ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование на языках высокого уровня (Python)»

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

Выполнил студент Шефнер А.
Факультет: АИТ
Группа: ИВБ-211

Проверил: Баталов Д.И.

Санкт-Петербург

2023

Оценочный лист результатов ЛР № 5

Ф.И.О. студента _____Шефнер Альберт_____

Группа _____ИВБ-211_____

№ п/п	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Лабораторная работа№	Соответствие методике выполнения	Соответствует	7	
			Не соответствует	0	
		Срок выполнения	Выполнена в срок	2	
			Выполнена с опозданием на 2 недели	0	
		оформление	Соответствует требованиям	1 0	
			Не соответствует		
	ИТОГО количество баллов			10	

Доцент кафедры

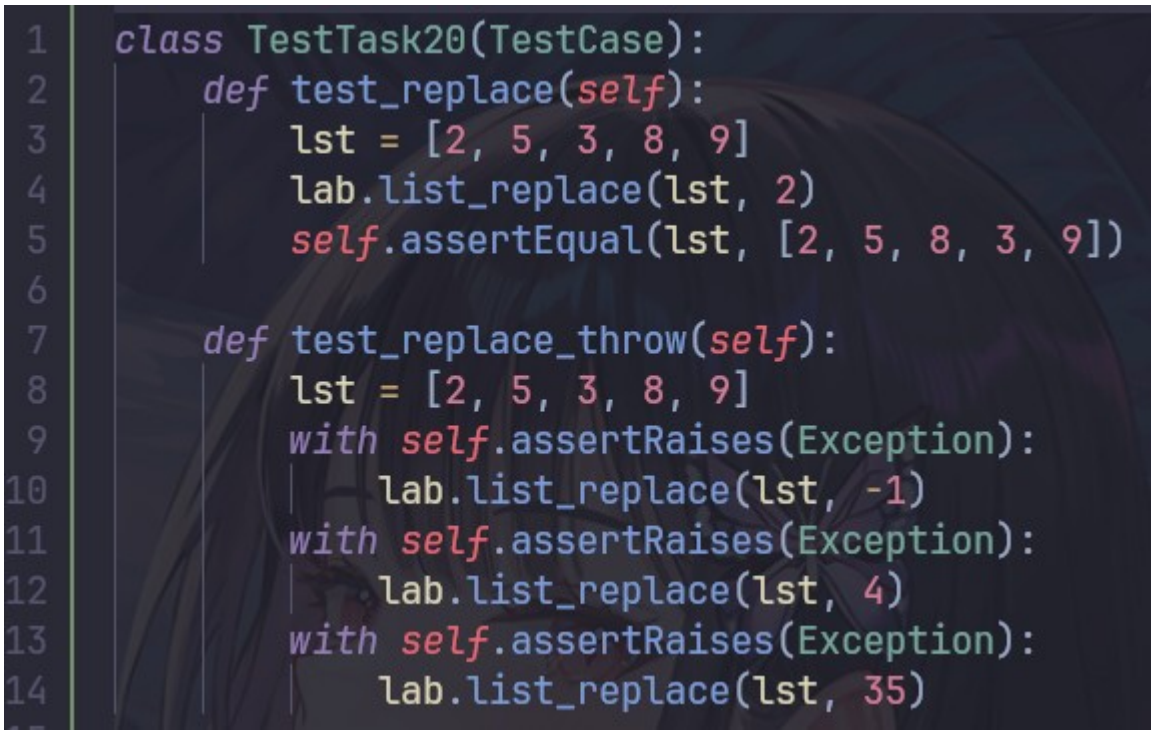
«Информационные и вычислительные

системы» Баталов Д.И. «__»

_____2023 г.

Тестирование

Для тестов используется модуль встроенной библиотеки unittest. Формат тестов везде одинаковы: сначала вызов функции или процедуры, потом сверка её результата с ожидаемым с помощью assert. Если assert имеет два параметра, тестируемое значение всегда слева, ожидаемое всегда справа. Вот пример одного из тестов:



```
class TestTask20(TestCase):
    def test_replace(self):
        lst = [2, 5, 3, 8, 9]
        lab.list_replace(lst, 2)
        self.assertEqual(lst, [2, 5, 8, 3, 9])

    def test_replace_throw(self):
        lst = [2, 5, 3, 8, 9]
        with self.assertRaises(Exception):
            lab.list_replace(lst, -1)
        with self.assertRaises(Exception):
            lab.list_replace(lst, 4)
        with self.assertRaises(Exception):
            lab.list_replace(lst, 35)
```

