Билет 1

1. Что такое Seaborn. Для чего используется Seaborn.

   Seaborn - это библиотека визуализации данных Python, предоставляющая высокоуровневый интерфейс для рисования привлекательных и информативных статистических графиков.

2. Какой командой в Seaborn можно установить тему графика?

```
import seaborn as sns
sns.set_theme()
```

Практика

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme()


class DataAnalysis:
    '''
    Напишите функцию pos_add(a, b), которая возвращает положительное значение
сложения двух целых чисел.
    '''

    def __init__(self, filename):
        self._filename = filename
        self._data = sns.load_dataset('tips')

    def buildGraph(self):
        graph = sns.relplot(
            data=self._data,
            x='total_bill',
            y='tip',
            col='time',
            hue='sex',
            style='smoker',
            size='size'
        )
        plt.savefig('ticket1.png')
        plt.show(block=True)

    @property
    def data(self):
        return self._data
```

```
task = DataAnalysis('../assets/tips.csv')
task.buildGraph()
```

Билет 2

1. Сколько (встроенных) типов чисел есть в языке Python. Напишите их на латинице.
    - INT
    - FLOAT
    - COMPLEX
2. Если частное двух целых чисел (int) не является целым числом, то оно будет преобразовано...
    - float

Практика

```
class Number:
    '''
    Напишите функцию pos_add(a, b), которая возвращает положительное значение
сложения двух целых чисел.
    '''

    def __init__(self, num=None):
        self._num = num

    def __add__(self, other):
        return abs(self._num + other._num)


print(Number(3) + Number(1))
```

Билет 3

1. Строки в языке Python (полное определение)

   Строка – это последовательность символов, заключенных в одинарные или двойные кавычки.

2. Строки байтов – bytes и bytearray.

   Байт - минимальная единица хранения и обработки цифровой информации. Последовательность байт представляет собой какую-либо информацию.

```
>>> b'bytes'
b'bytes'
>>> 'Байты'.encode('utf-8')
b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'
>>> bytes('bytes', encoding = 'utf-8')
b'bytes'
>>> bytes([50, 100, 76, 72, 41])
b'2dLH)'
```

Практика

```
class DataAnalysis:
    '''
    Определить, встречается ли в строке буква 'я'.
    Вывести на экран ее позицию (индекс) в строке.
    Определить, сколько раз в строке встречается буква 'у'.
    Определить длину строки.
    '''

    def __init__(self, string=None):
        self._string = string

    def findLetter(self, letter):
        return self._string.index(letter) if True else letter in self._string

    def countLetter(self, letter):
        return f'Длина строки: {len(self._string)}, буква: {letter} встретилась ->
{self._string.find(letter)}'


task = DataAnalysis('У лукоморья 123 дуб зеленый 456')
print(task.findLetter('я'))
print(task.countLetter('у'))
```

Билет 4

1. Списки в Python — это. С помощью чего можно создать список?
    - []
    - list()
    - [x for x in string]
2. Функция list(). Для чего используется?

    Создать список

Практика

```
class DataAnalysis:
    '''
    Напишите функцию change(lst), которая принимает список и меняет местами его
первый и последний элемент.
    В исходном списке минимум 2 элемента.
    '''

    def __init__(self, array=None):
        self._array = array

    def change(self):
        self._array[0], self._array[1] = self._array[1], self._array[0]


list_ = [1, 4, 5, 2, 7]
task = DataAnalysis(list_)
task.change()
print(list_)
```

Билет 5

1. Словари в Python

   Словарь - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

2. Ключевые отличия словарей от списков

   Список и словарь представляют собой принципиально разные структуры данных. Список может хранить последовательность объектов в определенном порядке, чтобы вы могли индексироваться в список или перебирать список. Более того, List является изменчивым типом, означающим, что списки могут быть изменены после их создания. Словарь Python представляет собой реализацию хеш-таблицы и является хранилищем ключей. Он не упорядочен и требует, чтобы клавиши были хэш-таблицами. Кроме того, это быстрый поиск по ключу.

   Элементы в списке имеют следующие характеристики:

   - Они сохраняют свой порядок, если явно не упорядочены (например, путем сортировки списка).
   - Они могут быть любого типа, и типы могут быть смешаны.
   - Доступ к ним осуществляется с помощью числовых (нулевых) индексов.

   Элементы в словаре имеют следующие характеристики:

   - Каждая запись имеет ключ и значение.
   - Доступ к элементам осуществляется с использованием значений ключа.
   - Ключевыми значениями могут быть любые типы хеш-таблиц (т. е. не dict), и типы могут быть смешаны.
   - Значения могут быть любого типа (включая другие dict's), а типы могут быть смешанный.

   Использование:

   - Используйте словарь, если у вас есть набор уникальных ключей, которые сопоставляются с значениями и используют список, если у вас есть упорядоченный набор элементов.

Практика

```
class DataAnalysis:
    '''
    Напишите функцию to_dict(lst), которая принимает аргумент в виде списка и
возвращает словарь,
    в котором каждый элемент списка является и ключом, и значением.
    Предполагается, что элементы списка будут соответствовать правилам задания
ключей в словарях.
    '''

    def __init__(self, list_=None):
```

```
        self._list = list_
        print(self._list)

    def to_dict(self):
        return {key: value for key, value in zip(self._list, self._list)}


task = DataAnalysis([1, 3, 'fgd', 'dfg', (3, 4)])
print(task.to_dict())
```

Билет 6

1. Множества в Python

   Множество в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке.

2. Что будет если попытаться создать множества из элементов изменяемого типа?

   Элементами множеств могут быть строки, числа, списки, словари.

   Возьмет уникальные значения, в случае словорей - ключи.

Практика

