

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Гайчук Дарья Дмитриевна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика
и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента
цифровых, роботехнических систем и
электроники института перспективной
инженерии

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: «Наследование и полиморфизм в языке Python»

Цель работы: приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

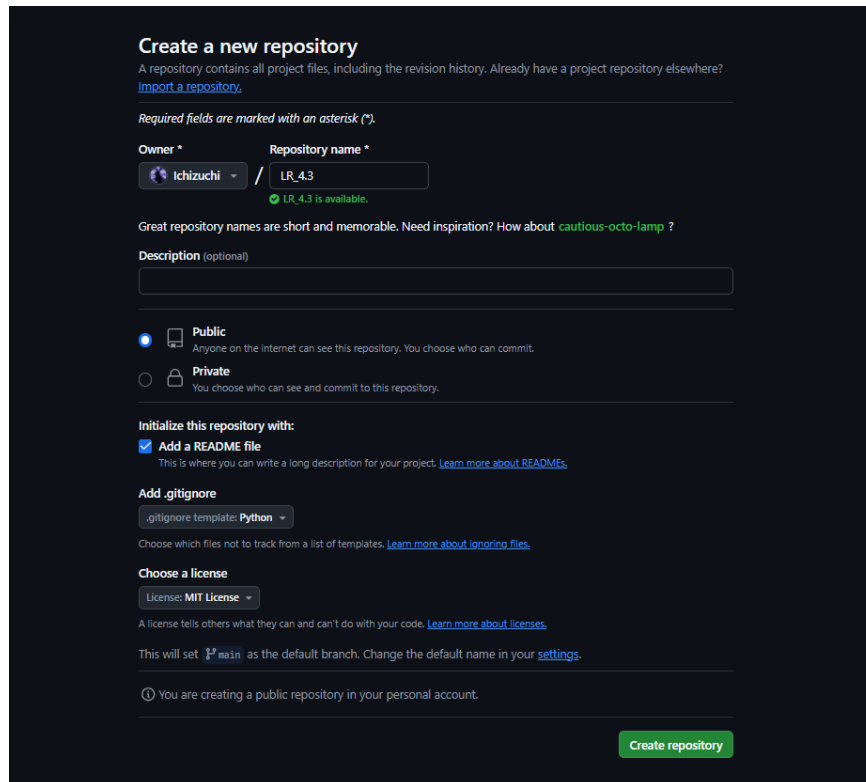


Рисунок 1. Создание репозитория

2. Клонировала репозиторий на свой компьютер.

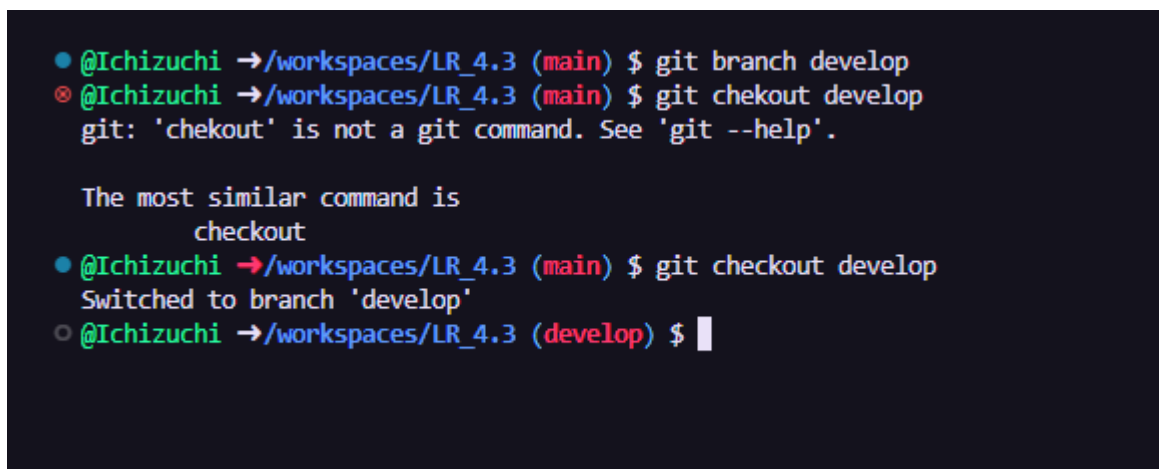


Рисунок 2. Модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

```
@Ichizuchi →/workspaces/LR_4.3 (develop) $ conda create -n myenv python=3.10
Retrieving notices: ...working... done
Channels:
- defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /opt/conda/envs/myenv

added / updated specs:
- python=3.10

The following packages will be downloaded:
```