Программа успешно проходит все приведённые ниже тесты.

```
[albert@fedora Lab 5]$ python -m unittest test.py
.......................
----------------------------------------------------------------------
Ran 23 tests in 0.003s

OK
```

Более подробные результаты:

```
[albert@fedora Lab 5]$ python -m unittest test.py -v
test_feed (test.TestAnimals.test_feed) ... ok
test_make_sound (test.TestAnimals.test_make_sound) ... ok
test_car (test.TestCar.test_car) ... ok
test_ride (test.TestCarWithState.test_ride) ... ok
test_turns (test.TestCarWithState.test_turns) ... ok
test_drive (test.TestCars.test_drive) ... ok
test_stop (test.TestCars.test_stop) ... ok
test_counter (test.TestCounter.test_counter) ... ok
test_employee (test.TestEmployee.test_employee) ... ok
test_gcd (test.TestGcdFinder.test_gcd) ... ok
test_operations (test.TestIntNum.test_operations) ... ok
test_jsonfile (test.TestJsonFile.test_jsonfile) ... ok
test_rect_area (test.TestMaxBy.test_rect_area) ... ok
test_converter (test.TestNumSysConverter.test_converter) ... ok
test_operations (test.TestOperations.test_operations) ... ok
test_propertyholder (test.TestPropertyHolder.test_propertyholder) ... ok
test_equation (test.TestQuadraticEquation.test_equation) ... ok
test_area (test.TestRectangle.test_area) ... ok
test_operations (test.TestRectangle.test_operations) ... ok
test_area (test.TestShapes.test_area) ... ok
test_perimeter (test.TestShapes.test_perimeter) ... ok
test_shops (test.TestShops.test_shops) ... ok
test_subsets (test.TestSubsets.test_subsets) ... ok

----------------------------------------------------------------------
Ran 23 tests in 0.003s

OK
```

1. Напишите класс, позволяющий сформировать все уникальные подмножества из списка целых чисел, например: [2, 4, 10, 1].

Решение:

```
class SubSets:
    __list: List[int]

    def __init__(self, lst: List[int]) → None:
        self.__list = lst

    def get_subsets(self) → List[List[float]]:
        def compose_subset(comb):
            for e in zip(comb, self.__list):
                if e[0]:
                    yield e[1]

        pair_lists = [[False, True]] * len(self.__list)
        return list(map(
            compose(list, compose_subset),
            itertools.product(*pair_lists)
        ))
```

Тесты:

```
class TestSubsets(TestCase):
    def test_subsets(self):
        lst = [2, 4, 10, 1]
        subsets = arithmetic.SubSets(lst)
        self.assertCountEqual(
            sorted(subsets.get_subsets()),
            sorted([
                [], [2], [4], [10], [1],
                [2, 4], [2, 10], [2, 1],
                [4, 10], [4, 1], [10, 1],
                [2, 4, 10], [2, 4, 1],
                [2, 10, 1], [4, 10, 1],
                [2, 4, 10, 1]
            ])
        )
```

2. Напишите класс, реализующий все арифметические операции над двумя значениями (a и b).

Решение:

```python
class Operations:
    __a: float
    __b: float

    def __init__(self, a: float, b: float) -> None:
        self.__a = a
        self.__b = b

    def add(self) -> float:
        return self.__a + self.__b

    def sub(self) -> float:
        return self.__a - self.__b

    def mul(self) -> float:
        return self.__a * self.__b

    def div(self) -> float:
        return self.__a / self.__b
```

Тесты:

```python
class TestOperations(TestCase):
    def test_operations(self):
        a, b = 3, 6
        op = arithmetic.Operations(a, b)
        self.assertEqual(op.add(), a + b)
        self.assertEqual(op.sub(), a - b)
        self.assertEqual(op.mul(), a * b)
        self.assertEqual(op.div(), a / b)
```

3. Напишите класс, описывающий такой объект, как автомобиль. Продумайте, какие методы и переменные он должен иметь.

