

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 4.1

**«Элементы объектно-ориентированного программирования в
языке Python»**

по дисциплине «Основы программной инженерии»

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

« » мая 2023 г.

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.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: <https://github.com/afk552/lab4.1>

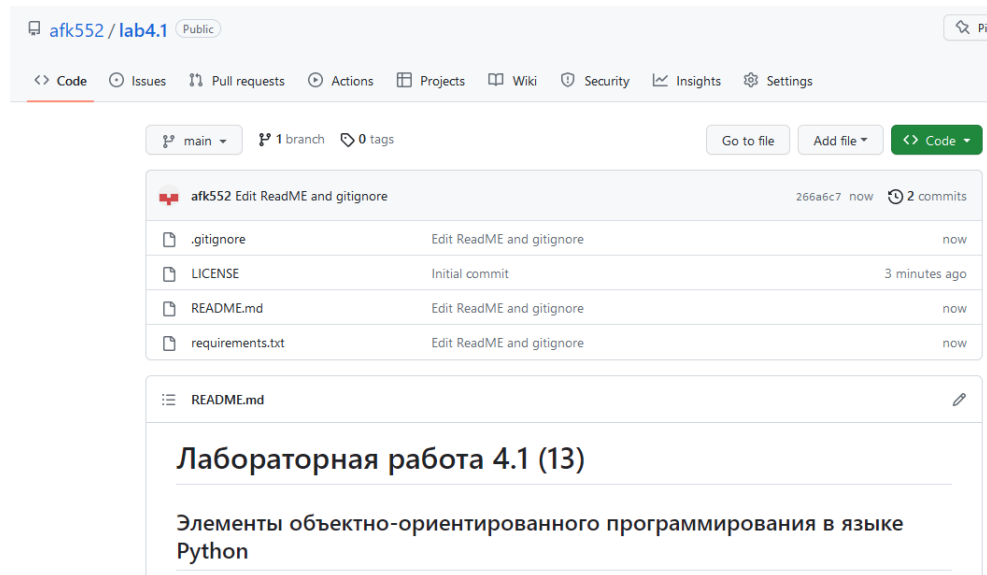


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

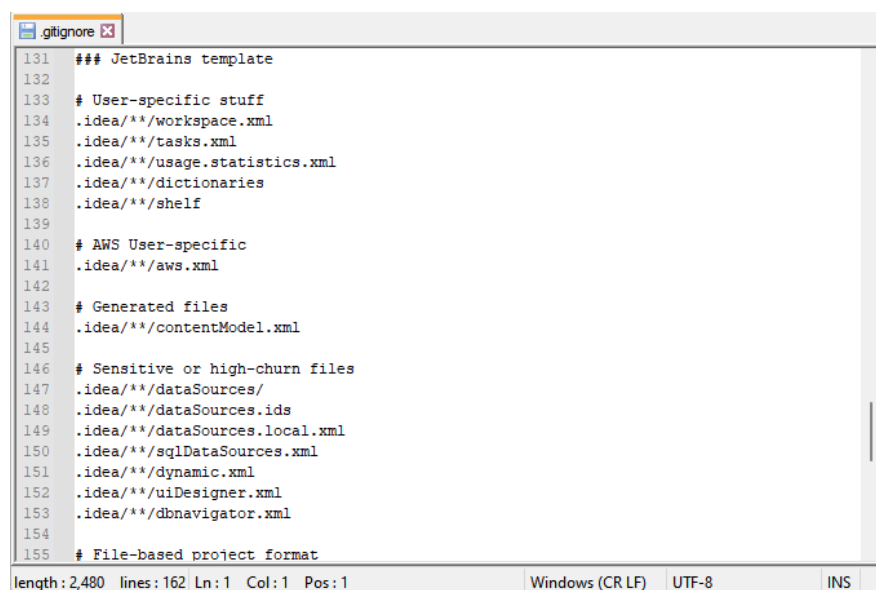


Рисунок 2 – Окно блокнота

Организируйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab4.1>git checkout -b develop  
Switched to a new branch 'develop'
```

Рисунок 3 – Окно командной строки

Примеры

```
1  #!/usr/bin/env python3  
2  # -*- coding: utf-8 -*-  
3  
4  
5  3 usages  
6  class Book:  
7      material = "paper"  
8      cover = "paperback"  
9      all_books = []  
10  
11  if __name__ == '__main__':  
12      print(Book.material)  
13      print(Book.cover)  
14      print(Book.all_books)  
15
```

Run ex1 x

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex1.py  
paper  
paperback  
[]  
  
Process finished with exit code 0
```