package	build	
-----	-----	
_libgcc_mutex-0.1	main	3 KB
_openmp_mutex-5.1	1_gnu	21 KB
bzip2-1.0.8	h5eee18b_6	262 KB
ld_impl_linux-64-2.38	h1181459_1	654 KB
libffi-3.4.4	h6a678d5_1	141 KB
libgcc-ng-11.2.0	h1234567_1	5.3 MB

Рисунок 3. Создание виртуального окружения

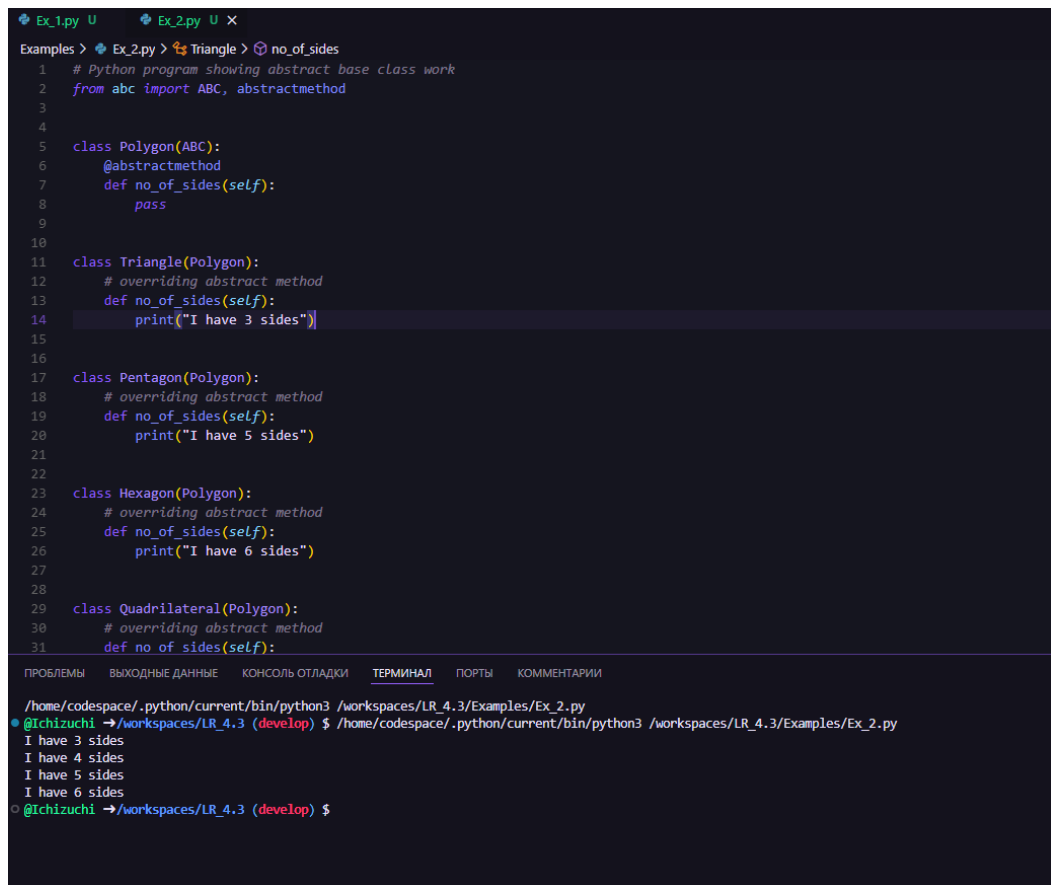
Выполнение примеров.

