Самоучитель по Python для начинающих. Часть 18: Основы ООП – инкапсуляция и наследование

Рассмотрим базовые понятия (классы, подклассы и методы) и приступим к изучению первых двух фундаментальных принципов объектно-ориентированного программирования. В конце статьи – 10 заданий, связанных с инкапсуляцией и наследованием.

Самоучитель по Python для начинающих. Часть 18: Основы ООП – инкапсуляция и наследование

Основы скрапинга и парсинга

Часть 19 →

Основы ООП – абстракция и полиморфизм

Объектно-ориентированное программирование (ООП) – это парадигма программирования, в которой для представления данных и для проведения операций над этими данными используются объекты.

Объекты, в свою очередь, являются экземплярами классов – с этой точки зрения классы можно назвать шаблонами для создания объектов определенного типа. Классы определяют:

структуру данных, которые характеризуют объект;

свойства (атрибуты) и статус (состояние) объекта;

операции, которые можно совершать с данными объекта (методы).

В этом примере класс Car (автомобиль) имеет атрибуты make, model, year (марка, модель, год выпуска):

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

Атрибуты – это свойства, характеристики объекта. Они определяют качества и состояние объекта. Атрибуты объекта перечисляют внутри\_\_init\_\_ метода класса – он вызывается каждый раз при создании экземпляра класса. Параметр self создает ссылку на экземпляр класса и позволяет получить доступ к атрибутам и методам объекта. Для создания экземпляра Car достаточно вызвать класс, передавая в скобках значения, соответствующие его атрибутам:

my\_car = Car("Toyota", "Corolla", 2023)

Теперь, когда атрибутам объекта присвоены значения, можно к ним обращаться – для этого используют выражение название\_объекта.атрибут:

print(f'Марка машины {my\_car.make},'

f'\nмодель {my\_car.model},'

f'\nгод выпуска - {my\_car.year}')

Результат:

Марка машины Toyota,

модель Corolla,

год выпуска – 2023

Car – пример простейшего класса: у него нет ни подклассов, ни методов, кроме обязательного \_\_init\_\_. Метод – это функция, которая определяет поведение объекта. Проиллюстрируем создание метода на примере класса WashingMachine – здесь методremaining\_warranty\_time() определяет срок истечения гарантии на стиральную машину:

import datetime

class WashingMachine:

def \_\_init\_\_(self, brand, model, purchase\_date, warranty\_length):

self.brand = brand

self.model = model

self.purchase\_date = purchase\_date

self.warranty\_length = warranty\_length

def remaining\_warranty\_time(self):

today = datetime.date.today()

warranty\_end\_date = self.purchase\_date + datetime.timedelta(days=self.warranty\_length)

remaining\_time = warranty\_end\_date - today

if remaining\_time.days < 0:

return "Срок действия гарантии истек."

else:

return "Срок действия гарантии истекает через {} дней.".format(remaining\_time.days)

# создаем объект стиральной машины

my\_washing\_machine = WashingMachine("LG", "FH4U2VCN2", datetime.date(2022, 5, 7), 1550)

# вызываем метод для проверки срока истечения гарантии

print(my\_washing\_machine.remaining\_warranty\_time())

Результат:

Срок действия гарантии истекает через 1218 дней.

Теперь рассмотрим чуть более сложный пример с подклассами и методами. Предположим, что нам нужно разработать CRM для автосалона. В ПО автосалона должен быть класс Vehicle (транспортное средство), который имеет набор атрибутов:

марка;

модель;

год выпуска;

стоимость.

Среди методов должна быть операция display\_info(), которая отображает информацию о конкретном транспортном средстве, а помимо классов, в ПО необходимо использовать подклассы.

Подкласс – это класс, который наследует все атрибуты и методы родительского класса (также известного как базовый класс или суперкласс), но при этом может иметь дополнительные, свои собственные, атрибуты и методы. Концепцию наследования мы подробнее разберем ниже.

В ПО для автосалона необходимо создать подкласс Car (легковой автомобиль), который наследует все атрибуты и методы класса Vehicle, и при этом имеет дополнительные атрибуты, например количество дверей и стиль кузова. Аналогично, мы можем создать подкласс Truck (грузовик), который наследует все атрибуты и методы класса Vehicle, и к тому же имеет свои атрибуты – длину кузова и тяговую мощность.

