

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

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.3**  
**дисциплины «Объектно-ориентированное программирование»**

Выполнил:  
Магомедов Имран Борисович  
3 курс, группа «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А., доцент департамента  
цифровых, робототехнических систем и  
электроники

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

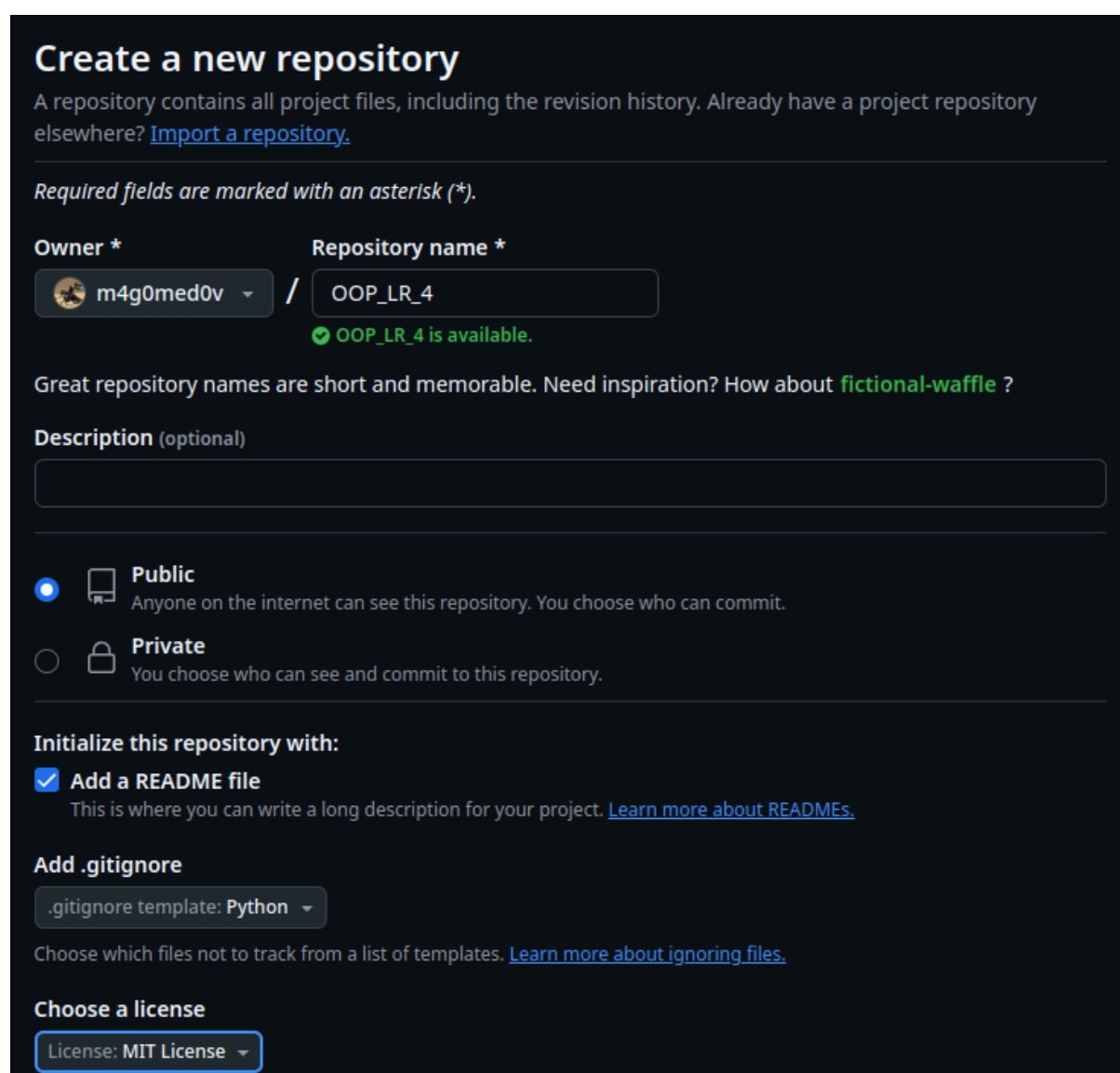
Ставрополь, 2024 г.

