**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА   
И ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**НИЖЕГОРОДСКИЙ ИНСТИТУТ УПРАВЛЕНИЯ – филиал РАНХиГС**

**Отчет по практической работе №1**

Студент: Комышков В.Д.

Иб – 311

«Прикладная информатика»

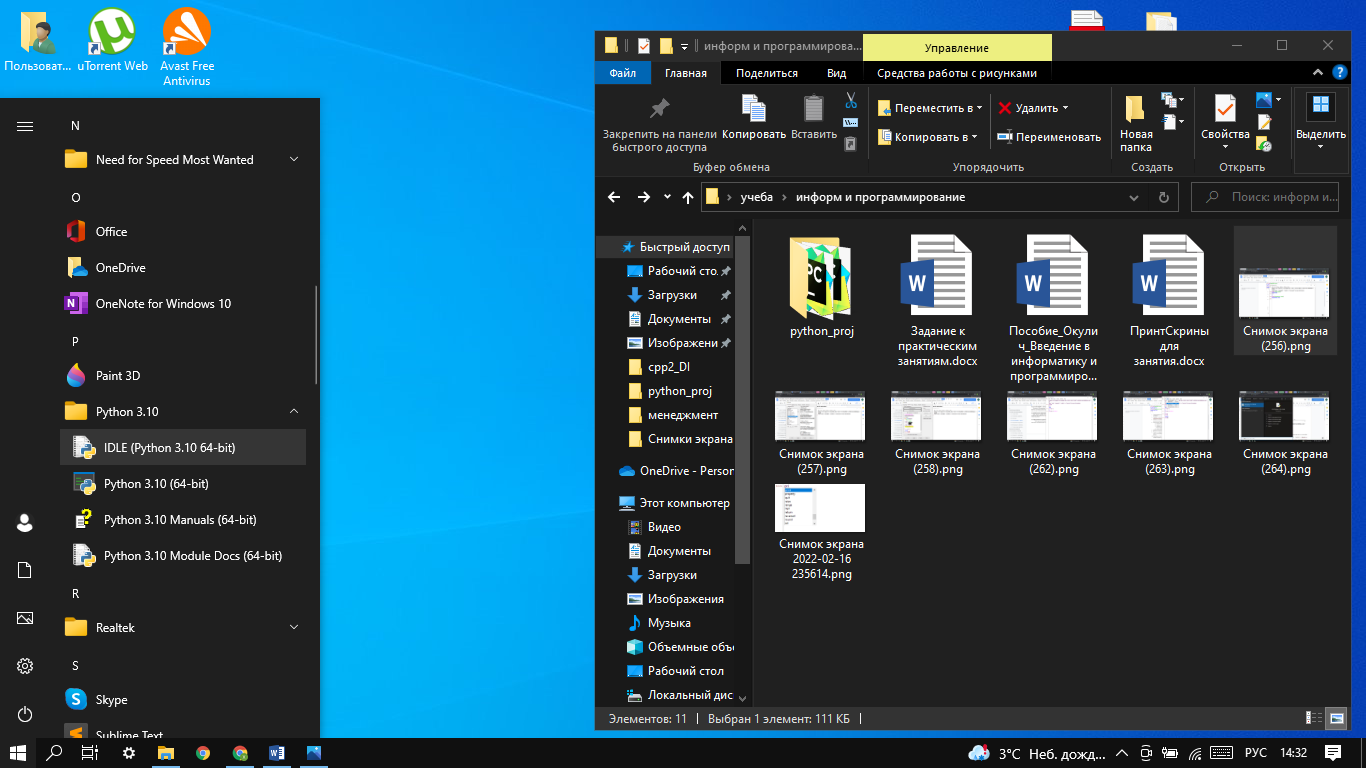
# **Знакомство с IDLE**

***IDLE (Integrated Development and Learning Environment) - это интегрированная среда для разработки (и обучения), которая поставляется вместе с Python. Сама IDLE написана на Python с использованием библиотеки Tkinter, поэтому является кроссплатформенным приложением (может работать на Windows, macOS, Linux).***

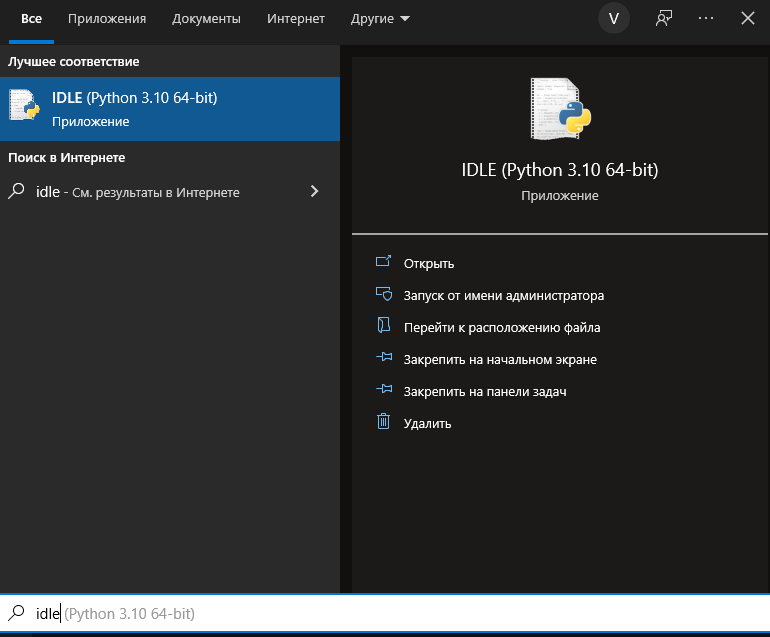
## Запуск

Запустим ***IDLE***!

Для этого найдем его в меню ***«Пуск»*** (Рис 1.1) или сочетанием клавиш *WIN+S* откроем ***поиск*** и найдем IDLE (Рис. 1.2).



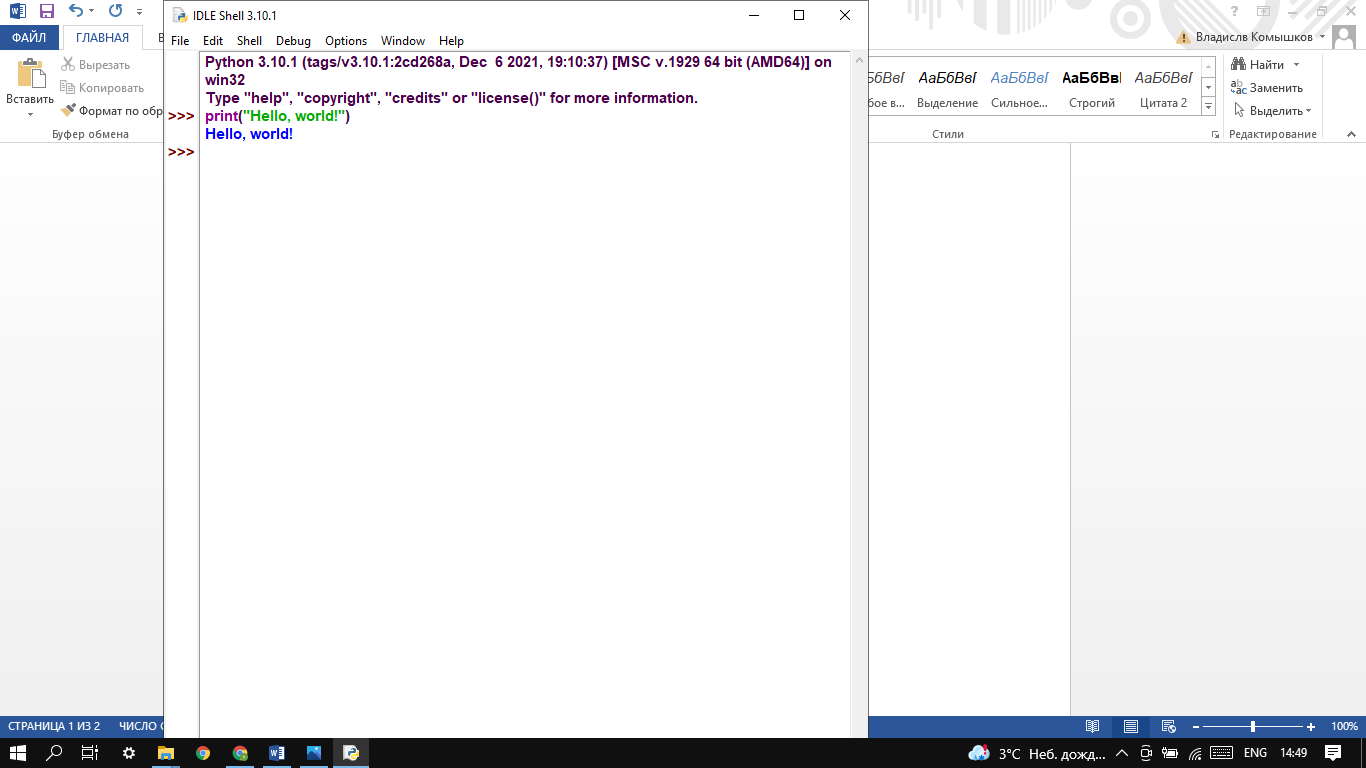
*(Рис. 1.1) Запуск IDLE в меню «Пуск»*



*(Рис. 1.2) Запуск IDLE через строку поиска*

## "Hello, world!"

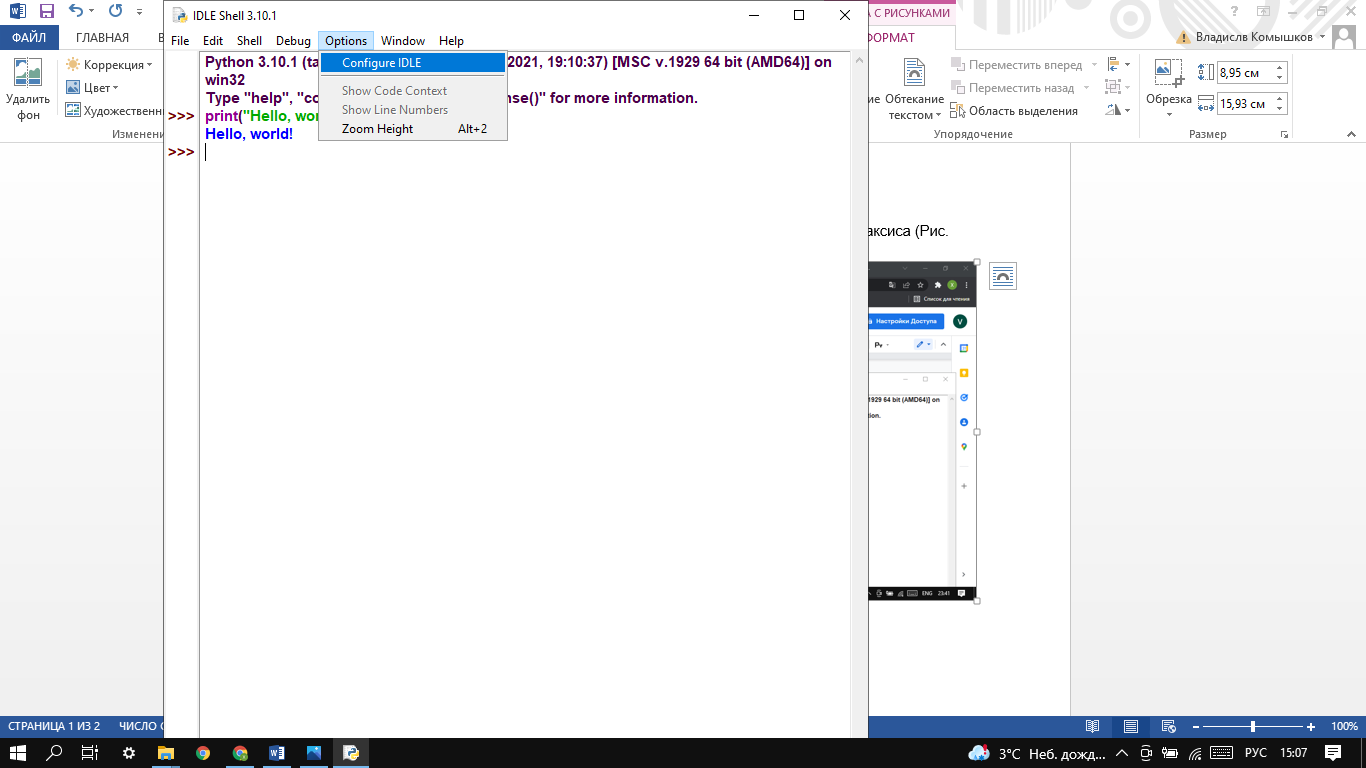
Выполним стандартный ***«Hello, world!»*** (Рис. 1.3).



