Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-1 «Системы автоматического управления»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Студент Шевченко Алексей Дмитриевич
фамилия, имя, отчество

Группа ИУ1-12Б

Тип практики Учебно-технологический практикум

Название предприятия Кафедра «Системы автоматического управления»

Студент 24/12/2021 Шевченко А.Д.
(Подпись, дата) (И.О. Фамилия)

Руководитель практики 24/12/2021 А.А. Николаев
(Подпись, дата) (И.О. Фамилия)

Оценка

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-1
(Индекс)

_____ К.А. Неусыпин
(И.О. Фамилия)

« 24 » сентября 20 21 г.

З А Д А Н И Е
на прохождение учебной практики

Студент группы ИУ1-12Б
Шевченко Алексей Дмитриевич
(Фамилия, имя, отчество)

Задание Ознакомление с системой контроля и доставки версий Git, удалённым репозитроием проектов GitHub, основами программирования на Python, а также библиотеками Pandas и Numpy. Реализация программного кода.

Оформление отчета по практике:
Отчет на 25 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
оформление графического материала в отчете по практике не предусмотрено

Дата выдачи задания « 24 » сентября 20 21 г.

Руководитель Практики 24/09/2021 А.А. Николаев
(Подпись, дата) (И.О. Фамилия)

Студент 24/09/2021 А.Д.Шевченко
(Подпись, дата) (И.О. Фамилия)

Содержание

Введение....2
Тема 1. Введение в python....3
1.1. Jupyter notebook....3
1.2. Типы данных....4
1.2.1 Целочисленный тип (int)....4
1.2.2 Действительные числа (float)....4
1.2.3 Логический тип (boolean)....5
1.2.4 Строка (str)....5
1.3. Операции над различными типами данных....5
1.3.1 Основные операции....6
1.3.2 Округление....6
1.3.3 And & or....6
1.4 Метод print....7
1.4 Приведение типов....8
1.5 Условные операторы if и else....8
1.6 Переменные....9
1.7 Input()....10
1.8 Массивы (list)....10
1.8.1 Виды массивов....11
1.8.2 Функции, методы и действия, применяемые к массивам....12
1.9 Циклы....15
1.10 Функции (def)....16
1.10 Решение уравнений.....16
Тема 2. Git и GitHub....17
2.1 Знакомство с Git....17
2.2 Знакомство с GitHub....18
2.2 Основные действия с Git....18
Тема 3. Практика. Определение вида треугольника....20
Тема 4. Библиотеки. Numpy & Pandas....20
4.1 Библиотеки....20
4.2 Библиотека Numpy....20
4.3 Библиотека Pandas....23
Заключение....24
Список литературы....25

Введение

Самым популярным языком программирования в наши дни безусловно можно назвать Python. По популярности он значительно опережает другие языки написания кода. Чем же обусловлена такая известность данного языка? Во-первых, это язык общего назначения, что значит адаптивность для многих задач. Во-вторых, он ориентирован на повышение производительности разработчика, читаемости кода и обеспечивает переносимость написанных на нем программ.

Python – высокоуровненвый язык программировани. Это означает, что синтаксис этого языка довольно минималистичен и понятен даже не особенно разбирающемуся в программировании человеку. Особенностью языка является то, что выделение блоков кода осуществляется при помощи пробельных отступов. Синтаксис языка не загроможден «лишними» деталями, что упрощает восприятие кода, снижает утомляемость и благоприятсвтует быстрому выполнению задачи.

Также в данном курсе лекций была рассмотрена система контроля и доставки версий git, а также github.com – веб-сайт для создания и управления git-репозиториев с возможностью совместной разработки проектов. Эти технологии упрощают разработку различных проектов, даруя возможность не только управлять версиями вашей работы, но и создавать различные ветви разработки, а следовательно, появляется возможность одновременно выполнять работу и не бояться, что всё придётся начинать с нуля. Такие инструменты являются незаменимыми вещами для любого программиста.

Впрочем, речь обо всем этом пойдет далее.

Тема 1. Введение в python

1.1. Jupyter notebook

Jupiter notebook – среда разработки, которая удобна тем, что можно сразу увидеть результат выполнения кода и его отдельных фрагментов. При запуске он запускает локальный сервер и открывается в браузере. Первым делом мы увидим следующее окно (рис. 1):

Рис. 1 Начальное окно notebook.

