**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

**Федеральное государственное автономное**

**образовательное учреждение высшего образования**

**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

# «Элементы объектно-ориентированного программирования в языке Python»

**Отчет по лабораторной работе № 4.1**

**по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Кучеренко С. Ю. _____« » 2023г.

Подпись студента_____

Работа защищена «  »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

**Цель работы:** приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

**Ход работы:**

1. Изучить теоретический материал работы.

2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

3. Выполните клонирование созданного репозитория.

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы.

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


class Book:
    material = "paper"
    cover = "paperback"
    all_books = []


if __name__ == '__main__':
    Book.material
    Book.cover
    Book.all_books
```

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

class River:
    all_rivers = []

    def __init__(self, name, lenght):
        self.name = name
        self.lenght = lenght
        River.all_rivers.append(self)

volga = River("Волга", 3530)
seine = River("Сена", 776)
nile = River("Нил", 6852)

for river in River.all_rivers:
    print(river.name)
```


Рисунок 1 – Результат работы программы

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


class River:
    all_rivers = []

    def __init__(self, name, length):
        self.name = name
        self.length = length
        River.all_rivers.append(self)

    def get_info(self):
        print("Длина {0} равна {1} км".format(self.name, self.length))


if __name__ == '__main__':
    volga = River("Волга", 3530)
    seine = River("Сена", 776)
    nile = River("Нил", 6852)
    volga.get_info()
    seine.get_info()
    nile.get_info()
```

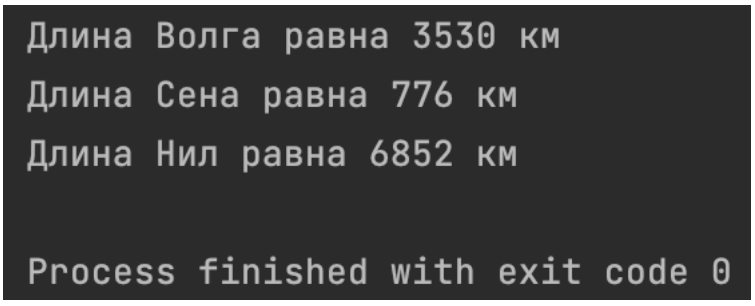


Рисунок 2 – Результат работы программы

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


class Ship:
    def __init__(self, name, capacity):
        self.name = name
        self.capacity = capacity
        self.cargo = 0

    def load_cargo(self, weight):
        if self.cargo + weight <= self.capacity:
            self.cargo += weight
```

```
            print("Loaded {} tons".format(weight))
        else:
            print("Cannot load that much")

    def unload_cargo(self, weight):
        if self.cargo - weight >= 0:
            self.cargo -= weight
            print("Unloaded {} tons".format(weight))
        else:
            print("Cannot unload that much")

    def name_captain(self, cap):
        self.captain = cap
        print("{} is the captain of the {}".format(self.captain, self.name))


if __name__ == '__main__':
    black_pearl = Ship("Black Pearl", 800)
    black_pearl.name_captain("Jack Sparrow")
    print(black_pearl.captain)
    black_pearl.load_cargo(600)
    black_pearl.unload_cargo(400)
    black_pearl.load_cargo(700)
    black_pearl.unload_cargo(300)
```

```
Jack Sparrow is the captain of the Black Pearl
Jack Sparrow
Loaded 600 tons
Unloaded 400 tons
Cannot load that much
Cannot unload that much

Process finished with exit code 0
```

Рисунок 3 – Результат работы программы

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


class Rectangle:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError
```

```
    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height


if __name__ == '__main__':
    rect = Rectangle(10, 20)
    print(rect.width)
    print(rect.height)
    rect.width = 50
    print(rect.width)
    rect.height = 70
    print(rect.height)
```

```
10
20
50
70

Process finished with exit code 0
```

Рисунок 4 – Результат работы 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()
```

```
3/4
Введите обыкновенную дробь: 5/6
5/6
19/12
1/12
5/8
10/9

Process finished with exit code 0
```

Рисунок 5 – Результат работы программы

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

Задание 1.

