|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 | Акимкина Алина Дмитриевна |
|  | *фамилия, имя, отчество* |

|  |  |  |
| --- | --- | --- |
| Группа | ИУ1-12Б |  |

|  |  |
| --- | --- |
| Тип практики | Учебно-технологический практикум |

|  |  |
| --- | --- |
| Название предприятия | Кафедра «Системы автоматического управления» |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  | 24/12/2021 |  | А.Д. Акимкина |
|  |  | (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. |

**ЗАДАНИЕ**

**на прохождение учебной практики**

|  |  |  |
| --- | --- | --- |
| Студент группы | ИУ1-12Б |  |
| Акимкина Алина Дмитриевна | | |
| (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 | 24 | 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. Акимкина |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

*Содержание*

# Начало работы в GitHub

## Установка

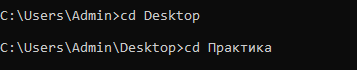
После установки GitHub на устройстве появится GitBash— это приложение для сред Microsoft Windows, эмулирующее работу командной строки. Открыв его и введя команду “git --version”, можно посмотреть установленную версию GitHub. Также это позволяет убедиться, что установка прошла успешно.

GitHub помогает контролировать состояние проекта, делать ветвления версий проекта и ставить точки, к которым мы можем вернуться после экспериментов.

Для того, чтобы начать контролировать, необходимо:

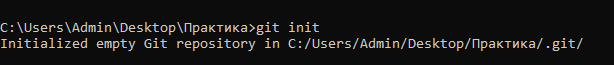
1. Перейти в директорию, где хранится проект.

Для этого используется команда “cd”. Применение “cd” совместно с двумя точками, то есть “cd ..”, позволяет перейти в более высокую директорию. Для открытия необходимой папки используется “cd” + название папки. Пример представлен на рисунке 1.



1. Инициализировать git, как систему контроля версий.

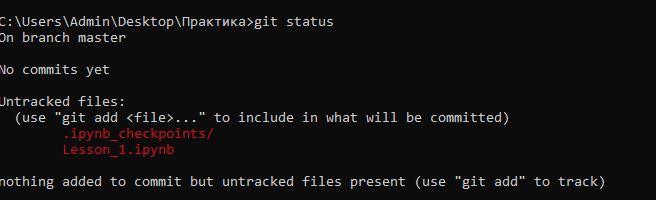
Для этого используется команда “git init”. После выполнения команды git сообщит о том, что директория инициализирована. Как это выглядит в командной строке, иллюстрирует рисунок 2.



В результате будет добавлена скрытая папка, в которой хранится множество других папок и файлов, необходимых для контроля проекта.

## Начало работы в git

На рисунке 3 представлен результат выполнения команды “git status”. “On branch master” обозначает, что пользователь находится в ветке «мастер», где происходит разработка текущего проекта. “No commits yet” говорит о том, что комитов ещё не было. “Untracked files” сообщает, что два файла “.ipynb\_checkpoints/” и “ Lesson\_1.ipynb” не отслеживаются.



Команды и методы для работы с файлами и директориями:

* “git add + имя файла” – добавление файла в систему контроля версий;
* “git add .” – добавление всех файлов в систему контроля версий;
* “git rm --cache + имя файла” – команда, после выполнения которой файл перестает отслеживаться;
* Если файлов много, но есть те, которые не желательно добавлять в систему контроля версий, то надо создать текстовый файл, поменять в нем разрешение на “.gitignore” и указать в нем названия файлов и директорий, которые git будет игнорировать.
* “git commit -m “название” ” – сохранение проекта в текущем состоянии;
* “git restore + имя файла” – возвращение к предыдущему сохраненному состоянию проекта;
* “git log” – показывает всю информацию о ранее сделанных сохранениях;
* “git log --oneline” – показывает только названия ранее созданных сохранений;
* “git commit –a –m “сообщение”” – сохранение изменений в уже отслеживаемых файлах без добавления новых;
* “git revert + hash commit” – возвращение к определенной версии проекта, удаление всех действий после определенного сохранения и фиксация нового состояния. Сохранение нового состояние происходит в редакторе vim. При этом будет предложено название сохранения(комита). Чтобы выйти из редактора необходимо напечатать двоеточие и после появления командной строки внизу написать “q!”.
* “git reset” – отмена последнего действия;
* “git reset --hard” – отмена последних изменение и возвращение к последнему сохранению;
* “git reset --hard HEAD~1” – возвращение на одно cохранение назад;
* “git reset --mixed HEAD~1” – отмена последнего commit без сохранения индексации файла, что позволяет доработать последние изменения;
* “git reset --soft HEAD~1” – отмена последнего сохранения с удержанием индексации файла;
* “git commit --amend” – добавление последнего изменения к последнему сохранению;
* “git tag –a + название версии –m “название тэга” ” – команда объявления тэга (метки). Тэг помечает коммит и указывает на его определенную версию. Чтобы создать тэг, для начала надо переключиться на необходимое сохранение командой “git checkout + ключ коммита” и только затем прописать указанную ранее команду. Далее по команде “git checkout +имя версии” можно переключаться на ту или иную версию.
* “git tag” – отображение всех тэгов, которые есть;
* “git push --tag” – добавление всех тэгов на удаленный репозиторий;