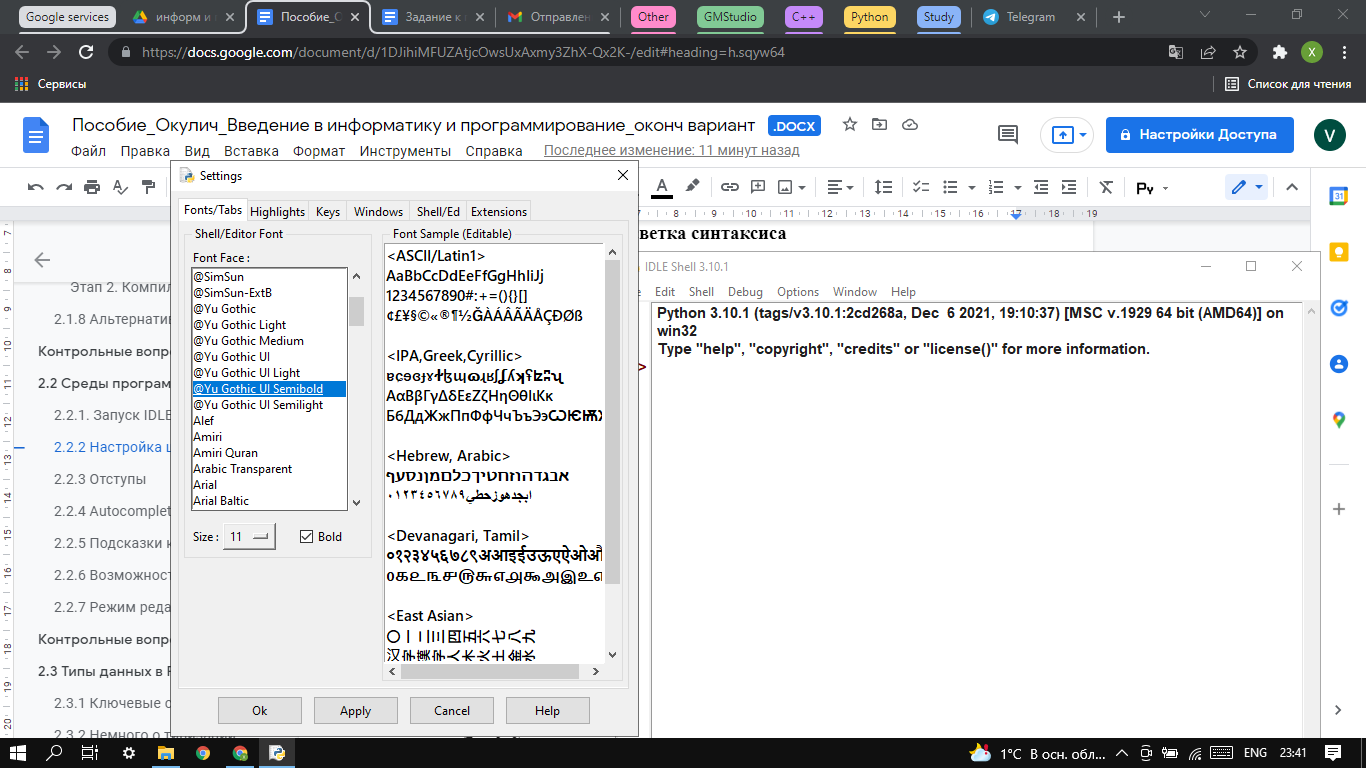
*(Рис. 1.3) «Hello, world!»*

## Настройки

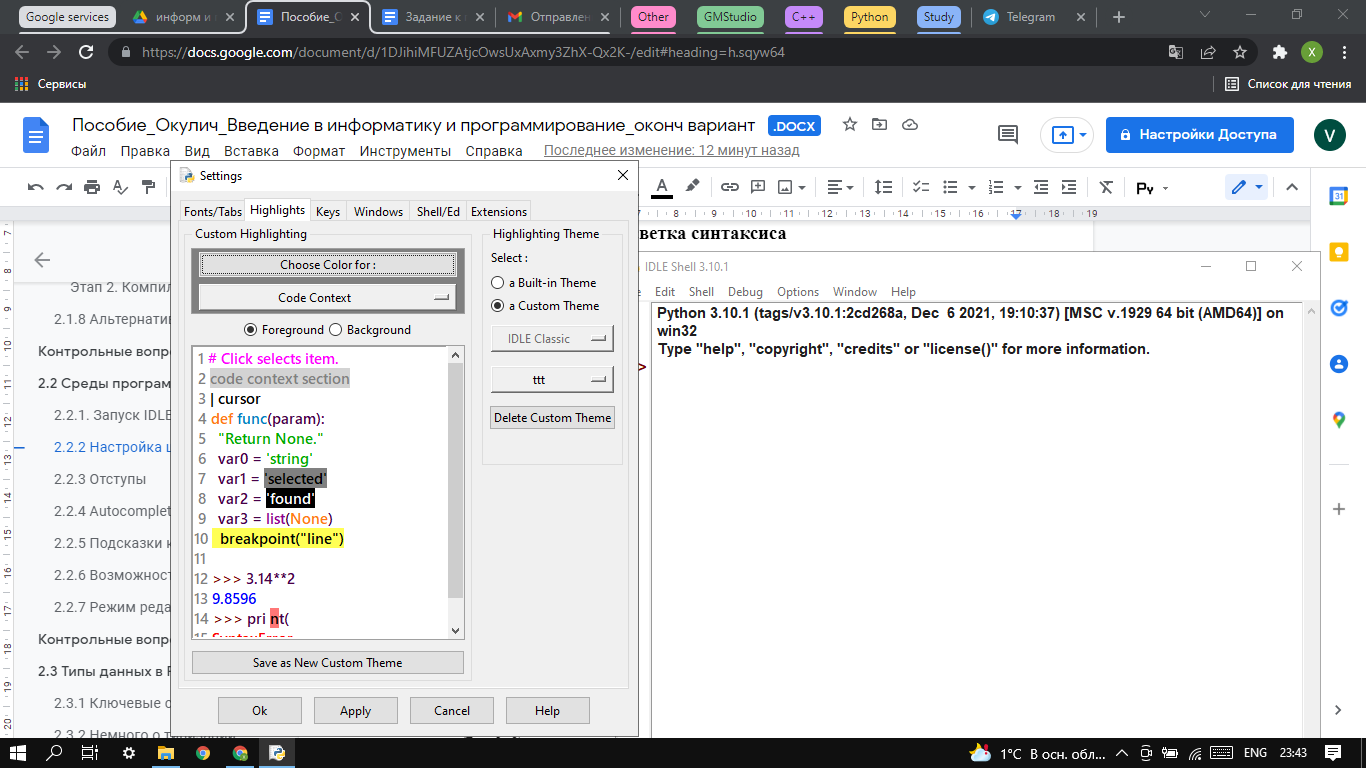
***Настройки*** IDLE (Рис. 1.4) позволяют изменять ***шрифт*** (Рис. 1.5) и ***подсветку синтаксиса*** (Рис. 1.6), что является довольно удобным для программиста.



*(Рис. 1.4) Настройки IDLE*



*(Рис 1.5) Шрифт*



*(Рис. 1.6) Подсветка синтаксиса*

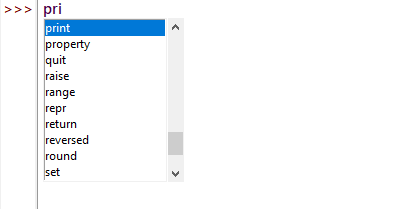
## Autocomplete, подсказки и история команд

Функция ***autocomplete –*** ***автозавершение***. Она позволяет увеличить скорость написания кода. Для ее использования достаточно написать часть слова, а затем нажать сочетание клавиш *ALT+/* (Рис. 1.7).



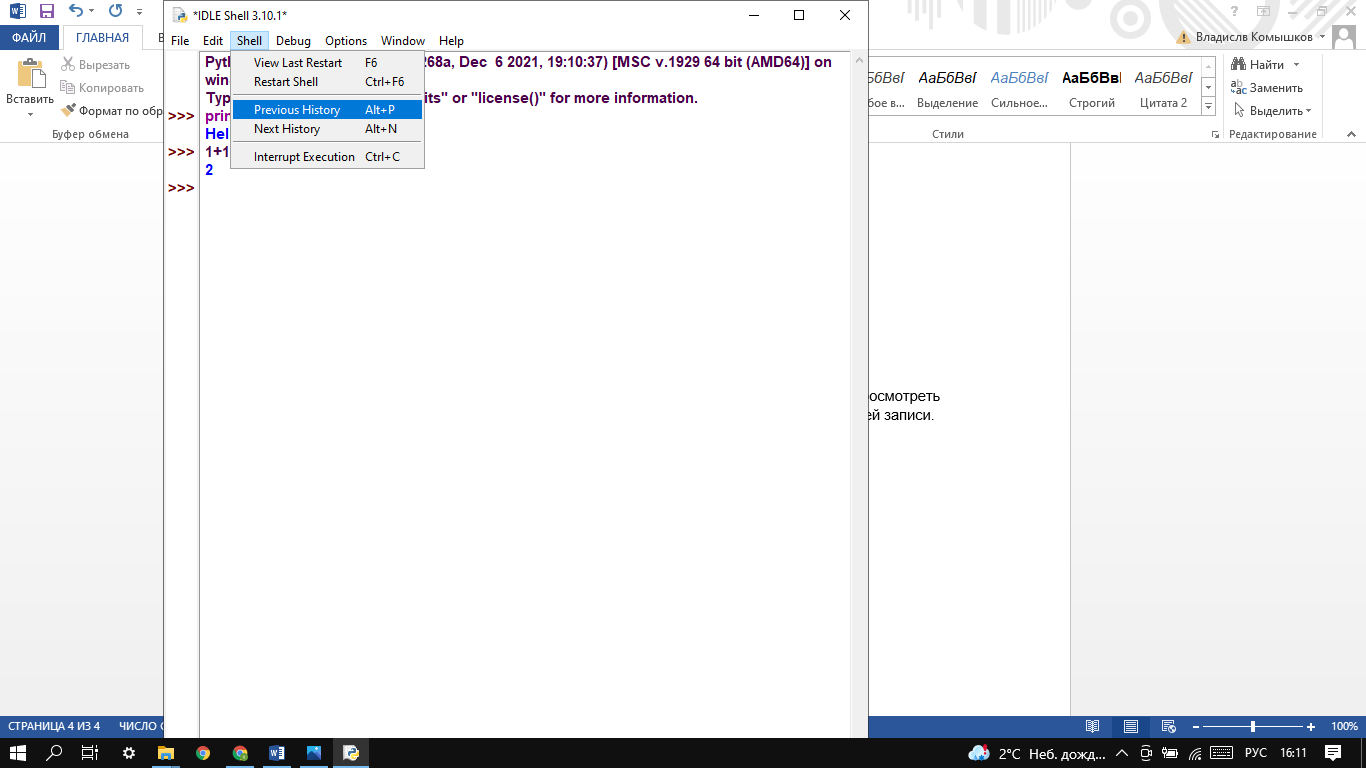
*(Рис. 1.7) Автозавершение*

Еще одной полезной функцией IDLE являются ***подсказки*** (Рис. 1.8). Нажатие клавиши *TAB* откроет небольшое окошко, где будут представлены возможные варианты функции, которую вы хотите написать.



*(Рис. 1.8) Подсказки*

Также IDLE запоминает то что мы писали ранее, и дает возможность просмотреть ранние команды и пройтись по последующим, соответствующим текущей записи. Проще говоря, просмотреть ***историю*** нашего кода (Рис. 1.9).



*(Рис. 1.9) История команд*

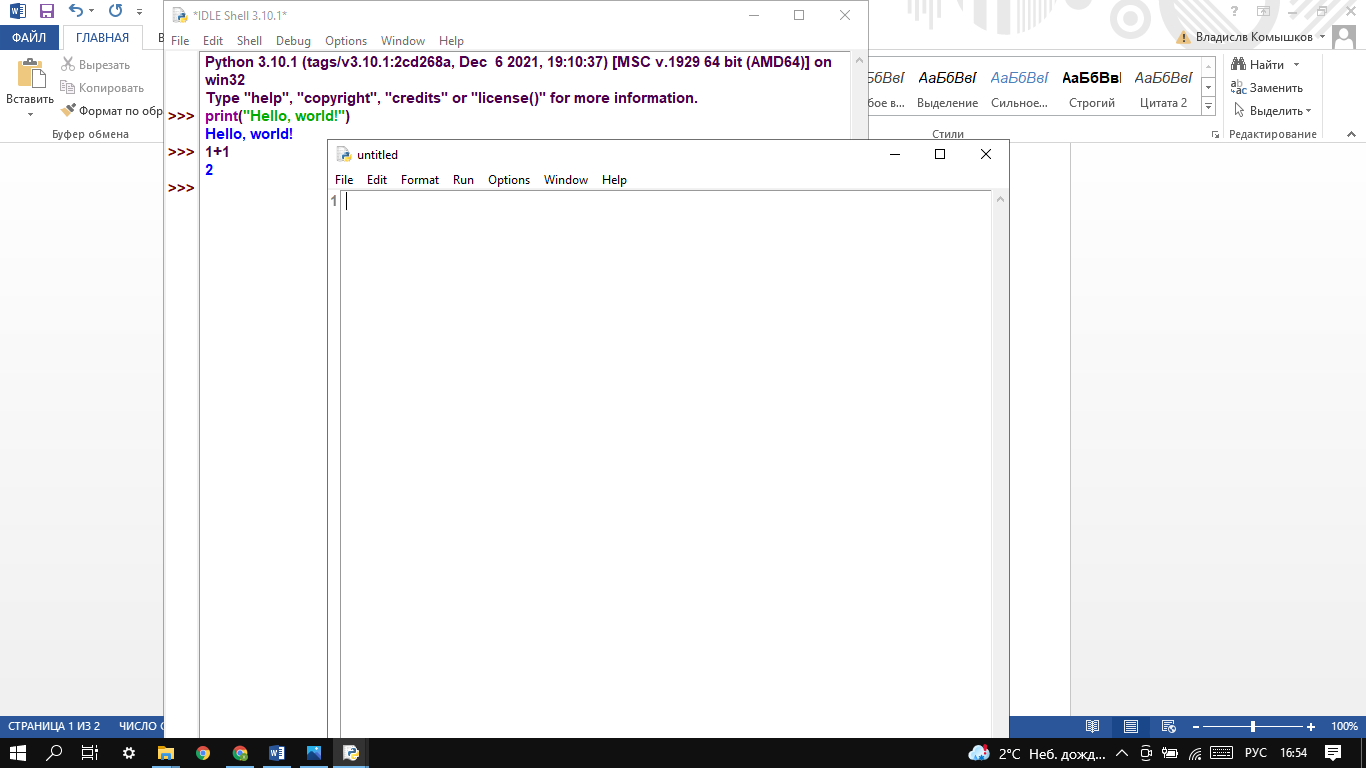
## Интерактивный режим

«IDLE обеспечивает интерактивный режим выполнения команд».

Что это значит?

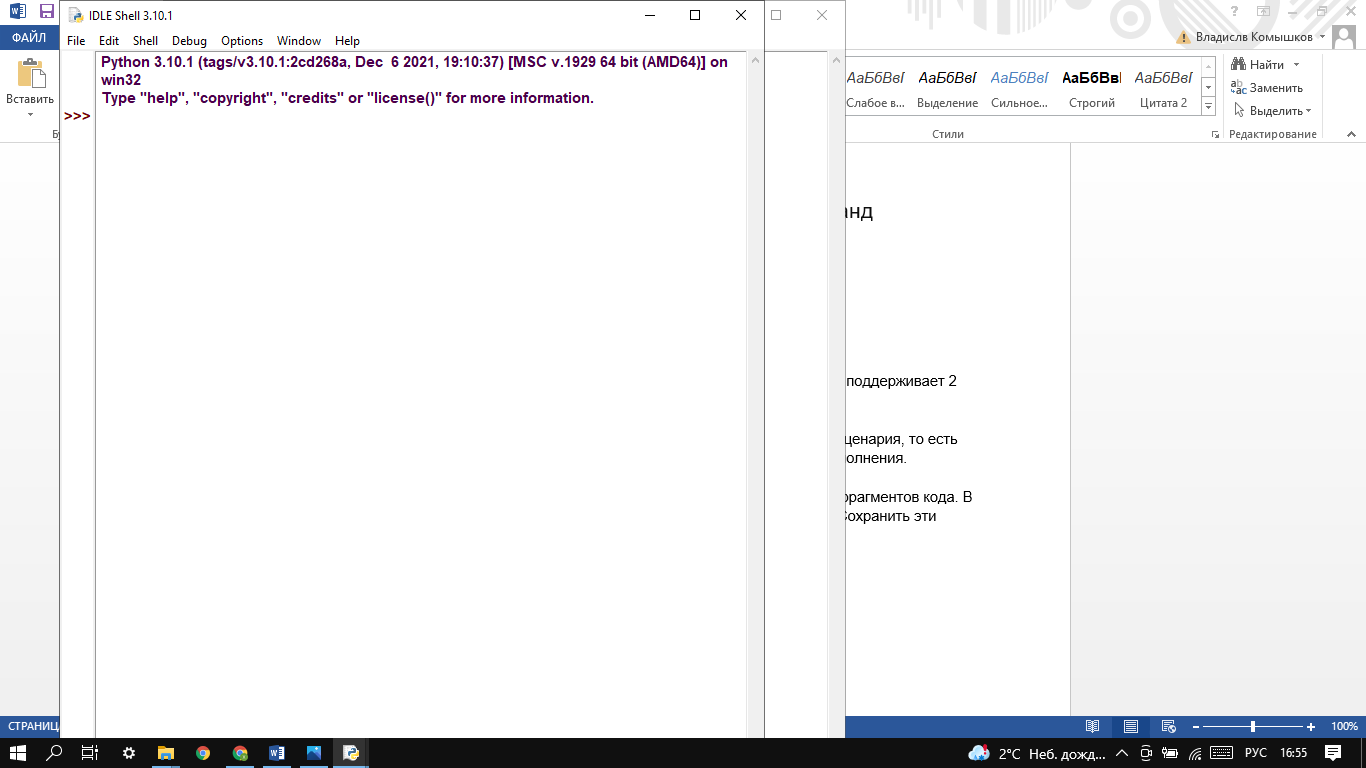
Среда разработки IDLE (Integrated Development Environment) Python поддерживает 2 режима работы: ***программный*** (Рис. 1.10) и ***интерактивный*** (Рис 1.11).

Программный режим предполагает запись скрипта или по-другому сценария, то есть полноценного кода программы. Затем его сохранения и полного выполнения.



*(Рис. 1.10) Программный режим*

Интерактивный режим предусматривает тестирование команд или фрагментов кода. В этом режиме код выполняется сразу и результат выводится сразу. Сохранить эти фрагменты нельзя.



*(Рис 1.11) Интерактивный режим*

## Программный режим

### Настроить режим

***Options >> Configure IDLE >> Windows***

Здесь нас интересует только первые три пункта (Рис. 1.6.1): ***At Startup***, ***Initial Window Size*** и ***Indent spaces***.