```
class DataAnalysis:
    '''
    На входе функция to_set() получает строку или список чисел.
    Преобразуйте их в множество.
    На выходе должно получиться множество и его мощность.
    '''

    def __init__(self, arg):
        self._arg = arg

    def to_set(self):
        return f'{set(self._arg)}, мощность: {len(set(self._arg))}'


task = DataAnalysis([4, 5, 5, 6, 'dgb'])
print(task.to_set())
```

Билет 7

1. Как распределить категориальные данные в Seaborn. Какой функцией можно объединить графики?

   График разброса по категориям удобен для небольших наборов данных, так как по мере увеличения количества точек они все равно начнут перекрывать друг друга и сливаться. Что бы преодолеть эти трудности, лучше воспользоваться графиками, которые сами содержат некоторую информацию о распределении внутри категорий. Один из таких графиков - это "ящик с усами" или boxplot. Его можно построить с помощью той же функции catplot с параметром kind, установленным в значение 'box'.

   Чтобы объединить несколько кривых на одном графике, просто перечислите их в функции plot(), при этом для каждой из них можно задать собственные парамеры кривых с помощью соответствующих "кодов". Следующий пример не только отрисовывает два графика в одном окне, но также делает подписи осей и отображает заголовок.

   ```
   plt.plot(X,Y, X,np.cos(X))       # объединение 2-х графиков
   plt.legend(('sin','cos'))           # подписи
   plt.title('Trigonometry')        # заголовок
   plt.xlabel('Time, s')               # наименование оси абсцисс
   plt.ylabel('Amplitude, c.u.')  # наименование оси ординат
   plt.show()
   ```

2. Если воспользоваться классами для управления сеткой ячеек, в которой располагаются графики, то можно...

   получить доступ к более тонким настройкам всей композиции графиков.

Практика

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme()


class DataAnalysis:
    '''
    Постройте простой график на основе встроенного набора данных flights,
    который содержит информацию о месячном объеме пассажироперевозок в период с
1945 по 1960 год.
    Строки – сколько пассажиров было перевезено в определенном месяце
определенного года,
    столбцы – год, месяц и количество.
```

```
    '''

    def __init__(self, filename):
        self._filename = filename
        self._flights = sns.load_dataset("flights")

    def buildGraph(self):
        sns.relplot(
            data=self._flights,
            x='year',
            y='passengers',
            kind='line'
            )
        plt.savefig(self._filename)
        plt.show(block=True)

    @property
    def data(self):
        return self._data

task = DataAnalysis('ticket7.png')
task.buildGraph()
```

Билет 8

1. Дайте определения понятия «анализ данных».
    - Область математики и информатики, занимающаяся построением и исследованием наиболее общих математических методов и вычислительных алгоритмов извлечения знаний из экспериментальных (в широком смысле) данных.
    - Процесс исследования, фильтрации, преобразования и моделирования данных с целью извлечения полезной информации и принятия решений.
    - Совокупность методов и приложений, связанных с алгоритмами обработки данных и не имеющих четко зафиксированного ответа на каждый входящий объект.
2. Три этапа процесса управления больших объемов данных.
    - Извлечение данных
    - Подготовка данных
    - Исследование и визуализация данных

Практика

```
class DataAnalysis:
    '''
    Дано 3 числа. Найти минимальное среди них и вывести на экран.
    '''

    def __init__(self, *args):
        self._num1, self._num2, self._num3 = args

    def getMinNum(self):
        return min([self._num1, self._num2, self._num3])


task = DataAnalysis(1, 2, 3)
print(task.getMinNum())
```

Билет 9

1. Сбор и подготовка данных. Виды. Источники данных.

   Подготовка данных включает такие процессы:

   - получение,
   - очистка,
   - нормализация,
   - превращение в оптимизированный набор данных.
   - Обычно это табличная форма, которая идеально подходит для этих методов, что были запланированы на этапе проектировки.

   Источник данных – это контейнер, содержащий данные, которые вы загружаете в Аналитику. Источники данных определяют, как загруженная информация объединяется с существующей. Источники настраиваются на уровне ресурса. Загружать данные из одного и того же источника данных можно многократно.

   Примеры: Радио, телепередачи, газеты, журналы, выставки, конференции, сайты Интернета.

2. Три основных этапа сбора и подготовки данных

Нормирование

Нормировка – это корректировка значений в соответствии с некоторыми функциями преобразования, с целью сделать их более удобными для сравнения.

Нормировка данных требуется, когда несовместимость единиц измерений переменных может отразиться на результатах и рекомендуется, когда итоговые отчеты могут быть улучшены, если выразить результаты в определенных понятных/совместимых единицах. Например, время реакции, записанное в миллисекундах, легче интерпретировать, чем число тактов процессора, в которых были получены данные эксперимента.

Например, разделив набор измерений о росте людей в дюймах на 2.54, мы получим значение роста в метрической системе.

Форматирование

Операции **форматирования** включают в себя выбор выравнивания абзацев, отступов и отбивок между абзацами, обтекания отдельных абзацев, а также видов и начертаний шрифтов. Эти операции выполняются различными текстовыми процессорами с разной степенью автоматизации.

Для удобства просмотра данных можно установить отображение отрицательных чисел красным цветом. Знак минус (-) при этом можно отображать или не отображать.

Кодирование – это представление категориальных данных в числовой форме. Например, при бинарной классификации один из классов можно представить числом `0`, а другой класс – числом `1`.

Кодирование

LabelEncoder

1. Категориальная особенность является порядковой (например, низкий, средний, высокий)
2. Количество категорий довольно велико, и **One-Hot-Encoding** может привести к высокому потреблению памяти.

Этот подход очень прост и включает преобразование каждого значения в столбце в число. Рассмотрим набор данных, в котором описываются категориальные признаки мостов:

Тип моста
Arch
Beam
Truss
Cantilever
Tied Arch
Suspension
Cable

Теперь закодируем текстовые значения. Позиция текстового значения в наборе данных будет являться его числовым значением:

Тип моста (Текст)	**Тип моста (Числовое значение)**
Arch	0
Beam	1
Truss	2
Cantilever	3
Tied Arch	4
Suspension	5
Cable	6

Проблема использования такого метода состоит в том, что они вводят сравнение между ними. Нет никакой связи между различными типами мостов, но, глядя на эти числа, можно подумать, что тип моста «Cable» имеет более высокий приоритет над типом моста «Arch». Алгоритм может неправильно понимать, что данные имеют какой-то порядок 0 <1 <2… <6 и могут придать «Cable» в 6 раз больший вес, чем типу моста «Arch».

Уровень безопасности (Текстовое значение)	**Уровень безопасности (Числовое значение)**
Никакой	0
Низкий	1
Средний	2
Высокий	3

Уровень безопасности (Текстовое значение)	Уровень безопасности (Числовое значение)
Очень высокий	4

Кодирование Label-Encoder в этом столбце также вызывает приоритет по числу, но правильным способом. Здесь числовой порядок не выглядит нестандартным, и будет иметь смысл, если алгоритм интерпретирует порядок безопасности 0 < 1 < 2 < 3 < 4, т. е. никакой <низкий <средний <высокий <очень высокий.

