

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 №3
дисциплины
«Объектно-ориентированное программирование»
Вариант 1

Выполнил:
Бабенко Артём Тимофеевич
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)

Проверил:
Ассистент департамента цифровых,
робототехнических систем и
электроники Богданов С.С

(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. Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a — числитель, b — знаменатель. Создать класс Rational для работы с рациональными дробями.

Код программы:

```
#!/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):

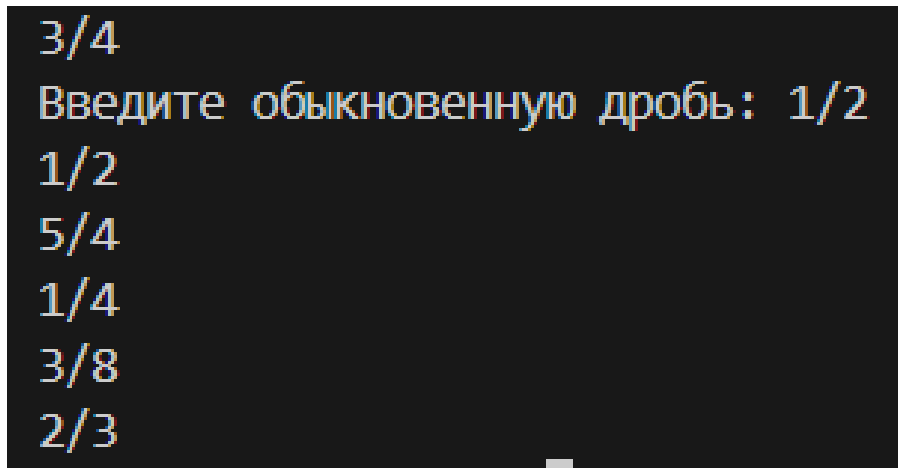
```

```

        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()

```

Результат работы программы:



```

3/4
Введите обыкновенную дробь: 1/2
1/2
5/4
1/4
3/8
2/3

```

Рисунок 1 – Результат работы программы

Пример 2.

Код программы:

```

# 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")

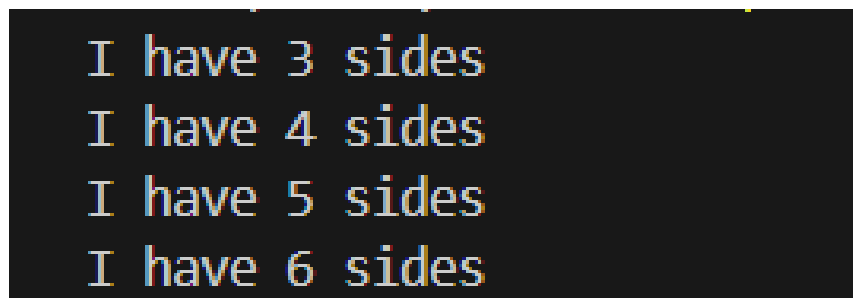
class Quadrilateral(Polygon):

    # 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()

```

Результат работы программы:



```

I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

```

Рисунок 2 – Результат работы программы

Пример 3.

Код программы:

```
# 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")

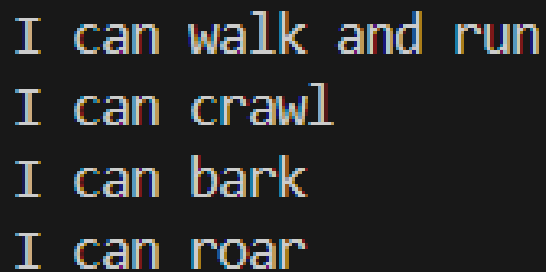
class Snake(Animal):
    def move(self):
        print("I can crawl")

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()
```

Результат работы программы:



```
I can walk and run
I can crawl
I can bark
I can roar
```

Рисунок 3 – Результат работы программы

Общее задание:

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.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from random import randint

class Warrior:
    def __init__(self, id="default", team="default"):
        self.__id = id
        self.__team = team
    @property
    def id(self):
        return self.__id
    @id.setter
    def id(self, value):
        self.__id = str(value)
    @property
    def team(self):
        return self.__team
    @team.setter
    def team(self, value):
        self.__team = str(value)

class Hero(Warrior):
```

```

def __init__(self, id="default", team="default", level=0, team_list = []):
    super().__init__(id, team)
    self.__level = level
    self.__team_list = team_list
@property
def level(self):
    return self.__level

@level.setter
def level(self, value):
    self.__level = int(value)

@property
def team_list(self):
    return self.__team_list

@team_list.setter
def team_list(self, value):
    self.__team_list = value

def level_up(self, n=1):
    self.level = self.level + n

class Soldier(Warrior):
    def __init__(self, id="default", team="default", hero = None):
        super().__init__(id, team)
        self.__hero = hero
    @property
    def hero(self):
        return self.__hero

    @hero.setter
    def hero(self, value):
        self.__hero = value

    def follow(self, hero):
        self.hero = hero
        self.team = hero.team

if __name__ == "__main__":
    first_hero = Hero("h001", "Red", 0)
    second_hero = Hero("h002", "Green", 0)
    red_soldiers = []
    green_soldiers = []
    for i in range(0, 16):
        temp_soldier = Soldier("00"+str(i))
        if randint(0, 1) == 0:
            red_soldiers.append(temp_soldier)
        else:

```



```

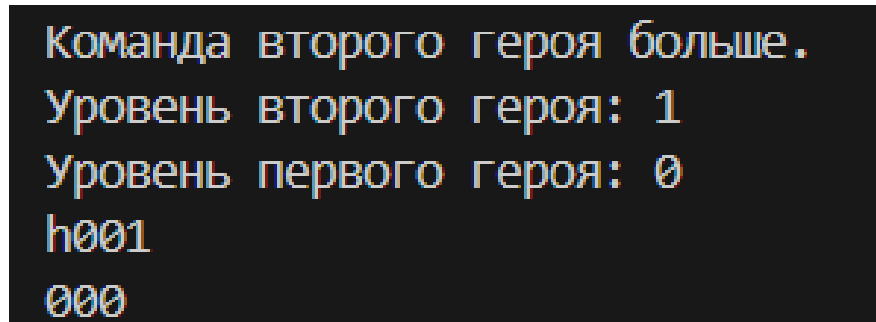
green_soldiers.append(temp_soldier)

first_hero.team_list = red_soldiers
second_hero.team_list = green_soldiers
if len(first_hero.team_list) < len(second_hero.team_list):
    second_hero.level += 1
    print(f"Команда второго героя больше.\nУровень второго героя:
{second_hero.level}\nУровень первого героя: {first_hero.level}")
elif len(first_hero.team_list) > len(second_hero.team_list):
    first_hero.level += 1
    print(f"Команда первого героя больше.\nУровень первого героя:
{first_hero.level}\nУровень второго героя: {second_hero.level}")
else:
    print("Размер команд одинаковый, уровни героев не изменились")

red_soldiers[0].follow(first_hero)
print(red_soldiers[0].hero.id)
print(red_soldiers[0].id)

```

Результат работы программы:



```

Команда второго героя больше.
Уровень второго героя: 1
Уровень первого героя: 0
h001
000

```

Рисунок 4 – Результат работы программы

Задание 1. Создать базовый класс Car (машина), характеризующийся торговой маркой (строка), числом цилиндров, мощностью. Определить методы переназначения и изменения мощности. Создать производный класс Lorry (грузовик), характеризующийся также грузоподъемностью кузова. Определить функции переназначения марки и изменения грузоподъемности.

Код программы:

```

# Базовый класс Car
class Car:
    def __init__(self, brand: str, cylinders: int, power: int):
        self._brand = brand # Торговая марка
        self._cylinders = cylinders # Число цилиндров
        self._power = power # Мощность

```

```

# Геттер для торговой марки
@property
def brand(self):
    return self._brand

# Сеттер для торговой марки
@brand.setter
def brand(self, new_brand: str):
    self._brand = new_brand

# Метод для изменения мощности
def change_power(self, new_power: int):
    if new_power > 0:
        self._power = new_power
    else:
        print("Мощность должна быть положительным числом.")