1. At Startup

Позволяет выбрать режим при запуске IDLE:

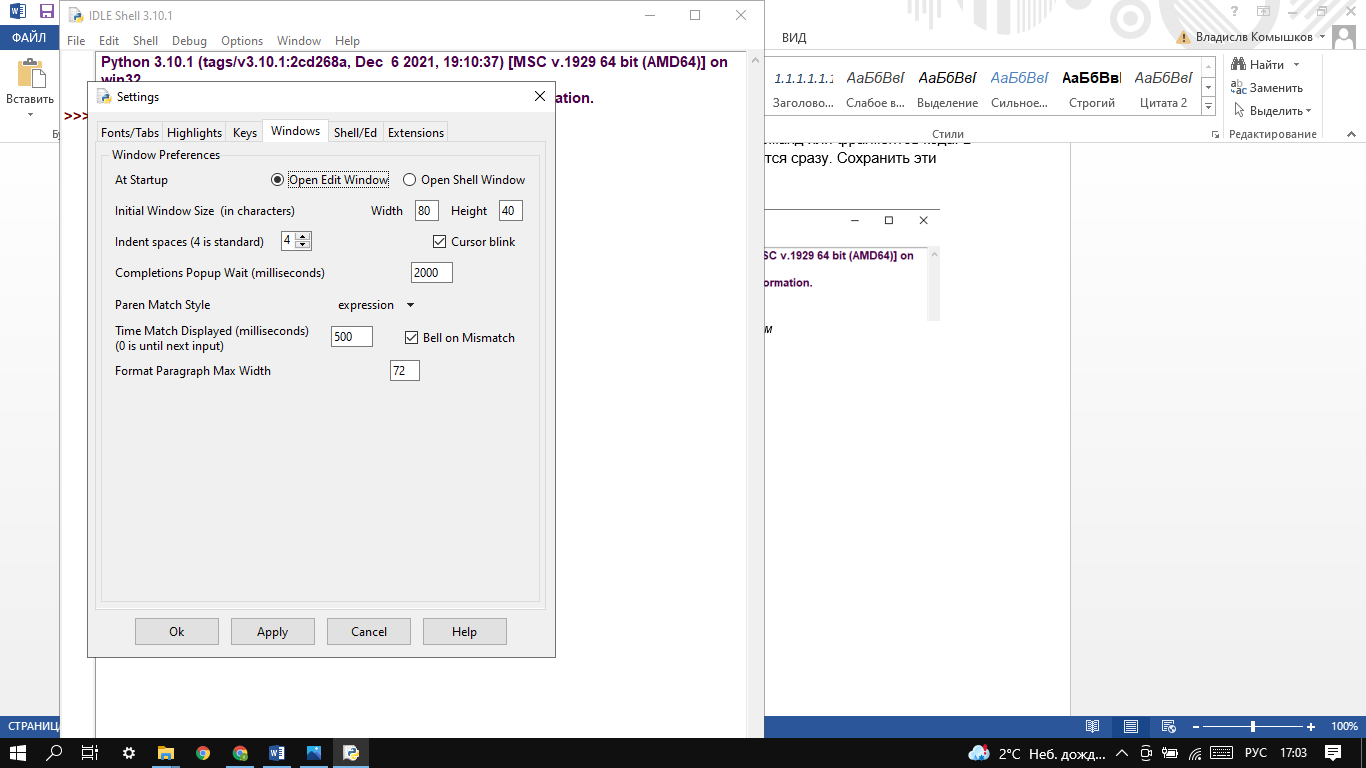
1. Open Edit Window – Интерактивный режим
2. Open Shell Window – Програмный режим
3. Initial Window Size

Размер окна (в символах):

* 1. Width - Ширина
  2. Hight – Высота

1. Indent spaces

Отступы (стандартно 4)



*(Рис. 1.6.1) Windows*

***Options >> Configure IDLE >> Shell/Ed***

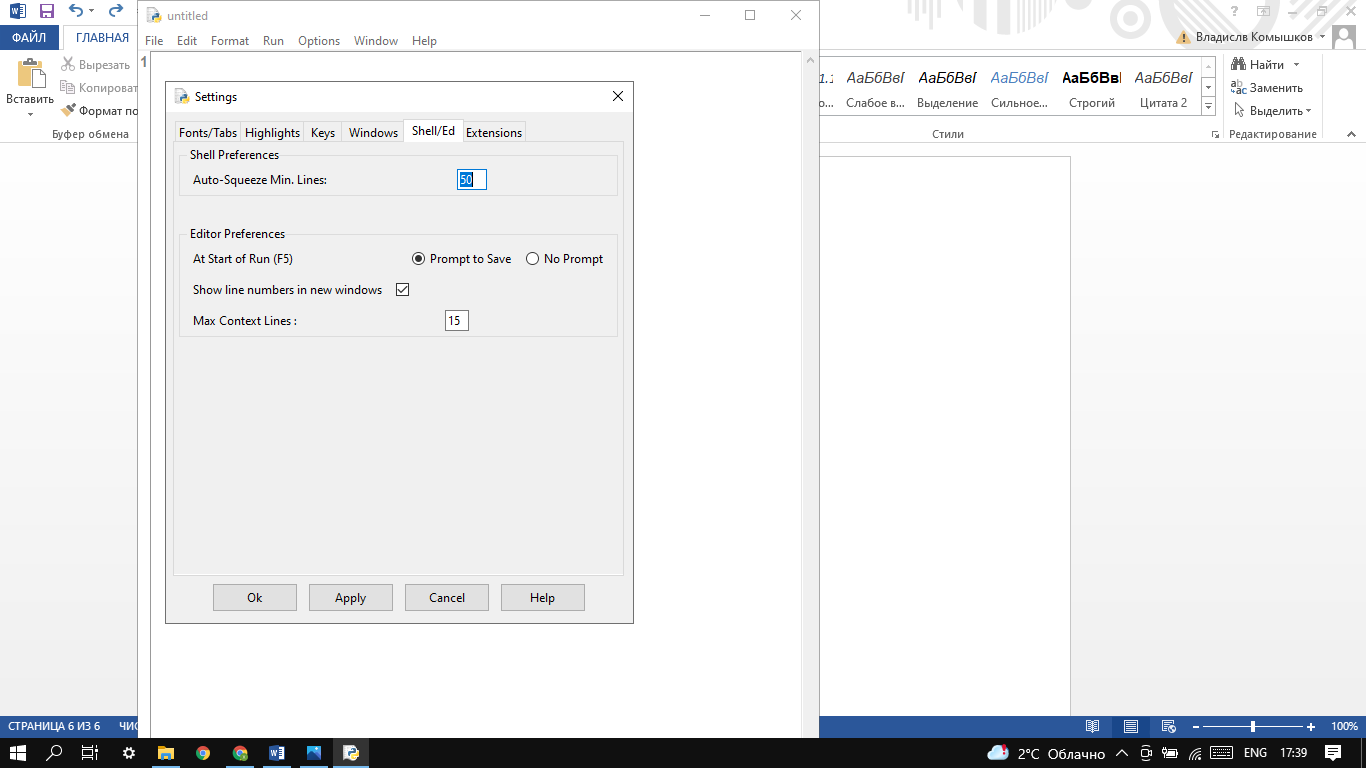
Самое важное для нас ***At Start of Run*** (Рис. 1.6.2).

At Start of Run

Напоминание о сохранение нашей программы:

1. Prompt to Save - напоминать
2. No Prompt – не напоминать

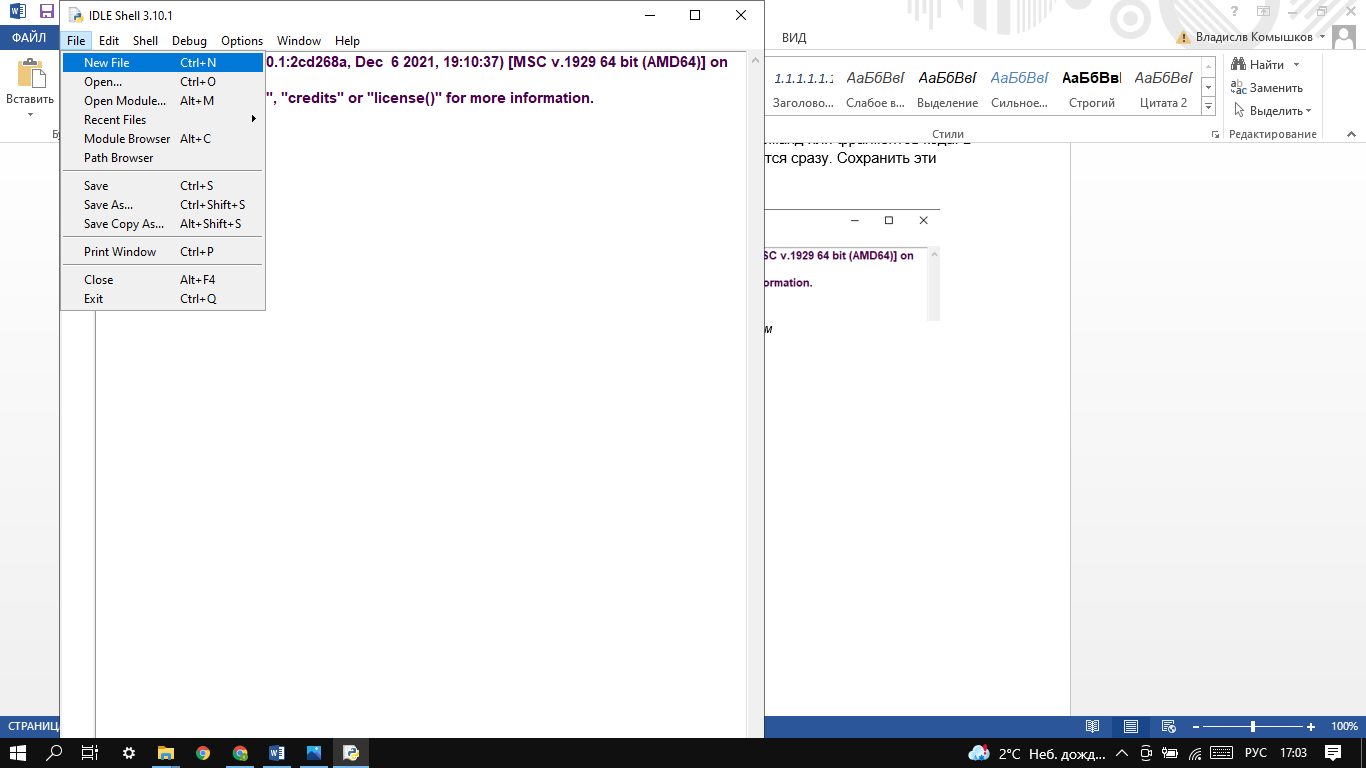
Нам надо чтобы IDLE напоминал об этом!



*(Рис. 1.6.2) Shell/Ed*

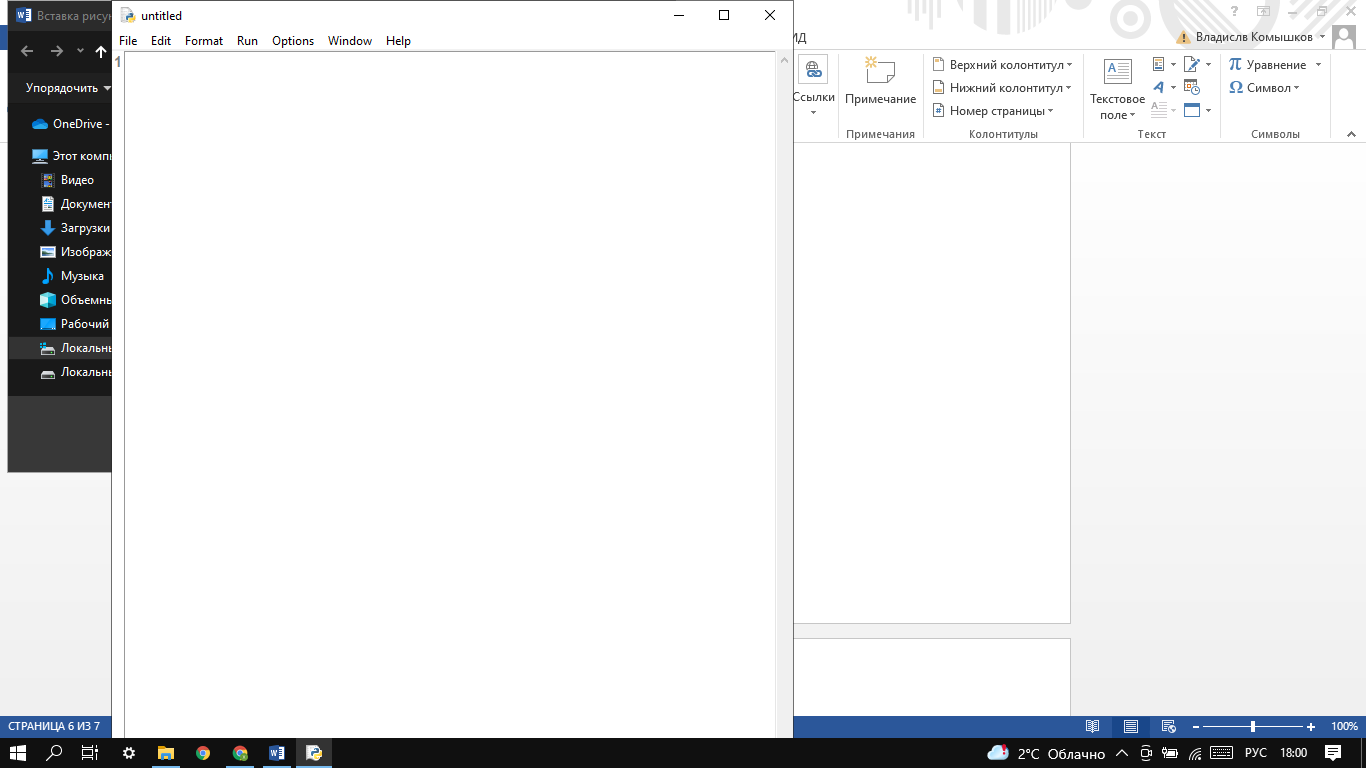
### Создать файл

Нажимаем ***New File*** (Рис. 1.6.3).



*(Рис 1.6.3) New File*

Появляется новое окошко (Рис. 1.6.4).



*(Рис. 1.6.4)*

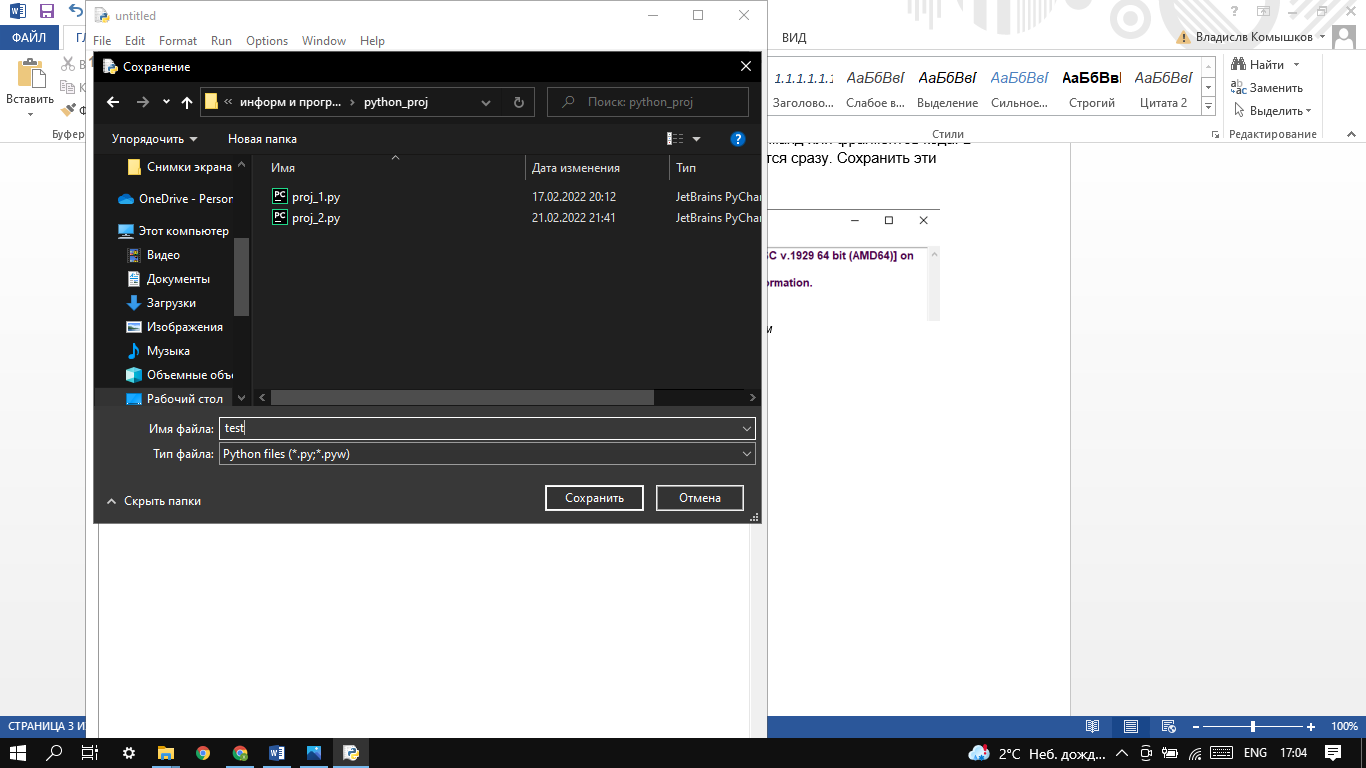
### Сохранить файл

Сначала необходимо сохранить файл. Для этого нажимаем ***Save as...*** или ***Save*** (Рис. 1.6.5).