Пример №1. Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a — числитель, b — знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции: сложения add, вычитания sub, умножения mul, деления div, сравнения equal, greater, less. Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

```
4 class Rational:
5     def __init__(self, a=0, b=1):
6         a = int(a)
7         b = int(b)
8         if b == 0:
9             raise ValueError("Denominator cannot be zero.")
10        self._numerator = abs(a)
11        self._denominator = abs(b)
12        self._reduce()
13
14    def _reduce(self):
15        def gcd(a, b):
16            if a == 0:
17                return b
18            elif b == 0:
19                return a
20            elif a > b:
21                return gcd(a % b, b)
22            else:
23                return gcd(a, b % a)
24        c = gcd(self._numerator, self._denominator)
25        self._numerator //= c
26        self._denominator //= c
27
28    @property
29    def numerator(self):
30        return self._numerator
31
32    @property
33    def denominator(self):
34        return self._denominator
35
36    def read(self, prompt=None):
37
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ  КОММЕНТАРИИ
/home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Examples/Ex_1.py
@ichizuchi → /workspaces/LR_4.3 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Examples/Ex_1.py
3/4
Введите обыкновенную дробь: 3/6
1/2
5/4
1/4
3/8
2/3
@ichizuchi → /workspaces/LR_4.3 (develop) $
```

Рисунок 4. Выполнение примера №1

Пример №2. По умолчанию Python не предоставляет абстрактных классов. Python поставляется с модулем, который обеспечивает основу для определения абстрактных базовых классов (АВС), и имя этого модуля - АВС. АВС работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы. Метод становится абстрактным, если он украшен ключевым словом `abstractmethod`.



```
Ex_1.py U Ex_2.py U X
Examples > Ex_2.py > Triangle > no_of_sides
1 # Python program showing abstract base class work
2 from abc import ABC, abstractmethod
3
4
5 class Polygon(ABC):
6     @abstractmethod
7     def no_of_sides(self):
8         pass
9
10
11 class Triangle(Polygon):
12     # overriding abstract method
13     def no_of_sides(self):
14         print("I have 3 sides")
15
16
17 class Pentagon(Polygon):
18     # overriding abstract method
19     def no_of_sides(self):
20         print("I have 5 sides")
21
22
23 class Hexagon(Polygon):
24     # overriding abstract method
25     def no_of_sides(self):
26         print("I have 6 sides")
27
28
29 class Quadrilateral(Polygon):
30     # overriding abstract method
31     def no_of_sides(self):
32
ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ КОММЕНТАРИИ
/home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Examples/Ex_2.py
@Ichizuchi → /workspaces/LR_4.3 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Examples/Ex_2.py
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
@Ichizuchi → /workspaces/LR_4.3 (develop) $
```

Рисунок 5. Выполнение примера №2

Выполнение индивидуального задания.

Вариант 3

Задание №1. Разработать программу по следующему описанию. В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня. В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки. Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, увеличивается уровень. Отправьте одного из солдат

первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

```
Tasks > Game.py > ...
3 class Soldier:
4     ...
5
6     def __init__(self, team):
7         self.id = Soldier._id_counter
8         Soldier._id_counter += 1
9         self.team = team
10
11     def follow_hero(self, hero):
12         print(f"Солдат {self.id} следует за героем {hero.id} команды {hero.team}")
13
14 class Hero:
15     _id_counter = 1
16
17     def __init__(self, team):
18         self.id = Hero._id_counter
19         Hero._id_counter += 1
20         self.team = team
21         self.level = 1
22
23     def level_up(self):
24         self.level += 1
25         print(f"Герой {self.id} команды {self.team} повысил уровень до {self.level}")
26
27 hero_team_1 = Hero(team=1)
28 hero_team_2 = Hero(team=2)
29
30 soldiers_team_1 = []
31 soldiers_team_2 = []
32
33 for _ in range(10):
34     team = random.choice([1, 2])
35     soldier = Soldier(team=team)
36
37     if team == 1:
38         soldiers_team_1.append(soldier)
39     else:
40         soldiers_team_2.append(soldier)
41
42 print(f"Солдаты команды 1: {len(soldiers_team_1)}")
43 print(f"Солдаты команды 2: {len(soldiers_team_2)}")
44
45 if len(soldiers_team_1) > len(soldiers_team_2):
```