В итоге, взаимодействие классов, подклассов и методов будет выглядеть так:

class Vehicle:

def \_\_init\_\_(self, make, model, year, price):

self.make = make

self.model = model

self.year = year

self.price = price

def display\_info(self):

print(f"Марка: {self.make}"

f"\nМодель: {self.model}"

f"\nГод выпуска: {self.year}"

f"\nСтоимость: {self.price} руб")

class Car(Vehicle):

def \_\_init\_\_(self, make, model, year, price, num\_doors, body\_style):

super().\_\_init\_\_(make, model, year, price)

self.num\_doors = num\_doors

self.body\_style = body\_style

class Truck(Vehicle):

def \_\_init\_\_(self, make, model, year, price, bed\_length, towing\_capacity):

super().\_\_init\_\_(make, model, year, price)

self.bed\_length = bed\_length

self.towing\_capacity = towing\_capacity

Создадим экземпляры классов и вызовем метод display\_info() для вывода информации о них:

# создаем объект "легковой автомобиль"

car = Car("Toyota", "Camry", 2022, 2900000, 4, "седан")

# создаем объект "грузовик"

truck = Truck("Ford", "F-MAX", 2023, 6000000, "6162", "13 т")

# выводим информацию о легковом автомобиле и грузовике

car.display\_info()

truck.display\_info()

Результат:

Марка: Toyota

Модель: Camry

Год выпуска: 2022

Стоимость: 2900000 руб

Марка: Ford

Модель: F-MAX

Год выпуска: 2023

Стоимость: 6000000 руб

В этом примере используется встроенная функция super(), которая позволяет вызывать методы родительского суперкласса из подкласса. Этот прием позволяет переиспользовать методы и расширять их функциональность. В данном случае вызывается метод инициализации super().\_\_init\_\_, который позволяет применить атрибуты суперкласса к подклассу. При необходимости, помимо унаследованных, можно определить новые свойства, которые относятся только к конкретному подклассу.

Рассмотрим еще один пример – библиотечную программу для хранения информации о книгах и их статусах (есть в наличии, выдана абоненту, получена от абонента и так далее). Здесь класс Book определяет различные характеристики книги – title, author, ISBN, а также задает методы check\_out() и check\_in(), которые выдают / принимают книги, и сообщают о статусах:

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.title = title

self.author = author

self.isbn = isbn

self.checked\_out = False

def check\_out(self):

if self.checked\_out:

print("Книга находится у абонента.")

else:

self.checked\_out = True

print("Выдаем книгу абоненту.")

def check\_in(self):

if not self.checked\_out:

print("Книга в наличии.")

else:

self.checked\_out = False

print("Принимаем книгу в библиотеку.")

Создадим объект книги и проверим статусы:

# создаем объект книги

book1 = Book("Война и мир", "Л.Н. Толстой", "978-0743273565")

# выдаем книгу, проверяем статус

book1.check\_out()

# проверяем статус повторно

book1.check\_out()

# принимаем книгу от читателя

book1.check\_in()

# проверяем статус книги повторно

book1.check\_in()

Результат:

Выдаем книгу абоненту.

Книга находится у абонента.

Принимаем книгу в библиотеку.

Книга в наличии.

Классы, объекты, атрибуты и методы – самые простые, самые базовые понятия ООП. Эти базовые концепции, в свою очередь, лежат в основе фундаментальных принципов ООП.

Фундаментальные принципы ООП

ООП основывается на четырех фундаментальных принципах: инкапсуляции, наследовании, полиморфизме и абстракции.

**Инкапсуляция** – механизм сокрытия деталей реализации класса от других объектов. Достигается путем использования модификаторов доступа public, private и protected, которые соответствуют публичным, приватным и защищенным атрибутам.

Наследование – процесс создания нового класса на основе существующего класса. Новый класс, называемый подклассом или производным классом, наследует свойства и методы существующего класса, называемого суперклассом или базовым классом.