*(Рис. 1.6.5) Save file*

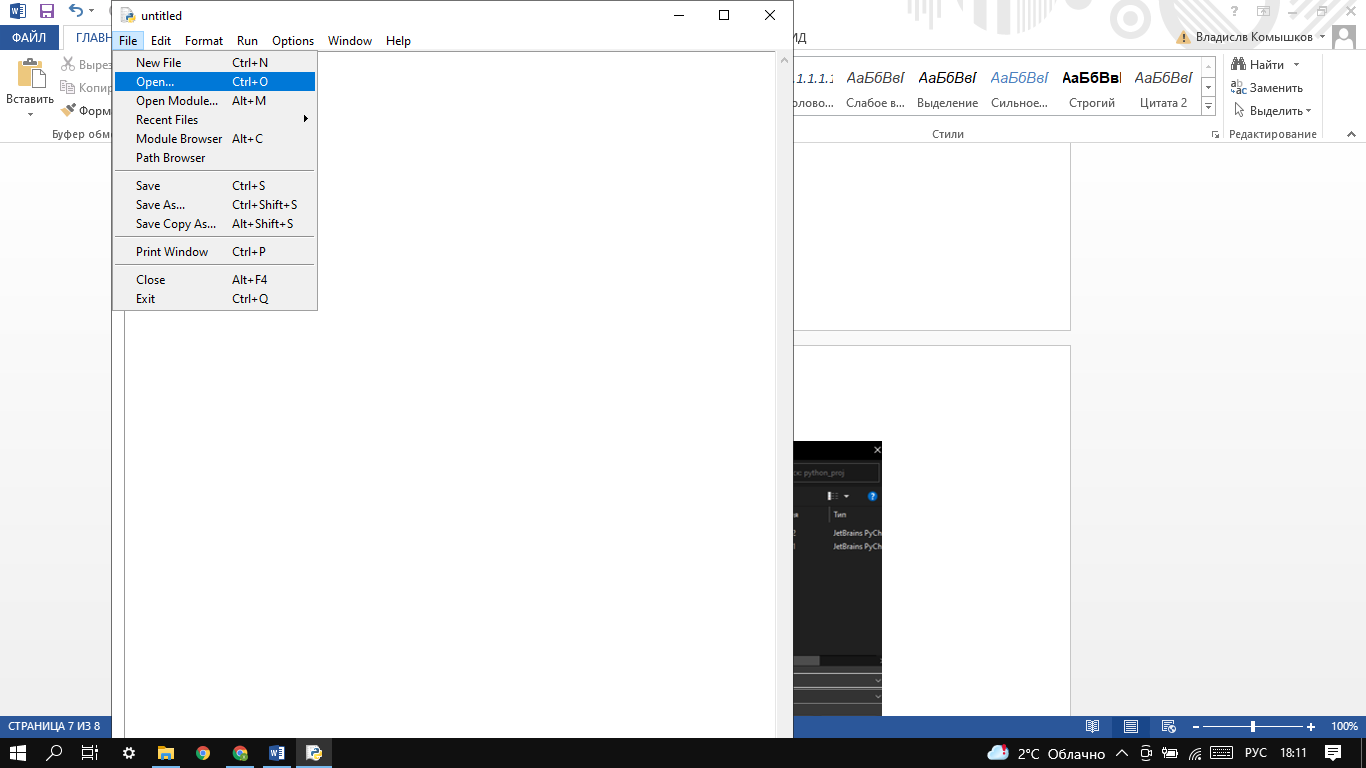
Находим нужную папку для сохранения и даем название файлу. Также выбираем тип файла Python files (Рис. 1.6.6).



*(Рис. 1.6.6) Сохранение*

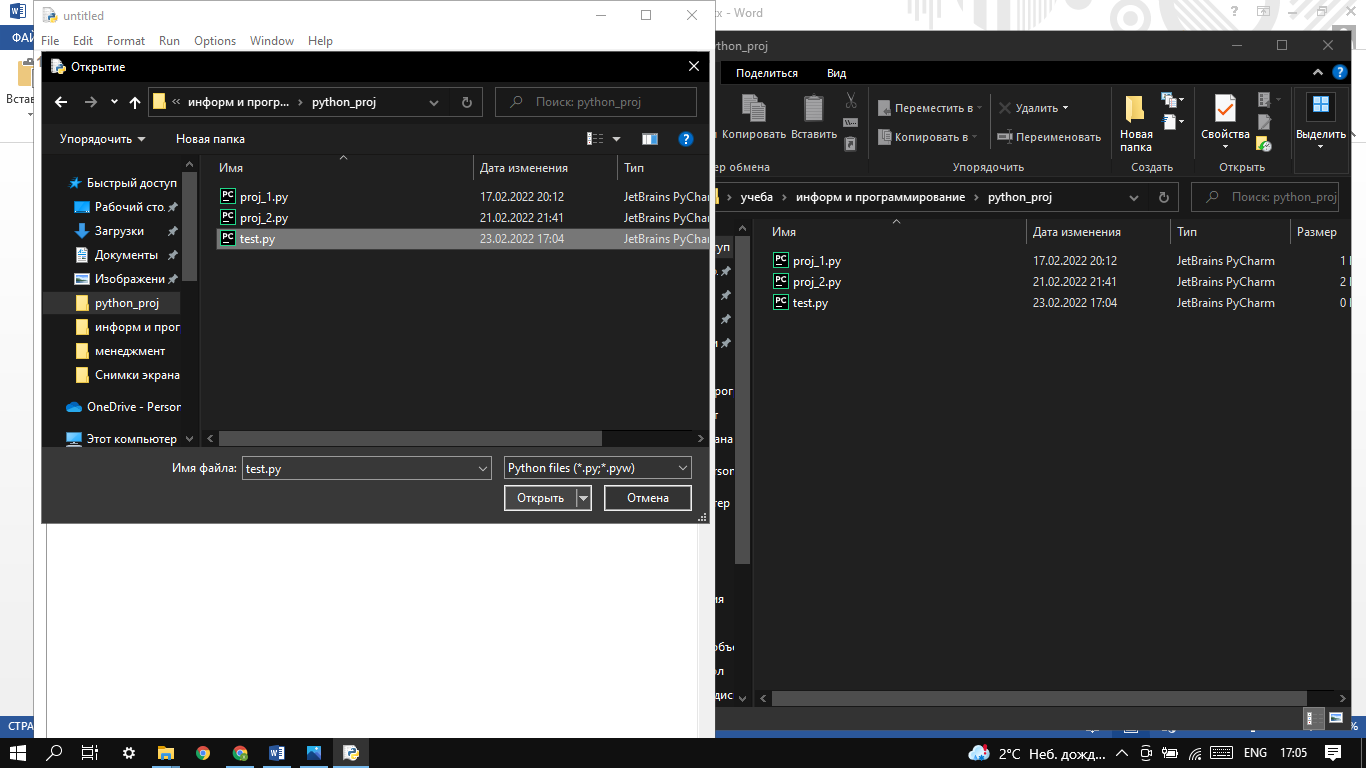
### Запустить файл

Чтобы в последующем запустить файл, нажимаем ***Open...*** (Рис. 1.6.7)



*(Рис. 1.6.7) Open file*

Находим файл и открываем (Рис. 1.6.8).



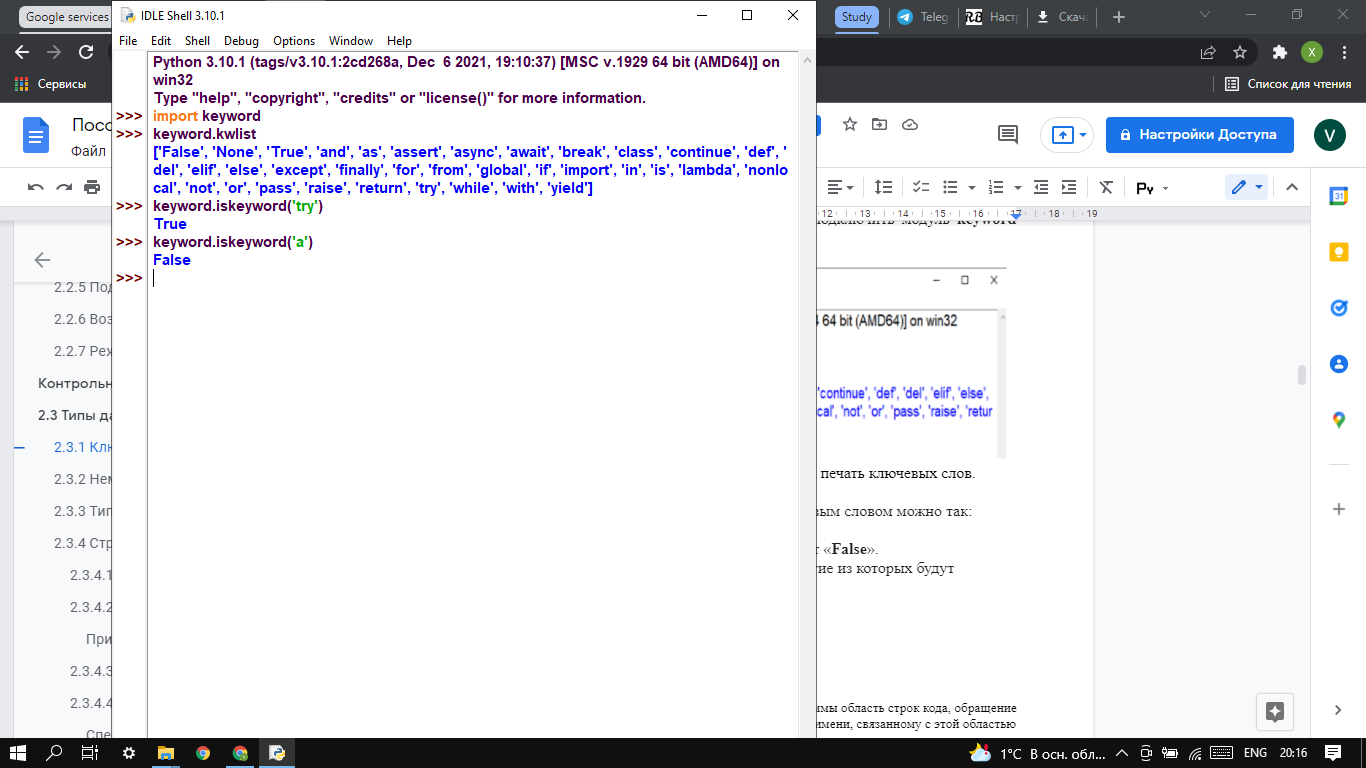
*(Рис. 1.6.8) Открыть файл*

## Ключевые слова

Ключевые слова — это часть языка. Их нельзя использовать для названия переменных, функций или классов.

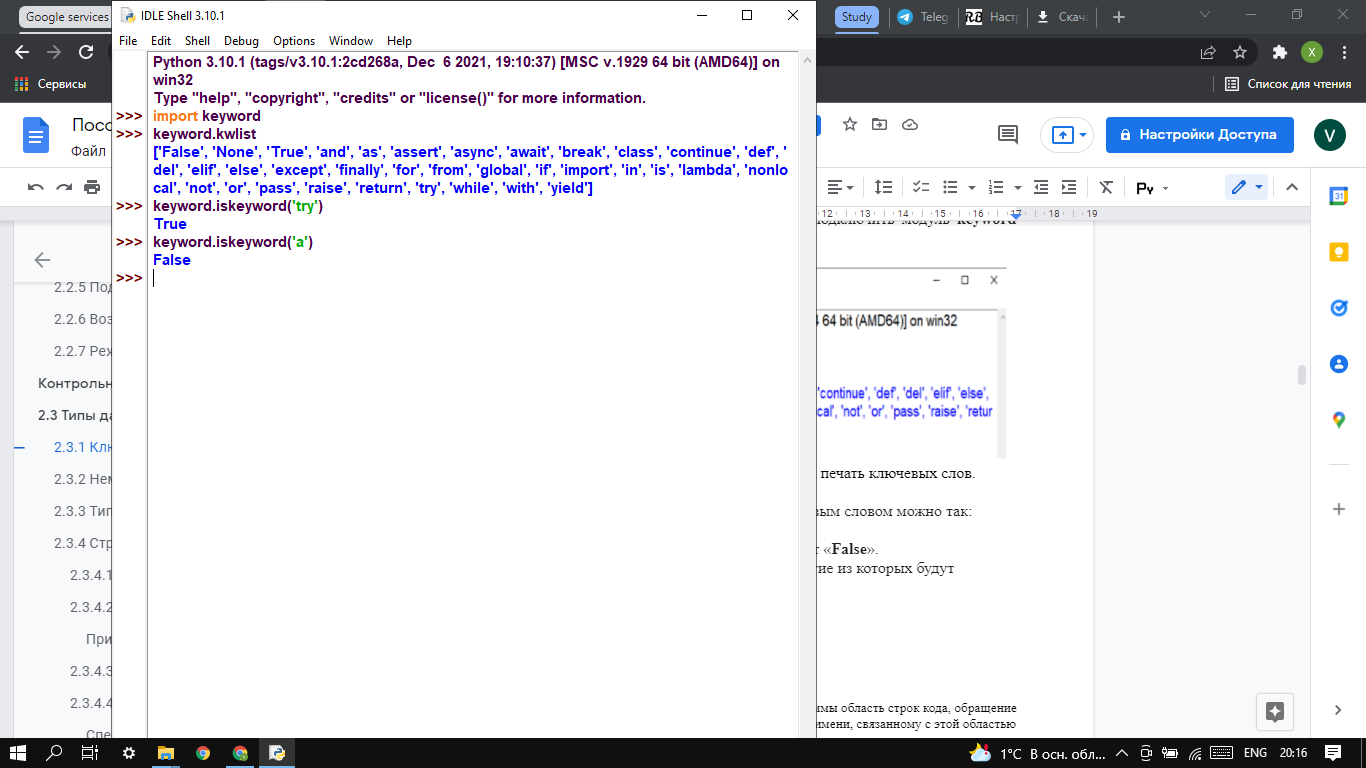
Вернемся в интерактивный режим и подключим модуль ***keyword***.

С помощью команды ***keyword.kwlist***, выведем список ключевых слов (Рис. 1.12).



*(Рис. 1.12) Ключевые слова*

Чтобы проверить совпадает ли наша переменная с ключевым словом, воспользуемся функцией ***keyword.iskeyword()*** (Рис. 1.13). Если совпадает, то система выведет ***«True»***, нет – ***«False»***.