**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.

### Методика выполнения работы


1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

**Owner \***  m4g0med0v / **Repository name \***

✔ OOP\_LR\_4 is available.

Great repository names are short and memorable. Need inspiration? How about **fictional-waffle** ?

**Description (optional)**

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

3. Выполните клонирование созданного репозитория.

```

~/ncfu ➤ git clone https://github.com/m4g0med0v/00P_LR_4.git
Клонирование в «00P_LR_4»...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 21 (delta 3), reused 16 (delta 2), pack-reused 0 (from 0)
Получение объектов: 100% (21/21), 9.48 КиБ | 64.00 КиБ/с, готово.
Определение изменений: 100% (3/3), готово.

```

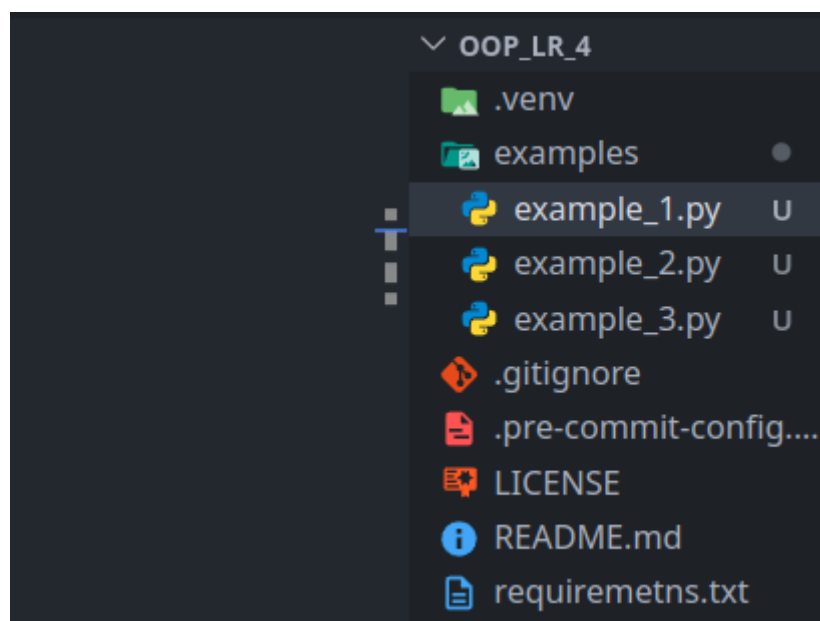
4. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```

~/ncfu/00P_LR_4 ➤ git switch develop
Переключились на ветку «develop»

```

5. Проработайте примеры лабораторной работы.



6. Решите задачу:

Разработайте 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.

```
18 import random
19
20
21 class Unit:
22     _id_counter = 1
23
24     def __init__(self, team):
25         self.id = Unit._id_counter
26         Unit._id_counter += 1
27         self.team = team
28
29 class Soldier(Unit):
30     def follow_hero(self, hero):
31         if isinstance(hero, Hero) and hero.team == self.team:
32             print(f"Солдат {self.id} следует за героем {hero.id}")
33
34 class Hero(Unit):
35     def __init__(self, team):
36         super().__init__(team)
37         self.level = 1
38
39     def increase_level(self):
40         self.level += 1
41         print(f"Герой {self.id} повысил уровень до {self.level}")
42
```

```
~/ncfu/00P_LR_4 git develop ?2 python src/task_1.py
Солдаты команды 1: 45
Солдаты команды 2: 55
Герой 2 повысил уровень до 2
Солдат 3 следует за героем 1
Герой: 1, Солдат: 3
```

7. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

**Индивидуальное задание №1.** Составить программу с использованием иерархии классов. Номер варианта необходимо получить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанных классов.

Создать класс `Triangle` с полями-сторонами. Определить методы изменения сторон, вычисления углов, вычисления периметра. Создать производный класс `RightAngled` (прямоугольный), имеющий поле площади. Определить метод вычисления площади.