Парой называется класс с двумя полями, которые обычно имеют имена *first* и *second*. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- метод инициализации `__init__`; метод должен контролировать значения аргументов на корректность;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Реализовать внешнюю функцию с именем `make_тип()`, где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ = '__main__':` добавить код, демонстрирующий возможности разработанного класса.

9. Поле first — целое положительное число, часы; поле second — целое положительное число, минуты. Реализовать метод `minutes()` — приведение времени в минуты.

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


def is_number(s):
    try:
        int(s)
    except ValueError:
```

```
        return False
    return True


def make_conversion(first, second):
    if is_number(first) and is_number(second) and first > 0 and second > 0:
        conversion = Conversion(first, second)
        return conversion
    else:
        raise ValueError


class Conversion:
    def __init__(self, first=0, second=0):
        if is_number(first) and is_number(second):
            if first > 0 and second > 0:
                self.__first = first
                self.__second = second
            else:
                raise ValueError
        else:
            raise ValueError

    def read(self):
        self.__first = int(input("Enter the first value: "))
        self.__second = int(input("Enter the second value: "))

    def display(self):
        print(f"First value: {self.__first}")
        print(f"Second value: {self.__second}")

    def cost(self):
        return 60 * self.__first + self.__second


if __name__ == '__main__':
    p1 = make_conversion(3, 45)
    p1.display()
    print(f"Time in minutes: {p1.cost()}")
    p1.read()
    print(f"Time in minutes: {p1.cost()}")
    p2 = make_conversion("fhhfhf", 4)
    p2.display()
    p2.cost()
```

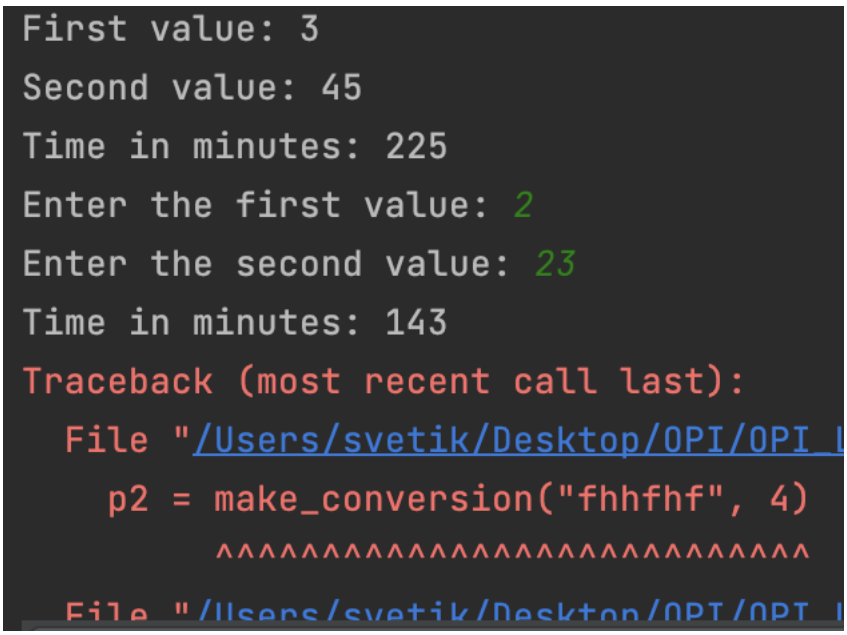


Рисунок 6 – Результат работы программы

Задание 2.

Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

- метод инициализации `__init__`;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ = '__main__':` добавить код, демонстрирующий возможности разработанного класса.

9. Реализовать класс Account, представляющий собой банковский счет. В классе должны быть четыре поля: фамилия владельца, номер счета, процент начисления и сумма в рублях. Открытие нового счета выполняется операцией инициализации. Необходимо выполнять следующие операции: сменить владельца счета, снять некоторую сумму денег со счета, положить деньги на счет, начислить проценты, перевести сумму в доллары, перевести сумму в евро, получить сумму прописью (преобразовать в числительное).

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


class Account:
    def __init__(self, owner, number, interest_rate, balance):
        self.owner = owner
        self.number = number
        self.interest_rate = interest_rate
        self.balance = balance
```

```
    def read(self):
        self.owner = input("Введите фамилию владельца счета: ")
        self.number = input("Введите номер счета: ")
        self.interest_rate = float(input("Введите процент начисления: "))
        self.balance = float(input("Введите сумму в рублях: "))

    def display(self):