**Полиморфизм** – способность объектов принимать различные формы. В ООП полиморфизм позволяет рассматривать объекты разных классов так, как если бы они были объектами одного класса.

**Абстракция** – процесс определения существенных характеристик объекта и игнорирования несущественных характеристик. Это позволяет создавать абстрактные классы, которые определяют общие свойства и поведение группы объектов, не уточняя детали каждого объекта.

В этой статье мы рассмотрим на конкретных примерах первые две концепции, а в следующей – остальные.

**Инкапсуляция**

Сделаем атрибуты title, author и isbn класса Book приватными – теперь доступ к ним возможен только внутри класса:

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.\_\_title = title # приватный

self.\_\_author = author # приватный

self.\_\_isbn = isbn # приватный

Чтобы получить доступ к этим атрибутам извне класса, мы определяем методы getter и setter, которые обеспечивают контролируемый доступ к атрибутам:

def get\_title(self):

return self.\_\_title

def set\_title(self, title):

self.\_\_title = title

def get\_author(self):

return self.\_\_author

def set\_author(self, author):

self.\_\_author = author

def get\_isbn(self):

return self.\_\_isbn

def set\_isbn(self, isbn):

self.\_\_isbn = isbn

В этом примере методы get\_title(), get\_author() и get\_isbn() являются получающими методами (геттерами), которые позволяют нам получать значения приватных атрибутов извне класса. Методы set\_title(), set\_author() и set\_isbn() – устанавливающие методы (сеттеры), которые позволяют нам устанавливать значения частных атрибутов извне класса.

Создадим экземпляр объекта и попытаемся получить доступ к его названию с помощью обычного метода:

book1 = Book("Террор", "Дэн Симмонс", "558-0743553565")

# пытаемся получить доступ к приватному атрибуту

print(book1.\_\_title)

Результат – ошибка:

AttributeError: 'Book' object has no attribute '\_\_title'

Воспользуемся геттерами:

# получаем приватные атрибуты с помощью геттеров

print(book1.get\_title())

print(book1.get\_author())

print(book1.get\_isbn())

Результат:

Террор

Дэн Симмонс

558-0743553565

Изменим название с помощью сеттера и выведем результат:

# изменяем название с помощью сеттера

book1.set\_title("Эндимион")

print(book1.get\_title())

Результат:

Эндимион

Наследование

Для иллюстрации концепции наследования мы определим класс Publication, который имеет свойства, общие для всех публикаций – title, author и year, а также общий метод display():

class Publication:

def \_\_init\_\_(self, title, author, year):

self.title = title

self.author = author

self.year = year

def display(self):

print("Название:", self.title)

print("Автор:", self.author)

print("Год выпуска:", self.year)

Теперь создадим два подкласса Book и Magazine, которые наследуют все свойства и методы от класса Publication, и кроме того, имеют свои атрибуты. Подкласс Book добавляет свойство isbn и переопределяет метод display() для включения свойства isbn. Подкласс Magazine добавляет свойство issue\_number (номер выпуска) и переопределяет методdisplay()для включения свойства issue\_number:

class Book(Publication):

def \_\_init\_\_(self, title, author, year, isbn):

super().\_\_init\_\_(title, author, year)

self.isbn = isbn

def display(self):

super().display()

print("ISBN:", self.isbn)

class Magazine(Publication):

def \_\_init\_\_(self, title, author, year, issue\_number):

super().\_\_init\_\_(title, author, year)

self.issue\_number = issue\_number

def display(self):

super().display()

print("Номер выпуска:", self.issue\_number)

Теперь, если мы создадим экземпляр класса Book или класса Magazine, мы сможем вызвать метод display()для отображения свойств объекта. Сначала будет вызван метод display() подкласса (Book или Magazine), который в свою очередь вызовет метод display()суперкласса Publication с помощью функции super(). Это позволяет нам повторно использовать код суперкласса и избежать дублирования кода в подклассах:

# создаем объект книги

book1 = Book("Выбор", "Эдит Эгер", 2019, "112-3333273566")

# создаем объект выпуска журнала

magazine1 = Magazine("Вокруг света", "коллектив авторов", 2023, 3)

# выводим информацию о книге и номере журнала

book1.display()

magazine1.display()

Результат:

Название: Выбор

Автор: Эдит Эгер

Год выпуска: 2019

ISBN: 112-3333273566

Название: Вокруг света

Автор: коллектив авторов

Год выпуска: 2023

Номер выпуска: 3

Практика

**Задание 1**

Напишите класс MusicAlbum, у которого есть:

Атрибуты title, artist, release\_year, genre, tracklist (название, исполнитель, год выхода, жанр, список треков.