Нас встречают различные директории нашего компьютера, и, нажав на кнопку «New», мы можем создать либо еще одну папку, либо же сам файл jupyter'а под названием «Python 3 (ipykernel)». Создав этот файл нас, встретит следующее окно (рис. 2):

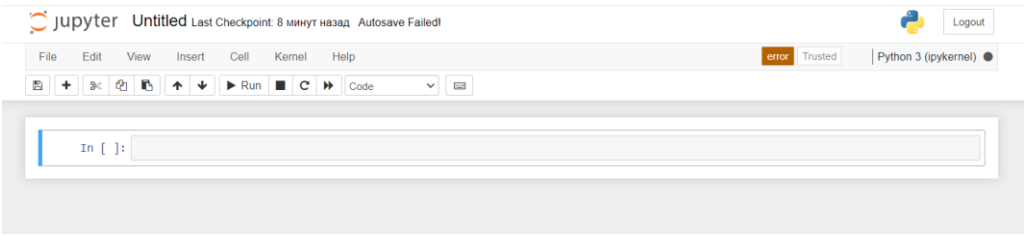

Рис. 2 Окно разработки.

Больше всего нас интересует, конечно же, поле «In [ ]», в котором мы и будем писать дальнейший код. Также, для упрощения работы с кодом, существуют некоторые быстрые операции, которые представлены на рисунке 3.

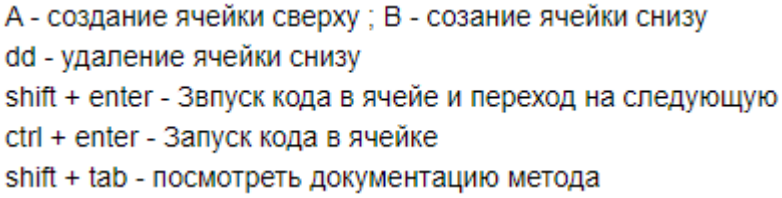

Рис. 3. Быстрые операции с ячейками.

1.2. Типы данных

Первым делом стоит познакомиться с синтаксисом языка программирования Python. Для начала мы рассмотрим различные типы данных. Для определения типа данных можно использовать функцию type(x), где x – какие-то данные, тип которых надо определить. Пример работы этой функции представлен на рисунке 4.

```
In [10]: type(1.3)
Out[10]: float
In [45]: type(None)
Out[45]: NoneType
```

Рис. 4. Определение типа с помощью функции.

Далее мы рассмотрим все основные типы данных в языке программирования Python.

1.2.1 Целочисленный тип (int)

Целочисленный тип данных включает в себя целые числа, такие как -2, 7, 10000 и т.д. Он является простейшим и самым распространенным типом данных во всех языках программирования. Однако в отличие от многих других языков программирования, Python не имеет как такового разграничения по целочисленному типу, как, например, разделение на short, int, long. Пример задания целочисленного типа представлен на рисунке 5.

```
In [1]: type(10000)
Out[1]: int
```

Рис. 5. Тип данных int.

1.2.2 Действительные числа (float)

Из названия следует, что к типу float относятся все действительные числа, такие как 1.7, 1/3, 0.5 и т.д. Такой тип данных удобен, когда вычисления становятся более конкретными и точными, чем в int. Задание типа float представлено на рисунке 6.

```
In [3]: type(1.777)
Out[3]: float
```

Рис. 6. Тип данных float.

1.2.3 Логический тип (boolean)

Этот тип создан для выполнения логических операций и может принимать только 2 значения: true, false. Значение True (или же 1) означает истину, значение False (или же 0), наоборот - ложь. Иногда True и False обозначают, как 1 или 0. Такое обозначение сложилось исторически, ведь 1 электросхемах значит присутствие сигнала, 0 же – его отсутствие. Пример задания такого типа представлен на рисунке 7.

```
In [2]: type(True)
Out[2]: bool
```

Рис. 7. Тип данных boolean.

1.2.4 Строка (str)

Str – это тип данных, который принимает в себя определенную строку текста и задается через кавычки. Задание типа str представлен на рисунке 8.

```
In [3]: type(1.777)
Out[3]: float
```

Рис. 8. Тип данных str.

1.3. Операции над различными типами данных

Над каждым из типов данных можно производить определённые операции, для каких-то типов они одинаковы, для каких-то различны. Особенности различных типов данных будут рассмотрены ниже.

1.3.1 Основные операции

Основные операции над различными типами данных представлены в таблице 1.

Тип операции	Пример	Результат выполнения
Сложение	2+8	10
Конкатенация	“str1” + “str2”	x1x2
Вычитание	3.5-6.4	-2.900000000000000004
Умножение	7*3	21
Деление	3/3	1
Целочисленное деление	5.5//2	2.0
Остаток от деления	3.4%1.3	0.7999999999999998
Возведение в степень	3.5**(2.3)	17.838424804458295
Сравнение	0.333333333333333<=1/3	True

Таблица 1. Операции над данными

1.3.2 Округление

Можно заметить, что некоторые операции дают не совсем тот результат, который мы ожидаем увидеть. Это связано с тем, что в компьютере может храниться только некоторое количество знаков после запятой из-за ограничений на отводимую память. Поэтому для получения более точного результата надо использовать округление, что представлено ниже (рис. 9)

```
In [11]: round(1/3, 5)
Out[11]: 0.33333
```

Рис. 9. Пример округления

На вход данная функция получает число для округления и количество знаков которые необходимо оставить после запятой. Последний элемент всегда окгруляется по прамивлам математического округления.

