Пономаренко Николай

Язык Python и библиотеки для анализа данных: задание 1

/Работа выполнена на третьей версии Python./

1 Python

1) Чем отличаются типы list и tuple? Какой из них можно использовать в качестве ключа в dict и добавлять в set? Почему? На какой структуре данных реализован dict в python? Чем отличается range от xrange в Python2.x? Что такое list comprehension? Как работают функции map и reduce в Python?

Во-первых, List является изменяемым типом, а тип tuple не является таковым. Во-вторых, они занимают в памяти разный размер (см. код ниже). Конечно, это не особо существенно, но операции с tuple происходят немного быстрее.

In [1]:

```
a = tuple(range(1000))
b = list(range(1000))

print(a.__sizeof__())
print(b.__sizeof__())
```

```
8024
9088
```

Так как list является изменяемым типом, то его нельзя использовать в качестве ключа в словаре (так как нехешируемый тип), когда tuple, к счастью, обладает таким свойством. Оба типа можно добавлять в set.

In [2]:

```
a = (1,2)
b = [1,2]

c = {a: 1}
print('OK')
c = set(a)
print('OK')
c = set(b)
print('OK')
try:
    c = {b: 1}
except Exception:
    print('Так нельзя:(')
```

```
OK
OK
OK
Так нельзя:(
```

Словарь в Python реализован в виде хэш-таблицы с методом открытой адресации.

xrange() очень похож на range(), но возвращает xrange object вместо list. Это, например, даёт преимущества при работе с большими циклами, так как xrange() не хранит весь list объектов, по которым произвоится итерация.

list comprehension обеспечивает краткий способ создания списков. Обычно применяется при создании новых списков, в которых каждый элемент является результатом некоторых операций, применяемых к каждому члену другой последовательности, или для создания подпоследовательности тех элементов, которые удовлетворяют определенному условию.

map(function, iterable, ...)

Применяет функцию к каждому элементу итерабельного объекта и возвращает объект типа list в качестве результата (во втором Питоне) или объект типа map (третий Питон).

In [3]:

```
arr = [i for i in range(10)]
print(arr)
mod_arr = list(map(lambda x: x ** 2, arr))
print(mod_arr)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

reduce(function, iterable[, initializer])

Применяет функцию двух аргументов кумулятивно к элементам итерабельного объекта слева направо, чтобы уменьшить итерабельный объект до одного значения. Пример: reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) вычисляет ((((1+2)+3)+4)+5).

In [4]:

```
from functools import reduce
reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
```

Out[4]:

```
15
```

2) Напишите рекурсивную функцию, возвращающую N-ное число Фибоначчи. Проверьте работу для небольших N. Выясните, для какого N начинаются проблемы с глубиной рекурсии. Дополнительное задание для любознательных: попробуйте выяснить, можно ли в Python изменить допустимую глубину рекурсии.

In [5]:

```
def fib_rec(N):
    if N == 0 or N == 1:
        return 1
    return fib_rec(N - 1) + fib_rec(N - 2)
```

In [6]:

```
# Проверим работу для первых 10 чисел.

for i in range(10):
    print(fib_rec(i))
```

```
1
1
2
3
5
8
13
21
34
55
```

Начиная с $N = 972$ появляются проблемы с глубиной рекурсии.

In [7]:

```
try:
    fib_rec(972)
except RecursionError:
    print('From here.')
```

```
From here.
```

In [8]:

```
import sys
sys.getrecursionlimit()
```

Out[8]:

```
1000
```

Функция, вызванная выше возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python. Этот предел предотвращает бесконечную рекурсию из-за переполнения стека. Максимальная глубина рекурсии может быть изменена с помощью sys.setrecursionlimit(limit).

In [9]:

```
print(sys.getrecursionlimit())
sys.setrecursionlimit(2000)
print(sys.getrecursionlimit())
sys.setrecursionlimit(1000)
```

```
1000
2000
```

3) Попробуйте измерить время работы предыдущей функции при разных N.

In [10]:

```
import time

for i in range(40):
    start_time = time.clock()
    fib_rec(i)
    print("N =", i, "is done in", time.clock() - start_time, "seconds")
```

```
N = 0 is done in 3.84e-06 seconds
N = 1 is done in 2.9866666666662767e-06 seconds
N = 2 is done in 4.693333333333216e-06 seconds
N = 3 is done in 4.266666666667349e-06 seconds
N = 4 is done in 6.4000000000001556e-06 seconds
N = 5 is done in 9.813333333334034e-06 seconds
N = 6 is done in 1.450666666666725e-05 seconds
N = 7 is done in 2.602666666666649e-05 seconds
N = 8 is done in 3.797333333333333e-05 seconds
N = 9 is done in 6.91199999999989e-05 seconds
N = 10 is done in 8.234666666666682e-05 seconds
N = 11 is done in 0.00015744 seconds
N = 12 is done in 0.00023850666666666576 seconds
N = 13 is done in 0.00037162666666666795 seconds
N = 14 is done in 0.0006438399999999997 seconds
N = 15 is done in 0.0009787733333333336 seconds
N = 16 is done in 0.0017949866666666682 seconds
N = 17 is done in 0.0021934933333333344 seconds
N = 18 is done in 0.003971413333333333 seconds
N = 19 is done in 0.008821333333333334 seconds
N = 20 is done in 0.011462826666666662 seconds
N = 21 is done in 0.01860053333333333 seconds
N = 22 is done in 0.028808533333333344 seconds
N = 23 is done in 0.04699136000000001 seconds
N = 24 is done in 0.07119658666666667 seconds
N = 25 is done in 0.10566314666666665 seconds
N = 26 is done in 0.2062890666666667 seconds
N = 27 is done in 0.15513941333333336 seconds
N = 28 is done in 0.20680789333333338 seconds
N = 29 is done in 0.3405183999999999 seconds
N = 30 is done in 0.54475904 seconds
N = 31 is done in 1.0015321599999998 seconds
N = 32 is done in 1.49713152 seconds
N = 33 is done in 2.4251741866666663 seconds
N = 34 is done in 4.531554986666667 seconds
N = 35 is done in 6.376645120000001 seconds
N = 36 is done in 9.950345813333335 seconds
N = 37 is done in 17.962469546666668 seconds
N = 38 is done in 29.778318506666672 seconds
N = 39 is done in 48.53847168 seconds
```

4) Напишите функцию, возвращающую (через запятую) модуль и аргумент комплексного числа по заданной мнимой и действительной части. Возможно, вам потребуется импортировать какие-то тригонометрические функции из модуля math. Приведите примеры работы. Каков тип возвращаемого вашей функцией значения?

In [11]:

```
# Для более красивого код будем использовать библиотеку cmath,
# так как в ней есть функция phase().
import cmath

def my_func_complex_num(x, y):
    modulus = abs(complex(x, y))

    # phase() не учитывает этот вариант.

    if (x == y == 0):
        raise Exception('Undefined.')

    arg = cmath.phase(complex(x, y))
    return modulus, arg
```

In [12]:

```
# z = 1 + 0j
print(my_func_complex_num(1, 0))

# z = 1 + 1j
print(my_func_complex_num(1, 1))

# z = -3 + -5j
print(my_func_complex_num(-3, -5))
```

```
(1.0, 0.0)
(1.4142135623730951, 0.7853981633974483)
(5.830951894845301, -2.1112158270654806)
```

Тип возвращаемого моей функцией значения - tuple.

In [13]:

```
type(my_func_complex_num(1, 1))
```

Out[13]:

```
tuple
```

5) Напишите docstrings к вашей последней функции. Изучите библиотеку doctest и попробуйте проверить, что написанные в docstrings примеры выполняются, без явного копирования кода оттуда (т.е. с помощью doctest).

In [14]:

```
# Для более красивого код будем использовать библиотеку cmath,
# так как в ней есть функция phase().
import cmath

"""
Common example.

>>> my_func_complex_num_dt(1, 1)
(1.4142135623730951, 0.7853981633974483)
"""

def my_func_complex_num_dt(x, y):

    """
    Returns modulus and arg of complex number x+iy.

    Another examples.

    >>> my_func_complex_num_dt(0, 1)
    (1.0, 1.5707963267948966)

    >>> my_func_complex_num_dt(0, 0)
    Traceback (most recent call last):
      ...
    Exception: Undefined.

    """

    modulus = abs(complex(x, y))

    # phase() не учитывает этот вариант.

    if (x == y == 0):
        raise Exception('Undefined.')

    arg = cmath.phase(complex(x, y))
    return modulus, arg


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

P.S. Я не очень понял, что значит "попробуйте проверить, что написанные в docstrings примеры выполняются, без явного копирования кода оттуда (т.е. с помощью doctest)". Поэтому я просто написал различные примеры проверок коректности моего кода, аналогично тому, что указано в документации.