Метод play\_random\_track() для вывода случайного названия песни.

Пример использования:

album4 = MusicAlbum("Deutschland", "Rammstein", 2019, "Neue Deutsche Härte",

["Deutschland", "Radio", "Zeig dich", "Ausländer", "Sex",

"Puppe", "Was ich liebe", "Diamant", "Weit weg", "Tattoo",

"Hallomann"])

print("Название:", album4.title)

print("Исполнитель:", album4.artist)

print("Год:", album4.release\_year)

print("Жанр:", album4.genre)

print("Треки:", album4.tracklist)

album4.play\_random\_track()

Вывод:

Название: Deutschland

Исполнитель: Rammstein

Год: 2019

Жанр: Neue Deutsche Härte

Треки: ['Deutschland', 'Radio', 'Zeig dich', 'Ausländer', 'Sex', 'Puppe', 'Was ich liebe', 'Diamant', 'Weit weg', 'Tattoo', 'Hallomann']

Воспроизводится трек 7: Was ich liebe

Решение:

import random

class MusicAlbum:

def \_\_init\_\_(self, title, artist, release\_year, genre, tracklist):

self.title = title

self.artist = artist

self.release\_year = release\_year

self.genre = genre

self.tracklist = tracklist

def play\_track(self, track\_number):

print(f"Воспроизводится трек {track\_number}: {self.tracklist[track\_number - 1]}")

def play\_random\_track(self):

track\_number = random.randint(1, len(self.tracklist))

self.play\_track(track\_number)

**Задание 2**

Создайте класс Student, который имеет:

атрибуты name, age, grade, scores (имя, возраст, класс, оценки);

метод average\_score – для вычисления среднего балла успеваемости.

Пример использования:

student2 = Student("Егор Данилов", 12, "5B", [5, 4, 4, 5])

print("Имя:", student2.name)

print("Возраст:", student2.age)

print("Класс:", student2.grade)

print("Оценки:", \*student2.scores)

print("Средний балл:", student2.average\_score())

Вывод:

Имя: Егор Данилов

Возраст: 12

Класс: 5B

Оценки: 5 4 4 5

Средний балл: 4.5

Решение:

class Student:

def \_\_init\_\_(self, name, age, grade, scores):

self.name = name

self.age = age

self.grade = grade

self.scores = scores

def average\_score(self):

return sum(self.scores) / len(self.scores)

**Задание 3**

Напишите класс Recipe с двумя методами:

print\_ingredients(self) – выводит список продуктов, необходимых для приготовления блюда;

cook(self) – сообщает название выбранного рецепта и уведомляет о готовности блюда.

Пример использования:

# создаем рецепт спагетти болоньезе

spaghetti = Recipe("Спагетти болоньезе", ["Спагетти", "Фарш", "Томатный соус", "Лук", "Чеснок", "Соль"])

# печатаем список продуктов для рецепта спагетти

spaghetti.print\_ingredients()

# готовим спагетти

spaghetti.cook()

# создаем рецепт для кекса

cake = Recipe("Кекс", ["Мука", "Яйца", "Молоко", "Сахар", "Сливочное масло", "Соль", "Ванилин"])

# печатаем рецепт кекса

cake.print\_ingredients()

# готовим кекс

cake.cook()

Вывод:

# готовим кекс

cake.cook()

Вывод:

Ингредиенты для Спагетти болоньезе:

- Спагетти

- Фарш

- Томатный соус

- Лук

- Чеснок

- Соль

Сегодня мы готовим Спагетти болоньезе.

Выполняем инструкцию по приготовлению блюда Спагетти болоньезе...

Блюдо Спагетти болоньезе готово!