Если в проекте создать папку, то git не будет её видеть, однако же, если в папке будет находится файл, то по команде “git status”, git отобразит её в неконтролируемых. В зависимости от задачи папку можно будет добавить в систему контроля версий.

В git есть возможность реализовывать параллельно разные идеи проекта. Для этого используются ветки. То есть на каждой ветке будут осуществляться разные решения задачи.

* “git branch” – показывает все ветки, которые имеются в проекте;
* “git branch + название новой ветки” – создание новой ветки;
* “git checkout + название необходимой ветки” – переключение на определенную ветку;
* “git switch + название определенной ветки” – переключение на определенную ветку;
* “git branch –D + название необходимой ветки” – удаление определенной ветки;
* “git merge + название необходимой ветки” – команда, которая позволяет объединить определенную ветку с той, в которой находишься сейчас;
* “git branch –m + новое название ветки” – команда, которая позволяет переименовать текущую ветку;

## GitHub

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git. С его помощью можно удаленно хранить версию проекта, следить за ней и редактировать с разных устройств.

Для того, чтобы объединить удаленный репозиторий и локальный, в первую очередь необходимо зарегистрироваться на сервере GitHub и создать репозиторий, где будет хранится проект. При создании указывается название, тип репозитория (публичный или личный), а также можно добавить файл .getignore, где будут находиться файлы, которые не будут сохраняться в систему контроля версий git, и файл readme для описания. После создания репозитория высветится подсказка, где будут указаны желательные дальнейшие действия.

По команде “git remote add origin + ссылка на репозиторий” можно подключить локальный репозиторий к созданному на GitHub. Далее благодаря команде “git push -u origin + название ветки” можно загрузить в него все файлы, связанные с проектом.

Чтобы продолжить работу с файлами на другом компьютере в командную строку через команду “git clone” надо ввести ссылку, которую можно скопировать из вкладки “code”. В результате на устройстве будет создана папка с именем репозитория, содержащая все файлы.

Текстовый файл “README.md” создается для описания файлов в проекте.

Команда “git push” добавляет все изменения на удаленный репозиторий, “git pull” – добавляет изменения из удаленного репозитория на локальный.

Более того, GitHub содержит все сохранения, которые были когда-либо сделаны. Чтобы вернуться к определенному состоянию проекта достаточно ввести команду “git checkout” и вставить ключ необходимого этапа, скопированный с GitHub.

Перейдя во вкладку “Code” в отделе “tag” можно создать релиз проекта. Для этого надо перейти в отдел “Releases”, нажать кнопку “Create a new release” и выбрать тэг; по необходимости можно добавить название и описание версии. Создание релизов помогает поддерживать версионность кода.

## Jupyter Notebook

Самый простой способ начать работу в Jupyter Notebook – это установить Anaconda Navigator и в главной вкладке приложения, на иконке Jupiter Notebook, выбрать “lauch”. После совершенных действий Jupiter Notebook запуститься в браузере, который выбран основным. В появившемся окне будет показан директорий компьютера. Именно там и создаются файлы для разработки проекта.

Чтобы создать новые файл, надо нажать “New” -> “Python 3” (рис х). Поменять имя файла можно в левом верхнем углу, нажав на edit box.

Интерфейс Jupiter Notebook состоит из ячеек, где пишется код, и верхней панели управления.

Быстрые команды для работы в Jupiter Notebook:

* “a” – создание ячейки сверху;
* “b” – создание ячейки снизу;
* “m” – изменение типа ячейки с “code” на “markdown”, что позволяет использовать ячейку для написания текста, описания;
* “dd” – удаление ячейки;
* Shift + Enter – запуск кода и переход в следующую ячейку;
* Ctrl + Enter – запуск кода без перехода в следующую ячейку;
* Shift + Tab – просмотр документации метода;
* Применение # в ячейке с типом markdown создает заголовок. От количества # зависит порядок заголовка.
* Запись комментариев в ячейке с типом code осуществляется после знака #.
* Формулы пишутся между знаками $$.
* <br> в конце строки в ячейке с типом markdown – переход на новую строку.