Решение:

```python
class Car:
    __speed: int
    __max_speed: int

    def __init__(self, max_speed):
        self.__max_speed = max_speed

    @property
    def speed(self):
        return self.__speed

    @property
    def max_speed(self):
        return self.__max_speed

    def ride(self):
        self.__speed = self.__max_speed
        print("Поехали!")

    def stop(self):
        self.__speed = 0
        print("Остановились!")
```

Тесты:

```python
class TestCar(TestCase):
    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_car(self, stdout):
        car1 = car.Car(100)
        car1.ride()
        car1.stop()
        self.assertEqual(
            stdout.getvalue(),
            "Поехали!\nОстановились!\n"
        )
```

4. Напишите класс, описывающий такой объект, как прямоугольник. Перегрузите у реализованного класса методы сравнения (сравнивать по площади), после чего создайте два экземпляра класса и проверьте, как работают перегруженные методы.

Решение:

```python
class Rectangle:
    __side_a: float
    __side_b: float

    def __init__(self, a: float, b: float) -> None:
        self.__side_a = a
        self.__side_b = b

    def area(self) -> float:
        return self.__side_a * self.__side_b

    def __eq__(self, other: "Rectangle") -> bool:
        return self.area() == other.area()

    def __ne__(self, other: "Rectangle") -> bool:
        return self.area() != other.area()

    def __gt__(self, other: "Rectangle") -> bool:
        return self.area() > other.area()

    def __ge__(self, other: "Rectangle") -> bool:
        return self.area() >= other.area()

    def __lt__(self, other: "Rectangle") -> bool:
        return self.area() < other.area()

    def __le__(self, other: "Rectangle") -> bool:
        return self.area() <= other.area()
```

Тесты:

```python
class TestRectangle(TestCase):
    def test_area(self):
        a, b = 4, 7
        rect = arithmetic.Rectangle(a, b)
        self.assertEqual(rect.area(), 28)

    def test_operations(self):
        r1 = arithmetic.Rectangle(10, 12)
        r2 = arithmetic.Rectangle(10, 12)
        r3 = arithmetic.Rectangle(5, 4)

        self.assertTrue(r1 == r2)
        self.assertFalse(r1 ≠ r2)
        self.assertTrue(r1 > r3)
        self.assertTrue(r2 ⩾ r1)
        self.assertTrue(r3 ⩽ r2 ⩽ r1)
        self.assertFalse(r1 < r3)
```

5. Напишите класс, описывающий такой объект, как автомобиль. У него может быть различное количество состояний, реализуемых посредством перечислений. Добавьте методы, позволяющие экземпляру класса менять свое текущее состояние (например: остановка, движение, поворот налево и т. д.)

Решение:

```python
    def display_state(self):
        state_str: str
        match self.__state:
            case CarState.STOPPED:
                state_str = "остановлена"
            case CarState.RIDES:
                state_str = "едет"
            case CarState.TURNS_LEFT:
                state_str = "поворачивает налево"
            case CarState.TURNS_RIGHT:
                state_str = "поворачивает направо"
            case _:
                raise Exception("self.__state was unknown.")

        print(f'Машина "{self.__name}" {state_str}')


    def ride(self):
        if self.__state == CarState.STOPPED:
            self.__state = CarState.RIDES

    def stop(self):
        self.__state = CarState.STOPPED

    def turn_left(self):
        if self.__state == CarState.RIDES or self.__state == CarState.TURNS_RIGHT:
            self.__state = CarState.TURNS_LEFT

    def turn_right(self):
        if self.__state == CarState.RIDES or self.__state == CarState.TURNS_LEFT:
            self.__state = CarState.TURNS_RIGHT
```

Тесты:

```python
class TestCarWithState(TestCase):
    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_ride(self, stdout):
        car1 = car.CarWithState("Феррари")
        car1.display_state()
        car1.ride()
        car1.display_state()

        self.assertEqual(
            stdout.getvalue(),
            'Машина "Феррари" остановлена\nМашина "Феррари" едет\n'
        )
```

```
13
14      @mock.patch("sys.stdout", new_callable=StringIO)
15      def test_turns(self, stdout):
16          car1 = car.CarWithState("Лада")
17          car1.turn_left()
18          car1.display_state()
19          car1.ride()
20          car1.turn_left()
21          car1.display_state()
22          car1.turn_right()
23          car1.display_state()
24
25          self.assertEqual(
26              stdout.getvalue(),
27              'Машина "Лада" остановлена\n' +
28              'Машина "Лада" поворачивает налево\n' +
29              'Машина "Лада" поворачивает направо\n'
30          )
```

6. Напишите класс, который подсчитывает текущее количество его экземпляров в приложении. Для корректного отображения этого числа перегрузите у класса метод __del__ и напишите необходимую логику.