```
15 import math
16
17 class Triangle:
18     def __init__(self, a, b, c):
19         self.a = a
20         self.b = b
21         self.c = c
22         if not self._is_valid_triangle():
23             raise ValueError("Стороны не образуют треугольник")
24
25     def set_sides(self, a, b, c):
26         """Изменяет длины сторон треугольника."""
27         self.a = a
28         self.b = b
29         self.c = c
30         if not self._is_valid_triangle():
31             raise ValueError("Стороны не образуют треугольник")
32
33     def _is_valid_triangle(self):
34         """Проверяет, образуют ли заданные стороны треугольник."""
35         return ((self.a + self.b > self.c)
36                 and (self.a + self.c > self.b)
37                 and (self.b + self.c > self.a)
38                 )
39
40     def perimeter(self):
41         """Вычисляет периметр треугольника."""
42         return self.a + self.b + self.c
43
44     def angles(self):
45         """Вычисляет углы треугольника в градусах."""
46         angle_A = math.degrees(math.acos((self.b**2 + self.c**2 - self.a**2)
47                                           / (2 * self.b * self.c)))
48         angle_B = math.degrees(math.acos((self.a**2 + self.c**2 - self.b**2)
49                                           / (2 * self.a * self.c)))
50         angle_C = 180 - angle_A - angle_B
51         return angle_A, angle_B, angle_C
```

```

54 class RightAngled(Triangle):
55     def __init__(self, a, b):
56         """
57         Инициализирует прямоугольный треугольник с катетами a и b,
58         вычисляет гипотенузу.
59         """
60         c = math.sqrt(a**2 + b**2)
61         super().__init__(a, b, c)
62         self.area = self.calculate_area()
63
64     def calculate_area(self):
65         """Вычисляет площадь прямоугольного треугольника."""
66         return 0.5 * self.a * self.b

```

```

~/ncfu/OOP_LR_4 git develop ?2 python src/individual_task_1.py
Периметр треугольника: 12
Углы треугольника: (36.86989764584401, 53.13010235415599, 90.0)
Периметр прямоугольного треугольника: 12.0
Площадь прямоугольного треугольника: 6.0
Углы прямоугольного треугольника: (36.86989764584401, 53.13010235415599, 90.0)

```

**Индивидуальное задание №2.** В следующих заданиях 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 Function (функция) с виртуальными методами вычисления значения функции  $y = f(x)$  в заданной точке  $x$  и вывода результата на экран. Определить производные классы Ellipse (эллипс), Hyperbola (гипербола) с собственными функциями вычисления  $y$  в зависимости от входного параметра  $x$ . Уравнение эллипса:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1; \text{ гиперболы: } \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1.$$

```

16 from abc import ABC, abstractmethod
17 import math
18
19 class Function(ABC):
20     @abstractmethod
21     def calculate_y(self, x):
22         """Вычисляет значение функции  $y=f(x)$  в точке  $x$ ."""
23         pass
24
25     @abstractmethod
26     def display_result(self, x):
27         """Выводит результат на экран."""
28         pass
29
30
31 class Ellipse(Function):
32     def __init__(self, a, b):
33         self.a = a
34         self.b = b
35
36     def calculate_y(self, x):
37         """Вычисляет значение  $y$  для эллипса в точке  $x$ , если возможно."""
38         if abs(x) > self.a:
39             raise ValueError("Значение  $x$  выходит за пределы допустимых значений для эллипса")
40         y = self.b * math.sqrt(1 - (x**2 / self.a**2))
41         return y
42
43     def display_result(self, x):
44         """Выводит значение  $y$  для эллипса при заданном  $x$ ."""
45         try:
46             y = self.calculate_y(x)
47             print(f"Для эллипса с параметрами  $a={self.a}$ ,  $b={self.b}$  и  $x={x}$ ,  $y = {y}$ ")
48         except ValueError as e:
49             print(f"Ошибка: {e}")

```