One-Hot Encoder

1. Категориальная особенность не является порядковой (как типы мостов в примере)
2. Небольшое количество категорий.

Хотя **Label-Encoder** является прямолинейным, но у него есть недостаток, заключающийся в том, что числовые значения могут быть неверно истолкованы алгоритмами как имеющие некоторый порядок. Эта проблема упорядочения решается с помощью другого распространенного альтернативного подхода, называемого «One-Hot-Encoding». В этом алгоритме каждое значение категории преобразуется в новый столбец, и столбцу присваивается значение `1` или `0` (обозначение `true` / `false`). Давайте рассмотрим предыдущий пример типов мостов и уровней безопасности с **One-Hot-Encoding**.

Тип моста (Текст)	Тип моста (Arch)	Тип моста (Beam)	Тип моста (Truss)	Тип моста (Cantileve)	Тип моста (Tied Arch)	Тип моста (Suspension)	Тип моста (Cable)
Arch	1	0	0	0	0	0	0
Beam	0	1	0	0	0	0	0
Truss	0	0	1	0	0	0	0
Cantilever	0	0	0	1	0	0	0
Tied Arch	0	0	0	0	1	0	0
Suspension	0	0	0	0	0	1	0
Cable	0	0	0	0	0	0	1

Уровень безопасности (Текст)	Уровень безопасности (Никакой)	Уровень безопасности (Низкий)	Уровень безопасности (Средний)	Уровень безопасности (Высокий)	Уровень безопасности (Очень высокий)
Никакой	1	0	0	0	0
Низкий	0	1	0	0	0
Средний	0	0	1	0	0
Высокий	0	0	0	1	0
Очень высокий	0	0	0	0	1

Строки с первым значением столбца (Arch/Никакой) будут иметь значение `1` (указывает на истинное значение), а столбцы других значений будут иметь значение `0` (указывающее на ложное значение). Аналогично для других строк сопоставляется значение со значением столбца.

Хотя этот подход устраняет проблемы порядка, но имеет недостаток в добавлении большего количества столбцов в набор данных. Это может привести к значительному увеличению количества столбцов, если у вас есть много уникальных значений в столбце категории. В приведенном выше примере это было управляемо, но управлять им будет действительно сложно, когда кодирование дает много столбцов

Практика

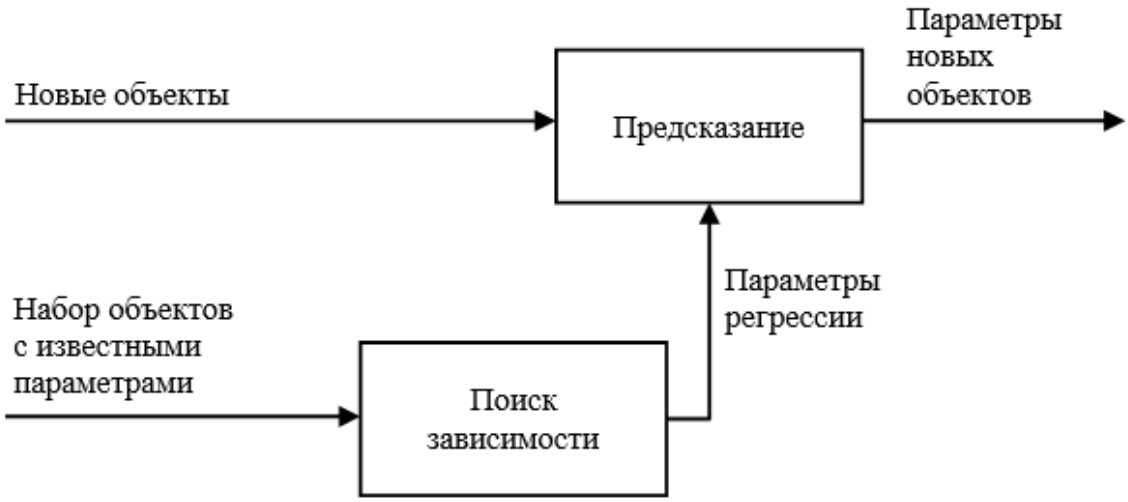
```
class DataAnalysis:
    '''
    Даны три целых числа. Выбрать из них те, которые принадлежат интервалу [1, 3].
    '''

    def __init__(self, lowerLimit=None, upperLimit=None, *nums):
        self._lowerLimit = lowerLimit
        self._upperLimit = upperLimit
        self._num1, self._num2, self._num3 = nums
        self._numsInBetween = []

    def getNumsInBetween(self):
        for num in [self._num1, self._num2, self._num3]:
            if num >= 1 and num <= 3:
                self._numsInBetween.append(num)
        return self._numsInBetween


task = DataAnalysis(1, 3, 1, 4, 6)
print(task.getNumsInBetween())
```

Билет 10

1. Что такое регрессионный анализ.

   Регрессионный анализ — метод моделирования измеряемых данных и исследования их свойств. Данные состоят из пар значений зависимой переменной (переменной отклика) и независимой переменной (объясняющей переменной).

2. Нарисуйте схему применения регрессии (два этапа).



Практика

```
class DataAnalysis:
    '''
    Найти сумму n элементов следующего ряда чисел: 1 -0.5 0.25 -0.125 ... n.
    Количество элементов (n) вводится с клавиатуры.
    Вывести на экран каждый член ряда и его сумму.
    Решить задачу используя циклическую конструкцию for.
    '''

    def __init__(self, array=None):
        self._array = array
        self._sum = 0
        self._calcSum()

    def _calcSum(self):
        for num in self._array:
            self._sum += num

    def _showArray(self):
        for num in self._array:
            print(num)
```