Рисунок 4 – Пример 1

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  5 usages
6  class River:
7      # список всех рек
8      all_rivers = []
9
10     def __init__(self, name, length):
11         self.name = name
12         self.length = length
13         # добавляем текущую реку в список всех рек
14         River.all_rivers.append(self)
15
16  ▶ if __name__ == '__main__':
17     volga = River("Волга", 3530)
18     seine = River("Сена", 776)
19     nile = River("Нил", 6852)
20     # далее печатаем все названия рек
21     for river in River.all_rivers:
22         print(river.name)
```

if __name__ == '__main__' > for river in River.all_rivers

Run ex2

C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex2.py

Волга
Сена
Нил

Process finished with exit code 0

Рисунок 5 – Пример 2

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  4 usages
6  class River:
7      all_rivers = []
8
9      def __init__(self, name, length):
10         self.name = name
11         self.length = length
12         River.all_rivers.append(self)
13
14     3 usages
15     def get_info(self):
16         print("Длина {0} равна {1} км".format(self.name, self.length))
17
18  ▶ if __name__ == '__main__':
19     volga = River("Волга", 3530)
20     seine = River("Сена", 776)
21     nile = River("Нил", 6852)
22     volga.get_info()
23     seine.get_info()
24     nile.get_info()
```

Run ex3

C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex3.py

Длина Волга равна 3530 км
Длина Сена равна 776 км
Длина Нил равна 6852 км

Process finished with exit code 0

Рисунок 6 – Пример 3

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  1 usage
6  class Ship:
7      def __init__(self, name, capacity):
8          self.name = name
9          self.capacity = capacity
10         self.cargo = 0
11
12         2 usages
13         def load_cargo(self, weight):
14             if self.cargo + weight >= self.capacity:
15                 self.cargo += weight
16                 print("Loaded {} tons".format(weight))
17             else:
18                 print("Cannot load that much")
19
20         3 usages
21         def unload_cargo(self, weight):
22             if self.cargo - weight <= 0:
23                 self.cargo -= weight
24                 print("Unloaded {} tons".format(weight))
25             else:
26                 print("Cannot unload that much")
27
28         1 usage
29         def name_captain(self, cap):
30             self.captain = cap
31             print("{} is the captain of the {}".format(self.captain, self.name))
32
33     if __name__ == '__main__':
34         black_pearl = Ship("Black Pearl", 800)
35         black_pearl.name_captain("Jack Sparrow")
36         print(black_pearl.captain)
37         black_pearl.load_cargo(600)
38         black_pearl.unload_cargo(400)
39         black_pearl.load_cargo(700)
40         black_pearl.unload_cargo(300)

```

Ship > name_captain()

Run ex4

C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex4.py

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

Рисунок 7 – Пример 4

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  1 usage
6  class Rectangle:
7      def __init__(self, width, height):
8          self.__width = width
9          self.__height = height
10
11         3 usages
12         @property
13         def width(self):
14             return self.__width
15
16         1 usage
17         @width.setter
18         def width(self, w):
19             if w > 0:
20                 self.__width = w
21             else:
22                 raise ValueError
23
24         3 usages
25         @property
26         def height(self):
27             return self.__height
28
29         1 usage
30         @height.setter
31         def height(self, h):
32             if h > 0:
33                 self.__height = h
34
35         29         else:
36             raise ValueError
37
38         32         def area(self):
39             return self.__width * self.__height
40
41     if __name__ == '__main__':
42         rect = Rectangle(10, 20)
43         print(rect.width)
44         print(rect.height)
45         rect.width = 50
46         print(rect.width)
47         rect.height = 70
48         print(rect.height)

