Тест:

1. Что такое функция и как ее объявить?

Ответ: Функция - это кусок кода, который решает конкретную задачу. Она может получать входные параметры (аргументы) и давать результат. Для создания функции в Python используется ключевое слово “def”.

1. Как создать функцию?

Ответ: Функция - это часть кода, которая делает определенную задачу. Она может иметь входные параметры (аргументы) и возвращать результат. Для создания функции в Python используется ключевое слово “def”, после которого идет имя функции, а потом в скобках указываются аргументы функции (если они есть). После этого ставится двоеточие и блок кода функции с пробелом.

Например, вот простая функция, которая выводит на экран приветствие:

def say\_hello():

print("Привет, мир!")

На выводе будет: Привет мир!

1. Как показать что созданная функция возвращает результат?

Ответ: Когда функция возвращает результат, это означает, что она дает какое-то значение, которое можно использовать в других местах программы. В Python, для того чтобы функция дала результат, используется ключевое слово “return”.

1. В чем разница между параметрами и аргументами функции?

Ответ: Параметры функции - это переменные, которые определяются в функции при ее создании. Они показывают, какие значения функция должна получить и обработать при ее работе. Параметры указываются в круглых скобках после имени функции. Аргументы функции - это значения, которые действительно передаются в функцию при ее использовании. Аргументы указываются в круглых скобках при использовании функции и соответствуют параметрам.

1. Как рекурсивные функции работают в Python?

Ответ: Рекурсивные функции в Python - это функции, которые вызывают саму себя в теле функции. Они используются для решения задач, которые могут быть разбиты на более простые или аналогичные подзадачи. Когда рекурсивная функция вызывает саму себя, она продолжает выполняться до тех пор, пока не достигнет базового случая, который обычно определяет, когда рекурсия будет остановлена. Когда базовый случай достигнут, функции начинают возвращаться в обратном порядке и выполнять остальную часть кода.

1. Что такое декораторы функций и как они используются? ( привести свой пример декоратора)

Декораторы функций - это специальные функции в Python, которые применяются к другим функциям в качестве аргументов и возвращают модифицированные функции. Они позволяют расширить или изменить функциональность других функций без изменения их кода. Декораторы оборачивают другие функции в один или несколько слоев дополнительного кода. Это может быть использовано для различных целей, таких как логирование, проверка аргументов, кеширование результатов и т.д.

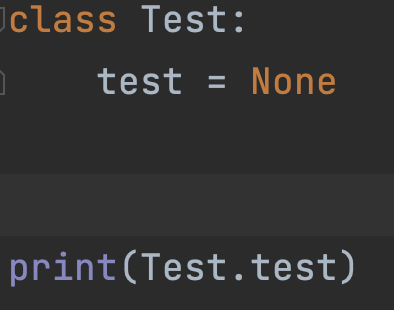
1. Как расшифровывается ООП?

Ответ: ООП расшифровывается как "Объектно-ориентированное программирование". ООП - это способ писать программы, в котором мы используем объекты как основные строительные блоки. Объекты - это сущности, которые имеют свои свойства (переменные) и действия (методы), которые они могут выполнять или взаимодействовать с другими объектами. Объекты создаются по образцу, называемому классом. Класс - это описание того, какой должен быть объект и что он может делать. ООП помогает нам делать код более четким и модульным, используя классы.

1. Что относится к основным принципам(парадигмам) ООП?

Ответ: ООП - это способ программирования, в котором мы работаем с объектами, которые имеют свои данные и методы. ООП основывается на четырех принципах: 1. Инкапсуляция: Это значит, что мы скрываем данные объекта от внешнего доступа и даем доступ к ним только через методы объекта. Это помогает нам контролировать данные и изменять код объекта без вреда для кода, который его использует. 2. Наследование: Это значит, что мы можем создавать новые классы на основе уже существующих классов. Класс-наследник получает все свойства и методы класса-родителя. Наследование помогает нам создавать иерархию классов и переопределять и дополнять функциональность родительского класса в классе-наследнике. 3. Полиморфизм: Это значит, что мы можем использовать объекты разных классов с одинаковым интерфейсом одинаково. Это означает, что объекты могут вести себя по-разному, в зависимости от того, к какому классу они относятся. Полиморфизм позволяет нам обращаться к объектам через их общий интерфейс, не зная их конкретного класса, и вызывать методы, которые будут иметь разную реализацию в каждом классе. 4. Абстракция: Это значит, что мы можем создавать абстрактные классы, которые определяют общий интерфейс для группы классов. Абстрактные классы могут содержать абстрактные методы, которые должны быть реализованы в классах-потомках. Абстракция позволяет нам скрыть детали реализации и фокусироваться на взаимодействии объектов через их общий интерфейс.