Решение:

```
1  class Counter:
2      __count: int = 0
3      def __init__(self) → None:
4          Counter.__count += 1
5
6      def __del__(self) → None:
7          global _count
8          Counter.__count -= 1
9
10     @staticmethod
11     def count() → int:
12         return Counter.__count
```

Тесты:

```
class TestCounter(TestCase):
    def test_counter(self):
        c1 = objcount.Counter()      "c1" is not accessed
        c2 = objcount.Counter()
        c3 = objcount.Counter()      "c3" is not accessed
        c4 = objcount.Counter()      "c4" is not accessed
        c5 = objcount.Counter()
        c6 = objcount.Counter()

        self.assertEqual(objcount.Counter.count(), 6)

        del c2
        del c5
        del c6

        self.assertEqual(objcount.Counter.count(), 3)
```

7. Напишите базовый класс, задающий интерфейс и часть характеристик (если надо) таких объектов, как геометрические фигуры, автомобиль и магазин, животное.

Решение:

```
class ShapeBase:
    @abstractmethod
    def area(self) → float:
        ...

    @abstractmethod
    def perimeter(self) → float:
        ...
```

```python
class CarBase:
    @abstractmethod
    def drive(self) -> None:
        ...

    @abstractmethod
    def stop(self) -> None:
        ...

    @property
    @abstractmethod
    def name(self) -> str:
        ...
```

```python
class ShopBase:
    @property
    @abstractmethod
    def items(self) -> dict:
        ...

    @property
    @abstractmethod
    def margin(self) -> int:
        ...
```

```python
class ShopItem:
    name: str
    amount: int
    price: int

    def __init__(self, name: str, amount: int, price: int) -> None:
        self.name = name
        self.amount = amount
        self.price = price

    def __str__(self):
        return  f"(name='{self.name}', amount={self.amount}, price={self.price})"
```

```python
class AnimalBase:
    @abstractmethod
    def feed(self, food: str, amount: int) -> None:
        ...

    @abstractmethod
    def make_sound(self) -> None:
        ...
```

8. Напишите несколько производных классов от базового класса геометрических фигур (например: прямоугольник и квадрат).

Решение:

```python
class Rectangle(ShapeBase):
    __side_a: float
    __side_b: float

    def __init__(self, a: float, b: float) -> None:
        super().__init__()
        self.__side_a = a
        self.__side_b = b

    @override
    def area(self) -> float:
        return self.__side_a * self.__side_b

    @override
    def perimeter(self) -> float:
        return 2 * (self.__side_a + self.__side_b)
```

```python
class Square(ShapeBase):
    __side: float

    def __init__(self, side: float) -> None:
        super().__init__()
        self.__side = side

    @override
    def area(self) -> float:
        return self.__side * self.__side

    @override
    def perimeter(self) -> float:
        return 4 * self.__side
```

```python
class Circle(ShapeBase):
    __radius: float

    def __init__(self, radius: float) -> None:
        super().__init__()
        self.__radius = radius

    @override
    def area(self) -> float:
        return self.__radius * self.__radius * math.pi

    @override
    def perimeter(self) -> float:
        return self.__radius * 2 * math.pi
```

Тесты:

```python
class TestShapes(TestCase):
    def test_area(self):
        shape_list = [
            shapes.Rectangle(2, 5),
            shapes.Square(10),
            shapes.Circle(4),
        ]

        self.assertEqual(shape_list[0].area(), 2 * 5)
        self.assertEqual(shape_list[1].area(), 10 * 10)
        self.assertEqual(shape_list[2].area(), math.pi * 4 * 4)

    def test_perimeter(self):
        shape_list: List[interfaces.ShapeBase] = [
            shapes.Rectangle(2, 5),
            shapes.Square(10),
            shapes.Circle(4),
        ]

        self.assertEqual(shape_list[0].perimeter(), 2 * (2 + 5))
        self.assertEqual(shape_list[1].perimeter(), 4 * 10)
        self.assertEqual(shape_list[2].perimeter(), math.pi * 2 * 4)
```

9. Напишите несколько производных классов от базового класса автомобилей (например: легковой и грузовой автомобиль).