1.3.3 And & or

Для операции сравнения существуют такие добавления, как and и or. Операция сравнения с добавлением and (логический и) принимает в себя два сравнения и возвращает

True в случае, если оба эти сравнения равны True, или False, если хотя бы одно из них равно False. Пример сравнения с добавлением and представлен на рисунке 10.

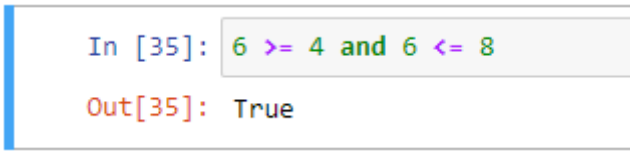

Рис. 10. Сравнение с and.

Как можно понять, операция сравнения с добавлением or (логическое или) возвращает true, когда хотя бы одно из представленных сравнений возвращает true. Пример сравнения с добавлением or представлен на рисунке 11.

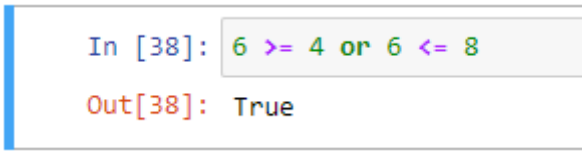

Рис. 11. Сравнение с or.

1.4 Метод print

Метод print выводит в консоль то, что мы ему передадим. Этот метод удобен тем, что мы далеко не всегда будем писать код по строкам отдельно в каждой ячейке. Но какие-то промежуточные значения нам будет необходимо выводить, потому что Jupyter notebook стандартно выводит только последнюю строку в ячейке. Как работает этот метод, смотрите на рисунке 12.

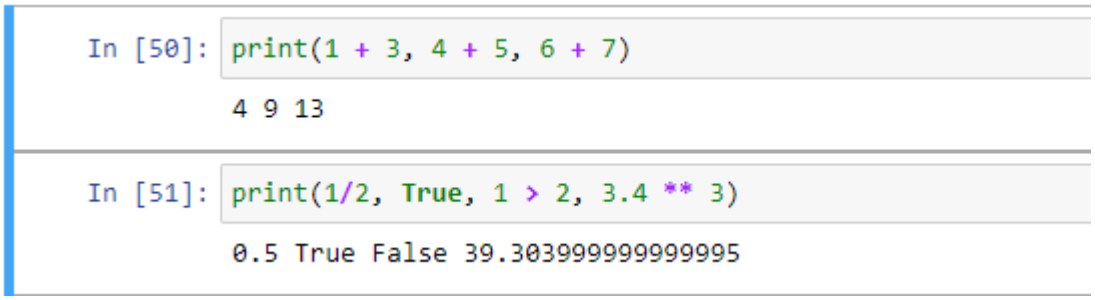

Рис. 12. Пример работы print.

Как можно заметить, этот метод действительно удобен и может применятся в различных ситуациях.

1.4 Приведение типов

Иногда можно столкнуться с тем, что нам будем необходимо перевести один тип в другой для каких-то дальнейших операций. Например, существуют следующие операции приведения типов:

1. Преобразование в строку (рис. 13).

```
In [58]: str(None)
Out[58]: 'None'

In [60]: str(1/3)
Out[60]: '0.3333333333333333'
```

Рис. 13. Приведение числа в строку

2. Преобразование в число (рис. 14).

```
In [63]: int("1")
Out[63]: 1

In [64]: type(int("1"))
Out[64]: int

In [64]: type(int("1"))
Out[64]: int
```

Рис. 14. Приведение строки в число

Таким образом можно менять типы данных и в дальнейшем использовать уже их преобразованные аналоги.

1.5 Условные операторы if и else

Условные операторы – операторы, которые проверяют, правдиво ли введенное условие или нет, и в зависимости от этого, выполняется определенный код. Выглядит данная структура в своем стандартном виде следующим образом:

If «условие»:

Код, который выполняется, если условие = True

Else:

Код, который выполняется, если условие = False

Стоит отметить, что условного оператора else может и не быть – тогда просто ничего не будет выполнятся при несправедливости условия. Также, в коде else может содержаться еще один условный оператор if (см. рис. 14)

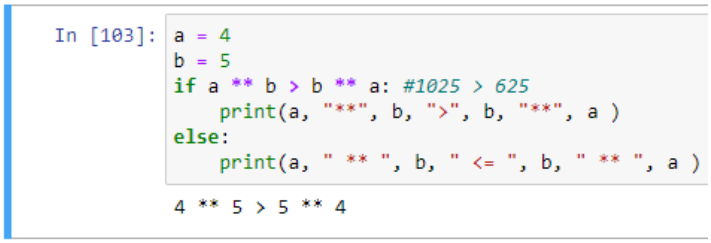

Рис. 14. Пример кода с использованием if и else if.