Ингредиенты для Кекс:

- Мука

- Яйца

- Молоко

- Сахар

- Сливочное масло

- Соль

- Ванилин

Сегодня мы готовим Кекс.

Выполняем инструкцию по приготовлению блюда Кекс...

Блюдо Кекс готово!

Решение:

class Recipe:

def \_\_init\_\_(self, name, ingredients):

self.name = name

self.ingredients = ingredients

def print\_ingredients(self):

print(f"Ингредиенты для {self.name}:")

for ingredient in self.ingredients:

print(f"- {ingredient}")

def cook(self):

print(f"Сегодня мы готовим {self.name}.")

print(f"Выполняем инструкцию по приготовлению блюда {self.name}...")

print(f"Блюдо {self.name} готово!")

**Задание 4**

Напишите суперкласс Publisher (издательство) и два подкласса BookPublisher (книжное издательство) и NewspaperPublisher (газетное издательство).

Родительский класс Publisher имеет два атрибута name и location (название, расположение) и два метода:

get\_info(self) – предоставляет информацию о названии и расположении издательства;

publish(self, message) – выводит информацию об издании, которое находится в печати.

Подклассы BookPublisher и NewspaperPublisher используют метод super().\_\_init\_\_(name, location) суперкласса для вывода информации о своих названии и расположении, и кроме того, имеют собственные атрибуты:

BookPublisher – num\_authors (количество авторов).

NewspaperPublisher– num\_pages (количество страниц в газете).

Пример использования:

publisher = Publisher("АБВГД Пресс", "Москва")

publisher.publish("Справочник писателя")

book\_publisher = BookPublisher("Важные Книги", "Самара", 52)

book\_publisher.publish("Приключения Чебурашки", "В.И. Пупкин")

newspaper\_publisher = NewspaperPublisher("Московские вести", "Москва", 12)

newspaper\_publisher.publish("Новая версия Midjourney будет платной")

Вывод:

Готовим "Справочник писателя" к публикации в АБВГД Пресс (Москва)

Передаем рукопись 'Приключения Чебурашки', написанную автором В.И. Пупкин в издательство Важные Книги (Самара)

Печатаем свежий номер со статьей "Новая версия Midjourney будет платной" на главной странице в издательстве Московские вести (Москва)

Решение:

class Publisher:

def \_\_init\_\_(self, name, location):

self.name = name

self.location = location

def get\_info(self):

return f"{self.name} ({self.location})"

def publish(self, message):

print(f'Готовим "{message}" к публикации в {self.get\_info()}')

class BookPublisher(Publisher):

def \_\_init\_\_(self, name, location, num\_authors):

super().\_\_init\_\_(name, location)

self.num\_authors = num\_authors

def publish(self, title, author):

print(f"Передаем рукопись '{title}', написанную автором {author} в издательство {self.get\_info()}")

class NewspaperPublisher(Publisher):

def \_\_init\_\_(self, name, location, num\_pages):

super().\_\_init\_\_(name, location)

self.num\_pages = num\_pages

def publish(self, headline):

print(f'Печатаем свежий номер со статьей "{headline}" на главной странице в издательстве {self.get\_info()}')

**Задание 5**

Создайте класс BankAccount, который имеет следующие свойства:

balance – приватный атрибут для хранения текущего баланса счета;

interest\_rate –приватный атрибут для процентной ставки;

transactions – приватный атрибут для списка всех операций, совершенных по счету.

Класс BankAccount должен иметь следующие методы:

deposit(amount) – добавляет сумму к балансу и регистрирует транзакцию;

withdraw(amount) – вычитает сумму из баланса и записывает транзакцию;

add\_interest()– добавляет проценты к счету на основе interest\_rate и записывает транзакцию;

history()– печатает список всех операций по счету.