Решение:

```python
class SportCar(CarBase):
    __name: str
    __wheels_need_replace: bool

    def __init__(self, name: str) -> None:
        super().__init__()

        self.__name = name
        self.__wheels_need_replace = False

    @override
    def drive(self) -> None:
        if self.__wheels_need_replace:
            print("Cannot move! Replace wheels.")
        else:
            self.__wheels_need_replace = True
            print("Need for speed!")

    @override
    def stop(self) -> None:
        print(f"Sportcar '{self.__name}' stopped! Why?")

    @property
    @override
    def name(self) -> str:
        return self.__name

    def replace_wheels(self) -> None:
        self.__wheels_need_replace = False
        print("Wheels successfully replaced!")
```

```python
class AutoVaz(CarBase):
    __carriage: float
    __max_carriage: float

    def __init__(self, max_carriage) -> None:
        super().__init__()

        self.__carriage = 0.0
        self.__max_carriage = max_carriage

    @override
    def drive(self) -> None:
        print(f"AutoVaz {self.name} tronulsya...")

    @override
    def stop(self) -> None:
        print(f"AutoVaz {self.name} ostanovilsya...")

    @property
    @override
    def name(self) -> str:
        return "Lastochka"

    def load(self, amount: float) -> None:
        if self.__carriage + amount > self.__max_carriage:
            print(f"AutoVaz {self.name} cannot hold that much!")
        else:
            self.__carriage += amount
            print(f"AutoVaz {self.name} successfully loaded.")

    def unload(self) -> float:
        carriage = self.__carriage
        self.__carriage = 0
        print(f"AutoVaz {self.name} unloaded.")
        return carriage

    @property
    def carriage(self) -> float:
        return self.__carriage
```

Тесты:

```python
class TestCars(TestCase):
    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_drive(self, stdout):
        car_list: List[interfaces.CarBase] = [
            cars.SportCar("Феррари"),
            cars.AutoVaz(100)
        ]

        car_list[0].drive()
        self.assertEqual(
            stdout.getvalue(),
            "Need for speed!\n"
        )

        stdout.truncate(0)
        stdout.seek(0)

        car_list[1].drive()
        self.assertEqual(
            stdout.getvalue(),
            "AutoVaz Lastochka tronulsya...\n"
        )

    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_stop(self, stdout):
        car_list: List[interfaces.CarBase] = [
            cars.SportCar("Феррари"),
            cars.AutoVaz(100)
```

```python
        car_list[0].stop()
        self.assertEqual(
            stdout.getvalue(),
            "Sportcar 'Феррари' stopped! Why?\n"
        )

        stdout.truncate(0)
        stdout.seek(0)

        car_list[1].stop()
        self.assertEqual(
            stdout.getvalue(),
            "AutoVaz Lastochka ostanovilsya...\n"
```

10. Напишите несколько производных классов от базового класса магазинов (например: ларек и супермаркет).

Решение:

```python
class Stall(ShopBase):
    __item_type: str
    __amount: int
    __price: int
    __margin: int

    def __init__(self, item_type: str, amount: int, price: int) -> None:
        super().__init__()

        self.__item_type = item_type
        self.__amount = amount
        self.__margin = 0
        self.__price = price

    @property
    @override
    def items(self) -> dict:
        return {
            self.__item_type: ShopItem(self.__item_type, self.__amount, self.__price)
        }

    @property
    @override
    def margin(self) -> int:
        return self.__margin

    def sell(self, amount: int) -> None:
        if amount > self.__amount:
            print("Cannot sell that much.")
            return

        self.__amount -= amount
        sold_margin = amount * self.__price
        self.__margin += sold_margin
        print(f"Successfully sold {amount} of {self.__item_type} for {sold_margin}$")

    def restock(self, amount: int) -> None:
        self.__amount += amount
```

```python
class Supermarket(ShopBase):
    __items: Dict[str, ShopItem]
    __margin: int

    def __init__(self, items: Dict[str, ShopItem]) -> None:
        super().__init__()
        self.__items = items
        self.__margin = 0

    @property
    @override
    def items(self) -> Dict[str, ShopItem]:
        return self.__items

    @property
    @override
    def margin(self) -> int:
        return self.__margin

    def sell(self, name: str, amount: int) -> None:
        if name not in self.__items:
            raise ValueError(f"Supermarket does not contain item with name '{name}'")

        item = self.__items[name]
        if amount > item.amount:
            raise ValueError(
                f"Supermarket cannot sold {amount} of {name}. ({item.amount} in stock)."
                )

        item.amount -= amount
        margin = amount * item.price
        self.__margin += margin
        print(f"Successfully sold {amount} of {name} for {margin}$.")

    def restock(self, name: str, amount: int) -> None:
        if name not in self.__items:
            raise ValueError(f"Supermarket does not contain item with name '{name}'")

        self.__items[name].amount += amount
```