# Метод для вывода информации о машине
def display_info(self):
    print(f"Марка: {self._brand}, Цилиндры: {self._cylinders}, Мощность: {self._power} л.с.")

# Производный класс Lorry (грузовик)
class Lorry(Car):
    def __init__(self, brand: str, cylinders: int, power: int, capacity: float):
        super().__init__(brand, cylinders, power) # Вызов конструктора базового класса
        self._capacity = capacity # Грузоподъемность кузова

# Геттер для грузоподъемности
@property
def capacity(self):
    return self._capacity

# Сеттер для грузоподъемности

```

```

@capacity.setter
def capacity(self, new_capacity: float):
    if new_capacity > 0:
        self._capacity = new_capacity
    else:
        print("Грузоподъемность должна быть положительным числом.")

# Переопределение метода для вывода информации о грузовике
def display_info(self):
    super().display_info() # Вызов метода базового класса
    print(f"Грузоподъемность: {self._capacity} тонн")

# Демонстрация возможностей классов
if __name__ == '__main__':
    # Создание объекта базового класса Car
    car = Car("Toyota", 4, 150)
    print("Информация о машине:")
    car.display_info()

    # Изменение мощности машины
    car.change_power(180)
    print("\nПосле изменения мощности машины:")
    car.display_info()

    # Переназначение марки машины
    car.brand = "Honda"
    print("\nПосле изменения марки машины:")
    car.display_info()

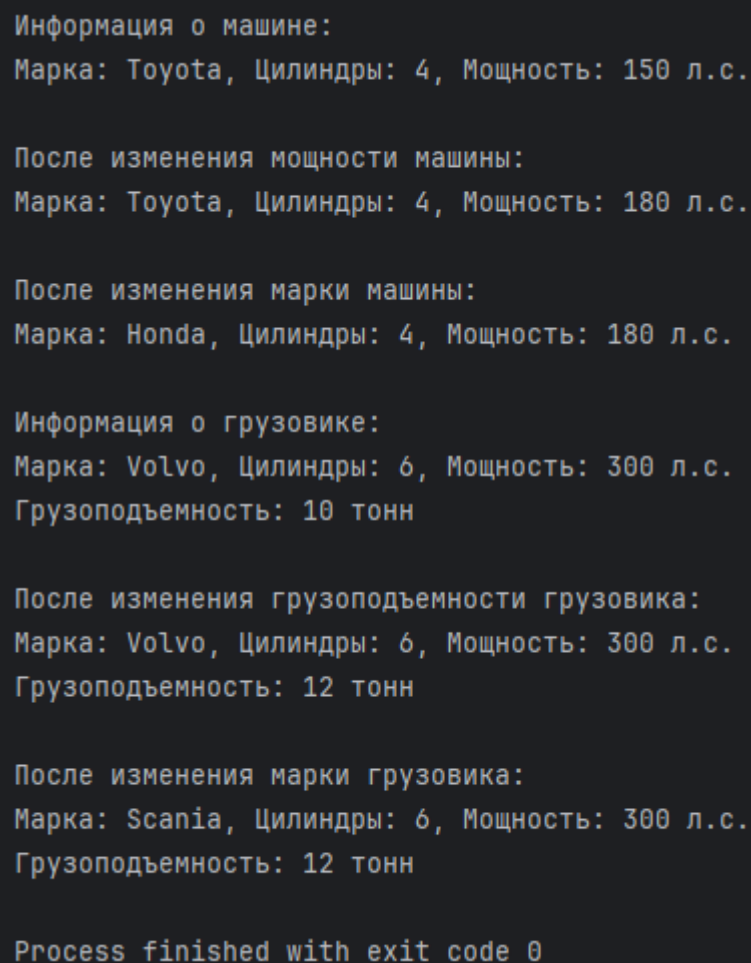
    # Создание объекта производного класса Lorry
    lorry = Lorry("Volvo", 6, 300, 10)
    print("\nИнформация о грузовике:")
    lorry.display_info()

```