```

Rectangle

Run ex5

C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex5.py

10

20

50

70

Рисунок 8 – Пример 5

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 13 usage:
6 class Rational:
7     def __init__(self, a=0, b=1):
8         a = int(a)
9         b = int(b)
10         if b == 0:
11             raise ValueError()
12         self.__numerator = abs(a)
13         self.__denominator = abs(b)
14         # Сокращение дроби
15
16 2 usage:
17 def __reduce(self):
18     # Функция для нахождения наибольшего общего делителя
19     def gcd(a, b):
20         if a == 0:
21             return b
22         elif b == 0:
23             return a
24         elif a >= b:
25             return gcd(a % b, b)
26         else:
27             return gcd(a, b % a)
28
29     c = gcd(self.__numerator, self.__denominator)
30     self.__numerator //= c
31     self.__denominator //= c
32
33 14 usage:
34 @property
35 def numerator(self):
36     return self.__numerator
37
38 10 usage:
39 @property
40 def denominator(self):
41     return self.__denominator
42
43 # Прочитать значение дроби с клавиатуры. Дробь вводится
44
45 # new a/b.
46 1 usage:
47 def read(self, prompt=None):
48     line = input() if prompt is None else input(prompt)
49     parts = list(map(int, line.split('/', maxsplit=1)))
50     if parts[1] == 0:
51         raise ValueError()
52     self.__numerator = abs(parts[0])
53     self.__denominator = abs(parts[1])
54     self.__reduce()
55
56 # Вывести дробь на экран
57 6 usage:
58 def display(self):
59     print(f"{self.__numerator}/{self.__denominator}")
60
61 # Сложение обыкновенных дробей.
62 1 usage:
63 def add(self, rhs):
64     if isinstance(rhs, Rational):
65         a = self.__numerator * rhs.__denominator + \
66             self.__denominator * rhs.__numerator
67         b = self.__denominator * rhs.__denominator
68         return Rational(a, b)
69     else:
70         raise ValueError()
71
72 # Вычитание обыкновенных дробей.
73 1 usage:
74 def sub(self, rhs):
75     if isinstance(rhs, Rational):
76         a = self.__numerator * rhs.__denominator - \
77             self.__denominator * rhs.__numerator
78         b = self.__denominator * rhs.__denominator
79         return Rational(a, b)
80     else:
81         raise ValueError()
82
83 # Умножение обыкновенных дробей.
84 1 usage:
85 def mul(self, rhs):
86     if isinstance(rhs, Rational):
87         a = self.__numerator * rhs.__numerator
88         b = self.__denominator * rhs.__denominator
89         return Rational(a, b)
90     else:
91         raise ValueError()
92
93 return Rational(a, b)
94 else:
95     raise ValueError()
96
97 # Деление обыкновенных дробей.
98 1 usage:
99 def div(self, rhs):
100     if isinstance(rhs, Rational):
101         a = self.__numerator * rhs.__denominator
102         b = self.__denominator * rhs.__numerator
103         return Rational(a, b)
104     else:
105         raise ValueError()
106
107 # Отношение обыкновенных дробей.
108 def equals(self, rhs):
109     if isinstance(rhs, Rational):
110         return (self.__numerator == rhs.__numerator) and \
111             (self.__denominator == rhs.__denominator)
112     else:
113         return False
114
115 def greater(self, rhs):
116     if isinstance(rhs, Rational):
117         v1 = self.__numerator / self.__denominator
118         v2 = rhs.__numerator / rhs.__denominator
119         return v1 > v2
120     else:
121         return False
122
123 def less(self, rhs):
124     if isinstance(rhs, Rational):
125         v1 = self.__numerator / self.__denominator
126         v2 = rhs.__numerator / rhs.__denominator
127         return v1 < v2
128     else:
129         return False
130
131 if __name__ == '__main__':
132     r1 = Rational(3, 4)
133     r1.display()
134     r2 = Rational()
135     r2.read("Введите обыкновенную дробь: ")
136     r2.display()
137     r3 = r2.add(r1)
138     r3.display()
139     r4 = r2.sub(r1)
140     r4.display()
141     r5 = r2.mul(r1)
142     r5.display()
143     r6 = r2.div(r1)
144     r6.display()

```

Рисунок 9 – Пример 6

Индивидуальное задание.

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.

7. Поле *first* — дробное число, левая граница диапазона; поле *second* — дробное число, правая граница диапазона. Реализовать метод `rangecheck()` — проверку заданного числа на принадлежность диапазону.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  1 usage
6
7  class Number:
8
9      def __init__(self, first=0.0, second=0.0):
10         try:
11             self.__first = float(first)
12             self.__second = float(second)
13         except ValueError:
14             ValueError("Проверьте правильность ввода!")
15
16
17  6 usages
18  @property
19  def first(self):
20      return self.__first
21
22  6 usages
23  @property
24  def second(self):
25      return self.__second
26
27  1 usage
28  def read(self):
29      lower_bound = float(input("Введите левую границу диапазона: "))
30      upper_bound = float(input("Введите правую границу диапазона: "))
31      if isinstance(lower_bound, float) and isinstance(upper_bound, float):
32          self.__first = float(lower_bound)
33          self.__second = float(upper_bound)
34      else:
35          raise ValueError("Проверьте введенные границы диапазона!")
36
37  1 usage
38  def display(self):
39      print(f"1: {self.first}")
40      print(f"2: {self.second}")
41
42  34
43  def __contains__(self, number):
44      return self.first <= number <= self.second
45
46
47  38
48  def rangecheck(self, number=None):
49      if self.first > self.second:
50          raise ValueError("Введенная левая граница больше правой!")
51      if isinstance(number, float):
52          pass
53      else:
54          try:
55              number = float(input("Задайте число: "))
56          except ValueError:
57              ValueError("Введено не число!")
58      if self.first <= float(number) <= self.second:
59          print(f"Число {number} в диапазоне [{self.first}, {self.second}]")
60      else:
61          print(f"Число {number} не в [{self.first}, {self.second}]")
62
63  54
64  if __name__ == "__main__":
65      test = Number()
66      test.read()
67      test.display()
68      test.rangecheck()
69      test.rangecheck(23.2)
```