Тесты:

```python
class TestShops(TestCase):
    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_shops(self, stdout):     ■ "stdout" is not accessed
        stall = shops.Stall("Арбуз", 10, 150)
        supermarket = shops.Supermarket({"Хлеб": interfaces.ShopItem("Хлеб", 10, 40)})

        stall.sell(4)
        supermarket.sell("Хлеб", 7)

        shop_list = [stall, supermarket]
        self.assertEqual([shop.margin for shop in shop_list], [600, 280])

        self.assertEqual(shop_list[0].items["Арбуз"].amount, 6)
        self.assertEqual(shop_list[1].items["Хлеб"].amount, 3)
```

11. Напишите несколько производных классов от базового класса автомобилей (например: лошадь и тигр).

Решение:

```python
class Horse(AnimalBase):
    def __init__(self) -> None:
        super().__init__()

    @override
    def feed(self, food: str, amount: int) -> None:
        match food.strip().lower():
            case "apple" | "apples":
                print(f"Horse just ate {amount} amount of {food}.")
            case _:
                print(f"Horse refused to eat {food}.")

    @override
    def make_sound(self):
        print("Neigh!!!")


class Tiger(AnimalBase):
    def __init__(self) -> None:
        super().__init__()

    @override
    def feed(self, food: str, amount: int) -> None:
        match food.strip().lower():
            case "pig" | "rabbit" | "gnu":
                print(f"Tiger just enjoed {amount} of {food}.")
            case "horse" | "lion":
                print(f"Tiger disliked {food}, but ate {amount} of.")
            case _:
                print(f"Tiger refused to eat {food}.")

    @override
    def make_sound(self) -> None:
        print("Roar!!!")
```

Тесты:

```python
class TestAnimals(TestCase):
    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_feed(self, stdout):
        animal_list: List[interfaces.AnimalBase] = [
            animals.Horse(),
            animals.Tiger()
        ]

        for animal in animal_list:
            animal.feed("apples", 20)

        self.assertEqual(
            stdout.getvalue(),
            "Horse just ate 20 amount of apples.\nTiger refused to eat apples.\n"
        )

    @mock.patch("sys.stdout", new_callable=StringIO)
    def test_make_sound(self, stdout):
        animal_list: List[interfaces.AnimalBase] = [
            animals.Horse(),
            animals.Tiger()
        ]

        for animal in animal_list:
            animal.make_sound()

        self.assertEqual(
            stdout.getvalue(),
            "Neigh!!!\nRoar!!!\n"
```

12. Напишите класс, доступ к атрибутам (переменным) которого осуществляется с помощью декоратора @property.

Решение:

```python
class PropertyHolder:
    __a: int
    __b: str

    def __init__(self, a: int, b: str) -> None:
        self.__a = a
        self.__b = b
```

```python
    @property
    def a(self) -> int:
        return self.__a

    @a.setter
    def a(self, value: int) -> None:
        if not isinstance(value, int):
            raise ValueError("'a' must be int.")

        self.__a = value

    @property
    def b(self) -> str:
        return self.__b

    @b.setter
    def b(self, value: str) -> None:
        if not isinstance(value, str):
            raise ValueError("'b' must be str.")

        self.__b = value
```

Тесты:

```python
class TestPropertyHolder(TestCase):
    def test_propertyholder(self):
        (holder) = properties.PropertyHolder(1, "xd")
        self.assertEqual((holder.a, holder.b), (1, "xd"))
        holder.b = "lol"
        self.assertEqual((holder.a, holder.b), (1, "lol"))
```

13. Напишите класс, хранящий целое число. Перегрузите у него методы арифметических операций. Объявите два экземпляра класса и проверьте, как работают перегруженные методы.

Решение:

```python
class IntNum:
    __num: int

    def __init__(self, num) -> None:
        self.__num = num

    @property
    def value(self) -> int:
        return self.__num

    def __add__(self, other) -> "IntNum":
        return IntNum(self.__num + other.__num)

    def __sub__(self, other) -> "IntNum":
        return IntNum(self.__num - other.__num)

    def __mul__(self, other) -> "IntNum":
        return IntNum(self.__num * other.__num)

    def __floordiv__(self, other) -> "IntNum":
        return IntNum(self.__num // other.__num)
```

Тесты:

```python
class TestIntNum(TestCase):
    def test_operations(self):
        a = arithmetic.IntNum(43)
        b = arithmetic.IntNum(10)

        self.assertEqual((a + b).value, 53)
        self.assertEqual((a - b).value, 33)
        self.assertEqual((a * b).value, 430)
        self.assertEqual((a // b).value, 4)
```

14. Напишите класс, который позволяет работать с json-файлом, осуществляя его чтение, запись, добавление, удаление и изменение значений.