1.6 Переменные

Переменная – выделенная область памяти, зависящая от типа переменной, которой дается определенное имя и значение. Таким образом, можно задавать и выполнять операции над переменными, далее сохраняя полученное значение, потом снова используя и меняя его. Такой подход значительно упрощает разработку даже простейших программ, не говоря уже о сложных. Пример программы с использованием переменных представлен ниже (рис. 15).

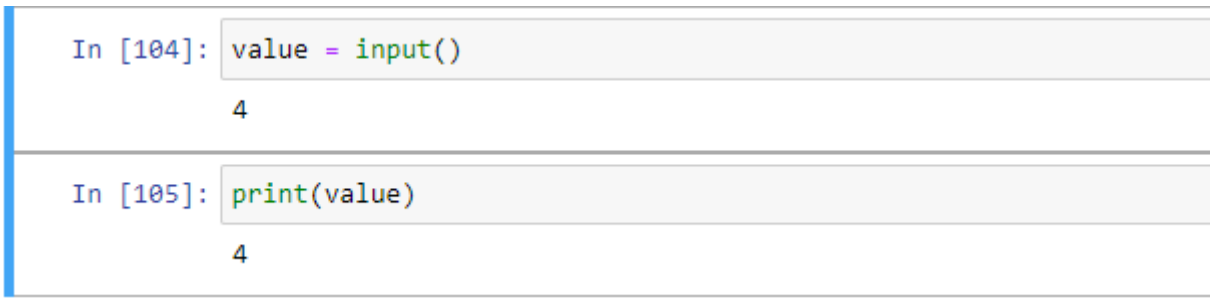

Рис. 15. Программа с переменными.

Также значительно большее применение переменные получают при использовании метода input.

1.7 Input()

Метод input() принимает значение, которое пользователь вводит с клавиатуры. Этот метод открывает огромные возможности в разработке приложений, ведь именно при помощи него можно дать пользователю не только видеть заранее прописанный вариант кода, но и самому влиять на то, к какому результату приведет та или иная программа. Тот же код, что и в предыдущем примере (рис. 15), но доработанный возможностью вводить различные числа, представлен на рисунке 16.

```
In [108]: a = int(input())
          b = int(input())
          if a ** b > b ** a: #1025 > 625
              print(str(a) + " ** " + str(b) + " > " + str(b) + " ** " + str(a))
          else:
              print(str(a) + " ** " + str(b) + " <= " + str(b) + " ** " + str(a))

          2
          3
          2 ** 3 <= 3 ** 2
```

Рис. 16. Метод input().

Input() принимает на вход строку и, как мы видим, использовав приведенный выше метод приведения переменных int(), можно спокойно перевести введенную строку в числовой тип.

1.8 Массивы (list)

Массив – структура данных, в которой хранятся различные данные. Каждому элементу массива присваивается определенный индекс, начиная с 0. В отличие от многих других языков, python позволяет в каждой ячейке массива хранить различные типы данных, от числового до строки. Пример создания массива представлен на рисунке 17.

```
In [1]: arr = [5, 6, 7, 8, 9]

In [5]: type(arr)

Out[5]: list
```

Рис. 17. Задание Массива.

Также, при определении массива, можно добавить в него некоторые данные с помощью функции append() (см. рис. 18)

```
arr.append(5)
arr
```

```
Out[3]: [5]
```

Рис. 18. Добавление данных в массив.

Давайте на данном примере рассмотрим структуру массива. Как было сказано выше, в этом массиве хранятся различные данные. 5, 6, 7, 8, 9 – все это числа типа int. Значит, этот массив состоит всего из 3 элементов, каждый из которых внутренне пронумерован от 0 до 4. 0 – первый элемент (5), 2 – третий элемент (6). Чтобы обратиться к какому-либо из этих элементов, нужно использовать такую запись:

arr[x], где x – число от 0 до 4.