Рисунок 10 – Код программы индивидуального задания 1

```
Введите левую границу диапазона: 12.5
Введите правую границу диапазона: 34.8
1: 12.5
2: 34.8
Задайте число: 13.7
Число 13.7 в диапазоне [12.5, 34.8]
Число 23.2 в диапазоне [12.5, 34.8]

Process finished with exit code 0
```

Рисунок 11 – Результат выполнения программы индивидуального задания 1

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` .

7. Создать класс *Date* для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями типа *unsigned int*: для года, месяца и дня. Класс должен включать не менее трех функций инициализации: числами, строкой вида «год.месяц.день» (например, «2004.08.31») и датой. Обязательными операциями являются: вычисление даты через заданное количество дней, вычитание заданного количества дней из даты, определение високосности года, присвоение и получение отдельных частей (год, месяц, день), сравнение дат (равно, до, после), вычисление количества дней между датами.

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\indiv_task\task2.py
1.1.2002
-----
Введите дату: (ГГГГ.ММ.ДД): 2002.12.03
3.12.2002
-----
Введите день: 13
Введите месяц: 04
Введите год: 2005
13.4.2005
-----
20.5.2002
*****
Разница между 2002-12-03 и 2002-01-01 составляет 336 дней.
Первая дата 2002-12-03 больше второй 2002-01-01.
2002-01-04
2001-12-27

Process finished with exit code 0
```

Рисунок 12 – Результат выполнения индивидуального задания 2