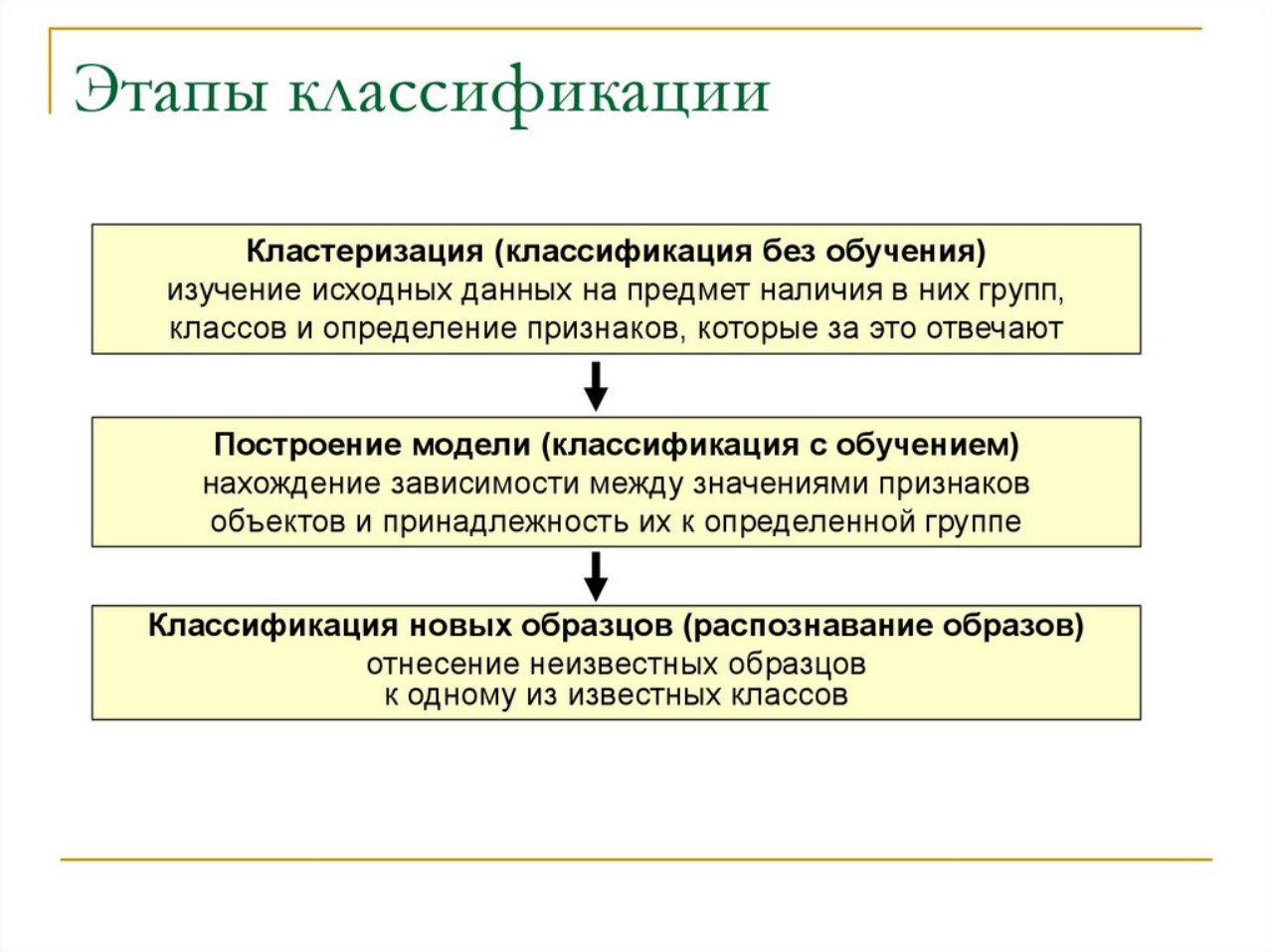
```
    def showArrayInfo(self):
        print('Ряд:')
        self._showArray()
        print(f'Сумма: {self.sum}')

    @property
    def sum(self):
        return self._sum


task = DataAnalysis(list(map(int, input('-> ').split())))
task.showArrayInfo()
```

Билет 11

1. Классификация данных – это.

   Классификации данных - разбиение множества объектов или наблюдений на априорно заданные группы, называемые классами, внутри каждой из которых они предполагаются похожими друг на друга, имеющими примерно одинаковые свойства и признаки.

2. Нарисуйте схему применения классификации данных.



Практика

```
class DataAnalysis:
    '''
    Дано целое число, не меньшее 2. Выведите его наименьший натуральный делитель, отличный от 1.
    '''

    def __init__(self, number=None):
        self._number = number
        self._divisor = self._calcDivisor()
```

```
    def _calcDivisor(self):
        for i in range(2, self._number + 1):
            if not self._number % i:
                return i

    @property
    def divisor(self):
        return self._divisor


task = DataAnalysis(94)
print(task.divisor)
```

Билет 12

1. Задача бинарной классификации.

   Классификация с бинарной переменной класса, т.е. категориальной выходной переменной, которая может принимать только два значения. Очевидно, что в таких задачах решается вопрос о принадлежности объекта к одному из двух классов. Чаще всего используют состояния `0` и `1`, но могут быть и другие, например, `Да` или `Нет`, `Y` или `N`, и т.д.

2. Логистическая регрессия. Алгоритм применения логистической регрессии.

   Логистическая регрессия или логит-модель (англ. logit model) — статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности бинарного события (`1` или `0`).

   Логистическая регрессия – это разновидность множественной регрессии, предназначенная для классификации записей на основании значений входных полей. При этом выходная переменная является категориальной или бинарной (т.е. может принимать только два значения).

   В бинарной классификации каждый объект или наблюдение должны быть отнесены к одному из двух классов (например, `А` и `Б`). Тогда с каждым исходом связано событие:

   - Объект принадлежит к классу `А` и объект принадлежит к классу `Б`.
   - Результатом будет оценка вероятности соответствующего исхода.

   Если в процессе анализа будет установлено, что вероятность принадлежности объекта с заданным набором значений признаков (входных переменных) к классу `А` больше, чем вероятность его принадлежности к классу `Б`, то он будет классифицирован, как объект класса `А`.

   Например, если рассматривается исход по займу, задается переменная y со значениями `1` и `0`, где `1` означает, что соответствующий заемщик расплатился по кредиту, а `0`, что имел место дефолт.

   Несомненным преимуществом логистической регрессии является наличие эффективного инструмента оценки качества моделей - ROC-анализа.

