости внедряются извне.

210. Как соблюдение принципа инверсии зависимостей влияет на проектирование классов?
a) Заставляет программировать на уровне интерфейсов, снижает зацепление, упрощает тестирование и замену реализаций.
b) Увеличивает жёсткие зависимости.
c) Усложняет тестирование.
d) Привязывает код к конкретным реализациям.

211. Какие основные классы вы бы выделили при моделировании интернет-

магазина?
a) Product, Category, Cart, Order, Customer, Payment.
b) только Product и Order.
c) Shop, Item, Money.
d) Database, UserInterface.

212. Как бы вы организовали связи между классами в модели интернет-магазина?
a) Cart содержит список Product; Order ссылается на Customer и содержит список CartItem; Payment ассоциирован с Order.
b) Все классы независимы.
c) Product наследует от Cart.
d) Customer наследует от Order.

213. Какие методы могли бы быть у ключевого класса в системе интернет-магазина?
a) У Cart: add_product, remove_product, calculate_total.
b) У Cart: только хранение продуктов.
c) У Product: process_payment.
d) У Customer: add_to_cart.

214. Какие основные классы вы бы выделили при моделировании библиотеки?
a) Book, Author, Reader, Loan, Library.
b) только Book и Shelf.
c) Page, Cover.
d) Database, Catalog.

215. Как бы вы организовали связи между классами в модели библиотеки?
a) Book имеет Author; Loan связывает Reader, Book и даты; Library управляет коллекцией Book и списком Reader.
b) Book наследует от Reader.
c) Author содержит Library.
d) Loan не связан с Book.

216. Какие методы могли бы быть у ключевого класса в системе библиотеки?
a) У Library: add_book, find_book, register_reader, lend_book.
b) У Book: read, turn_page.
c) У Author: write_book.
d) У Reader: borrow без связи с Library.

217. Какие основные классы вы бы выделили при моделировании системы бронирования отелей?
a) Hotel, Room, Reservation, Customer, Payment.
b) только Room и Bed.
c) Key, Reception.
d) City, Address.

218. Как бы вы организовали связи между классами в модели системы бронирования отелей?
a) Hotel содержит Room; Reservation ссылается на Customer, Room и даты; Payment ассоциирован с Reservation.
b) Room наследует от Hotel.

c) Customer содержит Hotel.
d) Reservation не связан с Room.

219. Какие методы могли бы быть у ключевого класса в системе системы бронирования отелей?
a) У Hotel: find_available_room, make_reservation, cancel_reservation.
b) У Room: open_door, clean.
c) У Customer: pay.
d) У Payment: process без связи с Reservation.

220. Какие основные классы вы бы выделили при моделировании школьного электронного журнала?
a) Student, Teacher, Subject, Grade, class (учебный класс).
b) только Pen и Paper.
c) Desk, Blackboard.
d) School, Principal.

221. Как бы вы организовали связи между классами в модели школьного электронного журнала?
a) class содержит список Student; Teacher ведёт Subject; Grade связывает Student, Subject, Teacher и оценку.
b) Student наследует от Teacher.
c) Subject содержит School.
d) Grade не связан с Student.

222. Какие методы могли бы быть у ключевого класса в системе школьного электронного журнала?
a) У Journal (или Gradebook): add_grade, get_grades_for_student, calculate_average.
b) У Student: study, answer.
c) У Teacher: teach, give_homework.
d) У Subject: start_lesson.

223. Какие основные классы вы бы выделили при моделировании банковской системы?
a) Account, Customer, Transaction, Bank, Card.
b) только Money и Safe.
c) Teller, Window.
d) City, Street.

224. Как бы вы организовали связи между классами в модели банковской системы?
a) Customer имеет Account; Transaction ссылается на Account (отправитель/получатель) и сумму; Bank управляет Account и Customer.
b) Account наследует от Customer.
c) Transaction содержит Bank.
d) Card не связан с Account.