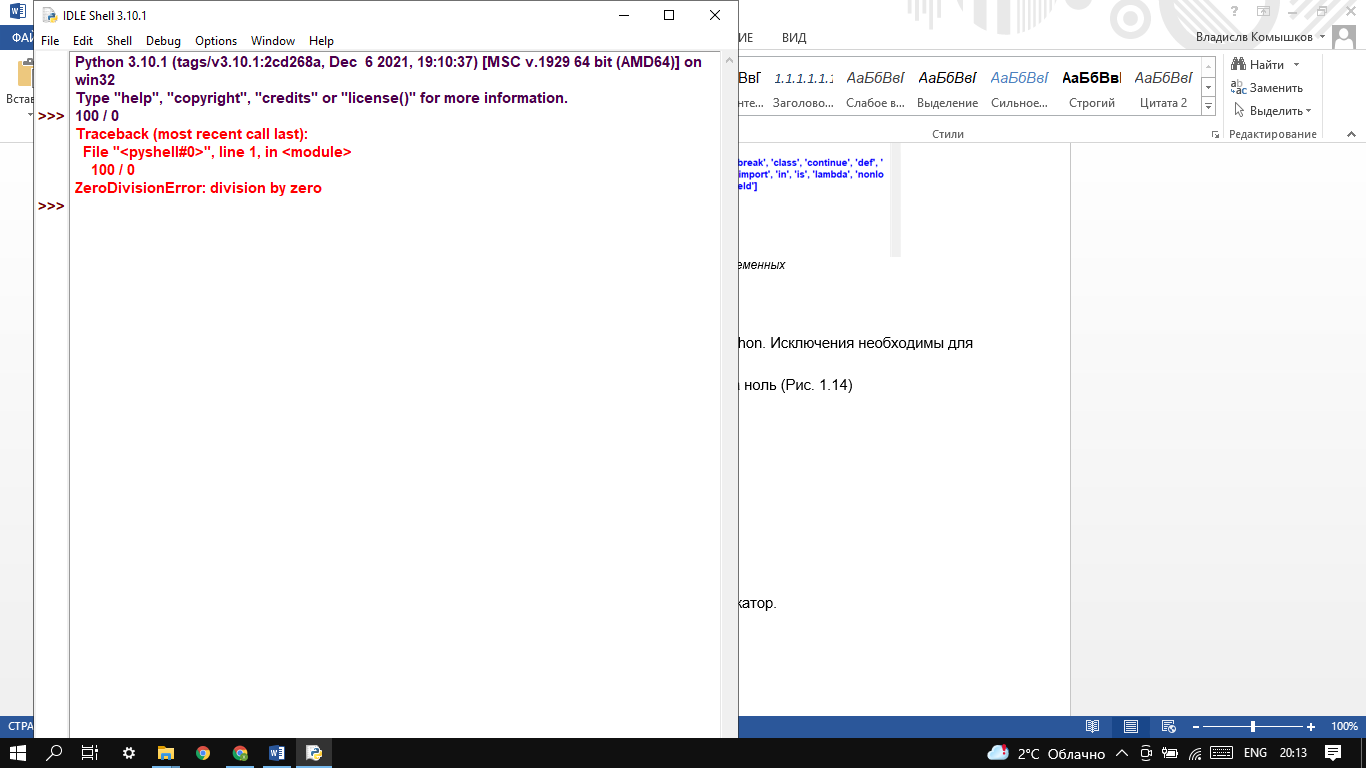


*(Рис. 1.13) Проверка имен переменных*

## Исключения в Python

***Исключения*** (exceptions) - ещё один тип данных в python. Исключения необходимы для того, чтобы сообщать программисту об ошибках.

Самый простейший пример - деление на ноль (Рис. 1.14)

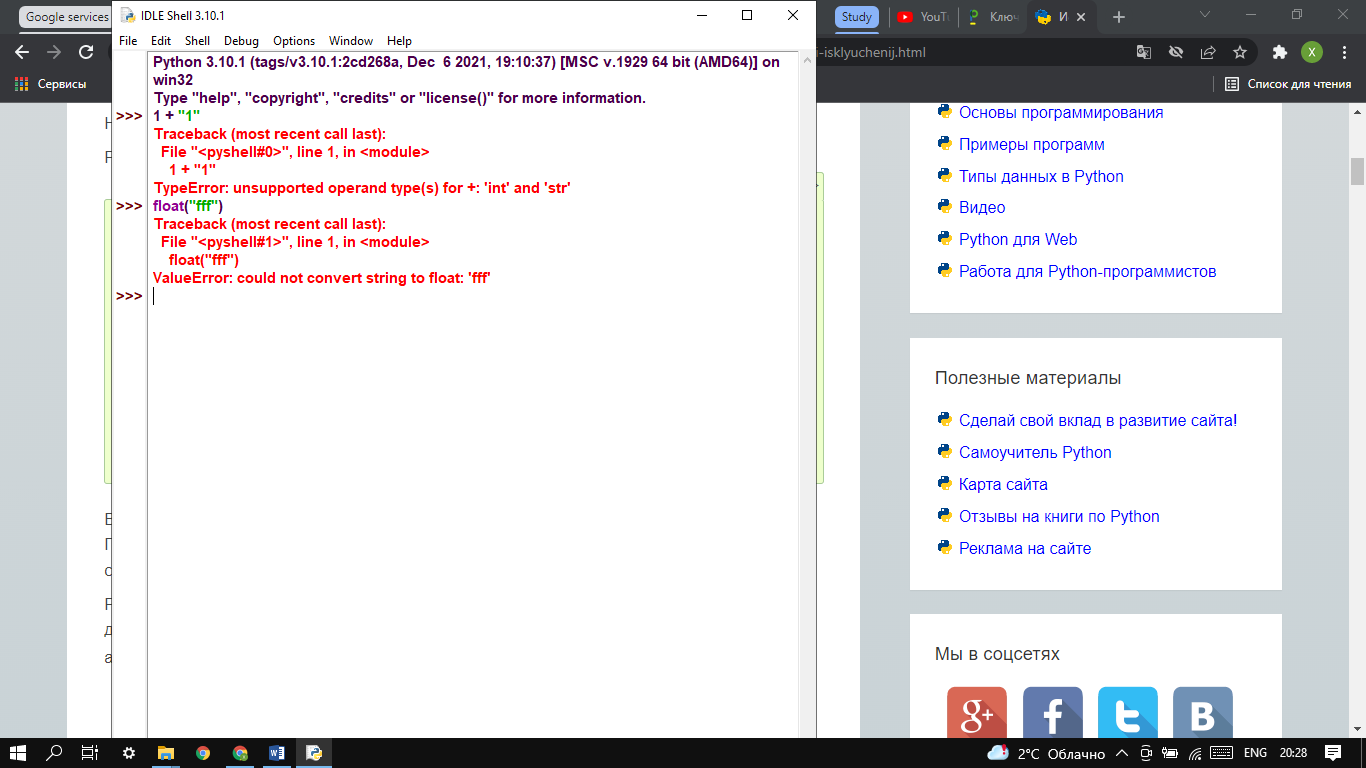


*(Рис. 1.14) Деление на ноль*

Разберём это сообщение подробнее:

1. Интерпретатор нам сообщает о том, что он поймал исключение и напечатал информацию (***Traceback (most recent call last*)**)
2. Далее имя файла (***File "<pyshell#0>"***) и строка в файле (***line 1***)
3. Выражение, в котором произошла ошибка (***100 / 0***)
4. Название исключения (***ZeroDivisionError***) и краткое описание исключения (***division by zero***)

Еще примеры исключений (Рис. 1.15).

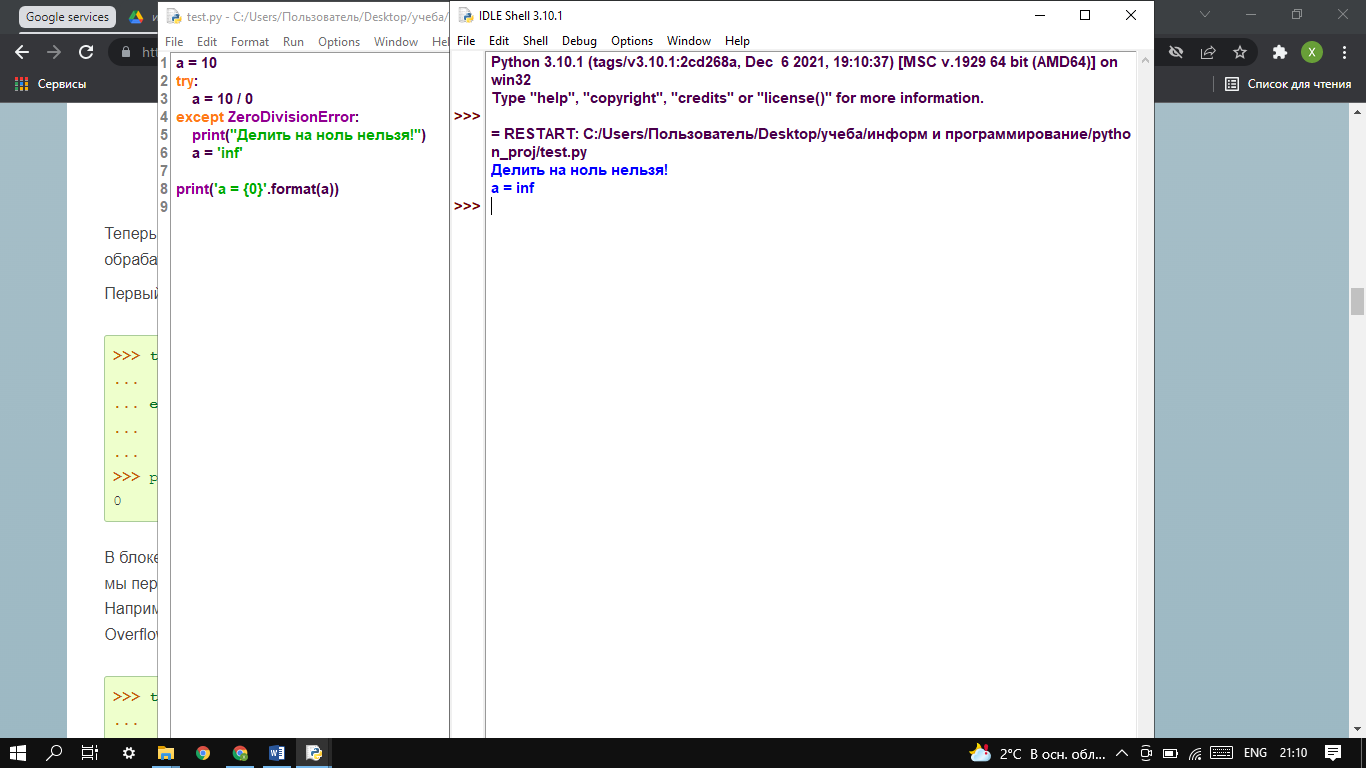


*(Рис. 1.15)*

Генерируются исключения ***TypeError*** и ***ValueError***. Как мы видим, подсказки дают нам полную информацию о том, где появилось исключение, и с чем оно связано.

В Python существует огромное множество исключений, также сам программист может создавать свои исключения.

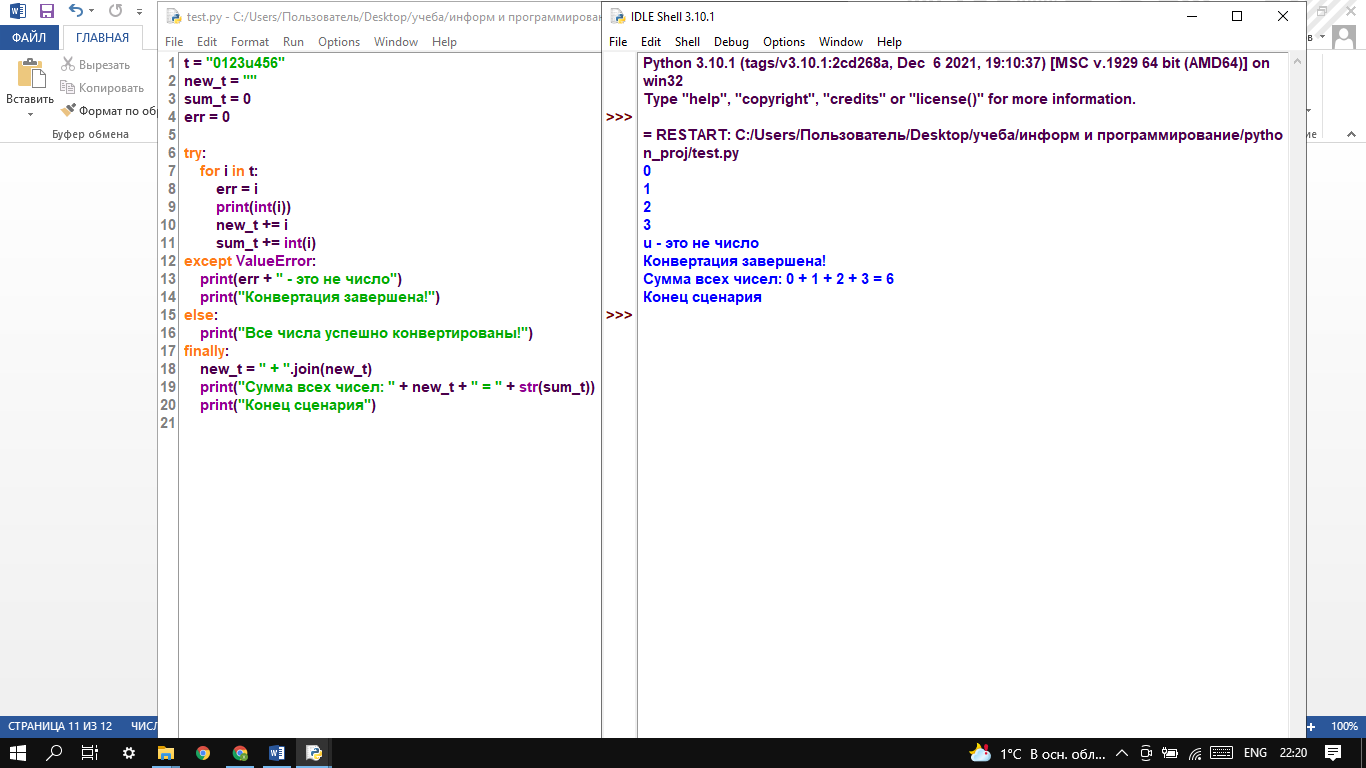
Зная, когда и при каких обстоятельствах могут возникнуть исключения, мы можем их обрабатывать. Для обработки исключений используется конструкция ***try – except*** (Рис. 1.16).



*(Рис. 1.16) try и except*

В блоке ***try*** мы выполняем инструкцию, которая может вызвать исключение. В блоке ***except*** мы перехватываем его. При этом перехватываются и его потомки. Например, перехватывая ***ArithmeticError***, мы также перехватываем ***FloatingPointError***, ***OverflowError*** и ***ZeroDivisionError***.

Инструкции ***finally*** и ***else*** (Рис. 1.17). ***Finally*** выполняет блок инструкций в любом случае, было ли исключение, или нет (применима, когда нужно непременно что-то сделать). Инструкция ***else*** выполняется в том случае, если исключения не было.



*(Рис. 1.17) finally и else*

# **ООП**

***ООП (объектно-ориентированное программирование) - парадигма программирования, в которой основными концепциями являются понятия объектов и классов.***

***Объект - это экземпляр класса. Класс можно сравнить с чертежом, по которому создаются объекты. Python соответствует принципам объектно-ориентированного программирования.***

***Типы данных в Python - это классификация, которая определяет, какой тип значения имеет переменная и какой тип математических, логических операций может быть применен.***

## Идентификаторы

В Pyhton все является объектами.

При создании объекта ему присваивается ***идентификатор***.