1.8.1 Виды массивов

Массивы бывают разных видов, основные из них будут представлены ниже

1. Одномерный массив

Такой массив данных состоит из одной «строки», то есть, просто из идущих друг за другом элементов (см. рис. 18)

2. Двумерный массив

Это массив, которой состоит как бы из строк и столбцов, или же из двух или более одномерных массивов. Пример задания и обращения к элементу такого массива представлен ниже (рис. 19)

```
In [128]: matrix = [[1, 2], [3, 4]]

In [129]: matrix
Out[129]: [[1, 2], [3, 4]]
```

Рис. 19. Двумерный массив.

Также, подобно одномерному массиву, мы можем обратиться к какому-либо элементу внутреннего массива, использовав следующую запись (рис. 20):

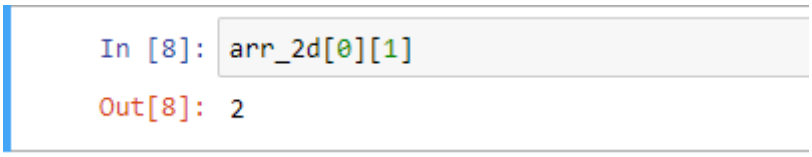

Рис. 20. Обращение к первому элементу первого массива.

1.8.2 Функции, методы и действия, применяемые к массивам

Например, существует функция Len(«массив»), которая возвращает количество элементов массива, или по-другому, его длину.

Методы, которые можно вызвать через «.»:

1. .append() – добавление указанного элемента в массив.
2. .clear() – очищает массив, удаляя из него все элементы.
3. .copy() – копирует массив.
4. .count() – возвращает количество встреченных в массиве указанных элементов.
5. .extend() – дополняет массив другим объектом.
6. .index() – возвращает индекс первого встреченного указанного значения.
7. .insert() – добавление элемента перед каким-либо индексом.
8. .pop() – удаление и возвращение значения элемента с индексом.
9. .remove() – удаление первое указанное значение.
10. .reverse() – переворот массива.
11. .sort() – сортировка массива в порядке возрастания элементов.
12. .sort(reverse = true) - сортировка массива в порядке убывания элементов

Также, над массивами можно проводить некоторые действия, которые представлены ниже:

1. Сложение (рис. 21)

```
In [4]: [1, 2] + [3, 4]
Out[4]: [1, 2, 3, 4]
```

Рис. 21 Сложение массивов.

2. Умножение на число (рис. 22)

```
In [6]: [1, 2, 3] * 5
Out[6]: [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Рис. 22. Умножение массива на число.

Далее представлены действия, производимые с помощью библиотеки Numpy.

3. Умножение массивов (рис. 23)

```
In [19]: import numpy as np
         A = np.array([1, 2])
         B = np.array([3, 4])
         A * B

Out[19]: array([3, 8])
```

Рис. 23. Умножение двух массивов.

4. Классическое умножение массивов (рис. 24)

```
In [20]: import numpy as np
         A = np.array([1, 2])
         B = np.array([3, 4])
         A.dot(B)

Out[20]: 11
```

Рис. 24. «Строка на столбец» умножение.

5. Вычитание из массива (рис. 25)

```
In [21]: import numpy as np
         A = np.array([1, 2])
         B = np.array([3, 4])
         A - 1

Out[21]: array([0, 1])
```

Рис. 25. Вычитание из массива.

6. Деление на число (рис. 26)

```
In [22]: import numpy as np
         A = np.array([1, 2])
         B = np.array([3, 4])
         A / 2

Out[22]: array([0.5, 1. ])
```

Рис. 26. Деление массива на число.

7. Нахождение обратной матрицы (определение из математики, читай – тот же массив) (рис. 27)

```
In [40]: import numpy as np
         from numpy import linalg as LA

         A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         np.linalg.inv(A)

Out[40]: array([[-4.50359963e+15,  9.00719925e+15, -4.50359963e+15],
                [ 9.00719925e+15, -1.80143985e+16,  9.00719925e+15],
                [-4.50359963e+15,  9.00719925e+15, -4.50359963e+15]])
```

Рис. 27. Обратная матрица.

8. Сумма элементов массива (рис. 28)

```
In [25]: sum([1, 2, 3])

Out[25]: 6
```

Рис. 28. Сложение элементов массива.

1.9 Циклы

Цикл – фрагмент кода, повторяющийся определенное количество раз в зависимости от каких-либо условий. Создание цикла представлено на рисунке 29.

```
In [70]: for element in arr:
             print(element) #Тело (body) цикла

4
3
2
-1
```

Рис. 29. Цикл for.

Также, есть некоторые функции, которые делают работу с массивами более многогранной. Они представлены ниже:

1. range()

   Эта функция создает новый объект, который имеет в себе определенное количество элементов. Пример работы представлен на рисунке 30.

```
In [72]: for i in range(10):
             print(i)

0
1
2
3
4
5
6
7
8
9
```

Рис. 30. range().

2. Срезы

   Этот прием позволяет обращаться к определенным элементам массива. Пример использования среза представлен на рисунке 31.