Пример использования:

# создаем объект счета с балансом 100000 и процентом по вкладу 0.05

account = BankAccount(100000, 0.05)

# вносим 15 тысяч на счет

account.deposit(15000)

# снимаем 7500 рублей

account.withdraw(7500)

# начисляем проценты по вкладу

account.add\_interest()

# печатаем историю операций

account.history()

Вывод:

Внесение наличных на счет: 15000

Снятие наличных: 7500

Начислены проценты по вкладу: 5375.0

Решение:

class BankAccount:

def \_\_init\_\_(self, balance, interest\_rate):

self.\_\_balance = balance

self.\_\_interest\_rate = interest\_rate

self.\_\_transactions = []

def deposit(self, amount):

self.\_\_balance += amount

self.\_\_transactions.append(f"Внесение наличных на счет: {amount}")

def withdraw(self, amount):

if self.\_\_balance >= amount:

self.\_\_balance -= amount

self.\_\_transactions.append(f"Снятие наличных: {amount}")

else:

print("Недостаточно средств на счете")

def add\_interest(self):

interest = self.\_\_balance \* self.\_\_interest\_rate

self.\_\_balance += interest

self.\_\_transactions.append(f"Начислены проценты по вкладу: {interest}")

def history(self):

for transaction in self.\_\_transactions:

print(transaction)

**Задание 6**

Создайте класс Employee (сотрудник), который имеет следующие приватные свойства:

name – имя сотрудника;

age – возраст;

salary – оклад;

bonus – премия.

Класс Employee должен иметь следующие методы:

get\_name()– возвращает имя сотрудника;

get\_age()– возвращает возраст;

get\_salary() – возвращает зарплату сотрудника;

set\_bonus(bonus) – устанавливает свойство bonus;

get\_bonus() – возвращает бонус для сотрудника;

get\_total\_salary() – возвращает общую зарплату сотрудника (оклад + бонус).

Пример использования:

# создаем сотрудника с именем, возрастом и зарплатой

employee = Employee("Марина Арефьева", 30, 90000)

# устанавливаем бонус для сотрудника

employee.set\_bonus(15000)

# выводим имя, возраст, зарплату, бонус и общую зарплату сотрудника

print("Имя:", employee.get\_name())

print("Возраст:", employee.get\_age())

print("Зарплата:", employee.get\_salary())

print("Бонус:", employee.get\_bonus())

print("Итого начислено:", employee.get\_total\_salary())

Вывод:

Имя: Марина Арефьева

Возраст: 30

Зарплата: 90000

Бонус: 15000

Итого начислено: 105000

Решение:

Имя: Марина Арефьева

Возраст: 30

Зарплата: 90000

Бонус: 15000

Итого начислено: 105000

Решение:

class Employee:

def \_\_init\_\_(self, name, age, salary):

self.\_\_name = name

self.\_\_age = age

self.\_\_salary = salary

self.\_\_bonus = 0

def get\_name(self):

return self.\_\_name

def get\_age(self):

return self.\_\_age

def get\_salary(self):

return self.\_\_salary

def set\_bonus(self, bonus):

self.\_\_bonus = bonus

def get\_bonus(self):

return self.\_\_bonus

def get\_total\_salary(self):

return self.\_\_salary + self.\_\_bonus

**Задание 7**

Напишите класс Animal, обладающий свойствами name, species, legs, в которых хранятся данные о кличке, виде и количестве ног животного. Класс также должен иметь два метода – voice() и move(), которые сообщают о том, что животное подает голос и двигается.

Создайте два подкласса – Dog и Bird. Подкласс Dog имеет атрибут breed (порода) и метод bark(), который сообщает о том, что собака лает. Подкласс Bird обладает свойством wingspan (размах крыльев) и методом fly(), который уведомляет о полете птицы.

Пример использования:

dog = Dog("Геральт", "доберман", 4)

bird = Bird("Вася", "попугай", 2)

dog.voice()

bird.voice()

dog.move()

bird.move()

dog.bark()

bird.fly()

Вывод:

Геральт подает голос

Вася подает голос

Геральт дергает хвостом

Вася дергает хвостом

доберман Геральт лает

попугай Вася летaeт

Решение:

class Animal:

def \_\_init\_\_(self, name, species, legs):

self.name = name

self.species = species

self.legs = legs

def voice(self):

print(f"{self.name} подает голос")

def move(self):

print(f"{self.name} дергает хвостом")

class Dog(Animal):

def \_\_init\_\_(self, name, breed, legs):

super().\_\_init\_\_(name, breed, legs)

self.breed = breed

def bark(self):

print(f"{self.breed} {self.name} лает")

class Bird(Animal):

def \_\_init\_\_(self, name, species, wingspan):

super().\_\_init\_\_(name, species, 2)

self.wingspan = wingspan

def fly(self):

print(f"{self.species} {self.name} летaeт")