```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import datetime as dt
5
6  """
7  Создать класс Date для работы с датами в формате «год.месяц.день».
8  Дата представляется структурой с тремя полями типа unsigned int:
9  для года, месяца и дня. Класс должен включать не менее трех функций
10 инициализации: числами, строкой вида «год.месяц.день»
11 (например, «2004.08.31») и датой. Обязательными операциями являются:
12 вычисление даты через заданное количество дней,
13 вычитание заданного количества дней из даты,
14 определение високосности года, присвоение и получение отдельных частей
15 (год, месяц, день), сравнение дат (равно, до, после),
16 вычисление количества дней между датами.
17 """
18
19
20 4 usages
21 class Date:
22     # Конструктор
23     def __init__(self, day=1, month=1, year=2000):
24         self.__day = int(day)
25         self.__month = int(month)
26         self.__year = int(year)
27
28     # Свойства (для присвоения и получения отдельных частей)
29     3 usages
30     @property
31     def day(self):
32         return self.__day
33
34     1 usage
35     @day.setter
36     def day(self, d):
37         if 1 <= d <= 31:
38             self.__day = int(d)
39         else:
40             raise ValueError
41
42     3 usages
43     @property
44     def month(self):
45         return self.__month
46
47     1 usage
48     @month.setter
49     def month(self, m):
50         if 1 <= m <= 12:
51             self.__month = int(m)
52         else:
53             raise ValueError
54
55     5 usages
56     @property
57     def year(self):
58         return self.__year
59
60     1 usage
61     @year.setter
62     def year(self, y):
63         self.__year = int(y)
64
65     # Ввод даты в виде строки -> экземпляр класса
66     1 usage
67     def read_str(self):
68         date_str = input("Введите дату: (ГГГГ.ММ.ДД): ")
69         try:
70             self.__year, self.__month, self.__day = map(
71                 int, date_str.split(".")
72             )
73         except ValueError:
74             raise ValueError("Задана строка неверного типа!")
75
76     # Ввод даты числами -> экземпляр класса
77     1 usage
78     def read_nums(self):
79         day = int(input("Введите день: "))
80         month = int(input("Введите месяц: "))
81         year = int(input("Введите год: "))
82         self.__day, self.__month, self.__year = int(day), int(month), int(year)
83
84     # Конвертация даты экземпляра класса -> дата типа datetime
85     8 usages (2 dynamic)
86     def dt_from_class(self):
87         return dt.date(self.year, self.month, self.day)
88
89
90     return True
91 else:
92     return False
93
94     # Вычисление даты через заданное количество дней
95     1 usage
96     def add_days(self, added_days):
97         curr_date = self.dt_from_class()
98         new_date = curr_date + dt.timedelta(days=added_days)
99         return new_date
100
101     # Вычитание заданного количества дней из даты
102     1 usage
103     def subtract_days(self, removed_days):
104         curr_date = self.dt_from_class()
105         new_date = curr_date - dt.timedelta(days=removed_days)
106         return new_date
107
108     # Сравнение дат (экземпляр класса vs другой экземпляр класса)
109     1 usage
110     def compare(self, to_compare):
111         curr_date = self.dt_from_class()
112         date_to_compare = to_compare.dt_from_class()
113         if curr_date == date_to_compare:
114             print("Даты одинаковы!")
115         elif curr_date > date_to_compare:
116             print(f"Первая дата {curr_date} больше второй {date_to_compare}.")
117         elif curr_date < date_to_compare:
118             print(f"Вторая дата {date_to_compare} больше первой {curr_date}.")
119
120     # Вычисление количества дней между датами
121     1 usage
122     def days_between(self, to_compare):
123         curr_date = self.dt_from_class()
124         date_to_compare = to_compare.dt_from_class()
125         delta = (curr_date - date_to_compare).days
126         return delta
127
128     if __name__ == "__main__":
129         # Установка даты при создании экземпляра класса
130         date1 = Date(1, 1, 2002)
131         date1.display()
132         print("-" * 20)
133         # Установка даты вводом строки вида "ГГГГ.ММ.ДД"
134         date2 = Date()
135         date2.read_str()
136         date2.display()
137         print("-" * 20)
138         # Установка даты вводом чисел по отдельности
139         date3 = Date()
140         date3.read_nums()
141         date3.display()
142         print("-" * 20)
143         # Установка даты при помощи свойств
144         date4 = Date()
145         date4.day = 20
146         date4.month = 5
147         date4.year = 2002
148         date4.display()
149         print("****" * 20)
150
151         if date1.is_leap_year():
152             print(f"Год {date1.year} - високосный!")
153         else:
154             print(f"Год {date1.year} не является високосным!")
155
156         print(
157             f"Разница между {date2.dt_from_class()} и {date1.dt_from_class()} "
158             f"составляет {date2.days_between(date1)} дней."
159         )
160
161         date2.compare(date1)
162         print(date1.add_days(3))
163         print(date1.subtract_days(5))

```

Рисунок 13 – Код программы индивидуального задания 2

Вывод: Были приобретены и применены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python.

Контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

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

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибут класса – это атрибут, общий для всех экземпляров класса. Атрибуты класса определены внутри класса, но вне каких-либо методов. Их значения одинаковы для всех экземпляров этого класса. Так что вы можете рассматривать их как тип значений по умолчанию для всех наших объектов.

Что касается переменных экземпляра, они хранят данные, уникальные для каждого объекта класса. Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово.

3. Каково назначение методов класса?

Методы определяют функциональность объектов, принадлежащих конкретному классу.

4. Для чего предназначен метод `__init__()` класса?

Метод `__init__` является конструктором. Конструкторы – это концепция объектноориентированного 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?

Аргумент `self` представляет конкретный экземпляр класса и позволяет нам получить доступ к его атрибутам и методам.

6. Как добавить атрибуты в класс?

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"
```

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

В Python 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 мешает изменить атрибут напрямую.

```
rect = Rectangle(10, 20)
```

```
rect.get_width() #10
```

```
rect._width #10
```

8. Каково назначение функции isinstance?

Встроенная функция `isinstance(obj, Cls)` , используемая при реализации методов арифметических операций и операций отношения, позволяет узнать что некоторый объект `obj` является либо экземпляром класса `Cls` либо экземпляром одного из потомков класса `Cls`.