1. Что будет результатом программы и почему?



Ответ: В этом примере создан класс Test, в котором есть статическое поле test, которому присвоено значение None. Выражение print(Test.test) покажет значение этого статического поля, которое в этом случае равно None

1. Какой принцип ООП описывает следующее предложение? Этот принцип является способностью использовать общий интерфейс для нескольких форм (типов данных).

Ответ: Это принцип полиморфизма, который позволяет работать с объектами разных классов через одинаковый интерфейс.

1. Какой из перечисленных вариантов является верным объявлением private поля?

* private field = 0
* field = 0
* \_field = 0
* \_\_field = 0

Ответ: \_field = 0

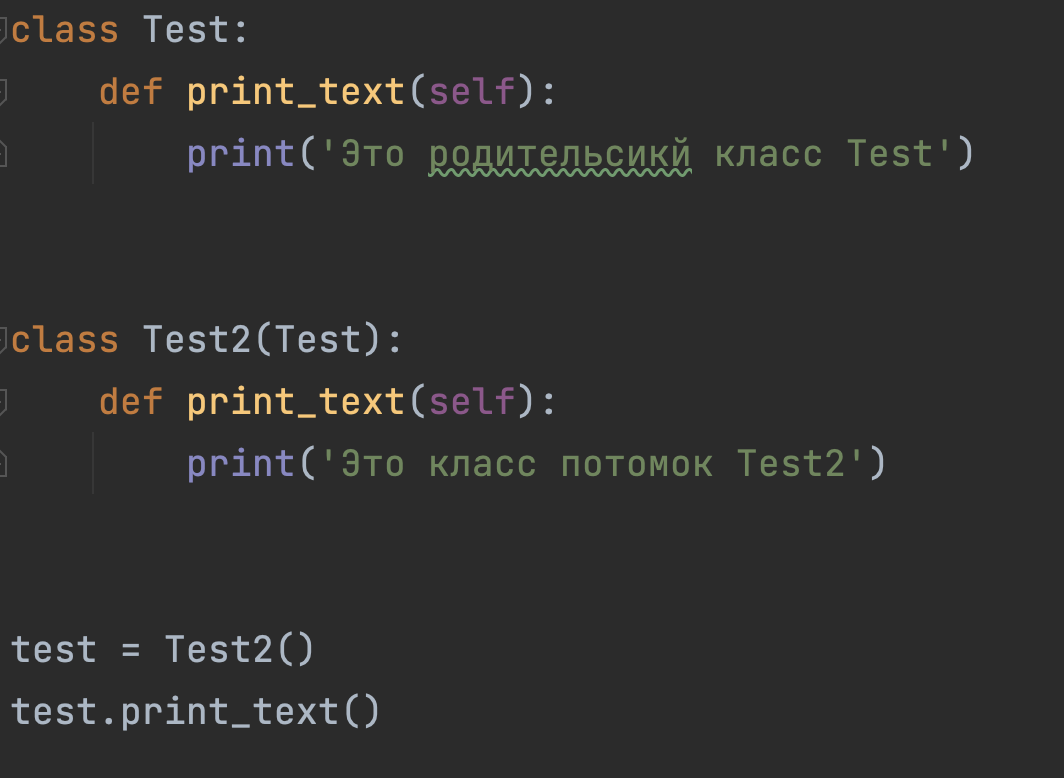
1. Как создать конструктор класса?

Ответ: Метод \_\_init\_\_ используется для создания конструктора класса в Python. Конструктор - это специальный метод, который вызывается при создании объекта класса. Он служит для инициализации полей объекта. Например: class MyClass: def init(self, initial\_value): self.my\_field = initial\_value

1. Как много конструкторов в классе может иметь Python?

Ответ: Python дает возможность создать только один конструктор (метод \_\_init\_\_) для каждого класса. Это значит, что в классе не может быть больше одного конструктора. Однако, можно использовать параметры по умолчанию в конструкторе для создания разных способов инициализации объекта.

1. Что будет результатом данной программы



Ответ: Данный код выведет "Это класс потомок Test2".

1. Какой параметр обязательно принимает в себя метод экземпляра?

* тип объекта
* сам экземпляр класса
* название класса
* таких аргументов нет

Ответ: сам экземпляр класса

1. Как использовать инкапсуляцию для защиты данных в классах?