225. Какие методы могли бы быть у ключевого класса в системе банковской системы?
a) У Bank: open_account, close_account, transfer_money, get_statement.
b) У Account: только хранить баланс.

c) У Customer: spend_money.
d) У Transaction: только запись суммы.

226. Какие основные классы вы бы выделили при моделировании чат-приложения?
a) User, Message, ChatRoom (или Conversation), ChatServer.
b) только Text и Emoji.
c) Phone, Network.
d) Screen, Keyboard.

227. Как бы вы организовали связи между классами в модели чат-приложения?
a) ChatRoom содержит список User и список Message; Message имеет отправителя (User) и текст; ChatServer управляет ChatRoom.
b) User наследует от Message.
c) Message содержит ChatServer.
d) ChatRoom не связан с User.

228. Какие методы могли бы быть у ключевого класса в системе чат-приложения?
a) У ChatRoom: add_user, remove_user, send_message, get_message_history.
b) У User: type_message, read_message.
c) У Message: только хранить текст.
d) У ChatServer: только запускаться.

229. Какие основные классы вы бы выделили при моделировании игры крестики-нолики?
a) Board, Player, Game, Cell.
b) только X и O.
c) Pencil, Paper.
d) Rule, Winner.

230. Как бы вы организовали связи между классами в модели игры крестики-нолики?
a) Board содержит матрицу Cell; Game имеет двух Player и Board; Player делает ход на Board.
b) Cell наследует от Board.
c) Player содержит Game.
d) Game не связан с Board.

231. Какие методы могли бы быть у ключевого класса в системе игры крестики-нолики?
a) У Board: make_move, check_winner, is_full.
b) У Player: только иметь символ (X или O).
c) У Cell: только хранить значение.
d) У Game: только создавать Board.

232. Какие основные классы вы бы выделили при моделировании системы управления задачами?
a) Task, Project, User, Tag, Comment.
b) только Paper и Pen.
c) Calendar, Reminder.
d) Database, Table.

233. Как бы вы организовали связи между классами в модели системы управления задачами?
a) Project содержит список Task; Task может иметь Tag, Comment, исполнителя (User); User имеет список Task.
b) Task наследует от Project.
c) Tag содержит User.
d) Comment не связан с Task.

234. Какие методы могли бы быть у ключевого класса в системе системы управления задачами?
a) У TaskManager (или Project): add_task, assign_task, change_status, filter_by_tag.
b) У Task: только описание.
c) У User: только имя.
d) У Tag: только название.

235. Какие основные классы вы бы выделили при моделировании онлайн-курса по программированию?
a) Course, Lesson, Student, Teacher, Assignment, Quiz.
b) только Video and Text.
c) Computer, Internet.
d) Certificate, Diploma.

236. Как бы вы организовали связи между классами в модели онлайн-курса по программированию?
a) Course содержит Lesson и Assignment; Student записывается на Course; Teacher ведёт Course; Quiz может быть частью Lesson.
b) Lesson наследует от Course.
c) Student содержит Teacher.
d) Assignment не связан с Course.

237. Какие методы могли бы быть у ключевого класса в системе онлайн-курса по программированию?
a) У Course: enroll_student, add_lesson, publish_assignment, calculate_progress.
b) У Lesson: только показывать видео.
c) У Student: только смотреть урок.
d) У Teacher: только создавать курс.

238. Какие основные классы вы бы выделили при моделировании интернет-форума?
a) User, Topic, Post, Comment, Forum.
b) только Text and Button.
c) Server, Database.
d) Moderator, Admin.

239. Как бы вы организовать связи между классами в модели интернет-форума?
a) Forum содержит Topic; Topic содержит Post; Post имеет автора (User) и может иметь Comment; Comment также имеет автора.
b) Post наследует от Topic.
c) User содержит Forum.
d) Comment не связан с Post.