```

52 class Hyperbola(Function):
53     def __init__(self, a, b):
54         self.a = a
55         self.b = b
56
57     def calculate_y(self, x):
58         """Вычисляет значение  $y$  для гиперболы в точке  $x$ ."""
59         y = self.b * math.sqrt((x**2 / self.a**2) - 1)
60         return y
61
62     def display_result(self, x):
63         """Выводит значение  $y$  для гиперболы при заданном  $x$ ."""
64         try:
65             y = self.calculate_y(x)
66             print(f"Для гиперболы с параметрами  $a={self.a}$ ,  $b={self.b}$  и  $x={x}$ ,  $y = {y}$ ")
67         except ValueError as e:
68             print(f"Ошибка: {e}")
69
70
71 # Функция для демонстрации виртуального вызова
72 def show_function_result(function_obj, x):
73     """Вызывает метод display_result для объекта базового класса Function."""
74     function_obj.display_result(x)
75

```

```
~/ncfu/00P_LR_4 git develop ?2 python src/individual_task_2.py
Вызов для эллипса:
Для эллипса с параметрами a=5, b=3 и x=3, y = 2.4000000000000004
Ошибка: Значение x выходит за пределы допустимых значений для эллипса
show_function_result(hyperbola, x=3)
Вызов для гиперболы:
Для гиперболы с параметрами a=5, b=3 и x=6, y = 1.9899748742132397
Ошибка: math domain error
```

8. Зафиксируйте сделанные изменения в репозитории.

```
~/ncfu/00P_LR_4 git develop +6 git commit -m "added examples, task and individual tasks"
check yaml.....(no files to check)Skipped
fix end of files.....Passed
trim trailing whitespace.....Passed
ruff.....Passed
ruff-format.....Passed
[develop 9f9c342] added examples, task and individual tasks
6 files changed, 481 insertions(+)
create mode 100644 examples/example_1.py
create mode 100644 examples/example_2.py
create mode 100644 examples/example_3.py
create mode 100644 src/individual_task_1.py
create mode 100644 src/individual_task_2.py
create mode 100644 src/task_1.py
```

9. Выполните слияние ветки для разработки с веткой main / master.

```
~/ncfu/00P_LR_4 git develop git switch main
Переключились на ветку «main»
Эта ветка соответствует «origin/main».

~/ncfu/00P_LR_4 git main git merge develop
Обновление bbc7516..9f9c342
Fast-forward
 .pre-commit-config.yaml | 20 ++++++
 examples/example_1.py | 138 ++++++
 examples/example_2.py | 41 ++++++
 examples/example_3.py | 39 ++++++
 requiremetns.txt | 1 +
 src/individual_task_1.py | 83 ++++++
 src/individual_task_2.py | 101 ++++++
 src/task_1.py | 79 ++++++
 8 files changed, 502 insertions(+)
create mode 100644 .pre-commit-config.yaml
create mode 100644 examples/example_1.py
create mode 100644 examples/example_2.py
create mode 100644 examples/example_3.py
create mode 100644 requiremetns.txt
create mode 100644 src/individual_task_1.py
create mode 100644 src/individual_task_2.py
create mode 100644 src/task_1.py
```



10. Отправьте сделанные изменения на сервер GitHub.

### Контрольные вопросы

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

**Наследование** — это один из основных принципов объектно-ориентированного программирования (ООП), который позволяет создавать новый класс на основе уже существующего. Новый класс (наследник) получает свойства и методы базового (родительского) класса. Это позволяет повторно использовать код, избегать дублирования и расширять функциональность базового класса.

В Python наследование реализуется путем указания родительского класса в скобках при определении нового класса:

```
1 class Parent:
2     def method(self):
3         print("Method in Parent")
4
5 class Child(Parent):
6     def another_method(self):
7         print("Method in Child")
8
9 # Пример использования
10 child = Child()
11 child.method() # Output: "Method in Parent"
```

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

**Полиморфизм** — это возможность использовать объекты разных типов в одном интерфейсе. Проще говоря, это свойство, которое позволяет одному и тому же методу работать с разными типами объектов. Полиморфизм позволяет нам писать код, который может работать с объектами любого класса, если эти объекты реализуют нужные методы.

В Python полиморфизм реализуется автоматически благодаря динамической типизации. Например:

```

1 class Dog:
2     def sound(self):
3         return "Woof"
4
5 class Cat:
6     def sound(self):
7         return "Meow"
8
9 def animal_sound(animal):
10     print(animal.sound())
11
12 # Пример использования
13 dog = Dog()
14 cat = Cat()
15 animal_sound(dog) # Output: "Woof"
16 animal_sound(cat) # Output: "Meow"

```

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

**"Утиная" типизация** (Duck Typing) — это концепция в Python, согласно которой объект считается соответствующим определенному типу, если он имеет все необходимые методы и свойства, а не если он является экземпляром определенного класса.

Принцип можно описать так: *"Если что-то выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, утка."*

В Python это означает, что тип объекта определяется по его поведению (наличию определенных методов), а не по классу:

```

1 class Duck:
2     def quack(self):
3         return "Quack"
4
5 class Human:
6     def quack(self):
7         return "I'm pretending to be a duck"
8
9 def make_it_quack(entity):
10     print(entity.quack())
11
12 duck = Duck()
13 person = Human()
14 make_it_quack(duck) # Output: "Quack"
15 make_it_quack(person) # Output: "I'm pretending to be a duck"

```

4. Каково назначение модуля abc языка программирования Python?

Модуль abc (Abstract Base Classes) в Python предоставляет основу для создания **абстрактных классов** и методов. Абстрактный класс — это класс, который не может быть инстанцирован (нельзя создать его экземпляр напрямую) и служит шаблоном для создания других классов. Он позволяет определять методы, которые должны быть реализованы в дочерних классах.

Модуль abc позволяет создавать такие классы и методы, используя декоратор `@abstractmethod`:

```
1  from abc import ABC, abstractmethod
2
3  class Animal(ABC):
4      @abstractmethod
5      def sound(self):
6          pass
7
8  class Dog(Animal):
9      def sound(self):
10         return "Woof"
11
12  # animal = Animal() # Ошибка: нельзя создать экземпляр абстрактного класса
13  dog = Dog() # Можно, потому что реализованы все абстрактные методы
14
15
```

5. Как сделать некоторый метод класса абстрактным?

Чтобы сделать метод абстрактным, нужно использовать модуль abc и декоратор `@abstractmethod`. Такой метод объявляется в абстрактном базовом классе и требует реализации в дочернем классе:

```
1  from abc import ABC, abstractmethod
2
3  class MyBaseClass(ABC):
4      @abstractmethod
5      def my_abstract_method(self):
6          pass
```

В этом примере `my_abstract_method` — это абстрактный метод, который обязан быть переопределен в любом классе, который наследуется от `MyBaseClass`.

#### 6. Как сделать некоторое свойство класса абстрактным?

Для создания абстрактного свойства в Python используется комбинация `@property` и `@abstractmethod`:

```
1  from abc import ABC, abstractmethod
2
3  class MyBaseClass(ABC):
4      @property
5      @abstractmethod
6      def my_abstract_property(self):
7          pass
```

В этом примере `my_abstract_property` — это абстрактное свойство, которое должно быть определено в дочерних классах.

#### 7. Каково назначение функции `isinstance` ?

Функция `isinstance` проверяет, является ли объект экземпляром указанного класса или его производного класса. Она возвращает `True`, если объект принадлежит указанному классу или его наследнику, и `False` в противном случае.