Ответ: 1. Использование знака подчеркивания перед именем атрибута или метода (например, \_attribute или \_method). Это соглашение указывает на то, что атрибут или метод являются внутренними и не следует обращаться к ним извне класса. 2. Использование двух подчеркиваний перед именем атрибута или метода (например, \_\_attribute или \_\_method). Такой атрибут или метод становятся "приватными". Они могут быть доступны только внутри класса и не могут быть обращены напрямую извне.

Какая разница между абстрактными классами и интерфейсами в Python?

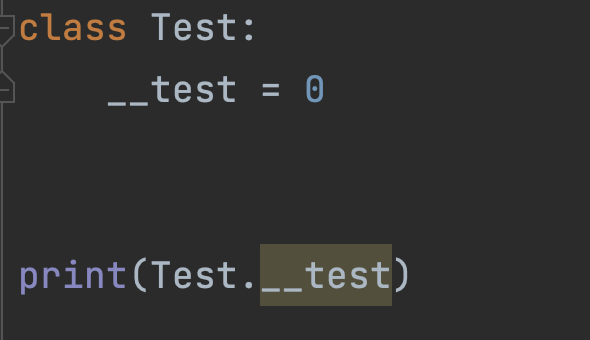
Ответ: В Python нет четкого понятия интерфейса, как в некоторых других языках программирования. В Python используются абстрактные классы и наследование для создания подобного эффекта. Но есть некоторые различия между абстрактными классами и интерфейсами: 1. Абстрактные классы могут иметь как абстрактные, так и обычные методы, а интерфейсы могут иметь только абстрактные методы. Абстрактный метод - это метод, который определен в базовом классе, но не имеет реализации и должен быть реализован в дочерних классах.

1. Как показать в классе что метод является абстрактым?

Ответ: В Python нет специального слова для создания абстрактных методов. Но можно использовать абстрактные базовые классы с помощью модуля abc (абстрактный базовый класс). Вот пример, как сделать метод абстрактным в Python: from abc import ABC, abstractmethod class MyClass(ABC): @abstractmethod def my\_method(self): pass class MyChildClass(MyClass): def my\_method(self): # реализация метода

Pass

1. Что будет результатом программы?



Ответ: При попытке обратиться к этому полю с помощью `Test.\_\_test` возникнет ошибка, так как поле \_\_test является приватным и недоступным для прямого доступа извне класса.

1. Что делает финализатор класса

Ответ: Финализатор (или метод `\_\_del\_\_`) в Python - это специальный метод, который вызывается при удалении объекта из памяти перед его освобождением. Финализатор позволяет выполнять определенные действия перед уничтожением объекта, например, закрытие файловых дескрипторов или освобождение ресурсов.

1. Распишите работу магических методов: new, call

Ответ: Магические методы в Python - это специальные методы, которые начинаются и заканчиваются двумя нижними подчеркиваниями, например, `\_\_new\_\_` и `\_\_call\_\_`. Эти методы позволяют определить специальное поведение для объектов в различных контекстах. Вот как работают магические методы `\_\_new\_\_` и `\_\_call\_\_`: 1. `\_\_new\_\_`: Этот метод вызывается при создании нового экземпляра класса и отвечает за выделение памяти для объекта. Он принимает класс в качестве первого аргумента, а затем любое количество аргументов, переданных при создании объекта. Метод `\_\_new\_\_` должен возвращать новый объект. Пример: class MyClass: def \_\_new\_\_(cls, \*args, \*\*kwargs): # Создание нового объекта instance = super().\_\_new\_\_(cls) # Кастомные действия при создании объекта return instance ``` 2. `\_\_call\_\_`: Этот метод вызывается при попытке вызвать объект как функцию. Он позволяет классам работать как вызываемые функции. Метод `\_\_call\_\_` принимает объект класса в качестве первого аргумента, а затем любое количество аргументов, переданных при вызове объекта в виде функции. Пример class MyClass: def \_\_call\_\_(self, \*args, \*\*kwargs): # Кастомные действия при вызове объекта pass

1. Если в классе определены два метода с одинаковыми именами и разными списками параметров, что будет результатом?