***Идентификатор*** - это адрес памяти объекта. После каждого запуска программы они отличаются.

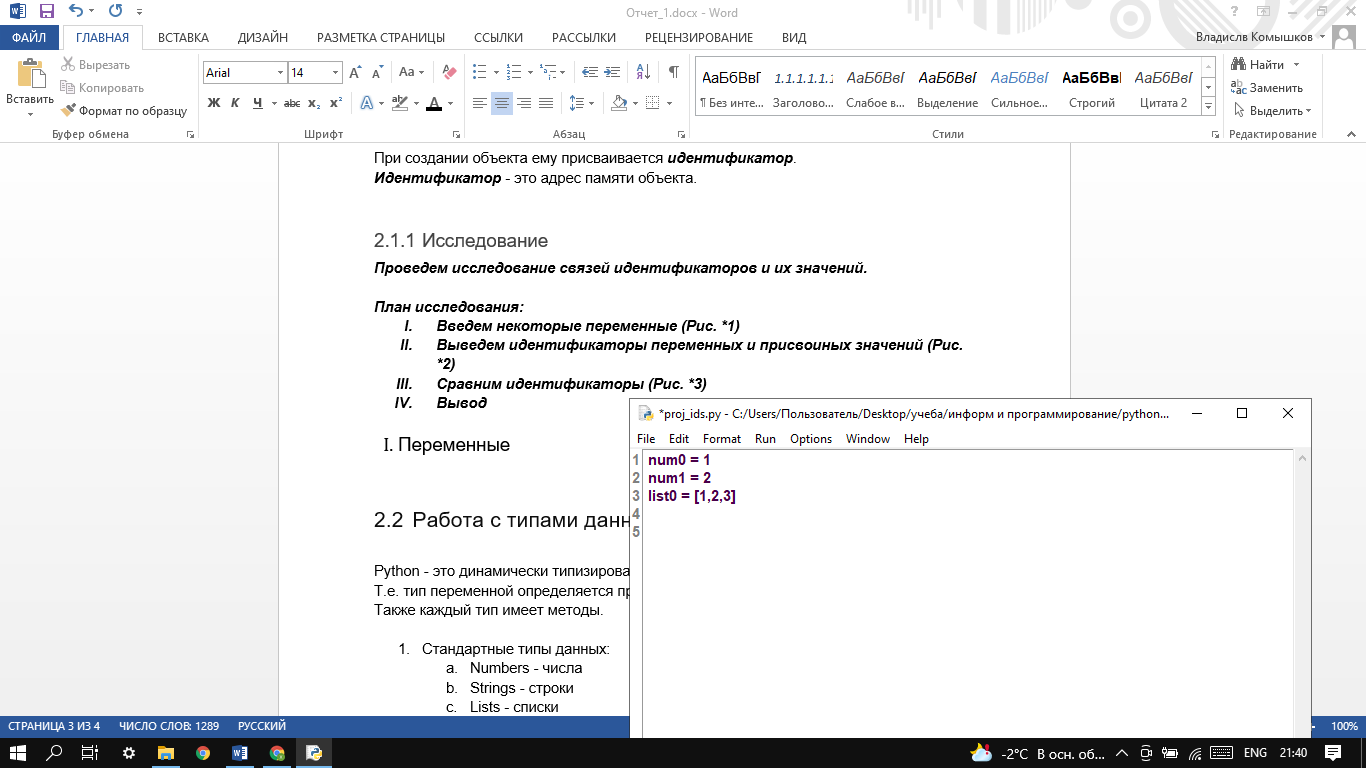
### Исследование

***Проведем небольшое исследование связей идентификаторов и их значений.***

***План исследования:***

1. ***Введем некоторые переменные (Рис. \*1)***
2. ***Выведем идентификаторы переменных (Рис. \*2)***
3. ***Сравним идентификаторы (Рис. \*3)***
4. ***Вывод***
5. Переменные

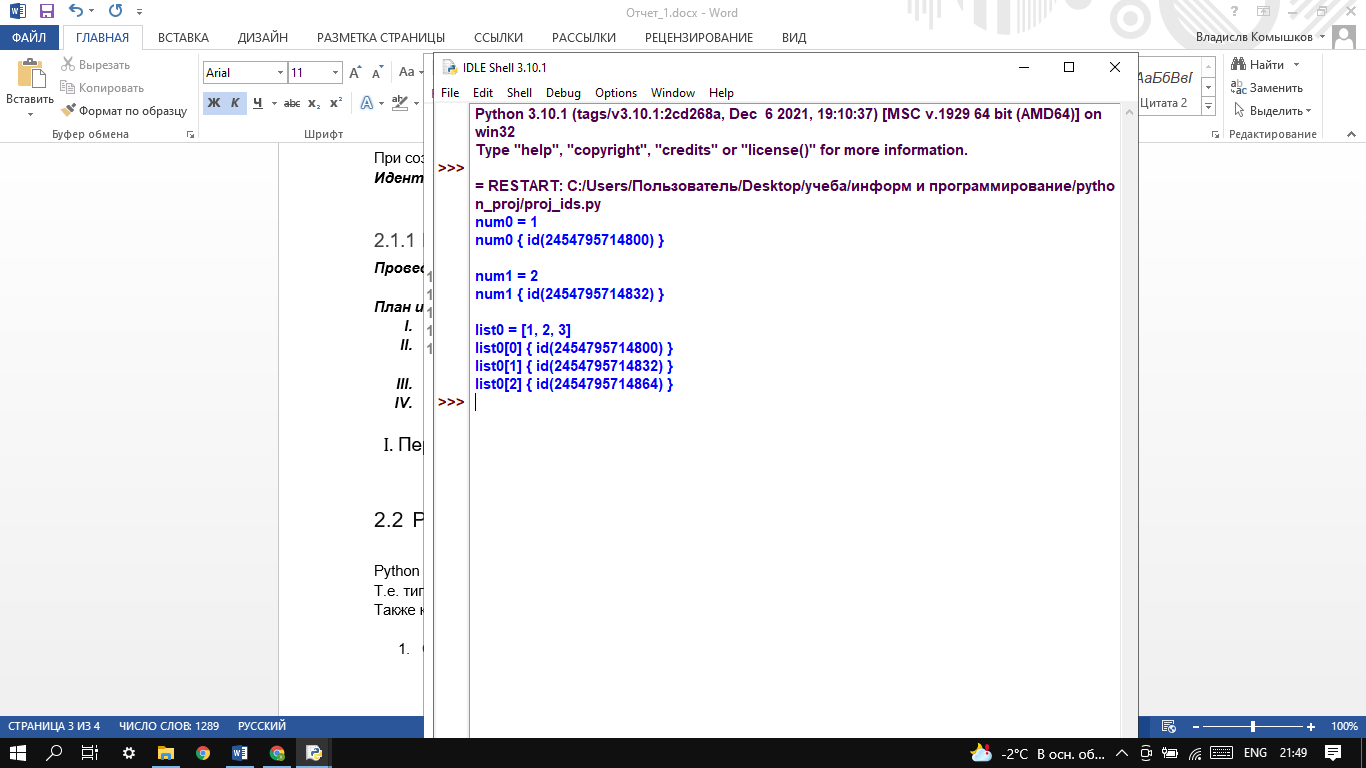
Введем некоторые переменные.



*(Рис. \*1)*

1. Идентификаторы

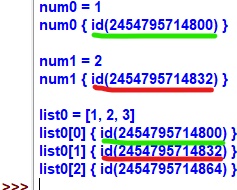
Выведем идентификаторы переменных с помощью функции ***id()***.



*(Рис. \*2)*

1. Сравнение

Заметим, что независимо от имени и типа переменной, некоторые идентификаторы одинаковые.



*(Рис. \*3)*

|  |  |  |
| --- | --- | --- |
| **ИМЯ** | **ID** | **Значение** |
| **num0, list[0]** | 2454795714800 | 1 |
| **num1, list[1]** | 2454795714832 | 2 |

1. Вывод

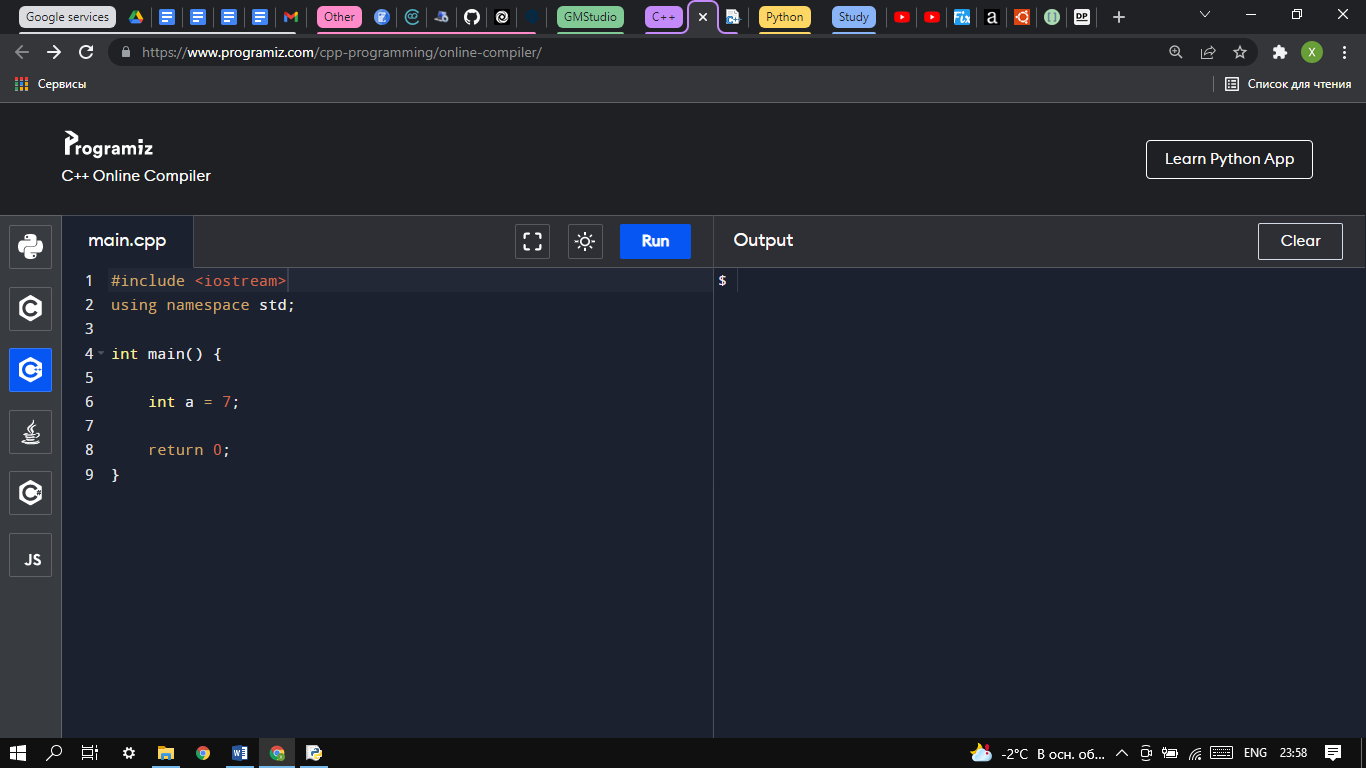
На сколько мы знаем, идентификатор является адресом памяти объекта. Идентичность идентификаторов указывает нам на то, что введенные переменные обращаются к одному и тому же адресу памяти, где хранится значение. Получается, имя переменной – это некоторая метка или ссылка к идентификатору. И таких ссылок возможно создать огромное множество и все они могут относится к одному объекту. Такой метод хранения данных является для нас большим плюсом, т.к. повышает эффективность и скорость работы наших программ.

## Работа с типами данных

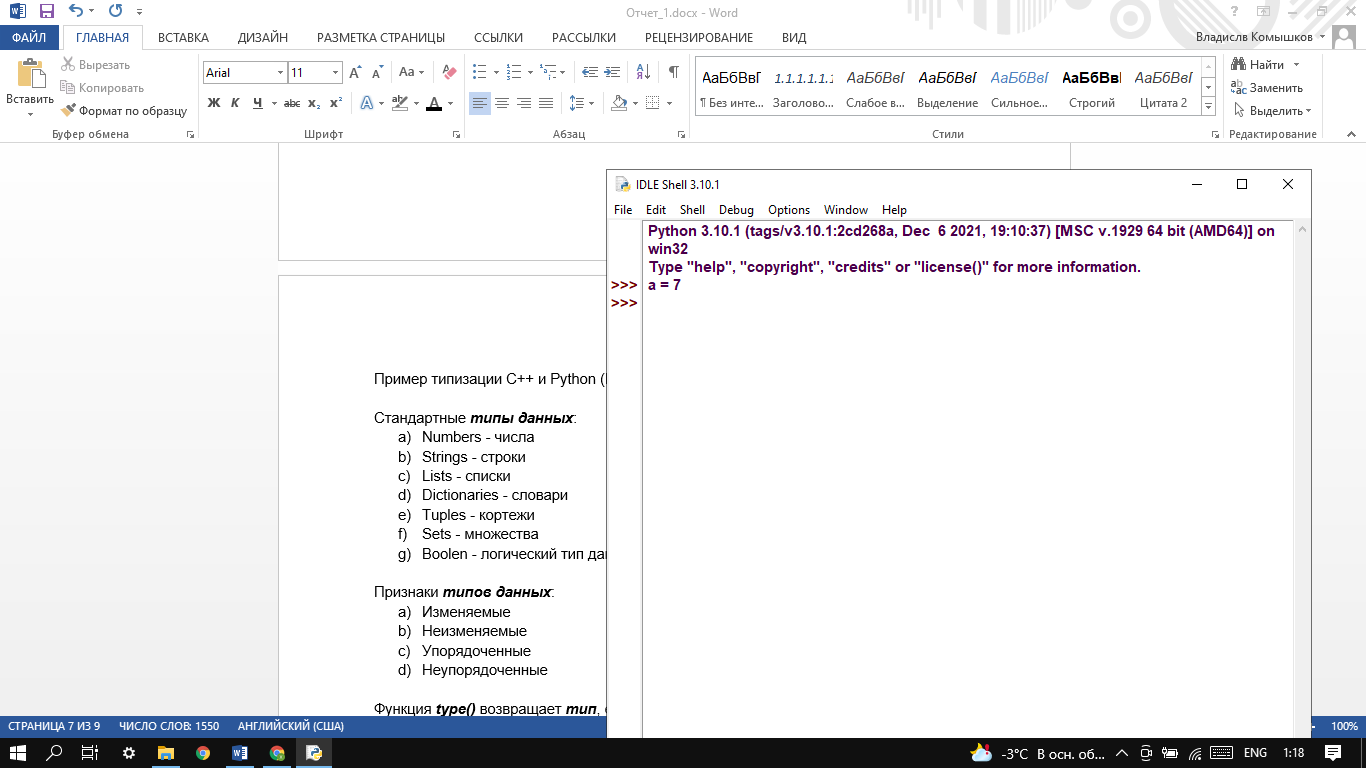
Python - это ***динамически типизированный язык***.

Т.е. ***тип переменной*** определяется при выполнении программы.

Пример типизации C++ и Python (Рис. 2.1 – 2.2).



*(Рис. 2.1) C++*



*(Рис 2.2) Python*

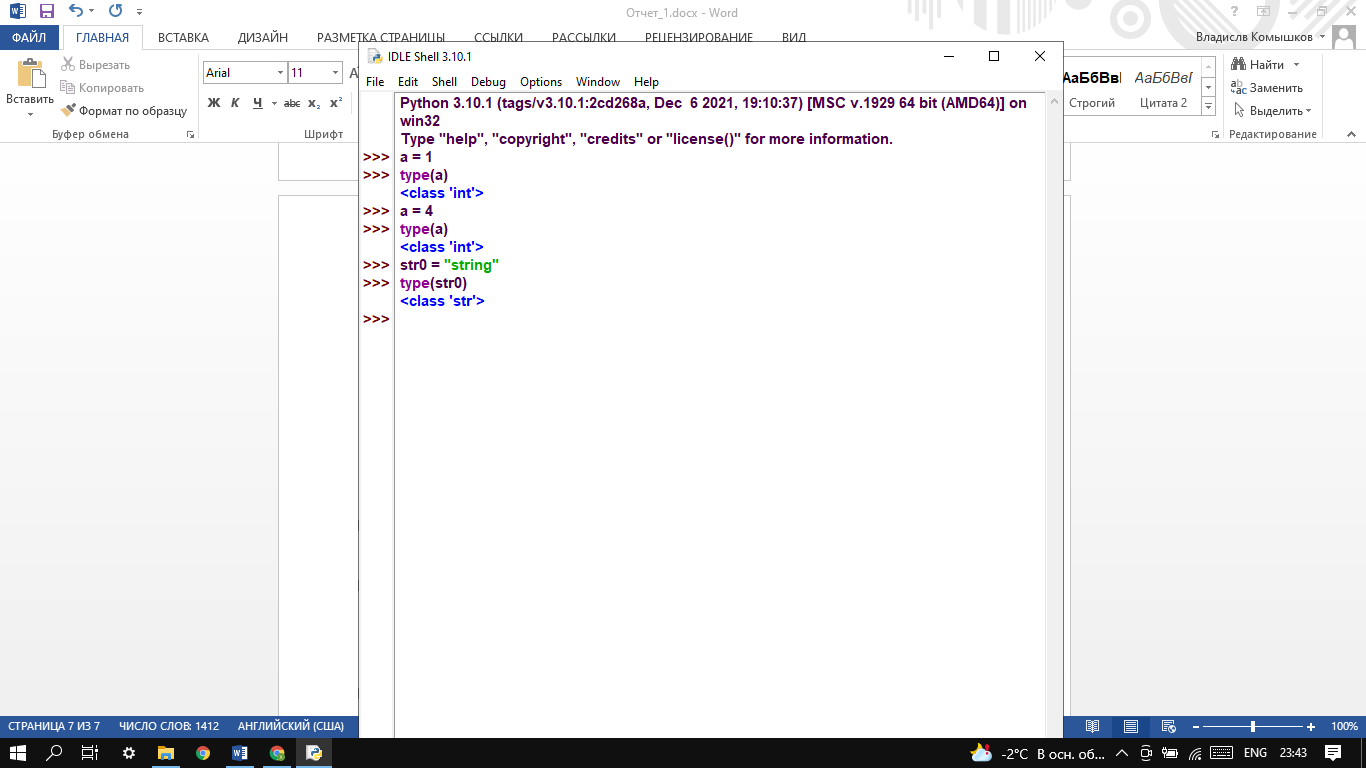
Стандартные ***типы данных***:

1. Numbers - числа | int, float
2. Strings - строки | str
3. Lists - списки | list
4. Dictionaries - словари | dict
5. Tuples - кортежи | tuple
6. Sets - множества | set
7. Boolen - логический тип данных | bool