## Python

### Типы данных:

1. Целочисленный тип – int.

Это целые числа. Например, 1, 5, 2354 и так далее.

Операции, которые можно выполнять с целочисленным типом:

* Сложение “ + ”;
* Вычитание “ - “;
* Умножение “ \* ”;
* Деление “ / ”;
* Целочисленное деление “ // “ – возвращает целое число от деления;
* Поиск остатка от деления “ % “ – возвращает остаток от деления;
* Возведение в степень – “ \*\* “. Пример применения: 2\*\*3 –> 8. Так же можно взять квадратный корень из числа, возведя его в степень 1/2 : 4\*\*(1/2) = 2.
* Сравнения:

“ == “ – проверка равенства;

“ >= “ – больше или равно;

“ <= “ – меньше или равно;

“ > “ – больше;

“ < “ – меньше.

Все примеры представлены на рисунке 4.

1. Вещественный тип – float.

Это все остальные числа, кроме целых. Например, 1.2, 3.44, 564.21 и так далее.

С вещественными числа можно выполнять все те же операции, что и с целыми числами.

1. Логический тип – bool.

Он может принимать лишь 2 значения – либо True, либо False.

Данный тип применяется в логических выражениях и выражениях отношения. Его использование удобно в циклах и условных операторах. Пример взаимодействия с логическим типом данных представлен на рисунке 5.

Чтобы получить True после выполнения оператора “and”, надо чтобы оба условия были истинными. Чтобы получить True после выполнения оператора “or”, необходимо чтобы хотя бы одно условие было истинным.

1. Строковый тип данных – string.

Операции, которые можно выполнять со строками:

* Сложение (рис 6).

### Методы работы с данными:

* print() – функция вывода необходимых данных;
* type() – проверка типа данных;
* round() – округление числа;
* str() – приведение некоторого объекта к строковому типу данных;
* int() – приведение некоторого объекта к целочисленному типу данных, выделяет целую часть числа, а также переводит из какой либо системы счисления в десятичную;
* float() – приведение некоторого объекта к вещественному типу данных;
* input() – ввод данных с консоли. Данные будут записаны стоковым типом.
* rjust(), ljust() – методы, применяемые к строкам, при которых будет добавлено такое количество пробелов слева и справа соответственно, что в сумме строка будет занимать количество символов, указанных в скобках.

Результаты работы методов приведены на рисунках 7-9.

### Условный оператор if:

Шаблон построения условного оператора:

if условие:

код, который выполняется, если результатом выполнения условия будет истина

else:

код, который выполняется в противном случае

Условие – это выражение, которое вернет либо True, либо False. Пример написания и работы условного оператора представлен на рисунке 9.

### Переменные:

Переменная — это определенная ячейка памяти, в которую можно записать различные значения. Все типы данных, которые перечислены выше, могут быть присвоены переменным.

Значения могут быть присвоены переменным с помощью оператора присваивания “ = “ или же введены с консоли с помощью метода input() (рис. 10).

### Массив

- это структура данных, состоящая из набора значений, которые принимают однотипные данные и к которым можно обращаться по индексу.

В Python массив инициализирует квадратными скобками “[ ]” или присваиванием типа массива; тип массива – list (рис. Ннный).

Индексация массива начинается с нуля, а с конца – с “-1”, то есть: (рис нный)

Можно обращаться к элементам по индексу, указывая индекс в квадратных скобках (рис. ннный).

Также можно создавать двумерный массив. Двумерный массив - это одномерный массив, элементами которого являются одномерные массивы. Другими словами, это набор однотипных данных, также являющихся массивами. Для поиска определенного элемента указывается два индекса (рис ннный).

Операции, которые можно выполнять с массивами:

* Сложение, например:
* Умножение, например:

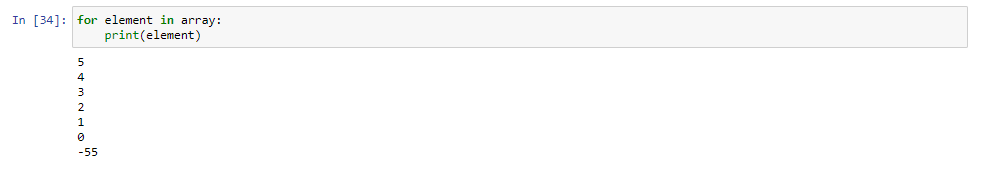
В результате вернется исходный массив, продублированный то количество раз, на которое он был умножен.

