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 «Наследование и полиморфизм в языке Python»**

**Вариант 9**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Иващенко Олег Андреевич  3 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информационные и вычислительные машины», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | | Руководитель практики:  Воронкин Роман Александрович, 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**: приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Пример 1. Реляционная (несократимая) дробь представляется парой целых чисел (a, b), где a – числитель, b – знаменатель. Создать класс Rational для работы с рациональным дробями. Обязательно должны быть реализованы операции:

* сложения add: (a, b) + (c, d) = (ad + bc, bd);
* вычитания sub: (a, b) – (c, d) = (ad – bc, bd);
* умножения mul: (a, b) x (c, d) = (ac, bd);
* деления div: (a, b) / (c, d) = (ad, bc);
* сравнения equal, greate, less.

Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

Листинг 1 – Код программы примера 1

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  class Rational:  def \_\_init\_\_(self, a=0, b=1):  a = int(a)  b = int(b)  if b == 0:  raise ValueError()    self.\_\_numerator = abs(a)  self.\_\_denominator = abs(b)  self.\_\_reduce()  # Сокращение дроби  def \_\_reduce(self):  # Функция для нахождения наибольшего общего делителя  def gcd(a, b):  if a == 0:  return b  elif b == 0:  return a  elif a >= b:  return gcd(a % b, b)  else:  return gcd(a, b % a)  c = gcd(self.\_\_numerator, self.\_\_denominator)  self.\_\_numerator //= c  self.\_\_denominator //= c  @property  def numerator(self):  return self.\_\_numerator  @property  def denominator(self):  return self.\_\_denominator  # Прочитать значение дроби с клавиатуры. Дробь вводится  # как a/b.  def read(self, prompt=None):  line = input() if prompt is None else input(prompt)  parts = list(map(int, line.split('/', maxsplit=1)))  if parts[1] == 0:  raise ValueError()  self.\_\_numerator = abs(parts[0])  self.\_\_denominator = abs(parts[1])  self.\_\_reduce()  # Вывести дробь на экран  def display(self):  print(f"{self.\_\_numerator}/{self.\_\_denominator}")  # Сложение обыкновенных дробей.  def add(self, rhs):  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator + \  self.denominator \* rhs.numerator  b = self.denominator \* rhs.denominator  return Rational(a, b)  else:  raise ValueError()  # Вычитание обыкновенных дробей.  def sub(self, rhs):  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator - \  self.denominator \* rhs.numerator  b = self.denominator \* rhs.denominator  return Rational(a, b)  else:  raise ValueError()  # Умножение обыкновенных дробей.  def mul(self, rhs):  if isinstance(rhs, Rational):  a = self.numerator \* rhs.numerator  b = self.denominator \* rhs.denominator  return Rational(a, b)  else:  raise ValueError()  # Деление обыкновенных дробей.  def div(self, rhs):  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator  b = self.denominator \* rhs.numerator  return Rational(a, b)  else:  raise ValueError()  # Отношение обыкновенных дробей.  def equals(self, rhs):  if isinstance(rhs, Rational):  return (self.numerator == rhs.numerator) and \  (self.denominator == rhs.denominator)  else:  return False  def greater(self, rhs):  if isinstance(rhs, Rational):  v1 = self.numerator / self.denominator  v2 = rhs.numerator / rhs.denominator  return v1 > v2  else:  return False  def less(self, rhs):  if isinstance(rhs, Rational):  v1 = self.numerator / self.denominator  v2 = rhs.numerator / rhs.denominator  return v1 < v2  else:  return False  if \_\_name\_\_ == '\_\_main\_\_':  r1 = Rational(3, 4)  r1.display()  r2 = Rational()  r2.read("Введите обыкновенную дробь: ")  r2.display()  r3 = r2.add(r1)  r3.display()  r4 = r2.sub(r1)  r4.display()  r5 = r2.mul(r1)  r5.display()  r6 = r2.div(r1)  r6.display() |

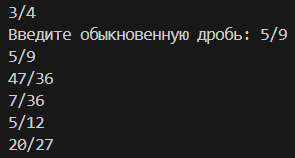


Рисунок 1 – Вывод программы примера 1

Пример 2. Используя абстрактные методы, создать несколько классов, наследуемые от абстрактного.

Листинг 3 – Код программы примера 2

|  |
| --- |
| # Python program showing  # abstract base class work  from abc import ABC, abstractmethod  class Polygon(ABC):  @abstractmethod  def noofsides(self):  pass  class Triangle(Polygon):  # overriding abstract method  def noofsides(self):  print("I have 3 sides")  class Pentagon(Polygon):  # overriding abstract method  def noofsides(self):  print("I have 5 sides")  class Hexagon(Polygon):  # overriding abstract method  def noofsides(self):  print("I have 6 sides")  class Quadrilateral(Polygon):  # overriding abstract method  def noofsides(self):  print("I have 4 sides")  # Driver code  R = Triangle()  R.noofsides()  K = Quadrilateral()  K.noofsides()  R = Pentagon()  R.noofsides()  K = Hexagon()  K.noofsides() |

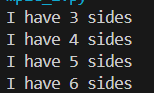


Рисунок 2 – Результат выполнения программы примера 2

Пример 3.