        print("Фамилия владельца: ", self.owner)
        print("Номер счета: ", self.number)
        print("Процент начисления: ", self.interest_rate)
        print("Сумма в рублях: ", self.balance)

    # Смена владельца счета
    def change_owner(self, new_owner):
        self.owner = new_owner

    # Снятие некоторой суммы со счета
    def withdrawal(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print("Снято", amount, "рублей")
        else:
            print("Недостаточно средств на счете")

    # Положить деньги на счет
    def crediting_money(self, amount):
        self.balance += amount
        print("Положено", amount, "рублей")

    # Начислить проценты
    def add_interest(self):
        interest = self.balance * (self.interest_rate / 100)
        self.balance += interest
        print("Начислены проценты:", interest, "рублей")

    # Перевести сумму в доллары
    def convert_to_dollars(self, exchange_rate):
        dollars = self.balance / exchange_rate
        print("Сумма в долларах:", dollars)

    # Перевести сумму в евро
    def convert_to_euros(self, exchange_rate):
        euros = self.balance / exchange_rate
        print("Сумма в евро:", euros)

    # Получить сумму прописью (преобразовать в числительное)
    def get_amount_in_words(self):
        ones = ['', 'один', 'два', 'три', 'четыре', 'пять', 'шесть',
                'семь', 'восемь', 'девять']
        teens = ['десять', 'одиннадцать', 'двенадцать', 'тринадцать',
                 'четырнадцать', 'пятнадцать', 'шестнадцать',
                 'семнадцать', 'восемнадцать', 'девятнадцать']
        tens = ['', 'десять', 'двадцать', 'тридцать', 'сорок', 'пятьдесят',
                'шестьдесят', 'семьдесят', 'восемьдесят', 'девяносто']
        hundreds = ['', 'сто', 'двести', 'триста', 'четыреста', 'пятьсот',
                    'шестьсот', 'семьсот', 'восемьсот', 'девятьсот']
        thousands = {
            0: '',
            1: 'одна тысяча',
            2: 'две тысячи',
            3: 'три тысячи',
            4: 'четыре тысячи',
            5: 'пять тысяч',
            6: 'шесть тысяч',
```

```
            7: 'семь тысяч',
            8: 'восемь тысяч',
            9: 'девять тысяч'
        }

        amount = int(self.balance)

        thousands_part = amount // 1000
        hundreds_part = (amount // 100) % 10
        tens_part = (amount // 10) % 10
        ones_part = amount % 10

        words = []

        if thousands_part > 0:
            words.append(thousands.get(thousands_part))

        if hundreds_part > 0:
            words.append(hundreds[hundreds_part])

        if tens_part == 1:
            words.append(teens[ones_part])
        else:
            if tens_part > 0:
                words.append(tens[tens_part])

            if ones_part > 0:
                words.append(ones[ones_part])

        amount_in_words = ' '.join(words)
        print("Сумма прописью:", amount_in_words)


if __name__ == '__main__':
    account = Account("", "", 0.0, 0.0)
    account.read()
    account.display()

    account.change_owner("Новый владелец")
    account.withdrawal(500)
    account.crediting_money(1000)
    account.add_interest()
    account.convert_to_dollars(80)
    account.convert_to_euros(86)
    account.get_amount_in_words()
    account.display()
```