Рисунок 6. Код программы

```
@Ichizuchi →/workspaces/LR_4.3 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Tasks/Game.py
Солдаты команды 1: 6
Солдаты команды 2: 4
Герой 1 команды 1 повысил уровень до 2
Солдат 2 следует за героем 1 команды 1
@Ichizuchi →/workspaces/LR_4.3 (develop) $
```

Рисунок 7. Вывод результата

Задание №2. Составить программу с использованием иерархии классов. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанных классов. Создать класс `Liquid` (жидкость), имеющий поля названия и плотности. Определить методы переназначения и изменения плотности. Создать производный класс `Alcohol` (спирт), имеющий крепость. Определить методы пере назначения и изменения крепости.

```

Tasks > Task_1.py > Alcohol > get_info
1  class Liquid:
2      def __init__(self, name: str, density: float):
3          self.name = name
4          self.density = density
5
6      def set_name(self, new_name: str):
7          self.name = new_name
8
9      def set_density(self, new_density: float):
10         if new_density > 0:
11             self.density = new_density
12         else:
13             print("Плотность должна быть положительным числом.")
14
15     def get_info(self):
16         return f"Название: {self.name}, Плотность: {self.density} кг/м³"
17
18
19 class Alcohol(Liquid):
20     def __init__(self, name: str, density: float, strength: float):
21         super().__init__(name, density)
22         self.strength = strength
23
24     def set_strength(self, new_strength: float):
25         if 0 <= new_strength <= 100:
26             self.strength = new_strength
27         else:
28             print("Крепость должна быть в диапазоне от 0 до 100 процентов.")
29
30     def get_info(self):
31         base_info = super().get_info()
32         return f"{base_info}, Крепость: {self.strength}%"
33
34
35 if __name__ == '__main__':
36     water = Liquid(name="Вода", density=1000)
37     print(water.get_info())
38
39     water.set_name("Пресная вода")
40     water.set_density(998)
41     print(water.get_info())
42
43     ethanol = Alcohol(name="Этанол", density=789, strength=96)
44     print(ethanol.get_info())
45
46     ethanol.set_strength(70)
47     ethanol.set_density(800)
48     print(ethanol.get_info())

```

Рисунок 8. Код программы

```

● @Ichizuchi →/workspaces/LR_4.3 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Tasks/Task_1.py
Название: Вода, Плотность: 1000 кг/м³
Название: Пресная вода, Плотность: 998 кг/м³
Название: Этанол, Плотность: 789 кг/м³, Крепость: 96%
Название: Этанол, Плотность: 800 кг/м³, Крепость: 70%
○ @Ichizuchi →/workspaces/LR_4.3 (develop) $

```

Рисунок 9. Вывод индивидуального задания №1

Задание №3. Требуется реализовать абстрактный базовый класс, определив в нем абстрактные методы и свойства. Эти методы определяются в производных классах. В базовых классах должны быть объявлены абстрактные методы ввода/вывода, которые реализуются в производных

классах. Вызывающая программа должна продемонстрировать все варианты вызова переопределенных абстрактных методов. Написать функцию вывода, получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов. Создать абстрактный базовый класс Body (тело) с абстрактными функциями вычисления площади поверхности и объема. Создать производные классы: Parallelepiped (параллелепипед) и Ball (шар) со своими функциями площади поверхности и объема.

```
Tasks > Task_2.py > Body
5 class Body(ABC):
18     def display(self):
19         """Абстрактный метод для вывода информации о теле"""
20         pass
21
22
23 # Производный класс Parallelepiped (Параллелепипед)
24 class Parallelepiped(Body):
25     def __init__(self, length: float, width: float, height: float):
26         self.length = length
27         self.width = width
28         self.height = height
29
30     def surface_area(self) -> float:
31         """Вычисление площади поверхности параллелепипеда"""
32         return 2 * (self.length * self.width + self.length * self.height + self.width * self.height)
33
34     def volume(self) -> float:
35         """Вычисление объема параллелепипеда"""
36         return self.length * self.width * self.height
37
38     def display(self):
39         """Вывод информации о параллелепипеде"""
40         print(f"Параллелепипед: длина = {self.length}, ширина = {self.width}, высота = {self.height}")
41         print(f"Площадь поверхности = {self.surface_area():.2f}")
42         print(f"Объем = {self.volume():.2f}")
43
44
45 # Производный класс Ball (Шар)
46 class Ball(Body):
47     def __init__(self, radius: float):
48         self.radius = radius
49
50     def surface_area(self) -> float:
51         """Вычисление площади поверхности шара"""
52         return 4 * math.pi * self.radius ** 2
53
54     def volume(self) -> float:
55         """Вычисление объема шара"""
56         return (4 / 3) * math.pi * self.radius ** 3
57
58     def display(self):
59         """Вывод информации о шаре"""
60         print(f"Шар: радиус = {self.radius}")
61         print(f"Площадь поверхности = {self.surface_area():.2f}")
62         print(f"Объем = {self.volume():.2f}")
63
64
```