#### Методы для работы с массивами:

* len() – передает количество элементов в массиве;
* append() – записывает в массив элементы, находящиеся в скобках;
* pop() – удаляет и возвращает элемент под определенным индексом. По умолчанию метод удаляет последний элемент, но, если в скобках указать индекс определенного элемента, то будет удален именно он.
* clear() – очищает весь массив;
* copy() – делает копию массива;
* count() – возвращает количество, сколько раз встречается элемент. Причем искомый элемент указывается в скобках.
* extend() – добавляет значения к массиву; в скобках прописываются элементы, которые надо добавить;
* index() – возвращает индекс указанного элемента; причем индекс первого вхождения элемента;
* insert() – вставляет элемент в определенное место массива. Через запятую в скобках сначала указывает до какого индекса надо вставить элемент, потом какой именно элемент должен быть добавлен.
* remove() – удаляет первый встретившийся в массиве элемент, указанный в скобках;
* reverse() – “переворачивает” массив;
* sort() – сортирует массив. По умолчанию метод сортирует массив по возрастанию, но если в скобках указать “reverse = True”, то массив будет сортироваться по убыванию.

### Цикл *for*:

Пример работы цикла *for*:



В программе на рисунке 1 цикл поочередно выводит элементы массива.

Метод range() задает промежуток, на котором будет выполняться цикл. При этом крайняя граница цикла не включается. Также методу можно передать шаг, с которым будут выводиться значения.

### Срезы:

- это часть массива. Пример представлен на рисунке ….

Причем конец указанного среза не выводится. Более того, в срезе можно указывать шаг. Если не указывать начало или конец среза, то по умолчанию будет выводится срез с начала массива по указанный индекс или с указанного индекса до конца соответственно.

### Функции:

Если в течение всего кода есть фрагмент, который повторяется, то его стоит определить в отдельную функцию. Это сократит количество строк и будет удобнее к применению. В Питоне создание функции определяется командой def, далее прописываются название функции и ее параметры. Для возвращения результата используется ключевое слово return. Пример создания функции представлен на рисунке х.

### Создание библиотек:

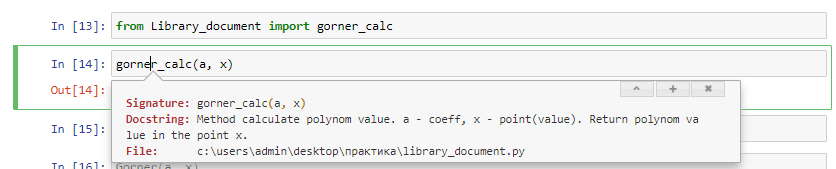
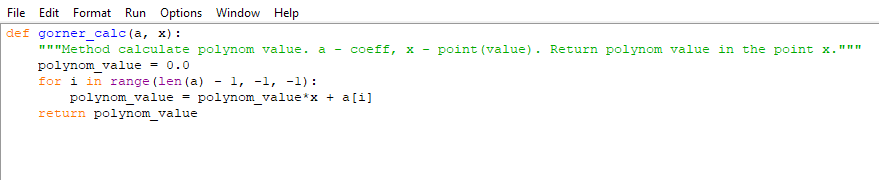
Для создания библиотеки надо создать текстовый файл в папке, где находятся все работы проекта, и изменить расширение файла на “.py”. Далее в этом файле прописываются полезные функции, которые планируются применяться в дальнейшем.

Чтобы импортировать определенную функцию, для начала надо в файле, где вы работаете, подключить библиотеку методом import. В дальнейшем коде обращение к функции будет осуществляться по следующему принципу: “название библиотеки”.“название функции”. Другой способ осуществления импортирования и дальнейшего применения функции выглядит следующим образом:

1. Мы импортируем функцию строкой «from “название библиотеки” import “название функции”».
2. Обращаем к функции как к обычному методу.

Оба примера подключения библиотеки и функций, хранящихся в ней, представлены на рисунке.  Более того, во время импортирования функции её можно переименовать. Делается это через оператор as: 

Написание документаций:

Документации прописываются в тройных кавычках в файле, содержащем функции, после объявления функции. Пример написания и пример отображения документации представлен на рисунках пук-пук. 

## Библиотека Numpy

- это библиотека, которая позволяет удобно работать с векторами, матрицами, делать различные алгебраические операции.

Для подключения к библиотеке numpy используется функция “import numpy”. Если же библиотека не установлена, то для её установки надо прописать “pip install numpy” в командной строке.