6) Реализуйте класс комплексных чисел с самым базовым функционалом. Цель не продемонстрировать умение спроектировать класс из каждого второго домашнего задания по любому языку программирования, а посмотреть на особенности реализации классов на питоне.

In [15]:

```
class Complex_numbers:
    """Class of complex numbers."""

    def __init__(self, x = 0, y = 0):
        """
        Constructor.
        x, y - Real and Img parts of complex number.
        Default values: x = y = 0.

        """
        self.x = x
        self.y = y


    def __add__(self, other):
        """
        Redefining of '+' operator.
        """

        return Complex_numbers(self.x + other.x, self.y + other.y)

    def __repr__(self):
        """
        Redefining of represenation.
        """

        return "{}{}{}i".format(self.x, '-' if self.y < 0 else '+', abs(self.y))
```

In [16]:

```
a = Complex_numbers(-1, -1)
b = Complex_numbers(0, 1)
c = Complex_numbers()
print(a)
print(b)
print(c)
print(a + b)
```

```
-1-1i
0+1i
0+0i
-1+0i
```

7) Изучите примеры работы с декораторами из официальной документации. Фактически, декоратор - это функция, которая принимает на вход другую функцию и возвращает её в модифицированном виде. Синтаксически предусмотрена возможность использовать декоратор, написав перед объявлением функции имя_декоратора. Попробуйте найти реализацию (или сделать сами) кэширующего декоратора и декоратора, вычисляющего время работы функции. Попробуйте применить эти декораторы в разной последовательности к функции, вычисляющей числа Фибоначчи, сравнить и объяснить результаты. Объясните также, как работают использованные вами декораторы.

In [17]:

```
def memoize(function):
    dic = {}
    def wrapper(*args, **kwargs):
        if args not in dic:
            dic[args] = function(*args, **kwargs)
        return dic[args]
    return wrapper
```

Данный декоратор работает следующим образом: создаётся словарь, в котором будут уже подсчитанные значения функции. Далее проверяется, вызывалась ли ранее функция с такими аргументами. Если нет, то вычисляем её значения при данных аргументах, а потом запоминаем его. Возвращаем значения функции от данных аргументов.

In [18]:

```
@memoize
def fib_rec_decor_m(N):
    if N == 0 or N == 1:
        return 1
    return fib_rec_decor_m(N - 1) + fib_rec_decor_m(N - 2)
```

In [19]:

```
fib_rec_decor_m(100)
```

Out[19]:

```
573147844013817084101
```

In [20]:

```
def timer(f):
    is_evaluating = False
    def g(x):
        nonlocal is_evaluating
        if is_evaluating:
            return f(x)
        else:
            start_time = time.clock()
            is_evaluating = True
            try:
                value = f(x)
            finally:
                is_evaluating = False
            end_time = time.clock()
            print('time taken: {time}'.format(time=end_time-start_time))
            return value
    return g
```

Отметим, что мы не должны считать рекурсивные вызовы функциии отдельно. Поэтому если функция вызывает сама себя, то это значит, что у мы уже начали вычислять время её работы. Тем самым мы замерим время от первого вызова функции до возвращения результата работы рекурсивной функции.

In [21]:

```
@timer
def fib_rec_decor_t(N):
    if N == 0 or N == 1:
        return 1
    return fib_rec_decor_t(N - 1) + fib_rec_decor_t(N - 2)
```

In [22]:

```
fib_rec_decor_t(32)
```

```
time taken: 2.4474073599999997
```

Out[22]:

```
3524578
```

In [23]:

```
@memoize
@timer
def fib_rec_decor_tm(N):
    if N == 0 or N == 1:
        return 1
    return fib_rec_decor_tm(N - 1) + fib_rec_decor_tm(N - 2)
```

In [24]:

```
fib_rec_decor_tm(100)
```

```
time taken: 0.002246826666677748
```

Out[24]:

```
573147844013817084101
```

8) Выясните, чем отличается задание поля x через "self.x =" от объявления метода с декоратором property.

Рассмотрим следующий пример.

In [25]:

```
class Person(object):
    """"""
    def __init__(self, first_name, last_name):
        """Конструктор"""
        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        """
        Возвращаем полное имя
        """
        return "%s %s" % (self.first_name, self.last_name)
```

In [26]:

```
person = Person("Mike", "Driscoll")

print(person.full_name) # Mike Driscoll
print(person.first_name) # Mike
try:
    person.full_name = "Jackalope"
except AttributeError:
    print('Нельзя изменить.')

person.first_name = "Jackalope"
print('А вот так можно.')
print(person.first_name)
```

```
Mike Driscoll
Mike
Нельзя изменить.
А вот так можно.
Jackalope
```

Декоратор @property превращает метод класса в «getter» в ТОЛЬКО ДЛЯ ЧТЕНИЯ атрибут с тем же именем, а вот заданное поле x через "self.x =" в последствие можно изменить.

9) Напишите setter и getter для полей класса комплексного числа.

In [27]:

```
class Complex_numbers_decor:
    """Class of complex numbers."""

    def __init__(self, x=0, y=0):
        """
        Constructor.
        x, y - Real and Img parts of complex number.
        Default values: x = y = 0.

        """
        self._x = x
        self._y = y

    @property
    def x(self):
        """I'm the 'x' property."""
        return self._x

    @property
    def y(self):
        """I'm the 'y' property."""
        return self._y

    @x.setter
    def x(self, val):
        """Set real part."""
        self._x = val

    @y.setter
    def y(self, val):
        """Set img part."""
        self._y = val

    def __add__(self, other):
        """
        Redefining of '+' operator.
        """

        return Complex_numbers(self.x + other.x, self.y + other.y)

    def __repr__(self):
        """
        Redefining of represenation.
        """

        return "{}{}{}i".format(self.x, '-' if self.y < 0 else '+', abs(self.y))
```

2 Requests/urllib + BeautifulSoup

Скачайте html-код страницы с новостью с любого новостного сайта и, распарсив его с помощью библиотеки Beautiful Soup или любой другой аналогичной, получите текст статьи (без html-тегов или вкраплений java script).

In [28]:

```
import requests
from bs4 import BeautifulSoup

# Почитаем про то, как наша фигуристка заняла лишь второе место :(.

url = "https://lenta.ru/articles/2018/12/09/grand_prix_final/"

# Скачиваю html-код страницы с новостью, преобразую её к типу 'str'.

text = requests.get(url).text

# Распарсиваем её с помощью библиотеки Beautiful Soup.

text_good = BeautifulSoup(text, "lxml")

# Печатаем именно текст статьи.
print(text_good.find(itemprop="articleBody").text)
```