240. Какие методы могли бы быть у ключевого класса в системе интернет-форума?
a) У Forum: create_topic, add_post, search_posts, get_recent_activity.
b) У User: только регистрироваться.
c) У Post: только хранить текст.
d) У Comment: только ответ.

241. Какие основные классы вы бы выделили при моделировании системы управления складом?
a) Product, Warehouse, Shelf, InventoryItem, Supplier, Order.
b) только Box and Pallet.
c) Forklift, Worker.
d) Address, Phone.

242. Как бы вы организовали связи между классами в модели системы управления складом?
a) Warehouse содержит Shelf; Shelf хранит InventoryItem; InventoryItem ссылается на Product; Order может содержать Product и ссылаться на Supplier.
b) Product наследует от Warehouse.
c) Shelf содержит Supplier.
d) InventoryItem не связан с Product.

243. Какие методы могли бы быть у ключевого класса в системе системы управления складом?
a) У Warehouse: add_product, remove_product, check_stock, reorder_from_supplier.
b) У Product: только название.
c) У Shelf: только номер.
d) У Supplier: только имя.

244. Какие основные классы вы бы выделили при моделировании кинопросмотра в кинотеатре?
a) Movie, CinemaHall, Showtime, Ticket, Customer, Seat.
b) только Popcorn and Drink.
c) Screen, Projector.
d) City, Street.

245. Как бы вы организовали связи между классами в модели кинопросмотра в кинотеатре?
a) CinemaHall имеет Seat; Showtime связывает Movie, CinemaHall и время; Ticket ссылается на Showtime, Seat и Customer.
b) Movie наследует от CinemaHall.
c) Seat содержит Customer.
d) Ticket не связан с Showtime.

246. Какие методы могли бы быть у ключевого класса в системе кинопросмотра в кинотеатре?
a) У BookingSystem (или Cinema): add_showtime, book_ticket, cancel_ticket, get_available_seats.
b) У Movie: только воспроизводиться.
c) У Seat: только номер.
d) У Customer: только покупать.

247. Какие основные классы вы бы выделили при моделировании социальной сети?
a) User, Post, Comment, Like, FriendRequest, NewsFeed.
b) только Photo and Text.
c) Server, Database.
d) Advertisement, Sponsor.

248. Как бы вы организовали связи между классами в модели социальной сети?
a) User может создавать Post, оставлять Comment, ставить Like; FriendRequest связывает двух User; NewsFeed агрегирует Post от друзей.
b) Post наследует от User.
c) Comment содержит NewsFeed.
d) Like не связан с Post.

249. Какие методы могли бы быть у ключевого класса в системе социальной сети?
a) У SocialNetwork (или UserProfile): create_post, add_friend, like_post, get_feed.
b) У User: только имя.
c) У Post: только текст.
d) У Comment: только ответ.

250. Какие основные классы вы бы выделили при моделировании сервиса доставки еды?
a) Restaurant, Dish, Order, Customer, Courier, Cart.
b) только Food and Box.
c) Car, Bicycle.
d) Menu, Price.

251. Как бы вы организовали связи между классами в модели сервиса доставки еды?
a) Restaurant предлагает Dish; Order содержит Dish, ссылается на Customer и Courier; Cart временно хранит выбранные Dish.
b) Dish наследует от Restaurant.
c) Courier содержит Order.
d) Cart не связан с Dish.

252. Какие методы могли бы быть у ключевого класса в системе сервиса доставки еды?
a) У OrderService (или Restaurant): create_order, assign_courier, track_order, calculate_total.
b) У Dish: только название.
c) У Customer: только адрес.
d) У Courier: только транспорт.

253. Какие основные классы вы бы выделили при моделировании системы онлайн-тестирования?
a) Test, Question, Answer, Student, Result, Teacher.
b) только Pen and Paper.
c) Timer, Score.
d) Certificate, Badge.