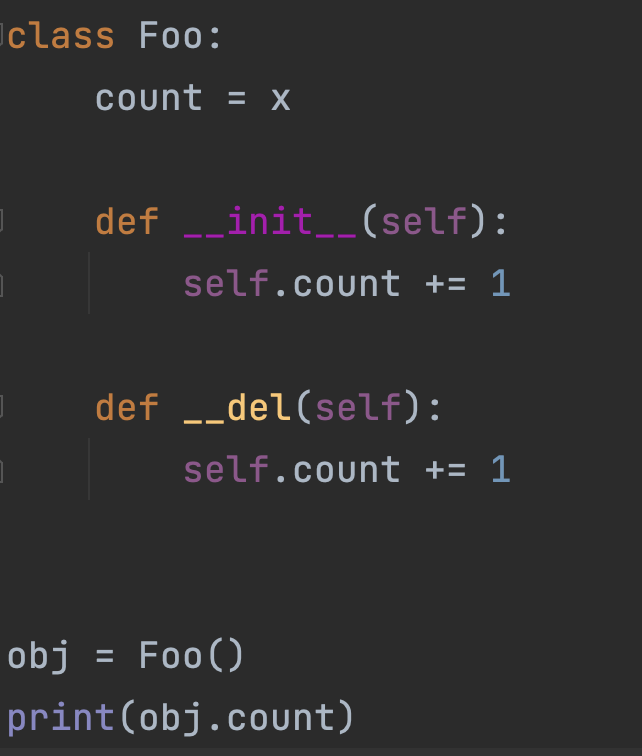
Ответ: Если в классе определены два метода с одинаковыми именами, но разными списками параметров, то это будет называться перегрузкой методов. В Python, в отличие от некоторых других языков программирования, перегрузка методов не поддерживается явно. При вызове метода, интерпретатор Python будет искать подходящий метод на основе переданных аргументов. Он будет сравнивать типы и количество аргументов, а также искать наиболее точное совпадение. Если в классе определены два метода с одинаковыми именами, и вызывается метод с конкретным списком аргументов, то Python выберет самый подходящий метод с наиболее точным совпадением параметров. Если не будет найдено точного совпадения, будет выброшено исключение `TypeError`, указывающее на неправильное количество или тип аргументов. Методы должны иметь различные списки параметров, чтобы их можно было различить, и это поможет интерпретатору Python выбрать правильный метод для вызова.

1. Значением поля класса по умолчанию может являться?

* значение переменной
* константа
* результат вызова функции
* возможность указания значений полей по умолчанию в Python не предусмотрена

Ответ: Все вышеперечисленное

1. Укажите результат выполнения скрипта и почему:



Ответ: В данном коде определен класс Foo с атрибутом `count`, который некорректно инициализируется значением `x`. Кроме того, в классе присутствуют неверно названные методы `\_\_del`, которые должны называться `\_\_del\_\_`. Кроме того, после создания объекта класса Foo (`obj = Foo()`), происходит попытка вывода значения атрибута `count` (`print(obj.count)`).

1. Как указать в программе что класс A наследуется от B?

Ответ: В Python для указания наследования класса A от класса B используется следующий синтаксис:

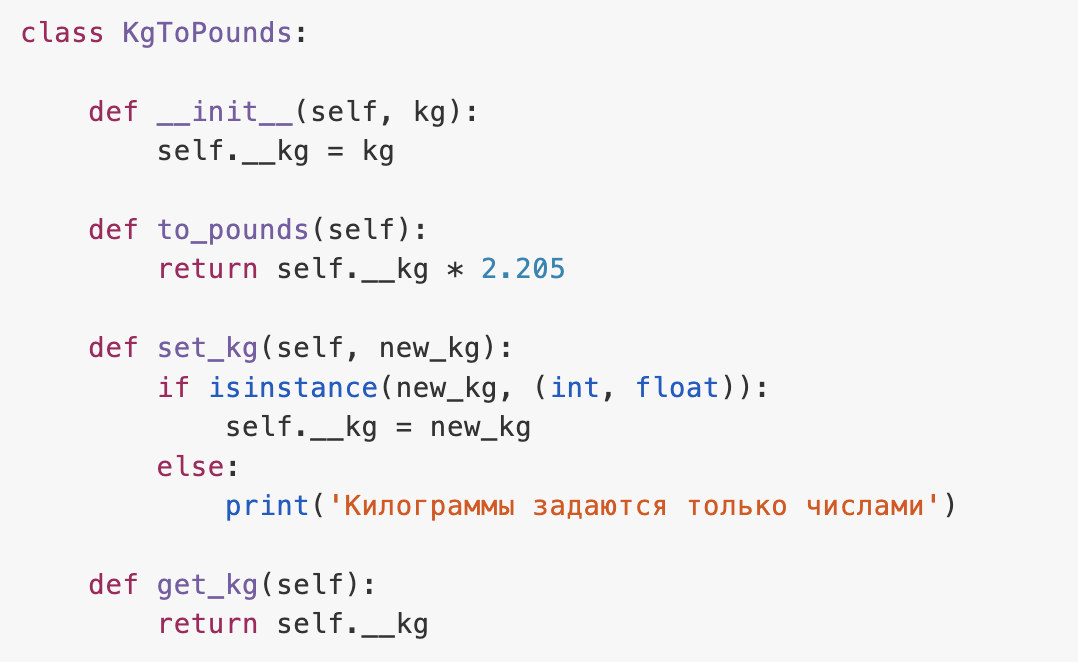
class A(B):