Три года подряд россиянки не отдавали никому золото в финале Гран-при по ф
игурному катанию. Елизавета Туктамышева, Евгения Медведева, Алина Загитов
а: фигуристки сборной России приучили отечественную публику не переживать
за судьбу медали высшей пробы. Загитова по всем законам жанра в этом году
должна была оформить золотой дубль на льду Ванкувера. Но на ее пути встала
Рика Кихира — японка с феноменальным техническим заделом. В чем сила Кихир
ы и почему действующая олимпийская чемпионка проиграла бывшей юниорке — ра
збиралась «Лента.ру».Проспорили«Алина Загитова или Евгения Медведева?» — т
олько в таком контексте было принято говорить о женском фигурном катании в
последние два сезона. Россиянки выясняли отношения на недосягаемом для все
х остальных уровне, будто соревновались в другом, собственном дивизионе. П
ротивостояние двух выдающихся фигуристок, а в этом году еще и двух тренеро
в, Этери Тутберидзе и Брайана Орсера, захватило внимание всего спортивного
мира, не оставив возможности даже подумать о существовании какой-либо пост
оронней силы.Говорят, что третий — всегда лишний, но... На этот раз лишним
и оказались первые две. Пока мир гудел, выясняя, кто из россиянок лучше и
чей тренер подкованнее, из тени, окрепнув и осмелев, выбралась новая звезд
а. Возможно, мирового масштаба. Японка Рика Кихира в финале Гран-при в Ван
кувере заявила: Россия в женском фигурном катании — больше не единоличный
лидер, а уникальная система Тутберидзе — не панацея.Российская публика не
привыкла к поражениям отечественных фигуристок, зато привыкла к их мировым
рекордам. Подопечные Тутберидзе весь прошлый сезон вплоть до Олимпиады по
очереди задирали планку все выше, подтверждая, что предела их возможностям
просто не существует. В текущем сезоне Загитова эту добрую традицию продол
жила. Медведева же из гонки выбыла, когда приняла решение переехать за оке
ан в группу канадских специалистов.Победа Загитовой в финале Гран-при под
сомнение не ставилась, хотя основания на то были. Звоночек прозвенел давн
о: в тот самый момент, когда 16-летняя японка Кихира заявила и сделала два
тройных акселя в произвольной программе. Это самый трудный и дорогой прыжо
к в женском фигурном катании. И в данный момент россиянка им не владеет.«Я
могу бороться с этими спортсменками»В финал Гран-при в 2018 году отобралис
ь шесть лучших фигуристок планеты: россиянки Алина Загитова, Елизавета Тук
тамышева, Софья Самодурова и японки Рика Кихира, Сатоко Мияхара и Каори Са
камото. Три на три. Алина Загитова Maxim Shemetov / Reuters Серьезно заду
маться о пьедестале еще до старта соревнований могли все шесть спортсмено
к, но это могли быть мысли только о бронзовой награде. Туктамышевой для по
беды недостаточно технической базы, особенно — вращательной, Самодуровой н
е хватает опыта и владения элементами ультра-си, а Мияхаре и Сакамото — ст
абильности и прыжкового арсенала. Поэтому главной интригой была, конечно,
готовность Загитовой и Кихиры. И после объявления оценок японки в короткой
программе те, кто заранее повесил на шею Загитовой золотую медаль, начали
переобуваться на ходу. 82,51 балла — так пал мировой рекорд, который росси
янка установила на этапе Гран-при в Москве меньше месяца назад.Кихира в ко
роткой программе действительно была безупречна. Коронный тройной аксель, ч
истейший каскад тройной флип — тройной тулуп и тройной лутц со сложным вые
здом в ласточку; все это позволило ей ощутимо оторваться от соперниц. Возм
ожно, программа японки не впечатляет хореографической зрелищностью, возмож
но, в ней нет драмы, нет остроты — таков уж «Лунный свет» Клода Дебюсси. З
ато в ней есть самый сложный в мире технический набор.Загитова же на лед в
ыходила последней и знала, что притязания на ее корону очень серьезные. Он
а блестяще справилась со сложнейшим каскадом тройной лутц — тройной риттбе
ргер, с тем самым элементом, который помог завоевать золото Олимпиады. Зат
ем тяжеловато, но все же приземлила двойной аксель, а под конец прыгнула т
ройной флип. Все это было сделано с настроением, уверенно, немного нагло:
так, как на Играх в Пхенчхане. Только вот повторить триумф не удалось. Заг
итова, откатавшись без видимых срывов, получила всего 77,93 балла. Сразу н
а пять баллов меньше, чем неопытная, но фантастически техничная японка.«Эт
и мысли (о тройном акселе — прим. «Ленты.ру») пощекотали мне нервы. Но я з
наю, что если делаю все идеально, могу бороться с этими спортсменками», —
сказала Загитова после короткой программы. Жаль, что настоящей борьбы не п
олучилось ни в один из соревновательных дней.«У Алины сильный соперник»Зад

ел из пяти баллов после короткой программы — это если не гарантия, то одно значно неплохое подспорье для победы. Загитова эти баллы проиграла на бума ге. И шансов исправить ситуацию, опять же математически, имела не так мног о.В произвольной последней каталась уже Кихира, а Загитова получила четвер тый стартовый номер. Как и накануне, внешне она была абсолютно спокойна. Н ичего другого ей не оставалось, ведь эта уверенность — ее единственный коз ырь. Получилось не все: Загитовой-Кармен не покорился тройной тулуп в каск аде с тройным лутцем. А ведь его в команде Тутберидзе называют разминочны м.«Я не очень хорошо приземлилась с лутца и попыталась вытащить тройной ту луп. Но в один момент у меня что-то щелкнуло, и...», — оправдывалась Загит ова после проката. Это «щелкнуло» для спортсменки, выступающей в статусе д ействующей олимпийской чемпионки, звучит несерьезно. Но что еще она могла сказать? Ведь в остальном прокат удался, хотя сравнить его с выступлением на Гран-при в Москве можно лишь с натяжкой. Рика Кихира Emmanuel Foudrot / Reuters Кихира на этот раз тоже не была идеальна. В самом начале программ ы она упала с тройного акселя и поэтому не смогла прицепить к нему тройной тулуп. Она сделала это позже, рискуя, но лишь увеличивая надбавку к оценк е. И даже с падением японка превзошла Загитову. По итогам произвольной — н а два балла, по сумме — на семь. Туктамышева с отставанием в 18 баллов зам кнула тройку призеров, а Самодурова стала пятой.«Думаю, причина моих успех ов в том, что у меня были провалы в последние два года. Я пообещала себе, что запомню их и больше подобных ошибок не повторю», — заявила новоиспечен ная победительница. Если так и дальше пойдет, судьба золота чемпионата мир а тоже окажется в ее руках.«Хочется всегда быть первой, но это невозможн о», — поделилась Загитова после объявления оценок японки. Вторя ей, специа листы отнеслись к этому поражению сдержанно: мол, ничего страшного, бывае т. «У Алины — сильный соперник, который владеет мощной технической базой. Не может человек все время быть первым», — отметила заслуженный тренер Тат ьяна Тарасова.Тем не менее Загитовой и тренерскому штабу Тутберидзе точно есть над чем подумать. За олимпийскую чемпионку тревожно. Ведь в Ванкувере она проиграла абсолютно по делу.

3 NumPy

1) Создайте numpy.ndarray размерностью 3x4x2 и продемонстрируйте разные способы обращения по индексам из документации NumPy

In [29]:

```
import numpy as np

a = np.ndarray(shape=(3, 4, 2), dtype=int, order='F')
for i in range(3):
    for j in range(4):
        for k in range(2):
            a[i,j,k] = i * 21 + j * 5 + k
print(a)
print(a.shape)
```

```
[[[ 0  1]
  [ 5  6]
  [10 11]
  [15 16]]

 [[21 22]
  [26 27]
  [31 32]
  [36 37]]

 [[42 43]
  [47 48]
  [52 53]
  [57 58]]]
(3, 4, 2)
```

Теперь продемонстрируем различные способы обращения к элементам данного объекта.

In [30]:

```
print(a[2,3,1])
```

```
58
```

In [31]:

```
# Да, можно обращаться к индексам в обратном порядке.

print(a[2,-2,1])
```

```
53
```

In [32]:

```
# В плане выдаваемого значения это эквивалентного первому примеру.

print(a[2][3][1])
```

```
58
```

In [33]:

```
# Поддерживаются слайсы.

a[1:]
```

Out[33]:

```
array([[[21, 22],
        [26, 27],
        [31, 32],
        [36, 37]],

       [[42, 43],
        [47, 48],
        [52, 53],
        [57, 58]]])
```

In [34]:

```
# Да, слайсы сразу по нескольким осям.

a[1::, 2:]
```

Out[34]:

```
array([[[31, 32],
        [36, 37]],

       [[52, 53],
        [57, 58]]])
```

In [35]:

```
# Не знаю, кто этим пользуется, но всё же...

a[1,...,1]
```

Out[35]:

```
array([22, 27, 32, 37])
```

2) Попробуйте сравнить по производительности numpy.array и list, выполняя какую-то стандартную операцию с каждым из типов много раз.

In [36]:

```
matrix = []
inner = []
for j in range(50):
    for i in range(10):
        inner.append(i+j*10)
    matrix.append(inner)
    inner = []
matrix
```

Out[36]:

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
 [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
 [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
 [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
 [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
 [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
 [90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
 [100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
 [110, 111, 112, 113, 114, 115, 116, 117, 118, 119],
 [120, 121, 122, 123, 124, 125, 126, 127, 128, 129],
 [130, 131, 132, 133, 134, 135, 136, 137, 138, 139],
 [140, 141, 142, 143, 144, 145, 146, 147, 148, 149],
 [150, 151, 152, 153, 154, 155, 156, 157, 158, 159],
 [160, 161, 162, 163, 164, 165, 166, 167, 168, 169],
 [170, 171, 172, 173, 174, 175, 176, 177, 178, 179],
 [180, 181, 182, 183, 184, 185, 186, 187, 188, 189],
 [190, 191, 192, 193, 194, 195, 196, 197, 198, 199],
 [200, 201, 202, 203, 204, 205, 206, 207, 208, 209],
 [210, 211, 212, 213, 214, 215, 216, 217, 218, 219],
 [220, 221, 222, 223, 224, 225, 226, 227, 228, 229],
 [230, 231, 232, 233, 234, 235, 236, 237, 238, 239],
 [240, 241, 242, 243, 244, 245, 246, 247, 248, 249],
 [250, 251, 252, 253, 254, 255, 256, 257, 258, 259],
 [260, 261, 262, 263, 264, 265, 266, 267, 268, 269],
 [270, 271, 272, 273, 274, 275, 276, 277, 278, 279],
 [280, 281, 282, 283, 284, 285, 286, 287, 288, 289],
 [290, 291, 292, 293, 294, 295, 296, 297, 298, 299],
 [300, 301, 302, 303, 304, 305, 306, 307, 308, 309],
 [310, 311, 312, 313, 314, 315, 316, 317, 318, 319],
 [320, 321, 322, 323, 324, 325, 326, 327, 328, 329],
 [330, 331, 332, 333, 334, 335, 336, 337, 338, 339],
 [340, 341, 342, 343, 344, 345, 346, 347, 348, 349],
 [350, 351, 352, 353, 354, 355, 356, 357, 358, 359],
 [360, 361, 362, 363, 364, 365, 366, 367, 368, 369],
 [370, 371, 372, 373, 374, 375, 376, 377, 378, 379],
 [380, 381, 382, 383, 384, 385, 386, 387, 388, 389],
 [390, 391, 392, 393, 394, 395, 396, 397, 398, 399],
 [400, 401, 402, 403, 404, 405, 406, 407, 408, 409],
 [410, 411, 412, 413, 414, 415, 416, 417, 418, 419],
 [420, 421, 422, 423, 424, 425, 426, 427, 428, 429],
 [430, 431, 432, 433, 434, 435, 436, 437, 438, 439],
 [440, 441, 442, 443, 444, 445, 446, 447, 448, 449],
 [450, 451, 452, 453, 454, 455, 456, 457, 458, 459],
 [460, 461, 462, 463, 464, 465, 466, 467, 468, 469],
 [470, 471, 472, 473, 474, 475, 476, 477, 478, 479],
 [480, 481, 482, 483, 484, 485, 486, 487, 488, 489],
 [490, 491, 492, 493, 494, 495, 496, 497, 498, 499]]
```

In [37]:

```
# То же самое, только через np.array.

np_array = np.arange(500).reshape(50,10)
np_array
```

Out[37]:

```
array([[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9],
       [ 10,  11,  12,  13,  14,  15,  16,  17,  18,  19],
       [ 20,  21,  22,  23,  24,  25,  26,  27,  28,  29],
       [ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49],
       [ 50,  51,  52,  53,  54,  55,  56,  57,  58,  59],
       [ 60,  61,  62,  63,  64,  65,  66,  67,  68,  69],
       [ 70,  71,  72,  73,  74,  75,  76,  77,  78,  79],
       [ 80,  81,  82,  83,  84,  85,  86,  87,  88,  89],
       [ 90,  91,  92,  93,  94,  95,  96,  97,  98,  99],
       [100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
       [110, 111, 112, 113, 114, 115, 116, 117, 118, 119],
       [120, 121, 122, 123, 124, 125, 126, 127, 128, 129],
       [130, 131, 132, 133, 134, 135, 136, 137, 138, 139],
       [140, 141, 142, 143, 144, 145, 146, 147, 148, 149],
       [150, 151, 152, 153, 154, 155, 156, 157, 158, 159],
       [160, 161, 162, 163, 164, 165, 166, 167, 168, 169],
       [170, 171, 172, 173, 174, 175, 176, 177, 178, 179],
       [180, 181, 182, 183, 184, 185, 186, 187, 188, 189],
       [190, 191, 192, 193, 194, 195, 196, 197, 198, 199],
       [200, 201, 202, 203, 204, 205, 206, 207, 208, 209],
       [210, 211, 212, 213, 214, 215, 216, 217, 218, 219],
       [220, 221, 222, 223, 224, 225, 226, 227, 228, 229],
       [230, 231, 232, 233, 234, 235, 236, 237, 238, 239],
       [240, 241, 242, 243, 244, 245, 246, 247, 248, 249],
       [250, 251, 252, 253, 254, 255, 256, 257, 258, 259],
       [260, 261, 262, 263, 264, 265, 266, 267, 268, 269],
       [270, 271, 272, 273, 274, 275, 276, 277, 278, 279],
       [280, 281, 282, 283, 284, 285, 286, 287, 288, 289],
       [290, 291, 292, 293, 294, 295, 296, 297, 298, 299],
       [300, 301, 302, 303, 304, 305, 306, 307, 308, 309],
       [310, 311, 312, 313, 314, 315, 316, 317, 318, 319],
       [320, 321, 322, 323, 324, 325, 326, 327, 328, 329],
       [330, 331, 332, 333, 334, 335, 336, 337, 338, 339],
       [340, 341, 342, 343, 344, 345, 346, 347, 348, 349],
       [350, 351, 352, 353, 354, 355, 356, 357, 358, 359],
       [360, 361, 362, 363, 364, 365, 366, 367, 368, 369],
       [370, 371, 372, 373, 374, 375, 376, 377, 378, 379],
       [380, 381, 382, 383, 384, 385, 386, 387, 388, 389],
       [390, 391, 392, 393, 394, 395, 396, 397, 398, 399],
       [400, 401, 402, 403, 404, 405, 406, 407, 408, 409],
       [410, 411, 412, 413, 414, 415, 416, 417, 418, 419],
       [420, 421, 422, 423, 424, 425, 426, 427, 428, 429],
       [430, 431, 432, 433, 434, 435, 436, 437, 438, 439],
       [440, 441, 442, 443, 444, 445, 446, 447, 448, 449],
       [450, 451, 452, 453, 454, 455, 456, 457, 458, 459],
       [460, 461, 462, 463, 464, 465, 466, 467, 468, 469],
       [470, 471, 472, 473, 474, 475, 476, 477, 478, 479],
       [480, 481, 482, 483, 484, 485, 486, 487, 488, 489],
       [490, 491, 492, 493, 494, 495, 496, 497, 498, 499]])
```

In [38]:

```
def add10(matrix):
    for count_x,x in enumerate(matrix):
        for count_y,y in enumerate(x):
            matrix[count_x][count_y]+=10
    return matrix
```

In [39]:

```
%timeit add10(matrix)
```

```
10000 loops, best of 3: 120 µs per loop
```

In [40]:

```
%timeit np_array+10
```

```
The slowest run took 20.89 times longer than the fastest. This could mean
that an intermediate result is being cached.
1000000 loops, best of 3: 1.39 µs per loop
```

Очевидно, что np.array() работает намного быстрее list().

3) Изучите numpy.linspace(), напишите свою реализацию аналогичной функции с помощью list comprehension, дающий тот же результат, только в виде list, а не numpy.ndarray. Сравните по производительности два варианта на достаточно больших массивах.

In [41]:

```
%timeit np.linspace(-100, 100, 1000000)
```

```
100 loops, best of 3: 13.6 ms per loop
```

In [42]:

```
def my_linspace(start = -100, end = 100, size = 1000000):
    return [start + i * (end - start) / (size - 1) for i in range(size)]
```

In [43]:

```
%timeit my_linspace()
```

```
1 loop, best of 3: 269 ms per loop
```

Очевидно, что встроенная функция np.linspace() работает намного быстрей.

4 Matplotlib

1) Что делает в Ipython Notebook команда %matplotlib inline?

С помощью этого мейджика вывод команд построения отображается в виде строки внутри интерфейсов, таких как блокнот Jupyter, непосредственно под ячейкой кода, которая его создала. Полученные графики также будут сохранены в документе записной книжки.

2) Используя numpy.linspace и генераторы, постройте в matplotlib график какой-нибудь элементарной функции. В минимальном варианте вам потребуются только pyplot из matplotlib и методы pyplot.plot() и pyplot.show(). Помните, что если импортируете модуль под каким-то именем (например, plt), то обращаться надо уже к нему, а не к pyplot.

In [44]:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('График функции f(x) = sin(x) на отрезке [0, 10].')
plt.plot(np.linspace(0, 10, 10000), [np.sin(i) for i in np.linspace(0, 10, 10000)], lab
el=r'$sin(x)$')
plt.legend(loc=0)
plt.show()
```

3) Постройте графики еще нескольких элементарных функций (на том же рисунке) и сделайте к ним подписи (легенду) с формулами (в подписях можно использовать LATEX, как обычно, в знаках $). Добавьте какие-нибудь подписи к осям координат и название графика. Сохраните изображение в файл одной строчкой на Python.