```
# Изменение грузоподъемности грузовика
lorry.capacity = 12
print("\nПосле изменения грузоподъемности грузовика:")
lorry.display_info()

# Переназначение марки грузовика
lorry.brand = "Scania"
print("\nПосле изменения марки грузовика:")
lorry.display_info()
```

Результат работы программы:



```
Информация о машине:
Марка: Toyota, Цилиндры: 4, Мощность: 150 л.с.

После изменения мощности машины:
Марка: Toyota, Цилиндры: 4, Мощность: 180 л.с.

После изменения марки машины:
Марка: Honda, Цилиндры: 4, Мощность: 180 л.с.

Информация о грузовике:
Марка: Volvo, Цилиндры: 6, Мощность: 300 л.с.
Грузоподъемность: 10 тонн

После изменения грузоподъемности грузовика:
Марка: Volvo, Цилиндры: 6, Мощность: 300 л.с.
Грузоподъемность: 12 тонн

После изменения марки грузовика:
Марка: Scania, Цилиндры: 6, Мощность: 300 л.с.
Грузоподъемность: 12 тонн

Process finished with exit code 0
```

Рисунок 5 – Результат работы программы

Задание 2. Создать абстрактный базовый класс Figure с абстрактными методами вычисления площади и периметра. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими

функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие — в производных. Площадь трапеции: $S = (a + b) \times h/2$.

Код программы:

```
from abc import ABC, abstractmethod
import math

# Абстрактный базовый класс Figure
class Figure(ABC):
    @abstractmethod
    def area(self):
        """Вычисление площади фигуры."""
        pass

    @abstractmethod
    def perimeter(self):
        """Вычисление периметра фигуры."""
        pass

    @abstractmethod
    def input_data(self):
        """Ввод данных фигуры."""
        pass

    @abstractmethod
    def output_data(self):
        """Вывод данных фигуры."""
        pass

# Производный класс Rectangle (прямоугольник)
class Rectangle(Figure):
    def __init__(self, width=0, height=0):
        self.width = width
```

```
self.height = height
```

```
def area(self):
```

```
    return self.width * self.height
```

```
def perimeter(self):
```

```
    return 2 * (self.width + self.height)
```

```
def input_data(self):
```

```
    self.width = float(input("Введите ширину прямоугольника: "))
```

```
    self.height = float(input("Введите высоту прямоугольника: "))
```

```
def output_data(self):
```

```
    print(f"Прямоугольник: ширина = {self.width}, высота = {self.height}")
```