Признаки ***типов данных***:

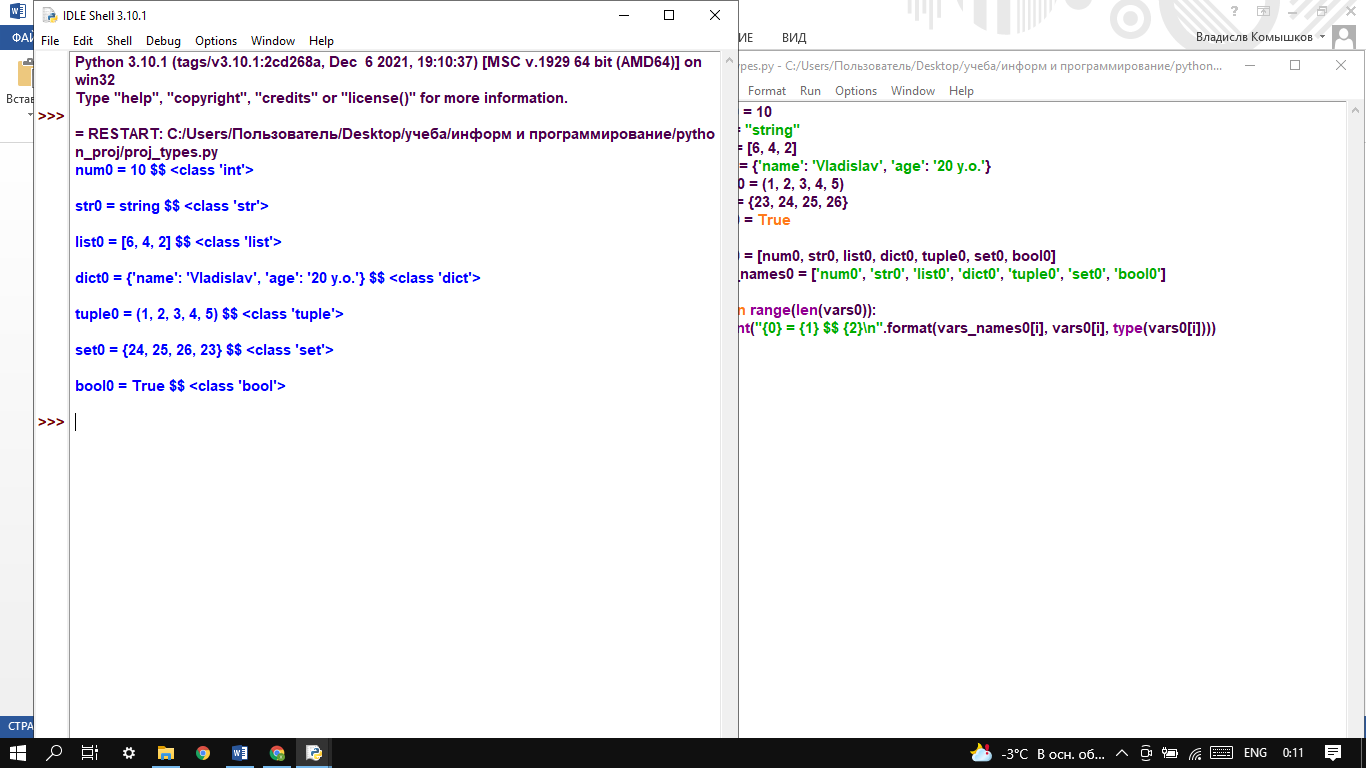
1. Изменяемые
2. Неизменяемые
3. Упорядоченные
4. Неупорядоченные

Функция ***type()*** возвращает ***тип***, его еще называют ***класс***, ***переменной*** (Рис. 2.3).



*(Рис. 2.3) Type*

Таким образом мы можем узнать ***все типы*** (Рис. 2.4).

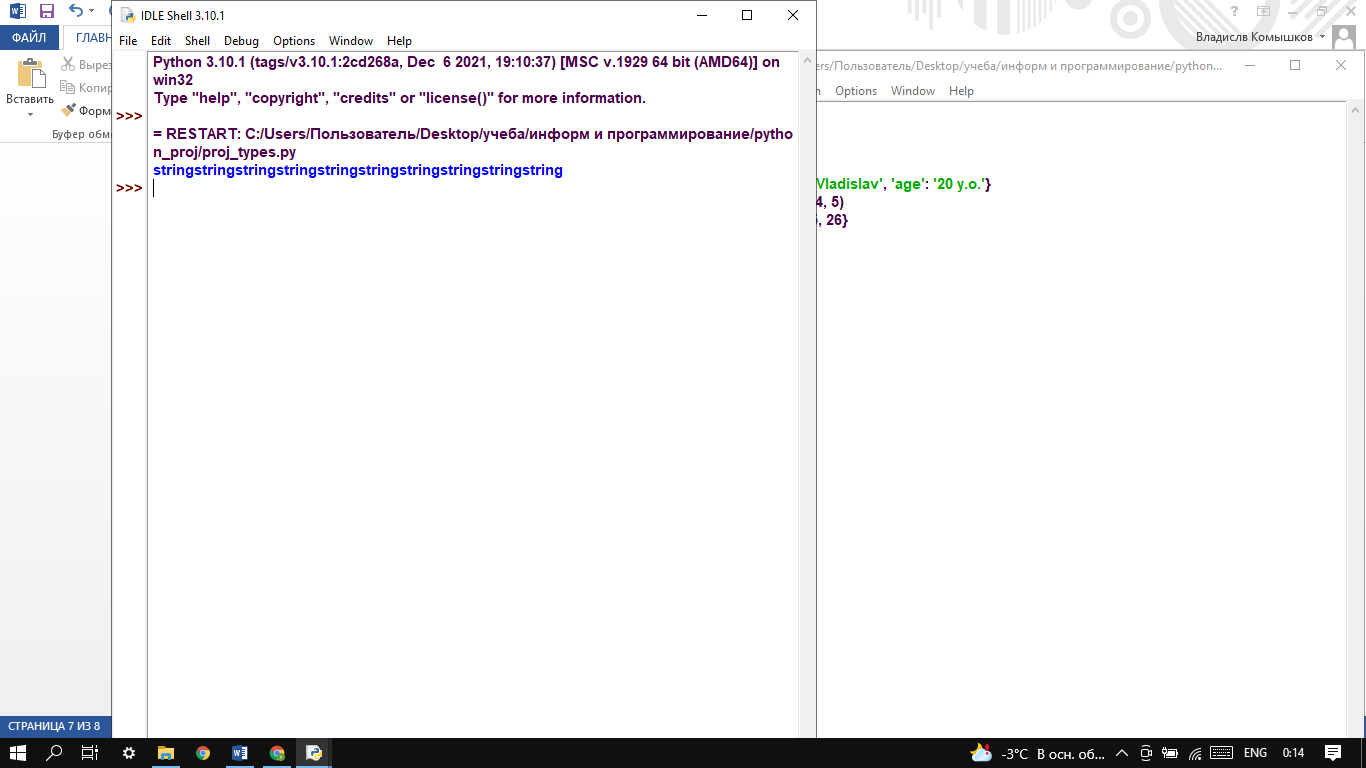


*(Рис. 2.4)*

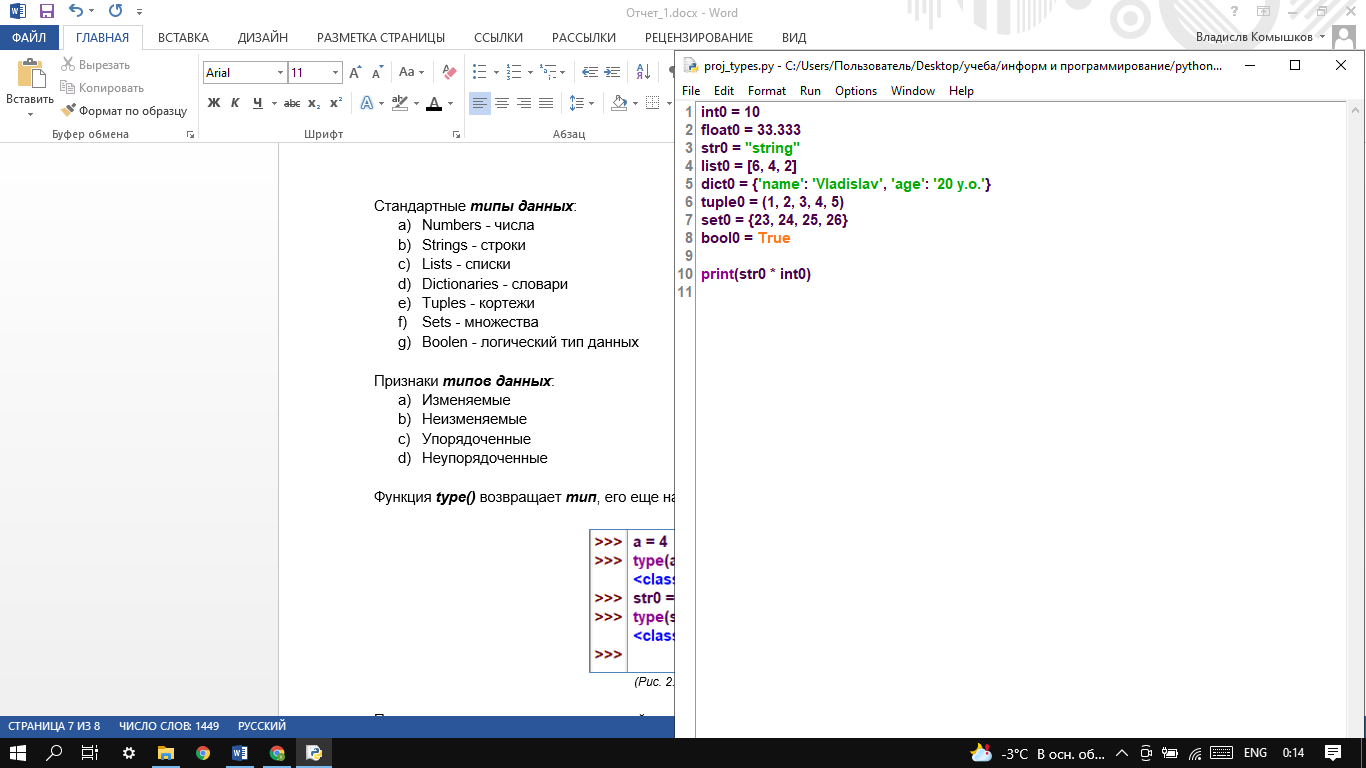
### Взаимодействия типов

Ниже будут приведены примеры ***взаимодействия*** типов переменных.

Умножение строк на целочисленные значения (Рис. 2.2.1 – 2.2.2).

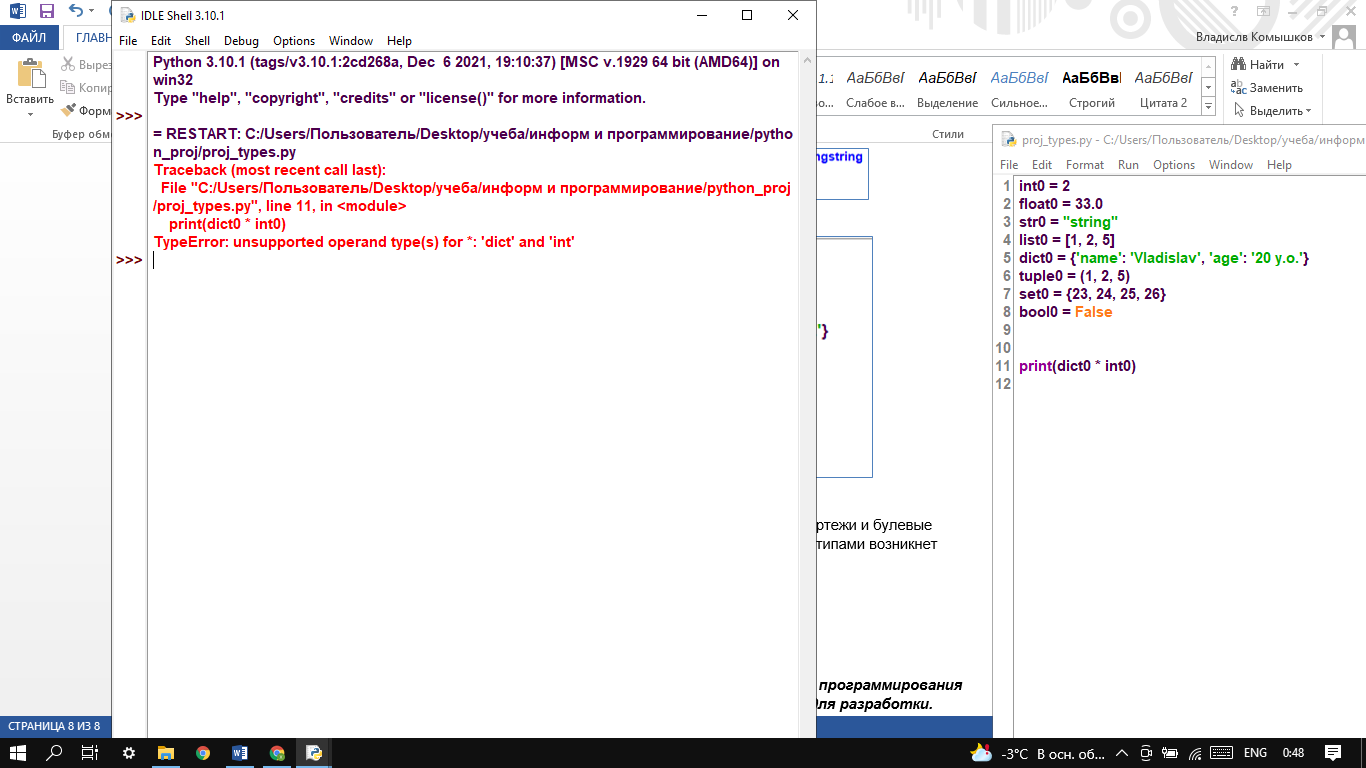


*(Рис. 2.2.1)*



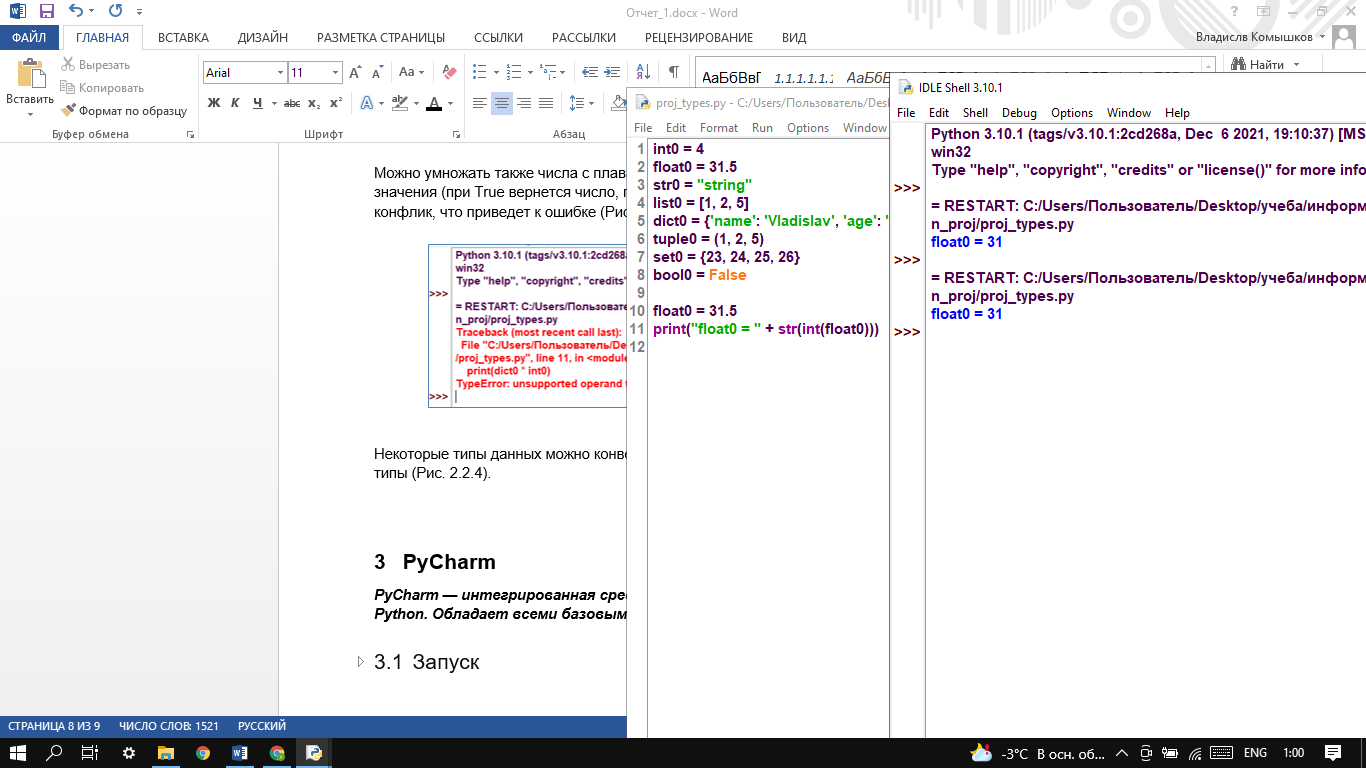
*(Рис. 2.2.2)*

Можно умножать также числа с плавающей запятой, списики, кортежи и булевые значения (при True вернется число, при False - 0). Но с другими типами возникнет конфлик, что приведет к ошибке (Рис. 2.2.3).