Рисунок 10. Код программы

```
@Ichizuchi → /workspaces/LR_4.3 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.3/Tasks/Task_2.py
Параллелепипед: длина = 3, ширина = 4, высота = 5
Площадь поверхности = 94.00
Объем = 60.00

Шар: радиус = 2
Площадь поверхности = 50.27
Объем = 33.51
@Ichizuchi → /workspaces/LR_4.3 (develop) $
```

Рисунок 11. Вывод индивидуального задания №2

```
@Ichizuchi → /workspaces/LR_4.3 (main) $ git merge develop
Already up to date.
@Ichizuchi → /workspaces/LR_4.3 (main) $ conda env export > environment.yml
@Ichizuchi → /workspaces/LR_4.3 (main) $ pip freeze > requirements.txt
@Ichizuchi → /workspaces/LR_4.3 (main) $ conda init
no change /opt/conda/condabin/conda
no change /opt/conda/bin/conda
```

Рисунок 12. Слияние веток

Ответы на контрольные вопросы

1. Что такое наследование и как оно реализовано в языке Python?

Наследование — это механизм, который позволяет одному классу (производному или дочернему) получать свойства и методы другого класса (базового или родительского). Это упрощает повторное использование кода и позволяет расширять его функциональность.

В Python наследование реализовано путем указания родительского класса в скобках после имени дочернего класса.

2. Что такое полиморфизм и как он реализован в языке Python?

Полиморфизм — это возможность объектов разных классов иметь методы с одинаковыми именами, которые могут вызываться единым способом, независимо от типа объекта. В Python полиморфизм реализован через методы с одинаковыми именами в разных классах, при этом вызов метода определяется во время выполнения программы.

3. Что такое "утиная" типизация в языке программирования Python?

"Утиная типизация" — это концепция, согласно которой объект определяет свою принадлежность к какому-либо типу не через наследование или явное указание типа, а через наличие необходимых методов и атрибутов. В Python это означает, что если объект "ведет себя" как ожидаемый тип (например, имеет метод «`sound()`»), то его можно использовать как объект этого типа, независимо от его реальной принадлежности к классу.

4. Каково назначение модуля «abc» языка программирования Python?

Модуль «abc» (Abstract Base Classes) позволяет создавать абстрактные базовые классы в Python. Он предоставляет инструменты для определения абстрактных методов и свойств, которые должны быть реализованы в производных классах. Это помогает создать интерфейсы и заставляет производные классы реализовывать конкретные методы.

Пример использования модуля «abc»:

```
from abc import ABC, abstractmethod

class Animal(ABC):

    @abstractmethod

    def sound(self):

        pass
```

5. Как сделать некоторый метод класса абстрактным?

Чтобы сделать метод класса абстрактным, нужно импортировать модуль «abc», а затем применить декоратор «@abstractmethod» к методу в абстрактном базовом классе. Абстрактные методы должны быть переопределены в производных классах.

6. Как сделать некоторое свойство класса абстрактным?

Для создания абстрактного свойства в классе нужно использовать декораторы «@property» и «@abstractmethod» из модуля «abc».

7. Каково назначение функции «isinstance»?

Функция «isinstance()» проверяет, является ли объект экземпляром указанного класса или его производного класса. Это используется для проверки типов объектов во время выполнения программы.

```
isinstance(object, class_or_tuple)
```

Вывод: в ходе работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.