Практика

```
class DataAnalysis:
    '''
    Дано вещественное число – цена 1 кг конфет.
    Вывести стоимость 1, 2, … 10 кг конфет.
    Решить задачу используя циклическую конструкцию for.
    '''

    def __init__(self, cost=None):
        self._cost = cost
        self._costPerKilogram = {}
```

```
        self._calcCosts()

    def _calcCosts(self):
        for i in range(1, 11):
            self._costPerKilogram[i] = i * self._cost

    def showCost(self):
        for key, value in self._costPerKilogram.items():
            print(f'Стоимость за {key} кг - {value}')


task = DataAnalysis(10)
task.showCost()
```

Билет 13

1. Качество классификации.

Доля правильно классифицированных объектов (accuracy)

Accuracy — самая простая оценка классификации:

$$Accuracy(a) = \frac{\sum_{i=1}^{N} \mathbb{I}[a(x_i) = y_i]}{N} = \frac{TP + TN}{TP + TN + FP + FN}$$

По сути это вероятность того, что класс будет предсказан правильно.

Например, если мы ловим сумасшедших, accuracy показывает долю правильных диагнозов.

- Работает для многоклассовой классификации.
- Плохо работает при высокой априорной вероятности у одного из классов. В таком случае константное предсказание может давать высокое значение **accuracy** (равное этой априорной вероятности).

Точность (Precision)

$$Precision(a) = \frac{TP}{TP + FP}$$

Точность показывает какую долю объектов, **распознанных** как объекты положительного класса, мы предсказали верно.

На примере: точность — это сколько из пойманных нами и посаженных в психушку людей реально сумасшедшие.

- Только бинарная классификация.
- Не зависит от априорной вероятности положительного класса.

Полнота (Recall)

$$Recall(a) = \frac{TP}{TP + FN}$$

Полнота показывает, какую долю объектов, **реально** относящихся к положительному классу, мы предсказали верно.

На примере: полнота — это сколько из сумасшедших людей, которых мы проверили, мы посадили в психушку.

- Только бинарная классификация

- Не зависит от априорной вероятности положительного класса.

F-мера

Точность и полнота хорошо оценивают качество классификатора для задач со смещенной априорной вероятностью, но если мы обучили модель с высокой точностью, то может случиться так, что полнота у такого классификатора низкая и наоборот. Чтобы связать точность с полнотой вводят F-меру как среднее гармоническое точности и полноты:

$$Fmeasure = \frac{2Precision \cdot Recall}{Precision + Recall}$$

В некоторых задачах одна метрика важнее другой (например при выдаче поисковых запросов полнота важнее точности, неинтересные страницы мы можем сами пропустить, а вот если поисковик пропустит несколько страниц то мы можем остаться без каких то важных деталей). Для установления важности конкретной метрики мы рассматриваем параметрическую F-меру:

$$Fmeasure_\beta = \frac{(1 + \beta^2)Precision \cdot Recall}{\beta^2 Precision + Recall}$$

Где $\beta \in [0, \infty)$ при $\beta = 0$ получаем точность, при $\beta = 1$ — непараметрическую F-меру, при $\beta = \infty$ — полноту.

2. Правильно классифицированные алгоритмом объекты. Неправильно классифицированные алгоритмом объекты.

   Общая правильность модели — это количество правильных предсказаний, деленное на общее количество предсказаний. Оценка правильности дает значение от `0` до `1`, где `1` — идеальная модель.

   > **Правильность = количество правильных прогнозов / общее количество прогнозов**

   Этот показатель редко следует использовать отдельно, поскольку на материале несбалансированных данных, где один класс значительно превосходит другой, правильность может быть очень обманчивой.

   Вернемся к примеру с раком. Представьте, что у вас есть набор данных, в котором только 1% образцов являются раковыми. Классификатор, который предскажет все результаты как доброкачественные, достигнет 99% правильности. Однако на практике такая модель была бы не просто неполезной, но и опасной, поскольку она никогда не обнаружила бы раковые образцы.

Практика

```
class DataAnalysis:
    '''
```

```
    Дана непустая последовательность целых чисел, оканчивающаяся нулем.
    Найти:
        а) сумму всех чисел последовательности;
        б) количество всех чисел последовательности.
    Решить задачу используя циклическую конструкцию while.
    '''

    def __init__(self):
        self._array = []
        self._inputArray()

    def _inputArray(self):
        number = None
        while number != 0:
            number = int(input('-> '))
            self._array.append(number)

    @property
    def sum(self):
        return sum(self._array)

    @property
    def len(self):
        return len(self._array)


array = DataAnalysis()
print(f'Сумма: {array.sum}, длина {array.len}')
```

Билет 14

1. Задача множественной классификации.

   Задача классифика́ции — задача, в которой имеется множество объектов (ситуаций), разделённых, некоторым образом, на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

   Когда число классов достигает многих тысяч (например, при распознавании иероглифов или слитной речи), задача классификации становится существенно более трудной.

2. Искусственная нейронная сеть (ИНС) – это ?

   Нейро́нная сеть (также искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

Практика

```
class DataAnalysis:
    '''
    Дана строка, содержащая русскоязычный текст.
    Найти количество слов, начинающихся с буквы "е".
    В строке заменить все двоеточия (:) знаком процента (%).
    Подсчитать количество замен.
    '''

    def __init__(self, string=None):
        self._string = string
        self._countReplace = 0
        self._countWordsWithNeedLetter = 0

    def findWordsWithNeedLetter(self, letterForFind):
        self._findWordsWithNeedLetter(letterForFind)
        return self._countWordsWithNeedLetter

    def _findWordsWithNeedLetter(self, letterForFind):
        self._countWordsWithNeedLetter = len([word for word in
self._string.split() if word[0].lower() == letterForFind.lower()])

    @property
    def countWordsWithNeedLetter(self):
        return self._countWordsWithNeedLetter

    def replace(self, oldLetter, newLetter):
        self._replace(oldLetter, newLetter)
```

```
        return self.string, self.countReplace

    def _replace(self, oldLetter, newLetter):
        self._countReplace = self._string.count(oldLetter)
        self._string = self._string.replace(oldLetter, newLetter)

    @property
    def countReplace(self):
        return self._countReplace

    @property
    def string(self):
        return self._string


letter, oldLetter, newLetter = 'e', ':', '%'
task = DataAnalysis('На еримере: точность — это сколько из пойманных нами и
посаженных в психушку людей реально сумасшедшие.')
replaceData = task.replace(oldLetter, newLetter)
print(
    f'Строка: {replaceData[0]}',
    f'Кол-во слов начинающихся на "{letter}":
{task.findWordsWithNeedLetter(letter)}',
    f'Кол-во замен "{oldLetter}" на "{newLetter}": {replaceData[1]}',
    sep='\n'
)
```

Билет 15

1. Кластерный анализ.