В библиотеке есть много различных методов.

Методы библиотеки:

* exp() –вычисляет экспоненту от заданного числа;
* array() – создание вектора. Преимущество в том, что с вектором мы можем выполнять операции, которые недоступны при обычном массиве типа list. Например, если умножить массив на какое-то число, то он продублируется это количество раз, если же вектор умножить на это число, то все элементы, хранящиеся в нем, будут умножены на это число.

Операции, возможные для работы с вектором:

* Умножение
* Деление
* Сложение
* Вычитание
* Нахождение средне арифметического. Для этого применяется метод mean().
* Нахождение стандартного отклонения. Применяется метод std().
* Определение максимума и минимума. Для этого применяются методы

## Метод Горнера

- это алгоритм вычисления значения многочлена, записанного в виде суммы одночленов. Например, задан многочлен :

,

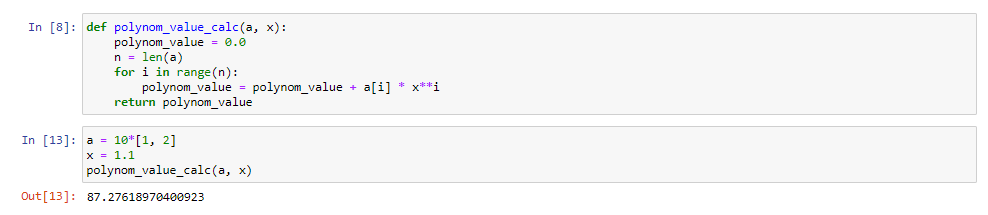
Пусть требуется вычислить значение данного многочлена при фиксированном значении . То по методу Горнера надо представить многочлен в следующем виде:

И только потом производить дальнейшие вычисления.

Как выглядит реализация решения многочлена первого уравнения на Python представлено на рисунке у.

Алгоритм решения:

1. Мы создаем функцию, которая будет производить счет. В нашем случае эта функция называется polynom\_value\_calc. Она зависит от двух переменных: от массива, который содержит в себе значения всех коэффициентов (на примере это коэффициенты ), и от .
2. Результат будет записываться в переменную polynom\_value, которую мы сначала приравниваем к нулю. Далее, проходя по всем элементам массива, мы поочередно умножаем их на x в степени индекса элемента и суммируем к переменной, которая содержит результат. По итогу функция вернет нам решение многочлена для заданных переменных.
3. Затем мы задаем массив и x. И, применяя созданную ранее функцию, получаем результат.



Алгоритм решения многочлена методом Горнера на Python следующий:

1. Необходимо создать функцию, которая так же на вход получает массив и значение x – на рисунке это функция gorner\_calc. Мы вводим переменную polynom\_value, которая будет сохранять результат, но для начала мы её обнуляем. Далее в цикле for мы постепенно двигаемся от конца в начало и к каждому значению массива прибавляем результат предыдущей суммы(polynom\_value) умноженный на x. Функция возвращает polynom\_value – конечный результат всех сложений.
2. Далее необходимо задать массив и неизвестную переменную x и применить функцию gorner\_calc. Таким образом мы получим результат работы функции на заданный массив и x.



Преимущество применение метода Горнера в том, что именно им процесс вычисления происходит быстрее, причем примерно в 2 раза. Это наглядно видно на рисунке. 

Для определения времени работы программы мы использовали метод “%%time”.

### Вычисление значения экспоненты с помощью ряда Тейлора в разных точках

Возьмем экспоненту, представленную в ряде Тейлора.

ex=1 + x1! + x22! + x33! + ... =n = 0xnn!, xC

Алгоритм написания программы для вычисления экспоненты в разных точках:

1. В первую очередь стоит написать функцию, которая будет считать факториал(рис ). Данная функция принимает на вход число, затем проверяет целочисленный ли тип. Если тип целочисленный, то проверяет какой знак у числа: если число равно нулю - то возвращается единица, если число отрицательное - то ничего не возвращается, так как факториал от отрицательного числа взять невозможно, если же число положительное - то вводится цикл, в котором некая переменная, изначально равная одному, постепенно умножается на все числа от 1 до самого числа. Таким образом находится факториал.
2. Затем мы определяем количество элементов в ряду и для каждого элемента вычисляем 1/факториал от числа, применяя ранее созданную функцию. Полученные результаты добавляем в массив. Таким образом, массив будет содержать коэффициенты многочлена, который соответствует разложению экспоненты в ряд Тейлора.
3. Далее присваиваем x некоторое значение. Теперь мы можем рассчитать значение экспоненты используя метод Горнера.