254. Как бы вы организовали связи между классами в модели системы онлайн-

тестирования?
a) Test содержит список Question; Question имеет варианты Answer; Result связывает Student, Test и оценку; Teacher создаёт Test.
b) Question наследует от Test.
c) Answer содержит Student.
d) Result не связан с Test.

255. Какие методы могли бы быть у ключевого класса в системе системы онлайн-тестирования?
a) У TestEngine: start_test, submit_answer, calculate_score, generate_report.
b) У Question: только текст.
c) У Student: только имя.
d) У Teacher: только создавать вопросы.

256. По каким признакам можно понять, что класс нарушает принцип единственной ответственности?
a) У класса слишком много методов, которые относятся к разным аспектам (например, работа с данными, логика, UI, сетевые запросы).
b) У класса только один метод.
c) Класс имеет только атрибуты, но не методы.
d) Класс очень маленький.

257. Почему наличие слишком большого количества методов у одного класса считается признаком проблемного дизайна?
a) Это часто указывает на нарушение SRP, класс становится сложным для понимания, тестирования и изменения.
b) Это означает, что класс очень мощный и полезный.
c) Это улучшает производительность.
d) Это сокращает количество файлов.

258. Что такое «божественный объект» (God Object) в ООП?
a) Класс, который знает или делает слишком много, централизуя в себе большую часть логики системы.
b) Класс, который ничего не делает.
c) Класс, который отвечает только за одну задачу.
d) Класс-синглтон.

259. Почему избыточное использование глобальных переменных плохо сочетается с ООП-подходом?
a) Оно нарушает инкапсуляцию, создаёт скрытые зависимости и затрудняет тестирование.
b) Оно ускоряет выполнение программы.
c) Оно улучшает читаемость.
d) Оно рекомендуется в ООП.

260. Как обнаружить дублирование кода между несколькими классами?
a) Обратить внимание на похожие методы или блоки кода в разных классах; использовать инструменты статического анализа.
b) Дублирования кода не бывает.
c) Код всегда дублируется, это нормально.

d) Дублирование видно только при запуске программы.

261. Какие приёмы используются для уменьшения дублирования кода в иерархии классов?
a) Вынос общего кода в базовый класс, использование композиции, создание отдельных служебных классов (утилит).
b) Копирование кода в каждый класс.
c) Удаление дублирующего кода из всех классов.
d) Использование глобальных переменных.

262. Зачем разбивать большие классы на более мелкие?
a) Чтобы улучшить соблюдение SRP, упростить тестирование, повысить переиспользуемость и читаемость.
b) Чтобы увеличить количество файлов.
c) Чтобы сделать классы более монолитными.
d) Чтобы усложнить архитектуру.

263. Что такое интерфейсный класс (pure interface) и когда он полезен?
a) Класс, который объявляет только методы (без реализации), используется для определения контракта; полезен для снижения зацепления.
b) Класс с полной реализацией всех методов.
c) Класс, который нельзя наследовать.
d) Класс для работы с GUI.

264. Почему слишком глубокая иерархия наследования считается нежелательной?
a) Усложняет понимание кода, делает его хрупким (изменения вверху влияют на многих потомков), может нарушать LSP.
b) Упрощает добавление новой функциональности.
c) Ускоряет выполнение программы.
d) Рекомендуется для всех проектов.

265. Когда наследование можно заменить композицией?
a) Когда нужно повторно использовать поведение, но не обязательно создавать отношение «является» (is-a).
b) Всегда, наследование следует избегать полностью.
c) Никогда, наследование всегда лучше.
d) Только при множественном наследовании.

266. Что такое инверсия управления (IoC) в ООП?
a) Принцип, при котором контроль за созданием и связыванием объектов передаётся внешнему контейнеру, а не самим классам.
b) Контроль остаётся внутри классов.
c) Управление памятью вручную.
d) Инвертирование порядка вызовов методов.