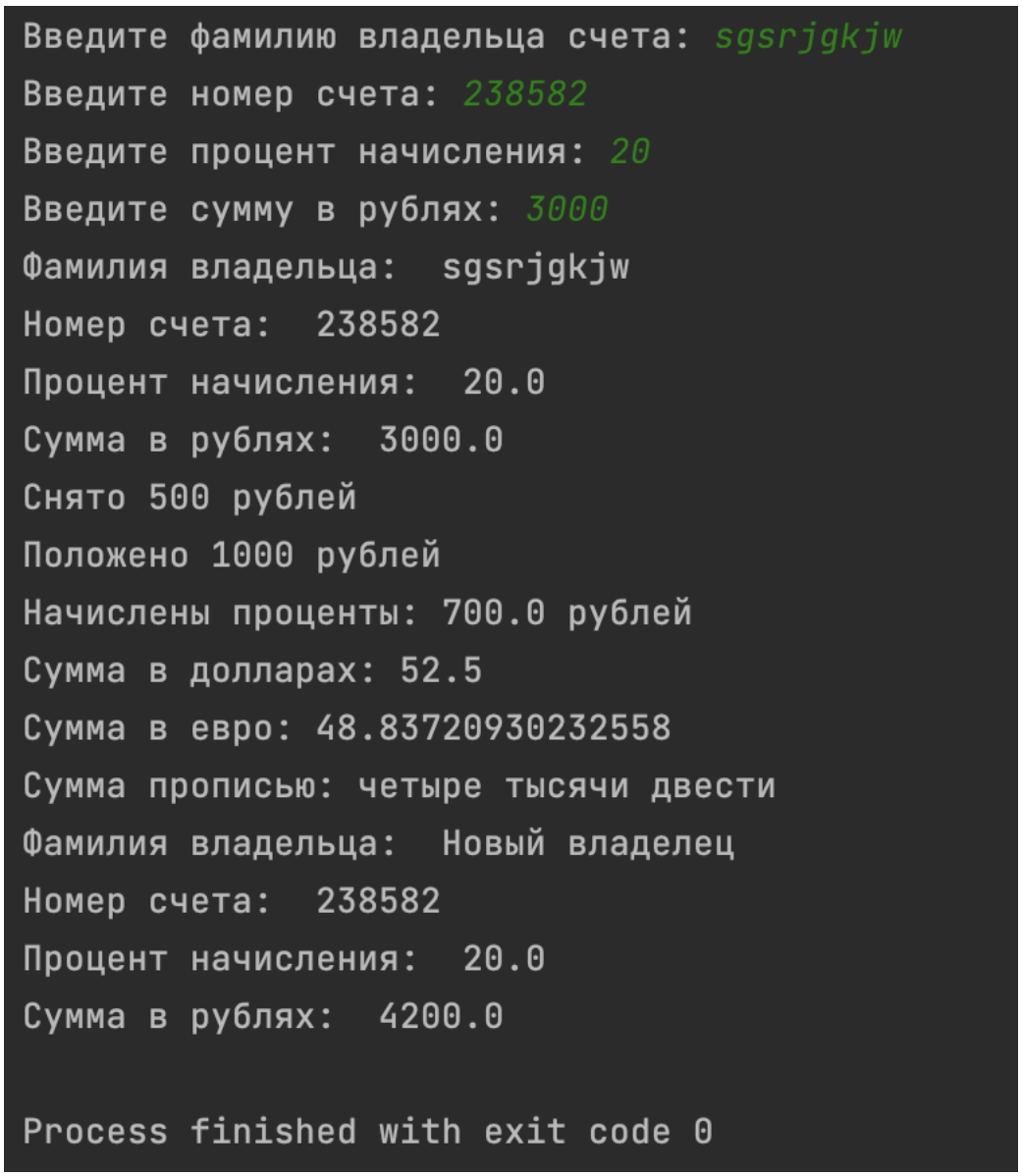


Рисунок 7 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

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

11. Отправьте сделанные изменения на сервер GitHub.

**Контрольные вопросы:**

1. Как осуществляется объявление класса в языке Python?

Классы объявляются с помощью ключевого слова `class` и имени класса:

```
# class syntax
class MyClass:
    var = ...  # некоторая переменная

    def do_smt(self):
    # какой-то метод
```

Как правило, имя класса начинается с заглавной буквы и обычно является существительным или словосочетанием . Имена классов соответствуют соглашению **CapWords** (или UpperCamelCase): это означает, что если это фраза, все слова в этой фразе пишутся с большой буквы и пишутся без подчеркивания между ними.