Задание 8

Создайте класс Shape (геометрическая фигура) со свойствами name и color (название и цвет). У этого класса должны быть три подкласса – Circle (окружность), Rectangle (прямоугольник), и Triangle (треугольник). Каждый подкласс наследует атрибут color и методdescribe() родительского класса Shape, и при этом имеет дополнительные свойства и методы:

Circle – атрибут radius и метод area() для вычисления площади.

Rectangle – атрибуты length и width, свой метод area().

Triangle – атрибуты base и height (основание и высота), собственный метод area().

Пример использования:

circle = Circle("красный", 5)

rectangle = Rectangle("синий", 3, 4)

triangle = Triangle("фиолетовый", 6, 8)

circle.describe()

rectangle.describe()

triangle.describe()

print(f"Площадь треугольника {triangle.area()}, окружности {circle.area()}, прямоугольника {rectangle.area()} см.")

Вывод:

Это геометрическая фигура, цвет - красный.

Это окружность. Радиус - 5 см, цвет - красный.

Это геометрическая фигура, цвет - синий.

Это синий прямоугольник. Длина - 3 см, ширина - 4 см.

Это геометрическая фигура, цвет - фиолетовый.

Это фиолетовый треугольник с основанием 6 см и высотой 8 см.

Площадь треугольника 24.0, окружности 78.5, прямоугольника 12 см.

Решение:

class Shape:

def \_\_init\_\_(self, color):

self.color = color

def describe(self):

print(f"Это геометрическая фигура, цвет - {self.color}.")

class Circle(Shape):

def \_\_init\_\_(self, color, radius):

super().\_\_init\_\_(color)

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

def describe(self):

super().describe()

print(f"Это окружность. Радиус - {self.radius} см, цвет - {self.color}.")

class Rectangle(Shape):

def \_\_init\_\_(self, color, length, width):

super().\_\_init\_\_(color)

self.length = length

self.width = width

def area(self):

return self.length \* self.width

def describe(self):

super().describe()

print(f"Это {self.color} прямоугольник. Длина - {self.length} см, ширина - {self.width} см.")

class Triangle(Shape):

def \_\_init\_\_(self, color, base, height):

super().\_\_init\_\_(color)

self.base = base

self.height = height

def area(self):

return 0.5 \* self.base \* self.height

def describe(self):

super().describe()

print(f"Это {self.color} треугольник с основанием {self.base} см и высотой {self.height} см.")

**Задание 9**

Для ПО кондитерской фабрики нужно написать родительский класс Candy (Конфеты). Этот класс имеет атрибуты name, price, weight (наименование, цена, вес). Подклассы Chocolate, Gummy, HardCandy (шоколад, жевательный мармелад, леденец) наследуют все атрибуты суперкласса Candy. Кроме того, у них есть и свои атрибуты:

Chocolate – cocoa\_percentage (процент содержания какао) и chocolate\_type (сорт шоколада).

Gummy – flavor и shape (вкус и форма).

HardCandy – flavor и filled (вкус и начинка).

Пример использования:

chocolate = Chocolate(name="Швейцарские луга", price=325.50, weight=220, cocoa\_percentage=40, chocolate\_type="молочный")

gummy = Gummy(name="Жуй-жуй", price=76.50, weight=50, flavor="вишня", shape="медведь")

hard\_candy = HardCandy(name="Crazy Фрукт", price=35.50, weight=25, flavor="манго", filled=True)

print("Шоколадные конфеты:")

print(f"Название конфет: {chocolate.name}")

print(f"Стоимость: {chocolate.price} руб")

print(f"Вес брутто: {chocolate.weight} г")

print(f"Процент содержания какао: {chocolate.cocoa\_percentage}")

print(f"Тип шоколада: {chocolate.chocolate\_type}")

print("\nМармелад жевательный:")

print(f"Название конфет: {gummy.name}")

print(f"Стоимость: {gummy.price} руб")

print(f"Вес брутто: {gummy.weight} г")

print(f"Вкус: {gummy.flavor}")

print(f"Форма: {gummy.shape}")

print("\nФруктовые леденцы:")

print(f"Название конфет: {hard\_candy.name}")

print(f"Стоимость: {hard\_candy.price} руб")

print(f"Вес брутто: {hard\_candy.weight} г")

print(f"Вкус: {hard\_candy.flavor}")

print(f"Начинка: {hard\_candy.filled}")