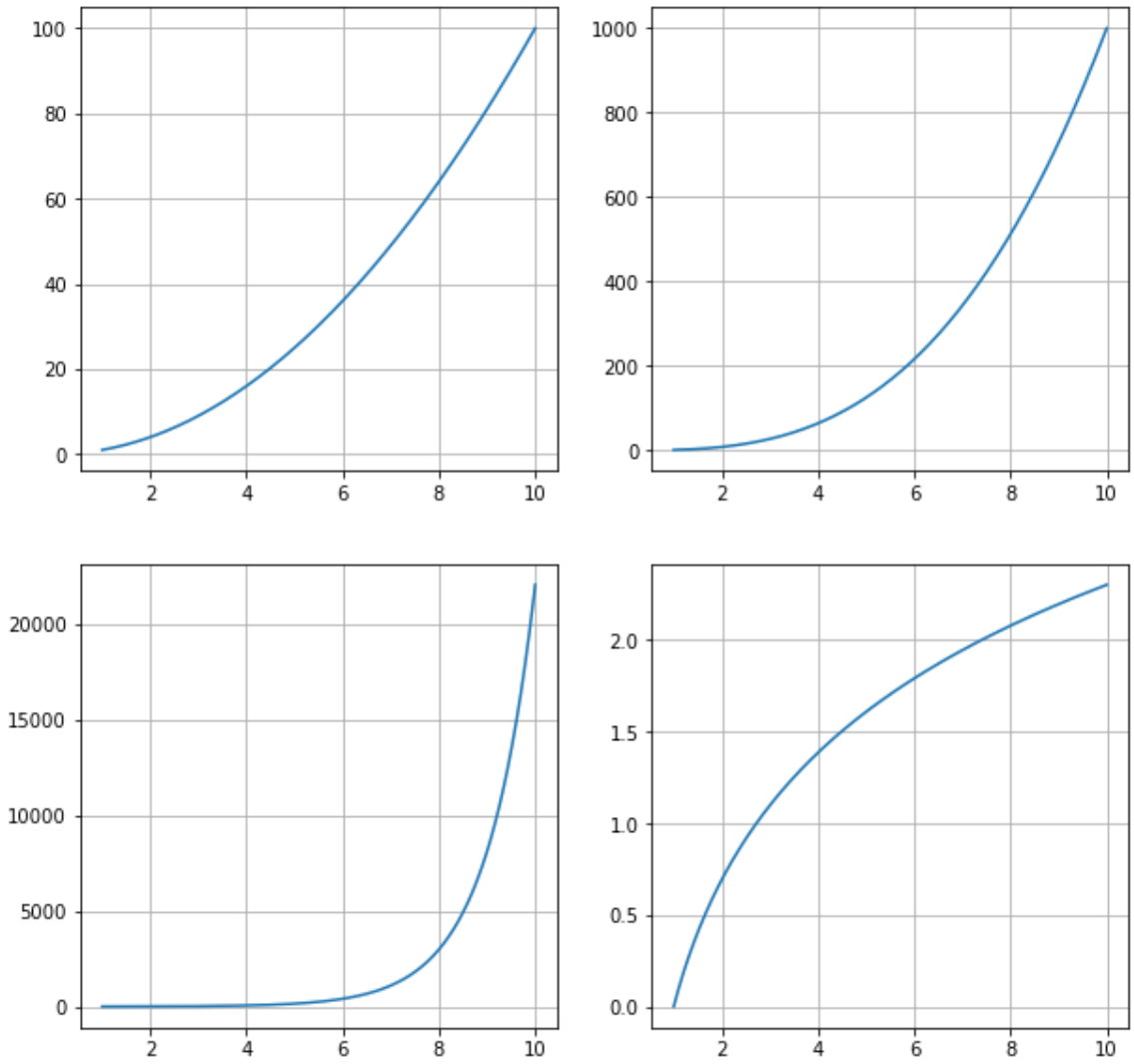
In [45]:

```
plt.title(r'$График \: различных \: функций \: на \: отрезке \: [0, 10].$')
plt.plot(np.linspace(0, 10, 10000), [np.sin(i) for i in np.linspace(0, 10, 10000)], lab
el=r'$sin(x)$')
plt.plot(np.linspace(0, 10, 10000), [np.cos(i) for i in np.linspace(0, 10, 10000)], lab
el=r'$cos(x)$')
plt.plot(np.linspace(0, 10, 10000), [np.exp(i)/10000 for i in np.linspace(0, 10, 10000
)], label=r'$exp(x)$')
plt.legend(loc=0)
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x)$')
#plt.show()

plt.savefig('my_functions.png', format='png', dpi=100)
```

4) Изучите документацию matplotlib и попробуйте построить на одном изображении 4 системы координат, на верхних двух - квадратичную и кубическую параболы, на нижних - экспоненту и логарифм

In [46]:

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))

ax1 = plt.subplot2grid((2,2), (0, 0))
ax2 = plt.subplot2grid((2,2), (0, 1))
ax3 = plt.subplot2grid((2,2), (1, 0))
ax4 = plt.subplot2grid((2,2), (1, 1))

for num, ax in enumerate(fig.axes):
    i += 1
    if num == 0:
        ax.plot(np.linspace(1, 10, 1000), list(map(lambda x: x ** 2, np.linspace(1, 10,
 1000))))
    elif num == 1:
        ax.plot(np.linspace(1, 10, 1000), list(map(lambda x: x ** 3, np.linspace(1, 10,
 1000))))
    elif num == 2:
        ax.plot(np.linspace(1, 10, 1000), list(map(lambda x: np.exp(x), np.linspace(1,
10, 1000))))
    else:
        ax.plot(np.linspace(1, 10, 1000), list(map(lambda x: np.log(x), np.linspace(1,
10, 1000))))
    ax.grid(True)

plt.show()
```



5) Изучите вопрос построения heatmap и изобразите его для функции $f(x, y) = 3xy + x - 2y$ в области $[0; 5]^2$.

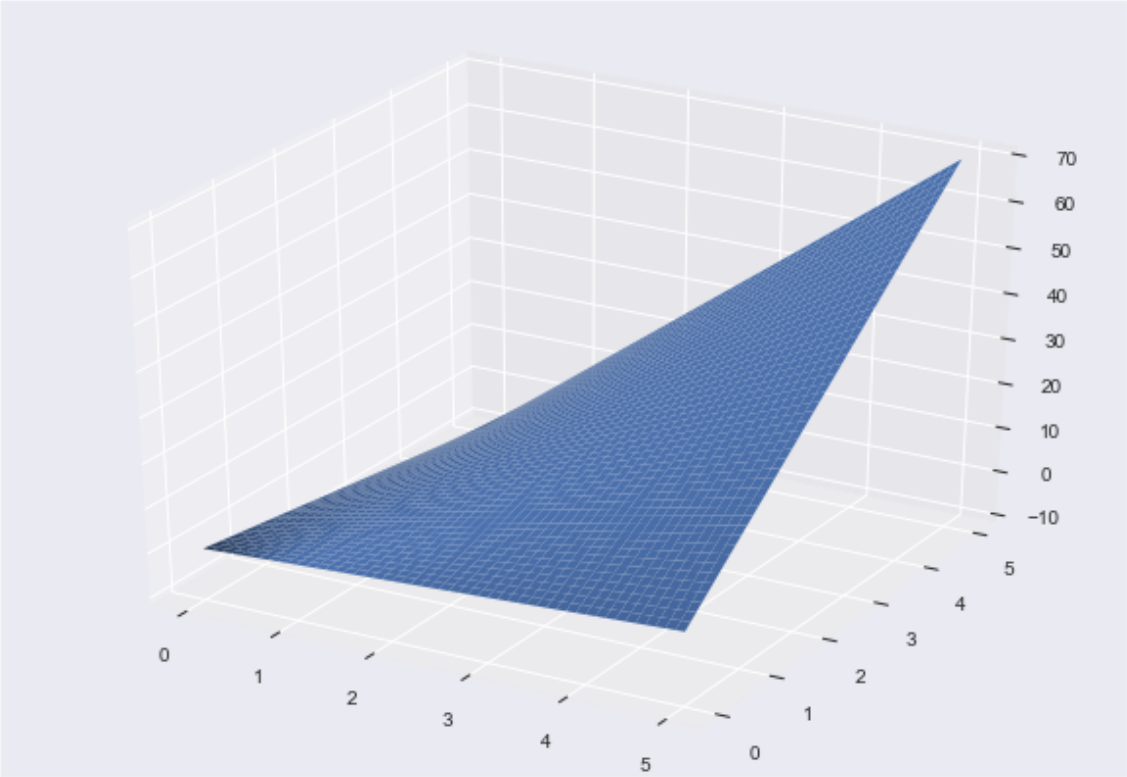
In [47]:

```
# Использую данную библиотеку, так как именно с ней был пример на семинаре.

import seaborn as sns

data = np.zeros(shape=(1000, 1000))
for x in range(1000):
    for y in range(1000):
        data[x, y] = 3 * x/1000*5 * y/1000*5 + x/1000*5 - 2 * y/1000*5

ax = sns.heatmap(data, xticklabels=False, yticklabels=False)
```

6) А теперь постройте трехмерный график той же функции в той же области.

In [50]:

```
from mpl_toolkits.mplot3d import Axes3D

def makeData ():
    x = np.linspace(0, 5, 1000)
    y = np.linspace(0, 5, 1000)
    xgrid, ygrid = np.meshgrid(x, y)

    zgrid = 3 * xgrid * ygrid + xgrid - 2 * ygrid
    return xgrid, ygrid, zgrid

x, y, z = makeData()
fig = plt.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z)
plt.show()
```



5 Всё вместе + SciPy

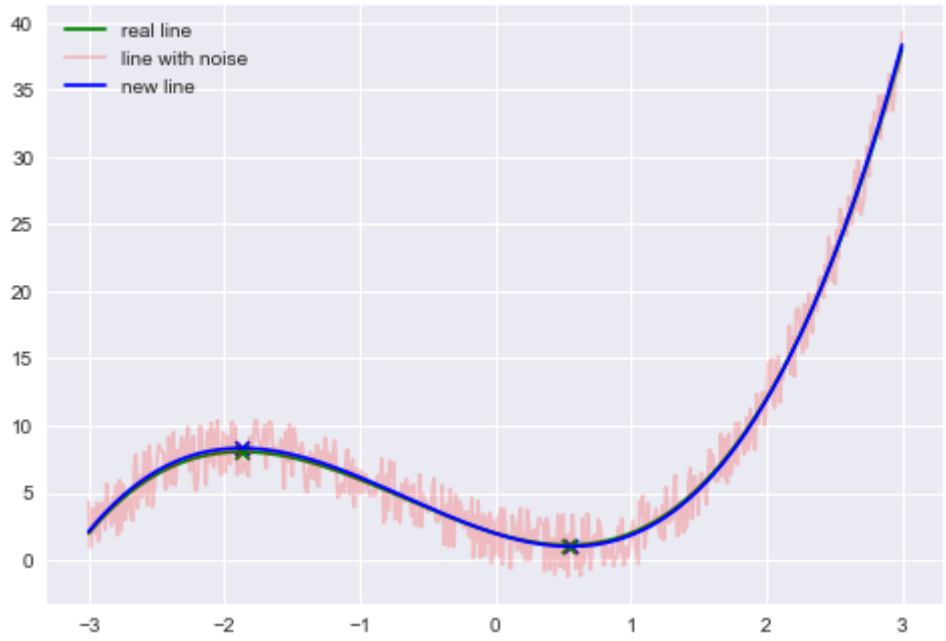
Сгенерируйте точки (y, x) (500 будет достаточно), удовлетворяющие зависимости $y = x^3 + 2x^2 - 3x + 2$, затем, с помощью numpy.random добавьте к координатам у нормальный шум. С помощью scipy восстановите исходную зависимость, считая известной степень полинома. Вам поможет curvefit или scipy.optimize.minimize, запущенный для суммы квадратичных отклонений. У восстановленной зависимости найдите экстремумы численно. Покажите на графике все вместе: исходные точки, восстановленную зависимость, найденные экстремумы.

Дополнительное задание: подберите коэффициенты многочлена, точно проходящего через точки (без добавления нормального шума), составив и решив с помощью scipy систему линейных уравнений.

In [51]:

```
# Я не очень понял условие сформулированной задачи, поэтому проинтерпретировал я его сл
едующим образом:
# 1) Для начала рассмотрим функцию на интервале [-3, 3], так как на нём она принимает н
ебольшие значения +
# имеет как максимум, так и минимум.
# 2) Далее внесём нормальный шум в координаты по оси ординат.
# 3) Потом, используя функцию curve_fit() посчитаем востановим исходную зависимость по
 данным с шумом.
# Отмечу, что я не понял, почему не было предложено использовать np.polyfit(func, 3), э
то же проще)
# 4) Далее нахожу с помощью scipy.optimize.minimize() локальные минимумы и максимумы фу
нкции, коэффициенты
# для которой мы нашли по данным с шумом.
# 5) Далее нахожу с помощью scipy.optimize.minimize() локальные минимумы и максимумы ис
ходной функции.
# 6) Изображаю всё на графике.

from scipy.optimize import curve_fit, minimize

x_real = np.linspace(-3, 3, 500)
y_real = np.array([x ** 3 + 2 * x ** 2 - 3 * x + 2 for x in x_real])
y_new = list(map(lambda y: y + (np.random.rand() - 0.5) * 5, y_real))

def func(x, a, b, c, d):
    return a * x ** 3 + b * x ** 2 + c * x + d

popt, pcov = curve_fit(func, x_real, y_new)
print('Found coeffs:', popt)

min_new = minimize(lambda x: +(popt[0] * x ** 3 + popt[1] * x ** 2 + popt[2] * x + popt
[3]), 0, method='Nelder-Mead').x[0]
max_new = minimize(lambda x: -(popt[0] * x ** 3 + popt[1] * x ** 2 + popt[2] * x + popt
[3]), 0, method='Nelder-Mead').x[0]
min_real = minimize(lambda x: +(x ** 3 + 2 * x ** 2 - 3 * x + 2), 0, method='Nelder-Mea
d').x[0]
max_real = minimize(lambda x: -(x ** 3 + 2 * x ** 2 - 3 * x + 2), 0, method='Nelder-Mea
d').x[0]

plt.plot(x_real, y_real, color='g', label='real line')
plt.plot(x_real, y_new, color='r', label='line with noise', alpha=0.2)
plt.plot(x_real, list(map(lambda x: +(popt[0] * x ** 3 + popt[1] * x ** 2 + popt[2] * x
 + popt[3]), x_real)), color='b',
         label='new line')
plt.scatter(min_new, popt[0] * min_new ** 3 + popt[1] * min_new ** 2 + popt[2] * min_ne
w + popt[3], linewidths=10, marker="x",
            color='b')
plt.scatter(max_new, popt[0] * max_new ** 3 + popt[1] * max_new ** 2 + popt[2] * max_ne
w + popt[3], linewidths=10, marker="x",
            color='b')
plt.scatter(min_real, min_real ** 3 + 2 * min_real ** 2 - 3 * min_real + 2, linewidths=
100, marker="x",
            color='g')
plt.scatter(max_real, max_real ** 3 + 2 * max_real ** 2 - 3 * max_real + 2, linewidths=
10, marker="x",
            color='g')
plt.legend(loc=0)
plt.show()
```

```
Found coeffs: [ 1.02447203  2.02707654 -3.1860579   1.98600316]
```



Так как многочлен ТОЧНО проходит через все точки, то я решал следующую систему:

In [52]:

```
# Как я понял, в доп. задании степень многочлена = 499.

X = np.vander(x_real)
np.linalg.solve(X, y_real)
```

Out[52]:

```
array([  9.40730227e-40,  -4.94858897e-39,  -6.19710645e-38,
         2.84090950e-37,   1.67302218e-36,  -5.89162728e-36,
        -2.36037456e-35,   3.87966769e-35,   1.93787240e-34,
         3.39635593e-34,  -1.28805770e-33,  -5.67296828e-33,
         1.20107276e-32,   8.74334420e-34,  -1.06421956e-31,
         3.67439072e-31,   6.25432041e-31,  -2.71082964e-30,
        -1.34996531e-30,   1.63065856e-29,  -3.45121632e-29,
        -7.35717723e-29,   2.10139165e-28,   1.87948051e-30,
         3.57370933e-27,  -4.36754612e-27,  -1.40748923e-26,
         5.23514232e-26,  -3.35248704e-25,  -2.12159916e-25,
        -6.30327671e-27,   4.36616274e-24,   3.15708537e-23,
        -2.61238208e-23,  -7.09541550e-23,  -2.23040105e-22,
        -2.86235208e-22,   7.46924341e-22,  -5.37924398e-21,
         2.00010600e-20,  -4.71475718e-21,  -5.49185830e-20,
         2.37907363e-19,  -1.33096419e-18,   3.36958873e-18,
         7.42468253e-18,  -1.81997801e-17,   6.02616760e-17,
        -1.61802584e-16,  -8.31597401e-16,   4.49616933e-16,
         4.02672179e-15,   1.54798160e-14,  -2.20143844e-14,
        -1.52512199e-13,   2.26983393e-13,   1.88914521e-13,
        -1.29456224e-12,   3.96728586e-12,  -4.68638364e-12,
        -3.23556459e-12,   8.39831850e-11,  -1.12313893e-11,
         5.80762913e-11,  -1.13550334e-09,  -3.80852669e-09,
         8.91515925e-10,   2.95882275e-08,   8.33045188e-10,
        -3.68241486e-07,   5.64602474e-07,   1.98713846e-06,
        -3.29034928e-06,   6.61399304e-06,   8.12662074e-06,
        -9.08023975e-05,   6.63292659e-05,   1.93286017e-04,
        -1.48201424e-03,   9.17628673e-04,   6.79398757e-04,
        -3.50225173e-03,   5.54602907e-02,  -4.89531536e-02,
         6.32558409e-02,  -1.34705102e-01,  -1.42101574e+00,
         5.87680688e+00,   1.89785945e+00,  -1.55326943e+01,
        -5.78493071e+01,  -3.85818401e+01,   1.11164419e+02,
        -7.08735620e+02,   3.21765067e+03,   7.61641217e+03,
        -1.85254347e+04,  -1.66563383e+04,   1.22363667e+05,
        -1.16560462e+05,  -4.25941035e+05,   1.85977452e+05,
        -5.11098916e+06,   1.10413290e+07,   2.77260185e+07,
        -5.72955430e+07,   1.36488833e+08,  -9.34816526e+07,
        -1.16185074e+09,   9.25984930e+08,   6.05993071e+08,
        -4.80396866e+09,   1.95449751e+10,   4.08059891e+10,
        -5.08460546e+10,   8.45281984e+10,  -5.14880289e+11,
        -2.11948476e+12,   3.18229002e+12,   6.36337053e+12,
         7.72389535e+12,  -6.74114976e+12,  -1.08079734e+14,
         4.56002014e+13,  -1.74678167e+14,  -5.63319170e+12,
         4.85294108e+15,  -1.88424861e+13,  -1.22242287e+16,
        -1.47128834e+16,  -1.44829600e+16,   9.08454806e+16,
        -8.97803496e+16,   5.46064232e+16,  -1.86058155e+17,
        -3.47639710e+18,   2.52641113e+18,   2.00035260e+19,
         6.21255174e+19,  -2.71755106e+19,  -2.63669630e+20,
        -2.15944843e+20,  -1.12464544e+21,   7.44582716e+20,
         5.55277641e+21,   7.44101020e+20,   1.07509366e+22,
         8.60156634e+21,  -9.64522063e+22,  -4.48727872e+22,
         5.29317896e+22,  -7.54521346e+23,   2.47866397e+24,
         4.23585780e+24,  -1.02108042e+25,   9.14364688e+24,
         9.93305871e+23,  -1.00759852e+26,  -1.12104803e+26,
         4.66454458e+25,   7.01011043e+26,   1.18657186e+27,
         2.14138127e+27,  -4.21692557e+27,  -5.84686028e+27,
         1.48047689e+28,  -6.24646643e+28,  -7.71691473e+28,
         1.28572320e+29,   3.13640813e+29,   3.59739009e+29,
        -8.05759136e+29,  -6.80137609e+29,  -4.87300452e+29,
        -3.63772792e+30,   4.96442489e+30,   4.38047879e+31,
```

```
 4.49487344e+31,  -2.67709115e+31,  -1.10897661e+32,
-9.55332437e+32,  -1.27545724e+33,   3.58437706e+32,
 5.23105749e+33,   1.17930951e+34,  -5.98196174e+33,
 1.58165254e+34,   4.53560604e+34,  -1.49958145e+35,
-1.51806632e+35,   1.92865736e+35,  -5.30911879e+35,
-2.33211553e+36,   1.48234208e+36,   9.19207092e+36,
 6.40592274e+36,   4.29048205e+36,   6.36294380e+36,
-8.31933411e+36,  -1.67722406e+38,  -2.48454057e+38,
 2.27342045e+38,   3.57213090e+38,   3.33659069e+38,
 5.70867709e+38,   2.19370938e+39,  -2.90750794e+38,
-1.07808040e+40,   1.00194305e+40,  -5.94500118e+39,
-5.41612631e+40,   8.52461268e+40,   9.23408989e+40,
-2.00804103e+41,  -3.73229426e+41,   3.44213317e+41,
 1.11636279e+42,   1.06094418e+42,   1.19194892e+42,
-7.37473464e+42,  -4.80470886e+42,   1.03083470e+42,
-1.30924739e+43,   5.50648973e+43,  -5.00527918e+42,
-6.76323905e+43,   1.90514087e+44,  -4.68156352e+43,
-3.13050180e+43,  -2.10174780e+43,  -9.13254140e+44,
-1.79961716e+45,  -1.14234454e+44,   8.15767667e+45,
 1.83130317e+45,   4.73943698e+45,   3.24871025e+45,
-6.53041239e+46,   2.26466194e+46,   7.93275032e+46,
-8.18013365e+46,  -2.69547755e+46,  -1.07057852e+47,
 1.98016759e+47,   4.75283130e+47,   9.78623039e+46,
-4.44861505e+47,  -1.38133025e+48,   1.77503469e+48,
-5.54843919e+47,  -1.17036202e+48,   1.57513962e+48,
-1.67293873e+49,   1.47226170e+49,   1.82950368e+49,
-3.75605163e+49,   5.57596361e+49,   1.41214681e+50,
 3.23533076e+49,  -4.56898748e+50,  -6.32385576e+50,
 3.61434045e+50,   1.16295152e+51,   5.31684403e+50,
-1.19864136e+50,   4.74678350e+49,  -4.13923767e+51,
-3.91709640e+50,   8.29982567e+51,  -4.08519456e+51,
 6.02902124e+51,  -1.00774756e+52,  -3.30802710e+52,
 5.71132984e+52,  -2.60150758e+52,  -5.11687938e+52,
 1.89805520e+53,  -7.24659025e+51,  -1.28868318e+53,
-1.43842662e+53,  -3.06688263e+53,   5.48395394e+53,
 3.24770233e+53,  -9.09752619e+53,   6.12802812e+53,
 9.10651696e+53,  -9.99606753e+53,   5.31518338e+53,
-1.24114485e+53,  -3.57327565e+54,  -1.45232638e+54,
 5.41094249e+54,   8.27997220e+54,  -6.47572835e+54,
-8.34422860e+54,   5.45454291e+54,  -1.01268283e+54,
 1.17113187e+55,  -1.97397122e+55,  -3.39799889e+55,
 6.43676622e+55,   3.57511420e+55,   4.34607817e+54,
-9.08702522e+55,  -2.23265644e+56,   2.20730169e+56,
 3.54898147e+56,  -1.54177088e+56,  -2.95903809e+56,
-1.15917445e+56,   2.25573563e+56,   1.28672410e+55,
-5.47839996e+55,   2.29639351e+56,  -3.30406307e+56,
 5.43197629e+56,   4.40517393e+56,  -1.59274072e+57,
-2.15894457e+56,   8.18343201e+56,   5.55252145e+56,
 6.47494019e+56,  -1.40023165e+57,  -6.55274446e+55,
 1.53049962e+57,  -7.57216230e+56,  -7.81071716e+56,
-1.42489958e+57,  -1.92834479e+56,   4.05778649e+57,
 1.05519960e+57,  -2.97217764e+57,  -8.69739702e+56,
 2.98213840e+55,  -7.25652381e+56,   6.29714240e+56,
 1.57954106e+57,   7.10930848e+56,  -3.74997459e+56,
-1.42176016e+57,  -6.35444284e+56,   9.16867736e+56,
-1.77274552e+56,  -1.08201425e+56,   1.14536291e+57,
-5.73332370e+56,  -9.61210324e+56,   1.22929061e+57,
 4.72168194e+56,  -1.75160253e+57,  -2.33769542e+56,
 1.65406530e+57,  -1.11844766e+56,  -7.66308902e+56,
 4.47833251e+56,  -2.33919002e+56,  -4.05372239e+56,
 5.65625435e+56,   1.79875609e+56,  -2.85381909e+56,
```