Листинг 3 – Код программы примера 3

|  |
| --- |
| # Python program showing  # abstract base class work  from abc import ABC, abstractmethod  class Animal(ABC):  def move(self):  pass  class Human(Animal):  def move(self):  print("I can walk and run")  class Snake(Animal):  def move(self):  print("I can crawl")  class Dog(Animal):  def move(self):  print("I can bark")  class Lion(Animal):  def move(self):  print("I can roar")  # Driver code  R = Human()  R.move()  K = Snake()  K.move()  R = Dog()  R.move()  K = Lion()  K.move() |

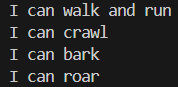


Рисунок 3 – Результат выполнения программы примера 3

Пример 4. Рассмотрим пример вызова метода, используя super().

Листинг 4 – Код программы примера 4

|  |
| --- |
| # Python program invoking a  # method using super()  import abc  from abc import ABC, abstractmethod  class R(ABC):  def rk(self):  print("Abstract Base Class")  class K(R):  def rk(self):  super().rk()  print("subclass")  # Driver code  r = K()  r.rk() |



Рисунок 4 – Результат выполнения программы примера 4

Общее задание. 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 создаётся 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.

Листинг 5 – Код программы общего задания

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import abc  import random  class Unit:  @abc.abstractmethod  def \_\_init\_\_():  pass  @abc.abstractmethod  def action():  pass  class Solder(Unit):  def \_\_init\_\_(self, id, command\_id):  self.id = id  self.command\_id = command\_id  def action(self, hero):  if isinstance(hero, Hero):  print("Солдат: Следую за Героем!")  else:  TypeError  class Hero(Unit):  def \_\_init\_\_(self, id, command\_id):  self.id = id  self.command\_id = command\_id  self.level = 1  def action(self):  self.level += 1  if \_\_name\_\_ == '\_\_main\_\_':  firstTeamUnitList = []  secondTeamUnitList = []  for i in range(30):  team = random.randint(0, 1)  if team == 0:  firstTeamUnitList.append(Solder(i, 0))  else:  secondTeamUnitList.append(Solder(i, 1))  hero1 = Hero(31, 0)  hero2 = Hero(32, 1)  firstCount = firstTeamUnitList.\_\_len\_\_()  secondCount = secondTeamUnitList.\_\_len\_\_()  if firstCount > secondCount:  hero1.action()  print("Система: Герой 1 повышает уровень")  elif secondCount > firstCount:  hero2.action()  print("Система: Герой 2 повышает уровень")  else:  print("Результат битвы: Ничья")  print("-----" \* 4, "[Команда 1]", "-----" \* 4)  print(f"Уровень героя: {hero1.level}")  print(f"Количество юнитов в команде: {firstTeamUnitList.\_\_len\_\_()}\n")  print("-----" \* 4, "[Команда 2]", "-----" \* 4)  print(f"Уровень героя: {hero2.level}")  print(f"Количество юнитов в команде: {secondTeamUnitList.\_\_len\_\_()}\n")  randomSolder = firstTeamUnitList[random.randint(0, firstCount)]  randomSolder.action(hero1)  print(f"ID Героя 1: {hero1.id}")  print("ID случайного солдата Героя 1: "  f"{randomSolder.id}") |

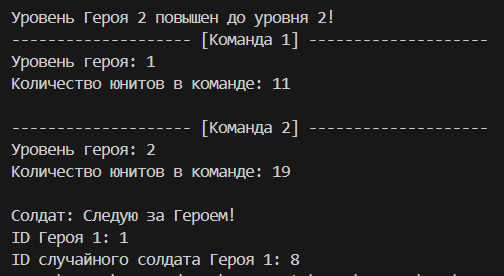


Рисунок 5 – Результат выполнения программы общего задания

Ответы на контрольные вопросы:

1. Что такое наследование как оно реализовано в языке Python?
2. Что такое полиморфизм и как он реализован в языке Python?
3. Что такое "утиная" типизация в языке программирования Python?
4. Каково назназначение модуля abc языка программирования Python?
5. Как сделать некоторый метод класса абстрактным?
6. Как сделать некоторое свойство класса абстрактным?
7. Каково назначение функции isinstance?

**Выводы**: В процессе выполнения лабораторной работы