Решение:

```python
class JsonFile:
    __path: str

    def __init__(self, path: str) -> None:
        self.__path = path

    def write(self, obj) -> None:
        with open(self.__path, 'w') as file:
            json.dump(obj, file)

    def read(self):
        with open(self.__path, 'r') as file:
            return json.load(file)
```

Тесты:

```python
class TestJsonFile(TestCase):
    def test_jsonfile(self):
        data = {"x": 4, "y": 8}
        file = models.JsonFile("test_task14.txt")

        file.write(data)

        s = file.read()
        self.assertEqual(s['x'], 4)
        self.assertEqual(s['y'], 8)
```

15. Напишите класс, который находит прямоугольник с максимальной площадью из списка.

Решение:

```python
class MaxBy[T, K]:
    __func: Callable[[T], K]
    def __init__(self, func: Callable[[T], K]) -> None:
        self.__func = func

    def get(self, lst: List[T]):
        return max(lst, key=self.__func) # type: ignore

    @staticmethod
    def rect_area() -> "MaxBy[Rectangle, float]":
        return MaxBy(lambda r : r.area())
```

Тесты:

```python
class TestMaxBy(TestCase):
    def test_rect_area(self):
        rects: List[arithmetic.Rectangle] = [
            arithmetic.Rectangle(2, 5),
            arithmetic.Rectangle(9, 1),
            arithmetic.Rectangle(2, 2),
            arithmetic.Rectangle(8, 8),
            arithmetic.Rectangle(3, 4),
        ]

        result = arithmetic.MaxBy.rect_area().get(rects)
        self.assertEqual(result.area(), 64)
```

16. Напишите класс, осуществляющий преобразование целого числа из десятичной системы счисления в двоичную и наоборот.

Решение:

```python
class NumSysConverter:
    ALPHABET = "0123456789ABCDEF"
    __num: int

    def __init__(self, repr: str, base: int) -> None:
        if base > len(NumSysConverter.ALPHABET):
            raise ValueError(f"Unsupported base '{base}'.")
        self.__num = NumSysConverter.get_num(repr, base)

    def convert_to(self, base: int) -> str:
        if base > len(NumSysConverter.ALPHABET):
            raise ValueError(f"Unsupported base '{base}'")

        repr = ""
        num = self.__num
        while num:
            repr += self.ALPHABET[num % base]
            num //= base

        return repr[::-1]


    @staticmethod
    @functools.lru_cache
    def get_digit(d: str):
        return NumSysConverter.ALPHABET.find(d)

    @staticmethod
    @functools.lru_cache
    def get_num(repr: str, base: int) -> int:
        exp = 1
        num = 0
        for d in repr[::-1]:
            num += NumSysConverter.get_digit(d) * exp
            exp *= base
        return num
```

Тесты:

```python
class TestNumSysConverter(TestCase):
    def test_converter(self):
        bin_1 = "1101"
        conv_1 = arithmetic.NumSysConverter(bin_1, 2)
        self.assertEqual(conv_1.convert_to(10), "13")

        dec_2 = "531"
        conv_2 = arithmetic.NumSysConverter(dec_2, 10)
        self.assertEqual(conv_2.convert_to(2), "1000010011")
```

17. Напишите класс, вычисляющий наименьший общий делитель (НОД). Значения, участвующие в поиске НОД должны устанавливаться отдельными методами или посредством декоратора @property до вызова метода расчета.

Решение:

```python
class GcdFinder:
    __a: int | None
    __b: int | None

    def __init__(self) -> None:
        self.__a = None
        self.__b = None

    @property
    def a(self) -> int | None:
        return self.__a

    @a.setter
    def a(self, value: int) -> None:
        self.__a = value

    @property
    def b(self) -> int | None:
        return self.__b

    @b.setter
    def b(self, value: int) -> None:
        self.__b = value

    def find(self):
        if self.__a is None or self.__b is None:
            raise ValueError("Must set a and b before finding GCD")

        return math.gcd(self.__a, self.__b)
```

Тесты:

```
class TestGcdFinder(TestCase):
    def test_gcd(self):
        finder = arithmetic.GcdFinder()
        finder.a = 3215
        finder.b = 1286
        self.assertEqual(finder.find(), 643)
```

18. Напишите класс, вычисляющий корни квадратного уравнения.