```
    print(f"Площадь: {self.area()}, Периметр: {self.perimeter()}")
```

```
# Производный класс Circle (круг)
```

```
class Circle(Figure):
```

```
    def __init__(self, radius=0):
```

```
        self.radius = radius
```

```
    def area(self):
```

```
        return math.pi * self.radius ** 2
```

```
    def perimeter(self):
```

```
        return 2 * math.pi * self.radius
```

```
    def input_data(self):
```

```
        self.radius = float(input("Введите радиус круга: "))
```

```
    def output_data(self):
```

```
        print(f"Круг: радиус = {self.radius}")
```

```
        print(f"Площадь: {self.area():.2f}, Периметр: {self.perimeter():.2f}")
```

Производный класс Trapezium (трапеция)

class Trapezium(Figure):

def __init__(self, base1=0, base2=0, height=0, side1=0, side2=0):

self.base1 = base1

self.base2 = base2

self.height = height

self.side1 = side1

self.side2 = side2

def area(self):

return (self.base1 + self.base2) * self.height / 2

def perimeter(self):

return self.base1 + self.base2 + self.side1 + self.side2

def input_data(self):

self.base1 = float(input("Введите длину первого основания трапеции: "))

self.base2 = float(input("Введите длину второго основания трапеции: "))

self.height = float(input("Введите высоту трапеции: "))

self.side1 = float(input("Введите длину первой боковой стороны: "))

self.side2 = float(input("Введите длину второй боковой стороны: "))

def output_data(self):

print(f"Трапеция: основание1 = {self.base1}, основание2 = {self.base2}, "

f"высота = {self.height}, сторона1 = {self.side1}, сторона2 = {self.side2}")

print(f"Площадь: {self.area():.2f}, Периметр: {self.perimeter():.2f}")

Функция вывода, демонстрирующая виртуальный вызов

def display_figure_info(figure: Figure):

figure.output_data()

Вызывающая программа

```
if __name__ == '__main__':  
    # Создание объектов производных классов  
    rectangle = Rectangle()  
    circle = Circle()  
    trapezium = Trapezium()  
  
    print("Введите данные для прямоугольника:")  
    rectangle.input_data()  
  
    print("\nВведите данные для круга:")  
    circle.input_data()  
  
    print("\nВведите данные для трапеции:")  
    trapezium.input_data()  
  
    print("\nИнформация о фигурах:")  
    display_figure_info(rectangle)  
    print()  
    display_figure_info(circle)  
    print()  
    display_figure_info(trapezium)Результат работы программы:
```



```
Введите данные для прямоугольника:
Введите ширину прямоугольника: 4
Введите высоту прямоугольника: 5

Введите данные для круга:
Введите радиус круга: 3

Введите данные для трапеции:
Введите длину первого основания трапеции: 3
Введите длину второго основания трапеции: 2
Введите высоту трапеции: 5
Введите длину первой боковой стороны: 3
Введите длину второй боковой стороны: 6

Информация о фигурах:
Прямоугольник: ширина = 4.0, высота = 5.0
Площадь: 20.0, Периметр: 18.0

Круг: радиус = 3.0
Площадь: 28.27, Периметр: 18.85

Трапеция: основание1 = 3.0, основание2 = 2.0, высота = 5.0, сторона1 = 3.0, сторона2 = 6.0
Площадь: 12.50, Периметр: 14.00

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

Ответы на контрольные вопросы:

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

Механизм объектно-ориентированного программирования, который позволяет создавать новый класс на основе существующего. Новый класс (называемый производным или подклассом) наследует атрибуты и методы родительского класса (или суперкласса). В языке Python наследование реализовано следующим образом: класс-потомок содержит в скобках у своего названия название родительского класса

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

Полиморфизм, как правило, используется с позиции переопределения методов базового класса в классе наследнике. В Python он реализован следующим образом: в классе-наследнике может быть метод с таким же названием, как и в родительском классе, но он может быть переопределен.

Метод `super()` обеспечивает доступ к функционалу методов родительского класса.

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

"Утиная" типизация - это концепция, согласно которой тип объекта определяется не его классом, а тем, какие методы и свойства он имеет. Если объект выглядит как утка и крикает как утка, то он может считаться уткой, независимо от его фактического типа.

4. Каково назначение модуля `abc` языка программирования Python?

Модуль `abc` (Abstract Base Classes) предоставляет инструменты для определения абстрактных базовых классов в Python.

5. Как сделать некоторый метод класса абстрактным?

Для создания абстрактного свойства можно использовать декоратор `@abstractmethod`.

6. Как сделать некоторое свойство класса абстрактным?

Для создания абстрактного свойства можно использовать декоратор `@property` вместе с `@abstractmethod`.

7. Каково назначение функции `isinstance`?

Функция `isinstance` используется для проверки того, является ли объект экземпляром определённого класса или его подкласса.

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