```
In [41]: arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
         arr[1:5]

Out[41]: [2, 3, 4, 5]

In [82]: arr[1:5:2]

Out[82]: [2, 4]
```

Рис. 31. Срез.

1.10 Функции (def)

Функция – фрагмент кода, к которому можно обращаться неограниченное количество раз, передавая ему какие-либо значения, которые в дальнейшем будут использоваться в данной функции для вычислений. Это удобно для того, чтобы разбить код на четкие и понятные фрагменты, чтобы не загромождать программу тем, чтобы каждый раз писать один и тот же код, ведь его можно просто выделить в отдельную функцию, пример которой представлен на рисунке 32.

```
In [88]: def f(x):#define
             y = x ** 2
             return y

In [89]: f(5)

Out[89]: 25
```

Рис. 32. Создание и использование функции.

В данном примере def – обозначение функции; f – имя функции; x – передаваемый параметр.

1.10 Решение уравнений.

Зная все вышеперечисленное, уже можно создавать свои программы для каких-либо целей. Пример использования таких знаний представлен на рисунке 33.

```
In [43]: a = [1, 2, 3, 4]
         a_0 = a[0]
         a_1 = a[1]
         a_2 = a[2]
         a_3 = a[3]
         x = 1.1
         polynom_value = a_0 + a_1*x + a_2 * x ** 2 + a_3 * x ** 3
         polynom_value

Out[43]: 12.154000000000003
```

Рисунок 33. Решение уравнения.

Эту же программу можно написать и более интересным способом (рис. 34)

```
In [48]: a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
         n = len(a)
         polynom_value = 0.0
         for i in range(n):
             polynom_value = polynom_value + a[i] * x ** i
         polynom_value

Out[48]: 76.42052309000005
```

Рисунок 34. Улучшенное решение уравнения.

Тема 2. Git и GitHub

2.1 Знакомство с Git

Git - система контроля и доставки версий, используемая для управления этапами разработки вашего проекта. Установив git с официального источника, мы можем запустить сам git или командную строку и написать в ней «git», что покажет нам весь арсенал наших возможностей. (рис. 35).

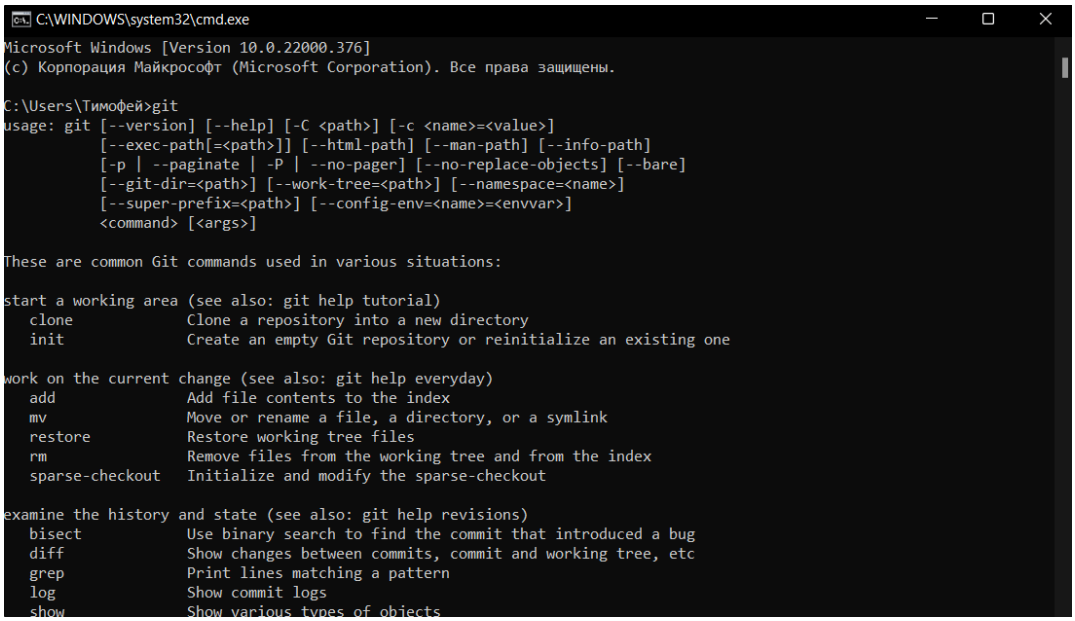

Рис. 35. Знакомство с гит.

2.2 Знакомство с GitHub

GitHub – сайт, позволяющий создавать удаленные репозитории, к которому можно подключить свой локальный репозиторий, который, в свою очередь, создается с помощью git. Пример такого удаленного репозитория см. на рис. 36.

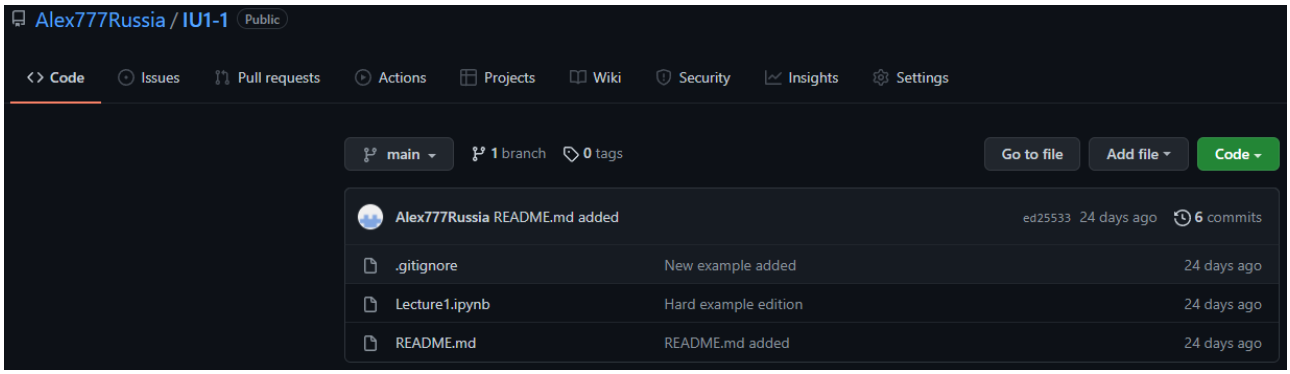

Рис. 36. Удаленный репозиторий.

2.2 Основные действия с Git

1. Git –version – данные о версии вашего гита.
2. Git status - сообщение о статусе данного проекта.
3. Git add (название файла, которого Вы хотите отслеживать) – отслеживание файлов
4. Если Вы вновь хотите перестать отслеживать какой-то файл нужно написать “git rm—cached (название файла)”.

5. Git add . – отслеживание сразу всех элементов.
6. Git commit – сохранение состояния проекта.
7. Git brunch – создание новой ветки разработки проекта.
8. Git switch – изменение разрабатываемой ветки на другую.
9. Git merge – объединение веток.
10. Git push – отправление проекта в удаленный репозиторий.
11. Git pull – загрузка проекта из удаленного репозитория.

Тема 3. Практика. Определение вида треугольника

Необходимо написать программу, которая по сторонам треугольника будет выводить, может ли такой треугольник существовать и какой у него вид. Код программы представлен на рисунке 37.

In [16]:

```
a = int(input())
b = int(input())
c = int(input())

if a <= 0 or b <= 0 or c <= 0:
    print("Треугольник с такими сторонами не существует")

else:
    if a == b or a == c or b == c:
        if a == b == c:
            print("Равносторонний")
        elif a == c:
            if 0 < ((a^2) + (c^2) - (b^2)) / (2*a*c) < 1:
                print("Равнобедренный остроугольный")
            elif -1 < ((a^2) + (c^2) - (b^2)) / (2*a*c) < 0:
                print("Равнобедренный тупоугольный")
            elif ((a^2) + (c^2) - (b^2)) / (2*a*c) == 0:
                print("Прямоугольный равнобедренный")
        elif a == b:
            if 0 < ((a^2) + (b^2) - (c^2)) / (2*a*b) < 1:
                print("Равнобедренный остроугольный")
            elif -1 < ((a^2) + (b^2) - (c^2)) / (2*a*b) < 0:
                print("Равнобедренный тупоугольный")
            elif ((a^2) + (b^2) - (c^2)) / (2*a*b) == 0:
                print("Прямоугольный равнобедренный")
        elif c == b:
            if 0 < ((c^2) + (b^2) - (a^2)) / (2*c*b) < 1:
                print("Равнобедренный остроугольный")
            elif -1 < ((c^2) + (b^2) - (a^2)) / (2*c*b) < 0:
                print("Равнобедренный тупоугольный")
            elif ((c^2) + (b^2) - (a^2)) / (2*c*b) == 0:
                print("Прямоугольный равнобедренный")

    else:
        if ((a^2) == (b^2) + (c^2)) or ((b^2) == (a^2) + (c^2)) or ((c^2) == (b^2) + (a^2)):
            print("Разносторонний прямоугольный")
        elif (-1 < (a^2) + (b^2) - (c^2) / (2*a*b) < 0) or (-1 < ((a^2) + (c^2) - (b^2)) / (2*a*c) < 0) or (-1 < ((c^2) + (b^2) - (a^2)) / (2*c*b) < 0)
            print("Разносторонний тупоугольный")
        elif (abs((a^2) + (b^2) - (c^2) / (2*a*b)) > 1) or (abs(((a^2) + (c^2) - (b^2)) / (2*a*c)) > 1) or (abs(((c^2) + (b^2) - (a^2)) / (2*c*b)) > 1)
            print("Треугольник с такими сторонами не существует")
        else:
            print("Разносторонний остроугольный")
```