Решение:

```
class QuadraticEquation:
    __a: complex
    __b: complex
    __c: complex

    def __init__(self, a: complex, b: complex, c: complex) → None:
        self.__a = a
        self.__b = b
        self.__c = c

    @property
    @functools.lru_cache
    def discriminant(self) → complex:
        return cmath.sqrt(self.__b * self.__b - 4 * self.__a * self.__c)

    @property
    @functools.lru_cache
    def solve(self) → Tuple[complex, complex]:
        return (
            (-self.__b - self.discriminant) / 2 * self.__a,
            (-self.__b + self.discriminant) / 2 * self.__a
        )
```

Тесты:

```
class TestQuadraticEquation(TestCase):
    def test_equation(self):
        equation = arithmetic.QuadraticEquation(1, -4, 3)
        self.assertEqual(equation.solve, (1, 3))
```

19. Напишите класс, хранящий данные сотрудника фирмы и имеющий метод, возвращающий характеристики текущего сотрудника в виде словаря.

Решение:

```python
class Employee:
    __name: str
    __salary: int
    __department: str

    def __init__(self, name: str, salary: int, department: str) -> None:
        self.__name = name
        self.__salary = salary
        self.__department = department

    @property
    def name(self) -> str:
        return self.__name

    @name.setter
    def name(self, value: str) -> None:
        self.__name = value

    @property
    def salary(self) -> int:
        return self.__salary

    @salary.setter
    def salary(self, value: int) -> None:
        self.__salary= value

    @property
    def department(self) -> str:
        return self.__department

    @department.setter
    def department(self, value: str) -> None:
        self.__department = value

    def to_dict(self) -> Dict[str, Any]:
        return {
            "name": self.__name,
            "salary": self.__salary,
            "department": self.__department
        }
```

Тесты:

```python
class TestEmployee(TestCase):
    def test_employee(self):
        employee = models.Employee("John", 1500, "Devops")
        self.assertDictEqual(
            employee.to_dict(),
            {"name": "John", "salary": 1500, "department": "Devops"}
        )
```

20. Напишите класс, представляющий собой записную книжку. Каждый элементпзаписной книжки должен содержать следующие поля: ФИО, номер телефона, e-mail, день рождения. Записная книжка может сохраняться на диск в виде json- файла, а также должна иметь метод загрузки данных из файла.

Решение:

```python
class NotebookItem:
    __name: str
    __phone: str
    __email: str
    __birthday: str

    def __init__(self, name: str, phone: str, email: str, birthday: str) -> None:
        self.__name = name
        self.__phone = phone
        self.__email = email
        self.__birthday = birthday

    @property
    def name(self) -> str:
        return self.__name

    @name.setter
    def name(self, value: str) -> None:
        self.__name = value

    @property
    def phone(self) -> str:
        return self.__phone

    @phone.setter
    def phone(self, value: str) -> None:
        self.__phone = value
```

```python
    @property
    def email(self) -> str:
        return self.__email

    @email.setter
    def email(self, value: str) -> None:
        self.__email = value

    @property
    def birthday(self) -> str:
        return self.__birthday

    @birthday.setter
    def birthday(self, value: str) -> None:
        self.__birthday = value
```

```python
class NotebookItemEncoder(json.JSONEncoder):
    def default(self, o):
        return json.dumps({
            "name": o.name,
            "phone": o.phone,
            "email": o.email,
            "birthday": o.birthday
        })
```

```python
class Notebook:
    __list: List[NotebookItem]

    def __init__(self, set_: List[NotebookItem]) -> None:
        self.__list = set_

    def add(self, item: NotebookItem):
        self.__list.append(item)

    def remove(self, name: str) -> None:
        self.__list = list(i for i in self.__list if not i.name == name)

    def json(self):
        return json.dumps(self.__list, cls=NotebookItemEncoder)

    @staticmethod
    def from_json(json_s: str) -> "Notebook":
        return Notebook(list(
            NotebookItem(o["name"], o["phone"], o["email"], o["birthday"])
            for o in json.loads(json_s)
        ))
```