267. Зачем использовать внедрение зависимостей (Dependency Injection) в Python?
a) Чтобы уменьшить зацепление, облегчить тестирование (подмена зависимостей) и следовать DIP.
b) Чтобы жёстко кодировать зависимости.
c) Чтобы ускорить создание объектов.

d) Чтобы избежать использования классов.

268. Какие преимущества даёт использование протоколов и duck typing в Python?
a) Гибкость: можно работать с любыми объектами, имеющими нужные методы, без жёсткого наследования; поддерживает полиморфизм без иерархий.
b) Жёсткая типизация и проверка на этапе компиляции.
c) Увеличение скорости выполнения.
d) Обязательность использования абстрактных базовых классов.

269. Чем структурная типизация отличается от номинальной в контексте Python?
a) Структурная (как в протоколах/duck typing) учитывает наличие методов/атрибутов, номинальная (как в ABC) — явное объявление наследования.
b) Структурная требует наследования, номинальная — нет.
c) Номинальная типизация слабее структурной.
d) В Python используется только номинальная типизация.

270. В чём преимущества использования абстрактных базовых классов по сравнению с обычными?
a) Чётко задают контракт (обязательные методы), предоставляют встроенные механизмы регистрации и проверки реализации.
b) Они быстрее выполняются.
c) Они не могут иметь реализаций.
d) Они проще в использовании, чем протоколы.

271. Когда лучше использовать протоколы вместо абстрактных классов?
a) Когда нужно определить интерфейс на основе наличия методов (структурная типизация), особенно для существующих классов или для уменьшения зацепления.
b) Всегда, протоколы лучше во всех случаях.
c) Когда требуется жёсткая иерархия наследования.
d) Когда нужно предоставить реализацию по умолчанию.

272. Как документировать интерфейс класса, чтобы им было удобно пользоваться?
a) Использовать docstrings, описывать назначение класса, параметры методов, возвращаемые значения, возможные исключения.
b) Не документировать, код должен говорить сам за себя.
c) Писать комментарии в случайных местах.
d) Использовать только внешнюю документацию.

273. Почему стоит избегать сильной связи с конкретными реализациями внутри классов?
a) Это затрудняет замену реализации, тестирование и адаптацию к изменениям, увеличивает зацепление.
b) Это делает код быстрее.
c) Это упрощает чтение кода.
d) Это рекомендуется для оптимизации.

274. Как сократить количество импортов конкретных классов в модуле за счёт абстракций?
a) Импортировать и использовать абстрактные классы или протоколы, а конкретные реализации передавать через DI.

b) Импортировать всё через import *.
c) Копировать код классов в модуль.
d) Использовать глобальные переменные вместо импортов.

275. Когда стоит разделить один модуль с классами на несколько модулей?
a) Когда модуль становится слишком большим, классы логически группируются по разным ответственностям, или чтобы уменьшить циклические зависимости.
b) Когда в модуле только один класс.
c) Всегда держать все классы в одном модуле.
d) Когда имена классов слишком длинные.

276. Что такое «тонкий» контроллер и «толстая» модель в архитектуре приложений?
a) Подход, при котором контроллер (обработчик запросов) содержит минимум логики, а основная бизнес-логика находится в моделях (классах предметной области).
b) Контроллер содержит всю логику, а модели только данные.
c) Модели не содержат логики, только геттеры/сеттеры.
d) Контроллер и модель равноправны.

277. Почему важно отделять бизнес-логику от логики представления (UI)?
a) Чтобы можно было менять UI (например, CLI на GUI) без переписывания бизнес-логики, и наоборот; улучшает тестируемость.
b) Чтобы UI мог содержать бизнес-логику.
c) Чтобы упростить код, смешав всё.
d) Это не важно для маленьких проектов.

278. Как реализовать простую систему плагинов с помощью ООП в Python?
a) Определить базовый класс или протокол для плагинов, загружать классы из указанных модулей (например, через importlib) и регистрировать их.
b) Жёстко прописать все плагины в коде.
c) Использовать наследование от главного класса приложения.
d) Использовать глобальный реестр функций.