   Кластерный анализ – группа методов, используемых для классификации объектов или событий в относительно гомогенные (однородные) группы, которые называют кластерами (clusters). В факторном анализе группируются столбцы, т. е. цель – анализ структуры множества признаков и выявление обобщенных факторов.

2. Схема кластерного анализа.

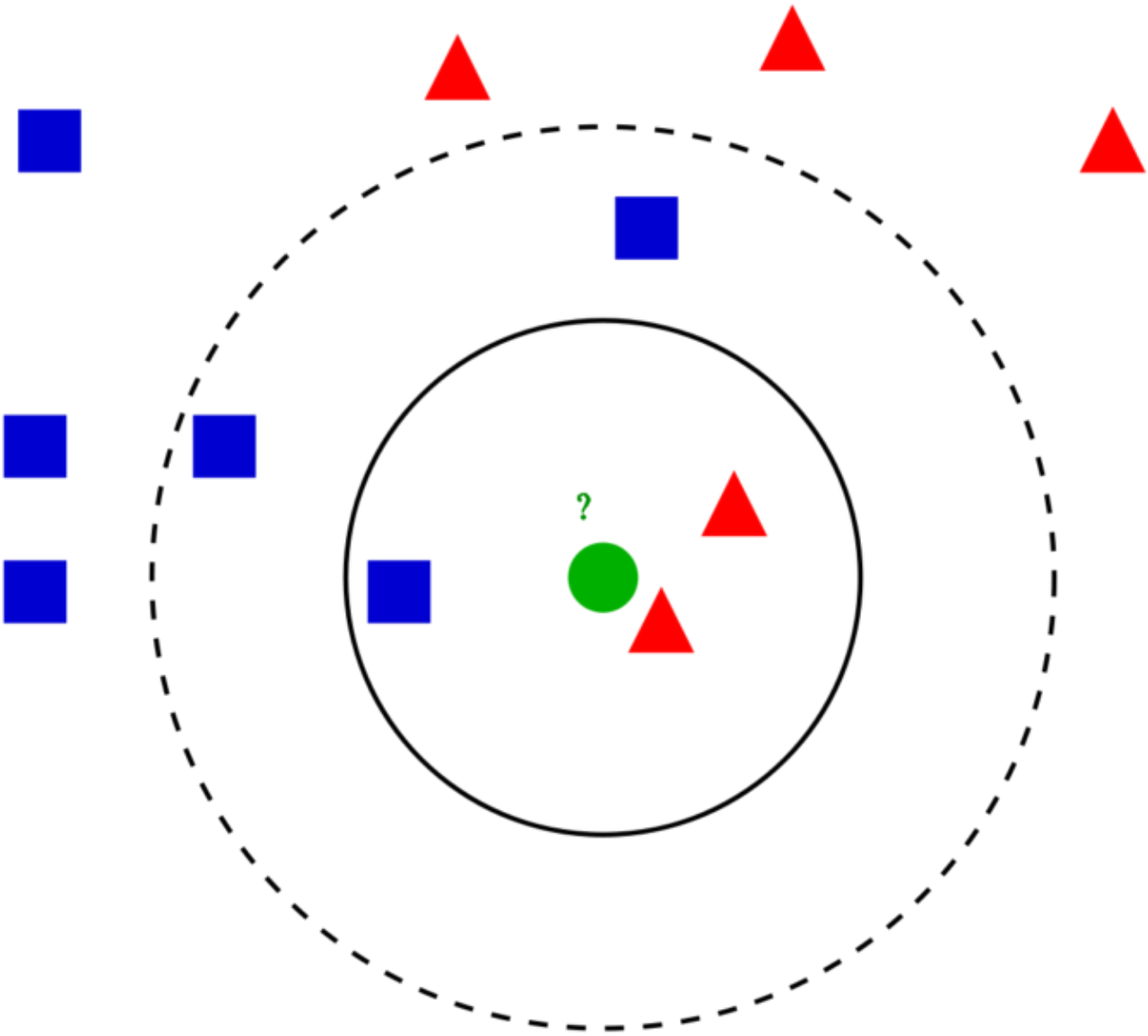
Практика

```
class DataAnalysis:
    '''
    В строке удалить символ точку (.) и подсчитать количество удаленных символов.
    В строке заменить букву (а) буквой (о).
    Подсчитать количество замен.
    Подсчитать сколько символов в строке.
    '''

    def __init__(self, string=None):
        self._string = string
        self._countDeletedLetter = 0
        self._countReplace = 0

    def deleteLetter(self, letter):
        self._deleteLetter(letter)
        return self.countDeletedLetter

    def _deleteLetter(self, letter):
        lenBeforeDelete = self.len
        self._string = self._string.replace(letter, '')
        self._countDeletedLetter = lenBeforeDelete - self.len

    @property
    def countDeletedLetter(self):
        return self._countDeletedLetter

    def replace(self, oldLetter, newLetter):
        self._replace(oldLetter, newLetter)
        return self.string
```

```
    def _replace(self, oldLetter, newLetter):
        self._string = self._string.replace(oldLetter, newLetter)

    @property
    def string(self):
        return self._string

    @property
    def len(self):
        return len(self._string)

letter, oldLetter, newLetter = '.', 'a', 'o'
task = DataAnalysis('Кластерный анализ – группа методов, используемых для
классификации\
 объектов или событий в относительно гомогенные (однородные) группы, которые
называют кластерами (clusters).')
print(
    f'Кол-во удалений "{letter}": task.deleteLetter(letter)',
    f'Строка: {task.replace(oldLetter, newLetter)}',
    f'Длина строки: {task.len}',
    sep='\n'
)
```

Билет 16

1. Метод к-средних. Его принцип.



Этот метод работает с помощью поиска кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированным объектами из обучающего набора.
Классифицируемый объект будет относится к тому классу, к которому принадлежит ближайший объект набора.

2. Последовательность операций метода к-средних.

    1. Выбирается число кластеров k.
    2. Из исходного множества данных случайным образом выбираются k наблюдений, которые будут служить начальными центрами кластеров.
    3. Для каждого наблюдения исходного множества определяется ближайший к нему центр кластера (расстояния измеряются в метрике Евклида). При этом записи, «притянутые» определенным центром, образуют начальные кластеры.
    4. Вычисляются центроиды — центры тяжести кластеров. Каждый центроид — это вектор, элементы которого представляют собой средние значения соответствующих признаков,

вычисленные по всем записям кластера.