```
-9.72436903e+55,  -3.36125214e+55,   1.12283782e+56,
 1.15992222e+56,  -8.33084263e+55,  -6.74056893e+55,
 2.40429194e+55,   1.86520349e+55,   9.08470380e+54,
 1.52351055e+54,  -1.30655577e+55,  -2.67190478e+54,
 6.02691697e+54,  -2.51942742e+54,   8.80407065e+53,
 4.14464212e+54,  -3.07635698e+54,  -1.88019741e+54,
 1.78371073e+54,   1.64546038e+53,  -6.14685970e+53,
-5.92652053e+52,   5.98467613e+53,   1.58316296e+53,
-6.81077943e+53,   8.54616729e+51,   3.99658394e+53,
-1.04860810e+53,  -1.01935538e+53,   5.78585902e+52,
 1.17236415e+51,  -8.75984499e+51,  -6.31208349e+51,
 2.21815986e+51,   1.34423654e+52,  -6.04780261e+51,
-7.32159935e+51,   4.23133034e+51,   1.46315590e+51,
-1.05152863e+51,   2.24693280e+50,  -1.40203048e+50,
-1.84642008e+50,   1.44786193e+50,   4.30492130e+49,
-2.46466638e+49,  -1.43656711e+49,  -1.59894918e+48,
 9.81192748e+48,  -1.55681538e+48,  -3.80524397e+48,
 1.99402320e+48,   2.84252123e+47,  -6.73919007e+47,
 4.21134701e+47,   6.09648876e+46,  -2.38247863e+47,
 2.25772760e+46,   7.08299937e+46,  -6.02291748e+45,
-1.37044109e+46,  -8.63942280e+44,   1.77426952e+45,
 7.57815284e+44,  -1.66085279e+44,  -1.98498843e+44,
 2.73804616e+43,   3.43815707e+43,  -1.13355324e+43,
-6.78782737e+42,   4.13671387e+42,   1.90467624e+42,
-1.16077000e+42,  -4.70279037e+41,   2.60908842e+41,
 7.46556769e+40,  -4.75277268e+40,  -4.16812386e+39,
 6.92160613e+39,  -1.31231343e+39,  -7.78961006e+38,
 4.58150494e+38,   6.30500632e+37,  -8.17764765e+37,
-3.01289850e+36,   1.00005817e+37,   2.29683534e+34,
-8.63920720e+35,  -3.21260940e+33,   4.61484336e+34,
 3.55007070e+33,  -9.46773144e+30,  -7.76863054e+32,
-3.13288159e+32,   1.02936506e+32,   4.08122893e+31,
-9.97240695e+30,  -3.41950332e+30,   7.57465429e+29,
 2.18932816e+29,  -4.66217021e+28,  -1.13022071e+28,
 2.36589810e+27,   4.81331589e+26,  -9.99058450e+25,
-1.70912315e+25,   3.52596853e+24,   5.08255283e+23,
-1.04122391e+23,  -1.26665312e+22,   2.56917044e+21,
 2.63998828e+20,  -5.27850993e+19,  -4.58163552e+18,
 8.98095942e+17,   6.57756884e+16,  -1.25593050e+16,
-7.74230003e+14,   1.42956883e+14,   7.38554829e+12,
-1.30819655e+12,  -5.62443713e+10,   9.47587785e+09,
 3.35351085e+08,  -5.32768604e+07,  -1.52583281e+06,
 2.26775686e+05,   5.11706494e+03,  -7.07469286e+02,
-1.20403400e+01,   1.54852599e+00,   1.84192995e-02,
-2.23509142e-03,  -1.59112891e-05,   1.93335793e-06,
 5.08798433e-09,   9.99999999e-01,   2.00000000e+00,
-3.00000000e+00,   2.00000000e+00])
```

6 Pandas + Scikit-learn

Считайте выборку из примера про детектирование кожи на фотографии с семинара с помощью библиотеки pandas. Продемонстрируйте индексацию по строкам и столбцам, labeled-based и index-based. Добавьте в DataFrame столбцы, соответствующие попарным произведениям признаков и модулям попарных разностей. Запустите на исходных признаках и на модифициро- ванных KNeighboursClassifier, LogisticRegression и Random Forest из sklearn и сравните качество работы каждого метода до и после преобразования признаков.

In [53]:

```
import pandas as pd
```

In [54]:

```
# Считаем данные.

data = pd.read_table('Skin_NonSkin.txt', header=None)
```

In [55]:

```
# Посмотрим на первые 5 строк в таблице.

data.head(5)
```

Out[55]:

	0	1	2	3
0	74	85	123	1
1	73	84	122	1
2	72	83	121	1
3	70	81	119	1
4	70	81	119	1

Продемонстирую основные примеры работы с pd.DataFrame().

In [56]:

```
# Посмотрим, например, на второй столбец.

data[1]
```

Out[56]:

```
0          85
1          84
2          83
3          81
4          81
5          80
6          81
7          81
8          87
9          87
10         88
11         88
12         88
13         89
14         85
15         86
16         86
17         85
18         84
19         85
20         88
21         91
22         91
23         92
24         92
25         91
26         88
27         86
28         86
29         87
          ...
245027    161
245028    161
245029    161
245030    161
245031    161
245032    161
245033    164
245034    164
245035    164
245036    164
245037    164
245038    163
245039    163
245040    163
245041    162
245042    162
245043    162
245044    162
245045    162
245046    162
245047    162
245048    162
245049    162
245050    162
245051    162
245052    162
245053    162
245054    162
```

```
245055    162
245056    255
Name: 1, Length: 245057, dtype: int64
```

In [57]:

```
# Посмотрим на 2-ую строчку. Отметим, что метод .ix устарел. Нужно использовать либо .l
oc,
# либо .iloc (в зависимости от того, что мы хотим получить: индексацию по значениям или
 по индексам.)

data.loc[1]
```

Out[57]:

```
0     73
1     84
2    122
3      1
Name: 1, dtype: int64
```

In [58]:

```
# Выберем только те строчки, в которых значение первого признака больше 155.

data_2 = data[data[0] > 155]
data_2
```

Out[58]:

	0	1	2	3
74	156	185	230	1
75	164	190	236	1
76	165	189	235	1
77	168	190	238	1
78	168	189	244	1
79	170	190	247	1
80	175	195	250	1
81	180	201	253	1
82	187	203	255	1
83	192	207	255	1
84	194	208	255	1
85	196	209	253	1
86	205	214	255	1
87	206	215	253	1
88	207	215	252	1
89	210	217	250	1
90	215	217	251	1
91	217	220	251	1
92	220	222	252	1
93	221	224	255	1
94	222	228	255	1
95	218	226	255	1
96	215	223	255	1
97	213	221	255	1
98	211	219	255	1
99	210	218	255	1
100	211	219	255	1
101	211	219	255	1
102	210	218	255	1
103	211	219	255	1
...	...	...	...	...
245027	162	161	110	2
245028	162	161	110	2

	0	1	2	3
245029	162	161	110	2
245030	162	161	110	2
245031	162	161	110	2
245032	162	161	110	2
245033	165	164	113	2
245034	165	164	113	2
245035	165	164	113	2
245036	165	164	113	2
245037	165	164	113	2
245038	164	163	112	2
245039	164	163	112	2
245040	164	163	112	2
245041	163	162	112	2
245042	163	162	112	2
245043	163	162	112	2
245044	163	162	112	2
245045	163	162	112	2
245046	163	162	112	2
245047	163	162	112	2
245048	163	162	112	2
245049	163	162	112	2
245050	163	162	112	2
245051	163	162	112	2
245052	163	162	112	2
245053	163	162	112	2
245054	163	162	112	2
245055	163	162	112	2
245056	255	255	255	2

101781 rows × 4 columns

In [59]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split
```

In [60]:

```
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.iloc[:, -1
], test_size=0.3, random_state=0)
```

Запускаю на исходных признаках KNeighborsClassifier.

In [61]:

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
0.999496721891
```

Запускаю на исходных данных LogisticRegression.

In [62]:

```
model = LogisticRegression()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
0.918645773824
```

Запускаю на исходных данных RandomForestClassifier.

In [63]:

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
0.999374302892
```

Добавим в DataFrame столбцы, соответствующие попарным произведениям признакам и модулям попарных разностей.

In [64]:

```
# Создадим новый объект DataFrame, в котором будет представлена изменённая таблица.

data_new = data.copy()
```

In [65]:

```
for i in range(4):
    for j in range(4):
        if i < j:
            data_new[r'${} \cdot {}$'.format(i, j)] = data_new[i] * data_new[j]
            data_new[r'$|{} - {}|$'.format(i, j)] = abs(data_new[i] - data_new[j])
```

In [66]:

```
data_new.head(5)
```

Out[66]:

	0	1	2	3	$0 \cdot 1$	$\|0 - 1\|$	$0 \cdot 2$	$\|0 - 2\|$	$0 \cdot 3$	$\|0 - 3\|$	$1 \cdot 2$	$\|1 - 2\|$	$1 \cdot 3$
0	74	85	123	1	6290	11	9102	49	74	73	10455	38	85
1	73	84	122	1	6132	11	8906	49	73	72	10248	38	84
2	72	83	121	1	5976	11	8712	49	72	71	10043	38	83
3	70	81	119	1	5670	11	8330	49	70	69	9639	38	81
4	70	81	119	1	5670	11	8330	49	70	69	9639	38	81

In [67]:

```
X_train, X_test, y_train, y_test = train_test_split(data_new.iloc[:, data_new.columns !
= 3],
                                                    data_new.iloc[:, 3], test_size=0.3,
 random_state=0)
```

Запускаю на модифицированных признаках KNeighborsClassifier.

In [68]:

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
0.999251883892
```

Запускаю на модифицированных данных LogisticRegression.

In [69]:

```
model = LogisticRegression()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
0.999007045894
```

Запускаю на модифицированных данных RandomForestClassifier.

In [70]:

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
predicted = model.predict(X_test)
print(sum(y_test == predicted) / len(y_test))
```

```
1.0
```

Для KNeighborsClassifier качество особо не изменилось. Для LogisticRegression качество повысилось. Для RandomForestClassifier качество стало просто запредельным:).