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты класса являются общими для всех объектов класса, а атрибуты экземпляра специфическими для каждого экземпляра. Более того, атрибуты класса определяются внутри класса, но вне каких-либо методов, а атрибуты экземпляра обычно определяются в методах, чаще всего в __init__ .

3. Каково назначение методов класса?

Методы определяют функциональность объектов, принадлежащих конкретному классу. Основной синтаксис выглядит так:

```
# basic method syntax
class MyClass:
    # the constructor
    def __init__(self, arg1):
        self.att = arg1

    # custom method
    def do_smt(self):
        # does something
```

4. Для чего предназначен метод __init__() класса?

Метод __init__ является конструктором. Конструкторы - это концепция объектно-ориентированного программирования. Класс может иметь один и

только один конструктор. Если __init__ определен внутри класса, он автоматически вызывается при создании нового экземпляра класса.

5. Каково назначение self ?

`self`

Возможно, вы заметили, что у нашего метода `__init__` был другой аргумент, кроме `name` и `length`: `self`. Аргумент `self` представляет конкретный экземпляр класса и позволяет нам получить доступ к его атрибутам и методам. В примере с `__init__` мы создаем атрибуты для конкретного экземпляра и присваиваем им значения аргументов метода. Важно использовать параметр `self` внутри метода, если мы хотим сохранить значения экземпляра для последующего использования.

В большинстве случаев нам также необходимо использовать параметр `self` в других методах, потому что при вызове метода первым аргументом, который ему передается, является сам объект. Давайте добавим метод к нашему классу **River** и посмотрим, как он будет работать. Синтаксис методов на данный момент не важен, просто обратите внимание на использование `self`:

6. Как добавить атрибуты в класс?

### Добавление атрибутов

В дополнение к изменению атрибутов мы также можем создавать атрибуты для класса или конкретного экземпляра. Например, мы хотим видеть информацию о всех видах наших питомцев. Мы могли бы записать ее в самом классе с самого начала или создать переменную следующим образом:

```
Pet.all_specs = [tom.spec, avocado.spec, ben.spec]

tom.all_specs      # ["cat", "dog", "goldfish"]
avocado.all_specs  # ["cat", "dog", "goldfish"]
ben.all_specs      # ["cat", "dog", "goldfish"]
```

Еще мы могли бы создать атрибут для конкретного экземпляра. Например, мы хотим вспомнить породу собаки под именем **Avocado**. Про породы чаще говорят применительно к собакам (у кошек тоже есть породы, но они не так сильно различаются), поэтому имеет смысл, чтобы только у нее был атрибут с такой информацией:

```
avocado.breed = "corgi"
```

Здесь мы создали атрибут `breed` для объекта `avocado` и присвоили ему значение `corgi`. Другие экземпляры класса **Pet** , а также самого класса не имеют этого атрибута, поэтому следующие строки кода могут вызвать ошибку:

```
Pet.breed  # AttributeError
tom.breed  # AttributeError
ben.breed  # AttributeError
```

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

Если вы знакомы с языками программирования *Java*, *C#*, *C++* то, наверное, уже задались вопросом: “а как управлять уровнем доступа?”. В перечисленных языка вы можете явно указать для переменной, что доступ к ней снаружи класса запрещен, это делается с помощью ключевых слов (*private*, *protected* и т.д.). В *Python* таких возможностей нет, и любой может обратиться к атрибутам и методам вашего класса, если возникнет такая необходимость. Это существенный недостаток этого языка, т.к. нарушается один из ключевых принципов ООП – инкапсуляция. Хорошим тоном считается, что для чтения/изменения какого-то атрибута должны использоваться специальные методы, которые называются *getter*/*setter*, их можно реализовать, но ничего не помешает изменить атрибут напрямую. При этом есть соглашение, что метод или атрибут, который начинается с нижнего подчеркивания, является скрытым, и снаружи класса трогать его не нужно (хотя сделать это можно).