```
5
1
1
Равнобедренный тупоугольный
```

Рис. 37. Код программы.

Тема 4. Библиотеки. Numpy & Pandas

4.1 Библиотеки

Библиотека - сборник подпрограмм или объектов, используемых для разработки программного обеспечения. Мы можем подключить к проекту либо уже созданную библиотеку для облегчения работы с кодом, либо же написать свою библиотеку.

Чтобы создать библиотеку надо создать файл с расширением “ру” и перенести туда программу. Далее, когда вы захотите ей воспользоваться, введите «import (название)».

4.2 Библиотека Numpy

Numpy – библиотека для работы с векторами, матрицами.

Для будущей работу с Numpy импортируем его и переименуем его в np. С помощью метода np.array() создадим вектор передав ему некоторый массив (рис 37).

```
In [55]: np_arr = [ 1, 2, 3, 4]
         np_arr * 2

Out[55]: [1, 2, 3, 4, 1, 2, 3, 4]

In [54]: import numpy as np
         value_list = np.array([1, 2, 3, 4])
         value_list * 2 # value_list + value_list

Out[54]: array([2, 4, 6, 8])
```

Рис. 37. Вектор и массив в сравнении.

Над векторами можно проводить почти те же операции, что и над массивами, но результат будет другим.

1. Умножение вектора (рис. 38).

```
In [56]: np_arr = np.array([1, 2, 3, 4])
         np_arr * 2

Out[56]: array([2, 4, 6, 8])
```

Рис. 38. Умножение.

2. Деление вектора (рис. 39).

```
In [57]: np_arr = np.array([1, 2, 3, 4])
         np_arr / 2

Out[57]: array([0.5, 1. , 1.5, 2. ])
```

Рис. 39. Деление.

3. Сложение векторов (рис. 40).

```
In [58]: np_arr = np.array([1, 2, 3, 4])
         np_arr + np_arr

Out[58]: array([2, 4, 6, 8])
```

Рис. 40. Сложение.

4. Разность векторов (рис. 41).

```
In [59]: np_arr = np.array([1, 2, 3, 4])
         np_arr - np_arr

Out[59]: array([0, 0, 0, 0])
```

Рис. 41. Разность.

5. Ср. арифметическое вектора (рис. 42).

```
In [60]: np_arr = np.array([1, 2, 3, 4])
         np_arr.mean()

Out[60]: 2.5
```

Рис. 42. Нахождение среднего арифметического.

6. Максимум / минимум вектора (рис. 43).

```
In [61]: np_arr = np.array([1, 2, 3, 4])
         np_arr.max()

Out[61]: 4
```

Рис. 43. Нахождение максимума / минимума.

4.3 Библиотека Pandas

Библиотека Pandas позволяет очень удобно работать с данными, загружать их из разных файлов, в том числе скачивать их из интернета.

Чтобы имплантировать библиотеку, нужно написать “import pandas”. Для более удобного пользования, переименуем данную библиотеку в “pd”.

Чтобы импортировать таблицу данные чаще всего используется тип csv(), в скобочках которого указывается ссылка на тот материал, который Вы хотите указать.

Если же нужно импортировать не все строки, а к примеру, несколько первых, используется df.head(), в скобочках указывается количество строк, которые нужно вывести.

Действия, которые можно производить над таблицу, которая выводится с помощью библиотеки Pandas, совпадают с действиями над массивами (см. страницы7-9).

Заключение

В ходе выполнения данной работы, мной были получены достаточно необходимые знания о синтаксисе и принципе работы языка программирования Python. На практике были закреплены многие теоретические данные, а также реализованы некоторые программы.

Помимо этого, я был ознакомлен с системой контроля версий Git’ом и сайтом для создания удаленных репозиториев GitHub’ом и слиянием удалённого репозитория и локального репозитория воедино.

Список литературы

1) Лекция Введение в Python https://youtu.be/l4u4fhnLqfM
2) Лекция Git & GitHub https://youtu.be/mRbs8uUZ4X0
3) Лекция Практикум номер 2 https://youtu.be/-tYQN8sYM4A
4) Лекция Быстродействие, метод Горнера, создание библиотек https://youtu.be/_-fT4JMWpo8
5) Лекция Numpy & Pandas https://youtu.be/om5G1Ifz7-E