5. Центр кластера смещается в его центроид, после чего центроид становится центром нового кластера.
6. 3-й и 4-й шаги итеративно повторяются. Очевидно, что на каждой итерации происходит изменение границ кластеров и смещение их центров. В результате минимизируется расстояние между элементами внутри кластеров и увеличиваются междукластерные расстояния.

Остановка алгоритма производится тогда, когда границы кластеров и расположения центроидов не перестанут изменяться от итерации к итерации, т.е. на каждой итерации в каждом кластере будет оставаться один и тот же набор наблюдений. На практике алгоритм обычно находит набор стабильных кластеров за несколько десятков итераций.

Практика

```
class DataAnalysis:
    '''
    Заполнить список квадратами чисел от 0 до 9, используя генератор списка.
    Заполнить список числами, где каждое последующее число больше на 2.
    '''

    def __init__(self):
        pass

    def getArraySquares(self):
        return self._getArraySquares()

    def _getArraySquares(self):
        return [x ** 2 for x in range(10)]

    def getArrayFromRangeWithCondition(self):
        return self._getArrayFromRangeWithCondition()

    def _getArrayFromRangeWithCondition(self):
        return [(x + 1) + x for x in range(10)]


task = DataAnalysis()
print(task.getArraySquares())
print(task.getArrayFromRangeWithCondition())
```

Билет 17

1. Дайте определение понятия «кластер».

   Кластер (англ. cluster — скопление, кисть, рой) — объединение нескольких однородных элементов, которое может рассматриваться как самостоятельная единица, обладающая определёнными свойствами.

2. Охарактеризуйте два любых алгоритма кластеризации.

    1. Иерархические и плоские.

        - Иерархические алгоритмы (также называемые алгоритмами таксономии) строят не одно разбиение выборки на непересекающиеся кластеры, а систему вложенных разбиений. Т.о. на выходе мы получаем дерево кластеров, корнем которого является вся выборка, а листьями — наиболее мелкие кластера.
        - Плоские алгоритмы строят одно разбиение объектов на кластеры.

    2. Четкие и нечеткие.

        - Четкие (или непересекающиеся) алгоритмы каждому объекту выборки ставят в соответствие номер кластера, т.е. каждый объект принадлежит только одному кластеру.
        - Нечеткие (или пересекающиеся) алгоритмы каждому объекту ставят в соответствие набор вещественных значений, показывающих степень отношения объекта к кластерам. Т.е. каждый объект относится к каждому кластеру с некоторой вероятностью.

Практика

```
class DataAnalysis:
    '''
    Определить длину строки.
    ЕСЛИ длина строки превышает 4 символа, ТО вывести строку в нижнем регистре.
    Заменить в строке первый символ на 'О'. Результат вывести на экран.
    '''

    def __init__(self, string=None, firstLetter=None):
        self._string = string
        self._firstLetter = firstLetter
        self._replaceFirstLetter()

    def _replaceFirstLetter(self):
        self._string = self._string[:0] + self._firstLetter + self._string[0 + 1:]

    @property
    def string(self):
        return self._string.lower() if self.len > 4 else self._string
```

```
    @property
    def len(self):
        return len(self._string)


task = DataAnalysis("У лукоморья 123 дуб зеленый 456", 'O')
print(task.string)
```

Билет 18

1. Вычислительная сложность – это?

   Вычисли́тельная сло́жность — понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных.

2. Перечислите факторы быстродействия систем анализа данных.

   - Вычислительной сложности использованных алгоритмов;
   - Способа программной реализации алгоритмов;
   - Аппаратного обеспечения.

Практика

```
from random import randint, sample


class DataAnalysis:
    '''
    Из массива X длиной n, среди элементов которого есть положительные,
отрицательные и равные нулю,
    сформировать новый массив Y, взяв в него только те элементы из X, которые
больше по модулю заданного числа M.
    Вывести на экран число M, данный и полученные массивы.
    '''

    def __init__(self, low=None, high=None):
        self._low = low
        self._high = high

        self._n = 0
        self._M = 0

        while self._n == 0:

            self._n, self._M = self._generateParams()

        self._X = self._generateArray()
        self._Y = [num for num in self._X if abs(num) > abs(self._M)]

    def _generateParams(self):
        if self._low > self._high:
            self._low, self._high = self._high, self._low
        return abs(randint(self._low, self._high)), randint(self._low, self._high)

    def _generateArray(self):
        return sample(range(self._low, self._high), self._n)
```

```
    def data(self):
        return f'M: {self._M}', f'n: {self._n}', f'X: {self._X}', f'Y: {self._Y}'


task = DataAnalysis(-10, 10)
for el in task.data():
    print(el)
```

Билет 19

1. Перечислите основные задачи анализа данных.

   К основным задачам анализа данных можно отнести прогнозирование, классификацию, поиск схожих черт, выдачу рекомендаций, выявление отклонений.

2. Приведите примеры применения методов анализа данных.

   Пример задач анализа данных в розничных магазинах:

   - оценка покупательских предпочтений,
   - анализ остатков товаров на складах,
   - выявление наиболее и наименее продаваемых товаров,
   - оценка динамики продаж,
   - сравнение объемов продаж по контрагентам.

   Пример задач анализа данных в интернет-магазинах:

   - выявление путей пользователей по сайту,
   - выявление причин брошенных корзин,
   - создание качественных рассылок на основании выявленных предпочтений покупателей,
   - настройка эффективной рекламы,
   - проведение A/B-тестирования.

Практика

```
class DataAnalysis:
    '''
    В массиве целых чисел все отрицательные элементы заменить на положительные.
    Вывести исходный массив и полученный.
    '''

    def __init__(self, array=None):
        self._array = array
        self._newAbsArray = self._absArray(self._array.copy())

    def _absArray(self, array):
        return [abs(x) for x in array]

    def __str__(self):
        return f'Исходный: {self._array}, новый: {self._newAbsArray}'


print(DataAnalysis([3, -3, 2, -6, -2, 1]))
```

Билет 20

1. Дайте определение понятию «источник данных».