279. Зачем использовать шаблон Adapter при работе с чужими классами или библиотеками?
a) Чтобы привести их интерфейс к виду, ожидаемому вашей системой, не изменяя их код и не загрязняя свою систему чужими деталями.
b) Чтобы изменить исходный код библиотеки.
c) Чтобы избежать использования библиотеки.
d) Чтобы увеличить скорость работы библиотеки.

280. Как с помощью паттерна Strategy упростить добавление новых алгоритмов в программу?
a) Выделить алгоритмы в отдельные классы, реализующие общий интерфейс, и позволить клиентскому коду выбирать нужный алгоритм динамически.
b) Писать все алгоритмы в одном классе с условными операторами.
c) Копировать код алгоритма для каждого случая.
d) Использовать наследование для каждого алгоритма.

281. Почему шаблон Observer часто используется в GUI и сетевых приложениях?
a) Потому что он позволяет гибко связывать источник событий (например, нажатие

кнопки, приход сообщения) с произвольным количеством обработчиков.
b) Потому что он запрещает связывание объектов.
c) Потому что он замедляет реакцию на события.
d) Потому что он сложен в реализации.

282. Как проверить, что класс слишком много знает о других классах?
a) Если он часто создаёт экземпляры других классов, вызывает их внутренние методы, зависит от многих конкретных классов.
b) Если он ничего не знает о других классах.
c) Если он имеет только одного друга.
d) Если он абстрактный.

283. Что такое Law of Demeter (закон Деметры) и как он относится к ООП?
a) Принцип, согласно которому объект должен иметь ограниченное знание о других объектах и общаться только с «ближайшими друзьями».
b) Принцип, что объект должен знать о всех других объектах.
c) Принцип максимального зацепления.
d) Принцип, запрещающий использование методов.

284. Как применение закона Деметры влияет на дизайн методов и классов?
a) Методы должны вызывать только методы своего объекта, его атрибутов, параметров, создаваемых ими объектов, но не методы объектов, полученных через цепочку вызовов.
b) Методы могут вызывать любые методы любых объектов.
c) Методы не должны вызывать другие методы.
d) Методы должны быть статическими.

285. Что такое неизменяемый объект (immutable) и когда он полезен?
a) Объект, состояние которого нельзя изменить после создания. Полезен для многопоточности, кэширования, использования как ключа в словаре.
b) Объект, который можно свободно менять. Полезен для частых обновлений.
c) Объект без состояния.
d) Объект, который меняется только через специальные методы.

286. Как в Python сделать класс, объекты которого нельзя изменить после создания?
a) Использовать __slots__, не определять методы, изменяющие атрибуты, и, возможно, использовать @property без сеттера или frozen dataclass.
b) Объявить все атрибуты приватными.
c) Использовать только статические методы.
d) Запретить создание экземпляров.

287. Почему полезно делать некоторые объекты неизменяемыми при разработке многопоточных программ?
a) Они безопасны для разделения между потоками без блокировок, так как состояние не меняется.
b) Они быстрее изменяются.
c) Они занимают меньше памяти.
d) Они обязательны для использования в потоках.

288. Как реализовать протокол последовательности (sequence protocol) в своём

классе?
a) Реализовать методы __len__ и __getitem__ (желательно также __contains__, __iter__, __reversed__).
b) Реализовать только __iter__.
c) Реализовать только __getitem__.
d) Унаследовать от collections.abc.Sequence.

289. Как реализовать протокол отображения (mapping protocol) в своём классе?
a) Реализовать методы __getitem__, __iter__, __len__ (и, возможно, __contains__, __keys__, __items__, __values__).
b) Реализовать только __getitem__.
c) Реализовать только __setitem__.
d) Унаследовать от collections.abc.Mapping.