Практическая часть:

1. Требуется проверить, возможно ли из представленных отрезков условной длины сформировать треугольник. Для этого создайте класс TriangleChecker, принимающий только положительные числа. С помощью метода is\_triangle() возвращаются следующие значения (в зависимости от ситуации):

* Ура, можно построить треугольник!
* С отрицательными числами ничего не выйдет!
* Жаль, но из этого треугольник не сделать.

1. Создан класс KgToPounds с параметром kg, куда передается определенное количество килограмм, а с помощью метода to\_pounds() они переводятся в фунты. Чтобы закрыть доступ к переменной kg реализованы методы set\_kg() – для задания нового значения килограммов, get\_kg() – для вывода текущего значения кг. Из-за этого возникло неудобство: нам нужно теперь использовать эти 2 метода для задания и вывода значений. Помогите переделать класс с использованием свойств-декораторов @property. Код приведен ниже.



1. Задача

Есть Помидор со следующими характеристиками:

1. Индекс
2. Стадия зрелости(стадии: Отсутствует, Цветение, Зеленый, Красный)

Помидор может:

1. Расти (переходить на следующую стадию созревания)
2. Предоставлять информацию о своей зрелости

Есть Куст с помидорами, который:

1. Содержит список томатов, которые на ней растут

И может:

1. Расти вместе с томатами
2. Предоставлять информацию о зрелости всех томатов
3. Предоставлять урожай

И также есть Садовник, который имеет:

1. Имя
2. Растение, за которым он ухаживает

И может:

1. Ухаживать за растением
2. Собирать с него урожай

Задание:

##### **Класс Tomato**

1. Создайте класс Tomato
2. Создайте статический атрибут states, который будет содержать все стадии созревания помидора
3. Создайте метод \_\_init\_\_(), внутри которого будут определены два приватных атрибута: 1) \_index - передается параметром и 2) \_state - принимает первое значение из словаря states
4. Создайте метод grow(), который будет переводить томат на следующую стадию созревания
5. Создайте метод is\_ripe(), который будет проверять, что томат созрел (достиг последней стадии созревания)

##### **Класс TomatoBush**

1. Создайте класс TomatoBush
2. Определите метод \_\_init\_\_(), который будет принимать в качестве параметра количество томатов и на его основе будет создавать список объектов класса Tomato. Данный список будет храниться внутри атрибута tomatoes.
3. Создайте метод grow\_all(), который будет переводить все объекты из списка томатов на следующий этап созревания
4. Создайте метод all\_are\_ripe(), который будет возвращать True, если все томаты из списка стали спелыми
5. Создайте метод give\_away\_all(), который будет чистить список томатов после сбора урожая

##### **Класс Gardener**

1. Создайте класс Gardener
2. Создайте метод \_\_init\_\_(), внутри которого будут определены два атрибута: 1) name - передается параметром, является публичным и 2) \_plant - принимает объект класса Tomato, является приватным
3. Создайте метод work(), который заставляет садовника работать, что позволяет растению становиться более зрелым
4. Создайте метод harvest(), который проверяет, все ли плоды созрели. Если все - садовник собирает урожай. Если нет - метод печатает предупреждение.
5. Создайте статический метод knowledge\_base(), который выведет в консоль справку по садоводству.

##### **Тесты (main)**

1. Вызовите справку по садоводству
2. Создайте объекты классов TomatoBush и Gardener
3. Используя объект класса Gardener, поухаживайте за кустом с помидорами
4. Попробуйте собрать урожай
5. Если томаты еще не дозрели, продолжайте ухаживать за ними
6. Соберите урожай

4) **Задача: покупка дома**

**Описание классовой структуры**

Есть Человек, характеристиками которого являются:

1. Имя
2. Возраст
3. Наличие денег
4. Наличие собственного жилья

Человек может:

1. Предоставить информацию о себе
2. Заработать деньги
3. Купить дом

Также же есть Дом, к свойствам которого относятся:

1. Площадь
2. Стоимость

Для Дома можно:

1. Применить скидку на покупку

Также есть Небольшой Типовой Дом, обязательной площадью 40м2.