Вывод:

Шоколадные конфеты:

Название конфет: Швейцарские луга

Стоимость: 325.5 руб

Вес брутто: 220 г

Процент содержания какао: 40

Тип шоколада: молочный

Мармелад жевательный:

Название конфет: Жуй-жуй

Стоимость: 76.5 руб

Вес брутто: 50 г

Вкус: вишня

Форма: медведь

Фруктовые леденцы:

Название конфет: Crazy Фрукт

Стоимость: 35.5 руб

Вес брутто: 25 г

Вкус: манго

Начинка: True

Решение:

class Candy:

def \_\_init\_\_(self, name, price, weight):

self.name = name

self.price = price

self.weight = weight

class Chocolate(Candy):

def \_\_init\_\_(self, name, price, weight, cocoa\_percentage, chocolate\_type):

super().\_\_init\_\_(name, price, weight)

self.cocoa\_percentage = cocoa\_percentage

self.chocolate\_type = chocolate\_type

class Gummy(Candy):

def \_\_init\_\_(self, name, price, weight, flavor, shape):

super().\_\_init\_\_(name, price, weight)

self.flavor = flavor

self.shape = shape

class HardCandy(Candy):

def \_\_init\_\_(self, name, price, weight, flavor, filled):

super().\_\_init\_\_(name, price, weight)

self.flavor = flavor

self.filled = filled

**Задание 10**

Для военной игры-стратегии нужно написать класс Soldier (солдат). Класс имеет атрибуты name, rank и service\_number (имя, воинское звание, порядковый номер), причем звание и номер – приватные свойства.

Напишите методы для:

получения воинского звания;

подтверждения порядкового номера;

повышения в звании;

понижения в звании.

Кроме того, нужно создать декоратор для вывода информации о персонаже.

Пример использования:

soldier1 = Soldier("Иван Сусанин", "рядовой", "12345")

soldier1.get\_rank()

soldier1.promote()

soldier1.demote()

Вывод:

Создан новый игровой персонаж типа Soldier с атрибутами: {'name': 'Иван Сусанин', '\_Soldier\_\_rank': 'рядовой', '\_Soldier\_\_service\_number': '12345'}

Персонаж Иван Сусанин имеет звание рядовой

Иван Сусанин повышен в звании, он теперь ефрейтор

Иван Сусанин понижен в звании, он теперь рядовой

Решение:

RANKS = ["рядовой", "ефрейтор", "младший сержант", "сержант", "старший сержант",

"прапорщик", "старший прапорщик"]

def print\_info(cls):

class NewClass(cls):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

print(f"Создан новый игровой персонаж типа {cls.\_\_name\_\_} с атрибутами: {self.\_\_dict\_\_}")

def get\_rank(self):

print(f"Персонаж {self.name} имеет звание {self.\_Soldier\_\_rank}")

def promote(self):

super().promote()

print(f"{self.name} повышен в звании, он теперь {self.\_Soldier\_\_rank}")

def demote(self):

super().demote()

print(f"{self.name} понижен в звании, он теперь {self.\_Soldier\_\_rank}")

return NewClass

@print\_info

class Soldier:

def \_\_init\_\_(self, name, rank, service\_number):

self.name = name

self.\_\_rank = rank

self.\_\_service\_number = service\_number

def verify\_service\_number(self, service\_number):

return self.\_\_service\_number == service\_number

def promote(self):

if self.\_\_rank in RANKS[:-1]:

self.\_\_rank = RANKS[RANKS.index(self.\_\_rank) + 1]

def demote(self):

if self.\_\_rank in RANKS[1:]:

self.\_\_rank = RANKS[RANKS.index(self.\_\_rank) - 1]