290. Какие преимущества даёт реализация специальных методов коллекций в пользовательских классах?
a) Позволяет использовать объекты в привычных конструкциях (for, in, len, [key]), интеграция со стандартной библиотекой.
b) Ускоряет выполнение программы.
c) Автоматически делает объект неизменяемым.
d) Позволяет избегать наследования.

291. Почему важно писать тесты для классов и их взаимодействия?
a) Чтобы убедиться в корректности работы, облегчить рефакторинг, документировать поведение и предотвращать регрессии.
b) Тесты замедляют разработку.
c) Код всегда работает правильно без тестов.
d) Тесты нужны только для библиотек.

292. Какие аспекты поведения класса нужно обязательно покрыть тестами?
a) Публичный интерфейс, граничные случаи, обработка ошибок, инварианты класса.
b) Только счастливый путь (happy path).
c) Приватные методы.
d) Только методы, которые часто меняются.

293. Как использование интерфейсов и абстрактных классов упрощает тестирование?
a) Позволяет легко подменять реальные реализации mock-объектами, изолируя тестируемый класс.
b) Усложняет создание тестовых двойников.
c) Требует тестировать только абстрактные классы.
d) Делает тесты зависимыми от конкретных реализаций.

294. Что такое mock-объекты и как они связаны с ООП?
a) Объекты-заглушки, имитирующие поведение реальных объектов; используются в тестах для изоляции и проверки взаимодействий между объектами.
b) Реальные объекты, используемые в продакшене.
c) Объекты, которые всегда выбрасывают исключения.
d) Объекты только для демонстрации.

295. Почему стоит минимизировать количество побочных эффектов в методах классов?
a) Чтобы методы были предсказуемыми, легче тестируемыми и переиспользуемыми; уменьшается связанность.
b) Побочные эффекты всегда полезны.
c) Они ускоряют выполнение.
d) Без побочных эффектов программа ничего не делает.

296. Как определить, какие зависимости класса стоит вынести во внешние объекты или сервисы?
a) Если зависимость не является неотъемлемой частью класса (например, логирование, отправка email, доступ к БД), её стоит вынести.
b) Все зависимости должны быть внутри класса.
c) Только зависимости от стандартной библиотеки.
d) Зависимости выносить не нужно.

297. По каким признакам можно судить о том, что класс слишком тесно связан с инфраструктурой (БД, сетью)?
a) Класс напрямую использует API конкретной БД или сетевой библиотеки, его трудно тестировать без реальной инфраструктуры.
b) Класс использует абстракции для доступа к данным.
c) Класс не имеет методов работы с данными.
d) Класс находится в отдельном модуле.

298. Почему полезно отделять доменную модель от слоёв доступа к данным?
a) Чтобы бизнес-логика не зависела от деталей хранения, улучшая гибкость и тестируемость.
b) Чтобы смешать логику и данные для скорости.
c) Чтобы уменьшить количество классов.
d) Это требование всех фреймворков.

299. Как ООП помогает управлять сложностью больших проектов на Python?
a) Позволяет структурировать код в виде модулей, классов и объектов, скрывать детали, переиспользовать код через наследование и композицию.
b) Делает код более запутанным.
c) Увеличивает объём кода.
d) Только через использование множественного наследования.

300. Какие приёмы ООП вы чаще всего используете для упрощения поддержки кода?
a) Инкапсуляция, выделение интерфейсов, соблюдение SRP, использование композиции, внедрение зависимостей.
b) Создание god-объектов.
c) Использование глобальных переменных.
d) Отказ от наследования.

Принята на заседании кафедры Информационных технологий и обработки медицинских данных ЦЦМиИИМ ИЦБиИИМ

от «23» ноября 2024 г., протокол № 5

Заведующий кафедрой		Лебедев Г.С.
Информационных технологий и обработки медицинских данных ЦЦМиИИМ ИЦБиИИМ	(подпись)	(фамилия, инициалы)