#### **Задание**

##### **Класс Human**

1. Создайте класс Human.
2. Определите для него два статических атрибута: default\_name и default\_age.
3. Создайте метод \_\_init\_\_(), который помимо self принимает еще два параметра: name и age. Для этих параметров задайте значения по умолчанию, используя атрибута default\_name и default\_age. В методе \_\_init\_\_() определите четыре атрибута: Публичные - name и age. Приватные - money и house.
4. Реализуйте справочный метод info(), который будет выводить поля name, age, house и money.
5. Реализуйте справочный статический метод default\_info(), который будет выводить статические атрибуты default\_name и default\_age.
6. Реализуйте приватный метод make\_deal(), который будет отвечать за техническую реализацию покупки дома: уменьшать количество денег на счету и присваивать ссылку на только что купленный дом. В качестве аргументов данный метод принимает объект дома и его цену.
7. Реализуйте метод earn\_money(), увеличивающий значение свойства money.
8. Реализуйте метод buy\_house(), который будет проверять, что у человека достаточно денег для покупки, и совершать сделку. Если денег слишком мало - нужно вывести предупреждение в консоль. Параметры метода: ссылка на дом и размер скидки

##### **Класс House**

1. Создайте класс House
2. Создайте метод \_\_init\_\_() и определите внутри него два атрибута: \_area и \_price. Свои начальные значения они получают из параметров метода \_\_init\_\_()
3. Создайте метод final\_price(), который принимает в качестве параметра размер скидки и возвращает цену с учетом данной скидки.

##### **Класс SmallHouse**

1. Создайте класс SmallHouse, унаследовав его функционал от класса House
2. Внутри класса SmallHouse переопределите метод \_\_init\_\_() так, чтобы он создавал объект с площадью 40м2

##### **Тесты**

1. Вызовите справочный метод default\_info() для класса Human()
2. Создайте объект класса Human
3. Выведите справочную информацию о созданном объекте (вызовите метод info()).
4. Создайте объект класса SmallHouse
5. Попробуйте купить созданный дом, убедитесь в получении предупреждения.
6. Поправьте финансовое положение объекта - вызовите метод earn\_money()
7. Снова попробуйте купить дом
8. Посмотрите, как изменилось состояние объекта класса Human

5) Создайте систему управления банковскими счетами, которая позволяет создавать, управлять и выполнять операции с банковскими счетами различных клиентов.

1. Реализуйте класс Client, представляющий клиента банка. Класс должен иметь атрибуты name (имя клиента) и id (уникальный идентификатор клиента).
2. Реализуйте класс BankAccount, представляющий банковский счет. Класс должен иметь атрибуты account\_number (номер счета), balance (баланс счета) и client (объект типа Client, которому принадлежит счет). Класс также должен иметь методы deposit(amount) и withdraw(amount), которые позволяют пополнить или снять деньги со счета.
3. Реализуйте класс Bank, представляющий банк. Класс должен иметь атрибут accounts, который является словарем, где ключами являются номера счетов, а значениями - объекты типа BankAccount. Класс также должен иметь методы create\_account(client, initial\_balance) для создания нового счета и get\_account(account\_number) для получения счета по его номеру.
4. Добавьте в класс Bank методы для выполнения переводов между счетами (transfer(sender\_account, receiver\_account, amount)), а также для получения общего баланса клиента (get\_total\_balance(client)), который включает сумму денег на всех его счетах.
5. Реализуйте обработку ошибок, например, недостаточно средств на счете при снятии денег или отсутствие счета при переводе.

6) Напишите программу с классом Student, в котором есть три атрибута: name, groupNumber и age. По умолчанию name = Ivan, age = 18, groupNumber = 10A. Необходимо создать пять методов: getName, getAge, getGroupNumber, setNameAge, setGroupNumber. Метод getName нужен для получения данных об имени конкретного студента, метод getAge нужен для получения данных о возрасте конкретного студента, vетод setGroupNumberнужен для получения данных о номере группы конкретного студента. Метод SetNameAge позволяет изменить данные атрибутов установленных по умолчанию, метод setGroupNumber позволяет изменить номер группы установленный по умолчанию. В программе необходимо создать пять экземпляров класса Student, установить им разные имена, возраст и номер группы.