   Источник данных – это контейнер, содержащий данные, которые вы загружаете в Аналитику. Источники данных определяют, как загруженная информация объединяется с существующей. Источники настраиваются на уровне ресурса. Загружать данные из одного и того же источника данных можно многократно.

   Примеры: Радио, телепередачи, газеты, журналы, выставки, конференции, сайты Интернета.

2. Приведите примеры категориальных данных.

   Категориальная (качественная) переменная — это бинарная или дихотомическая переменная, когда имеются только две возможные категории.

   Примеры включают `Да/Нет`, `Умер/Жив` или `Больной имеет заболевание/Больной не имеет никаких заболеваний`.

Практика

```
class DataAnalysis:
    '''
    Дан одномерный массив, состоящий из N целочисленных элементов.
    Ввести массив с клавиатуры.
    Найти максимальный элемент.
    Вывести массив на экран в обратном порядке.
    '''

    def __init__(self, array=None):
        self._array = array
        print(f'Max: {self._getMax()}')
        print(f'Reverse array: {self._getReverseArray()}')

    def _getMax(self):
        return max(self._array)

    def _getReverseArray(self):
        return self._array[::-1]


DataAnalysis(list(map(int, input('-> ').split())))
```

Билет 21

1. Массив в языке программирования – это?

   Массив — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.

2. Какими способами можно создать массив? Для чего можно применить многомерный массив?

   - Перечисление элементов
   - Использовать специальную функцию

Практика

```
from random import randrange


class DataAnalysis:
    '''
    В массиве действительных чисел все нулевые элементы заменить на
среднеарифметическое всех элементов массива.
    '''

    def __init__(self, low=None, high=None, count=None):
        self._low = low
        self._high = high
        self._count = count
        self._array = self._generateArray()
        self._mean = self._getMean()
        self._replaceZero()

    def _generateArray(self):
        return [randrange(self._low, self._high) for i in range(self._count)]

    def _getMean(self):
        return sum(self._array) / len(self._array)

    def _replaceZero(self):
        self._array = [self._mean if x == 0 else x for x in self._array]

    @property
    def array(self):
        return self._array


task = DataAnalysis(0, 2, 5)
print(task.array)
```

Билет 22

1. Процедура в языке программирования – это?

   Процедура – это подпрограмма, которая выполняет некоторые действия после вызова её из основной программы или другой процедуры. Каждая процедура имеет уникальное имя, может иметь произвольное количество входных параметров.

2. Опишите как можно вызвать процедуру. Сколько раз можно вызвать процедуру в одной программе?

   При вызове процедуры указываются фактические значения параметров. Вызывать можно не ограниченное кол-во раз.

Практика

```
class DataAnalysis:
    '''
    Создать процедуру для вывода сообщения об ошибке.
    Запрашивать у пользователя ввести положительное число, в случае ввода
отрицательного числа,
    вызывать процедуру для вывода сообщения об ошибке.
    '''

    def __init__(self):
        self._input()

    def _input(self):
        assert int(input('-> ')) >= 0, "Negative number"


task = DataAnalysis()
```

Билет 23

1. Функция в языке программирования – это?

   Фу́нкция в программировании, или подпрограмма — фрагмент программного кода, к которому можно обратиться из другого места программы.

2. Как создать функцию? Напишите на латинице часть встроенных функций в языке Python (минимум две).

   Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции `def`.

   `sum`, `min`

Практика

```
class DataAnalysis:
    '''
    Найти все простые натуральные числа, не превосходящие n, двоичная запись
которых представляет собой палиндром,
    т. е. читается одинаково слева направо и справа налево.
    '''

    def __init__(self, array=None):
        self._array = array

    def checkPalindrome(self):
        for num in self._array:
            copyNum = num
            result = 0

            while num != 0:
                digit = num % 10
                result = result * 10 + digit
                num = int(num / 10)

            if result == copyNum:
               print(copyNum)


task = DataAnalysis([123321, 3345, 121, 5643, 45654])
task.checkPalindrome()
```

Билет 24

1. Классы в Python — это. Чем отличается класс от объекта?

   Класс - кодовая структура и механизм, используемый для реализации новых видов **объектов**, которые поддерживают **наследование**.

   Класс - фабрика экземпляров. Атрибуты класса обеспечивают поведение, которое наследуется всеми экземплярами.

   Класс - это статическая часть кода, состоящая из атрибутов, которые не меняются во время выполнения программы - наподобие определений методов класса.

   Объект - это экземпляр класса. Однако термин object относится к реально существующему экземпляру класса. Каждый объект должен принадлежать классу.

2. Для чего необходимо ключевое слово self в классах?

   self - это ни в коем случае не зарезервированное слово. Это просто название переменной.

   В методах класса первый параметр функции по соглашению именуют self, и это ссылка на сам объект этого класса. Но это именно соглашение. Вы вольны называть параметры как угодно.

Практика

```
class DataAnalysis:
    '''
    Составить программу, которая изменяет последовательность слов в строке на
обратную.
    '''

    def __init__(self, string=None):
        self._string = string
        print(self._getString())

    def _getString(self):
        return ' '.join(self._string.split(" ")[::-1])


task = DataAnalysis('Составить программу, которая изменяет последовательность слов
в строке на обратную.')
```

Билет 25

1. Что представляет собой главный объект в NumPy. Дайте его определение.

   Главный объект NumPy - это однородный многомерный массив. Чаще всего это одномерная последовательность или двумерная таблица, заполненная элементами одного типа, как правило числами, которые проиндексированы кортежем положительных целых чисел.

2. Векторизованные вычисления что это? Какова их роль?

   Целью хранения числовых данных в массиве является возможность использовать краткие векторизованные выражения для обработки данных. Эффективное использование векторизованных выражений избавляет от необходимости использовать явные циклы for. Уменьшает длину кода, имеет лучшую ремонтопригодность и более высокую производительность.

Практика

```
import numpy as np


class DataAnalysis:
    '''
    Найти минимальное и максимальное значение, принимаемое каждым числовым типом
NumPy.
    '''

    def __init__(self):
        self._getData()

    def _getData(self):
        for dtype in [np.int8, np.int32, np.int64]:
            print(np.iinfo(dtype).min)
            print(np.iinfo(dtype).max)
        for dtype in [np.float32, np.float64]:
            print(np.finfo(dtype).min)
            print(np.finfo(dtype).max)
            print(np.finfo(dtype).eps)


task = DataAnalysis()
```