*(Рис. 2.2.3)*

Некоторые типы данных можно ***конвертировать***, т.е. преобразовывать их в другие типы (Рис. 2.2.4). Функция ***str()*** используется чаще всего. Она необходима для ***конкатенации*** (склеивания) строк и разных типов, для корректного вывода.



*(Рис. 2.2.4) Конвертация и конкатенация*

Динамически типизированный язык это удобно и экономично, с точки зрения времени. Python, как представитель такого вида, дает возможность пользователю не являющимся программистом, легко его освоить, для написания небольших программ.

Но не существовало бы других способов типизации, если бы все было так хорошо. Во время разработки более сложных программ могут возникать ошибки и для лучшей работы с типами данных подойдут другие языки, заточенные под это.

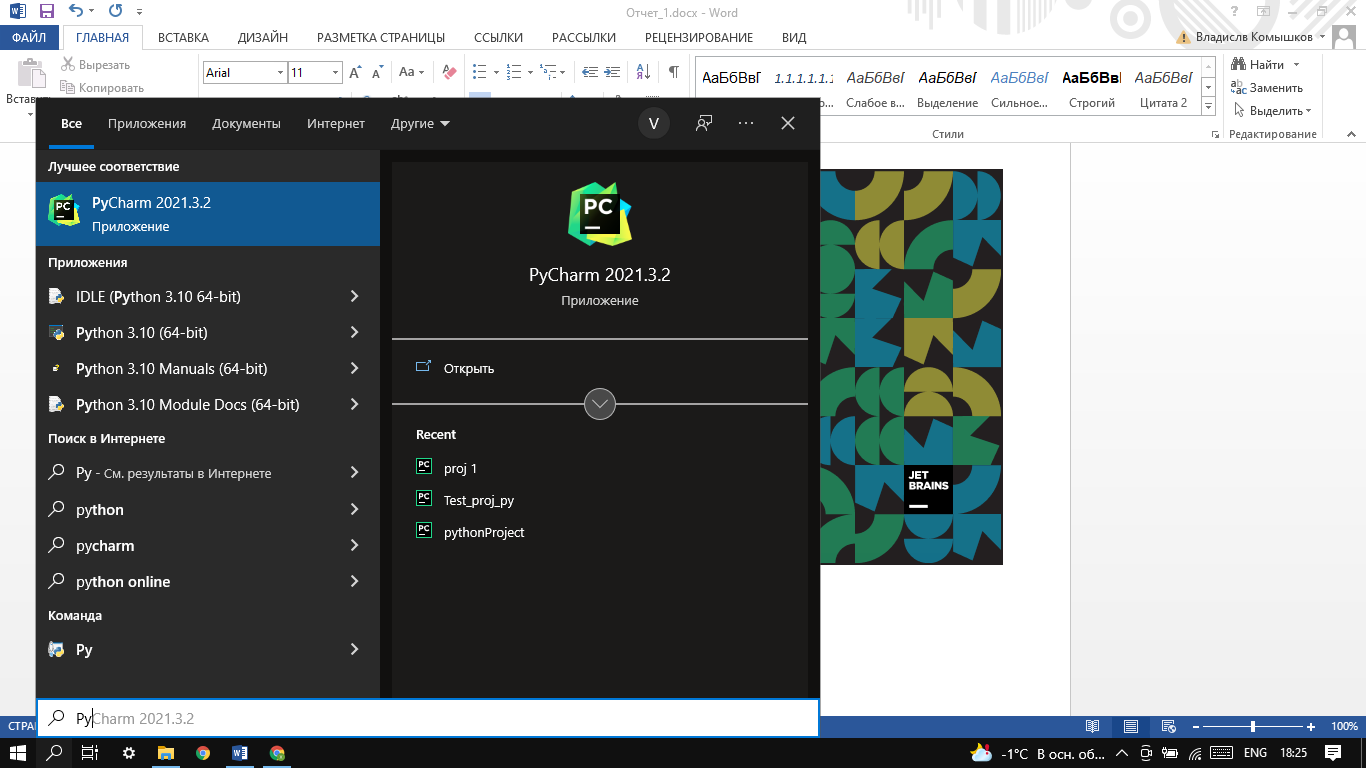
# **PyCharm**

***PyCharm — интегрированная среда разработки для языка программирования Python. Обладает всеми базовыми вещами необходимых для разработки.***

## Запуск

Запустим ***PyCharm***!

Повторим те же действия, что и с IDLE (Рис. 3.1).

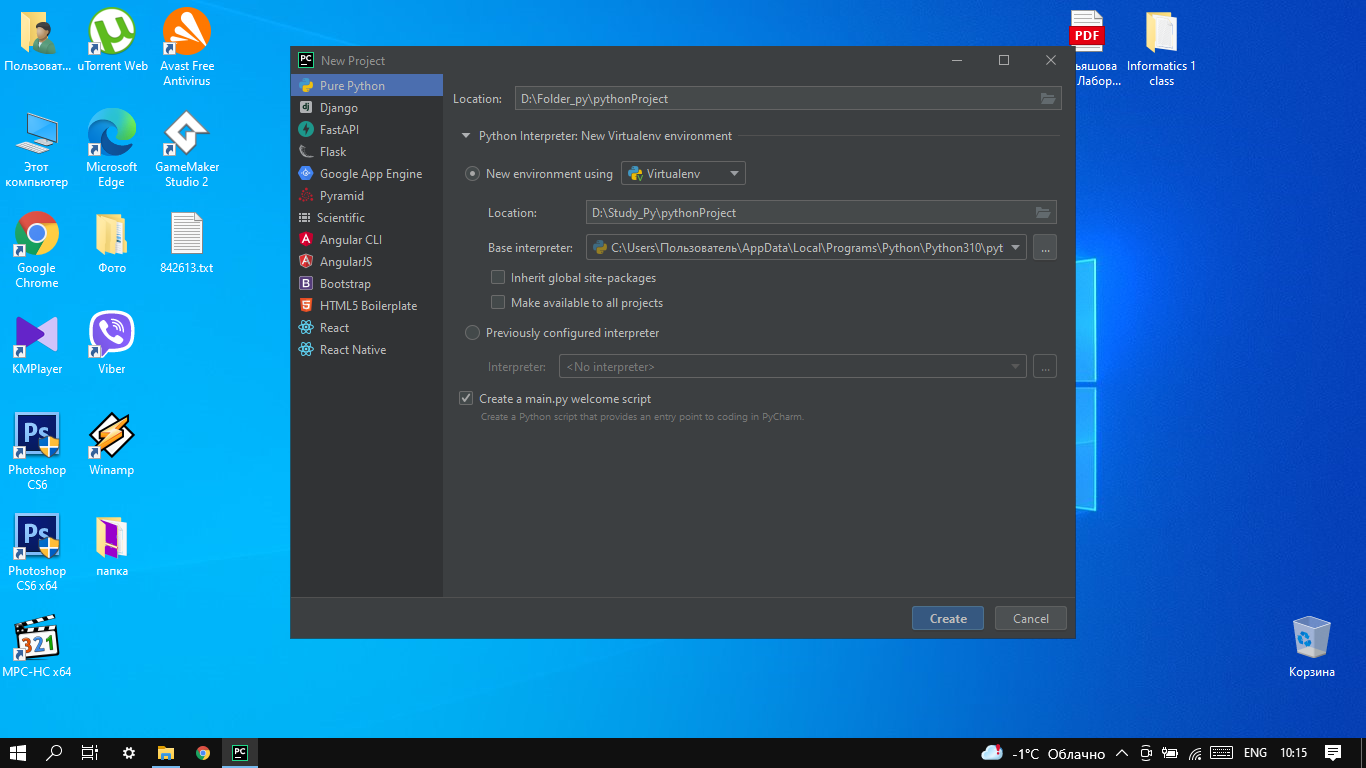


*(Рис. 3.1)*

## «Hello, world!»

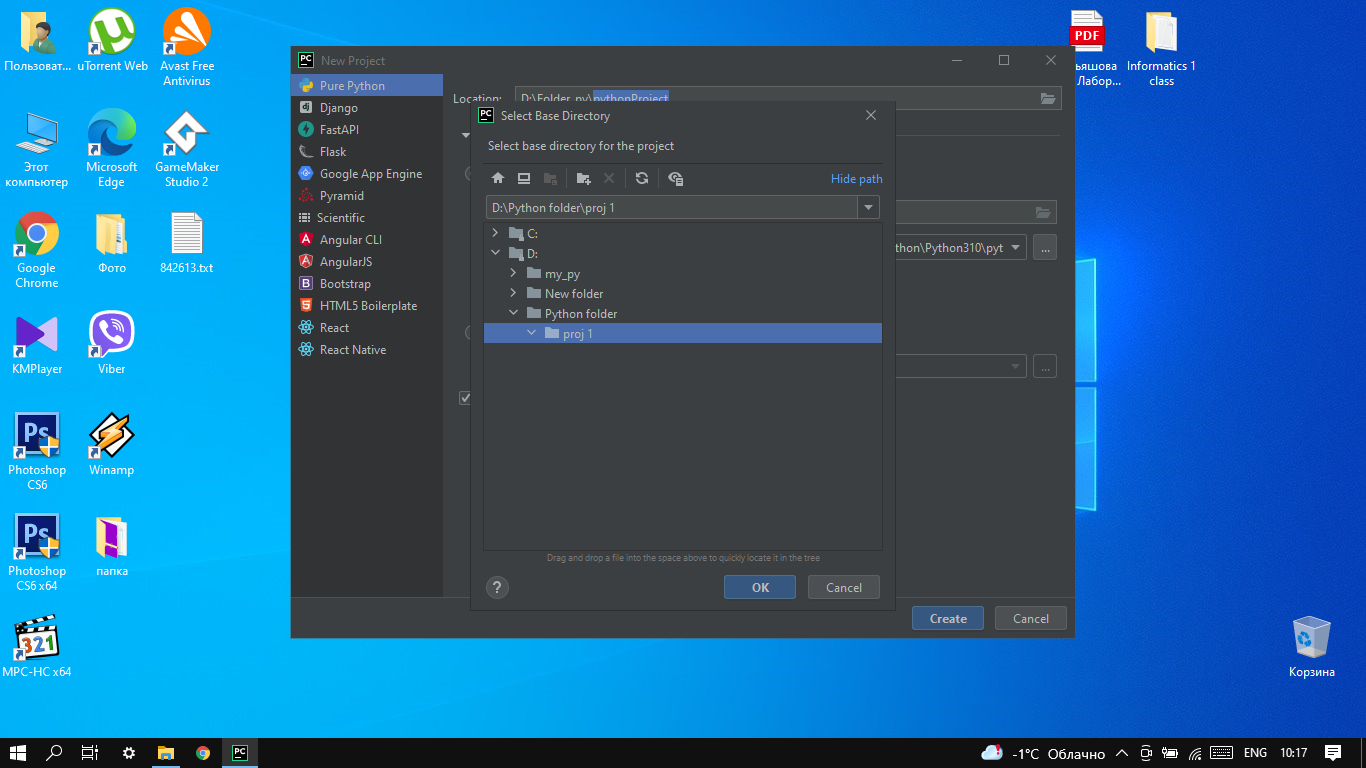
### Создать файл

Начнем с ***New Project*** (Рис. 3.2.1).



*(Рис. 3.2.1) New Project*

Для начала выберем расположение – ***Location*** (Рис. 3.2.2).



*(Рис. 3.2.2) Расположение файла*

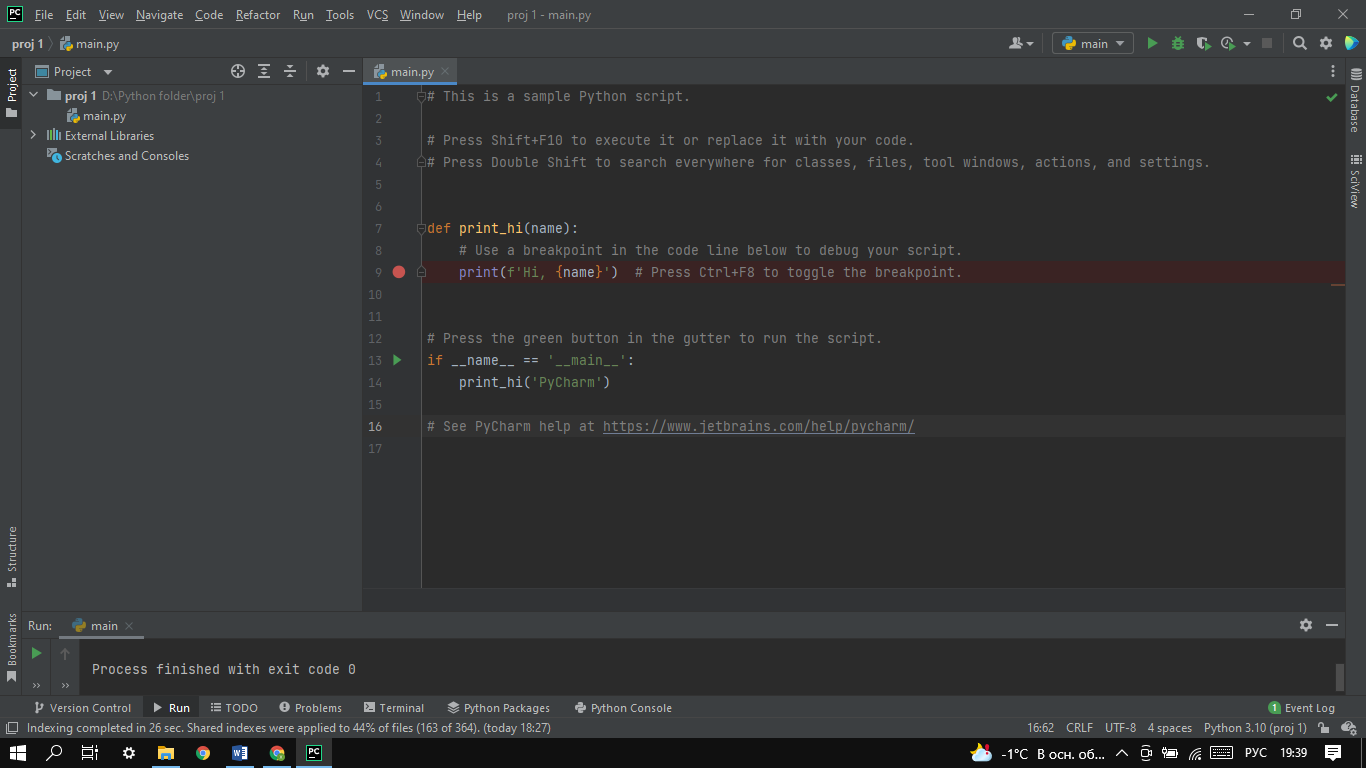
Введем 2 определения:

1. ***Interpreter* (интерпретатор)**- это такая программа, которая выполняет другие программы. Когда вы пишете программу на языке Python, интерпретатор читает вашу программу и выполняет содержащиеся в ней инструкции.
2. ***Virtualenv*** (виртуальная среда) — это инструмент Python для **управления зависимостями**и **изоляции проектов**. Он позволяет устанавливать **пакеты pip** (сторонние библиотеки) локально в изолированном каталоге для конкретного проекта, а не глобально (этих пакетов не будет в других проектах Python).

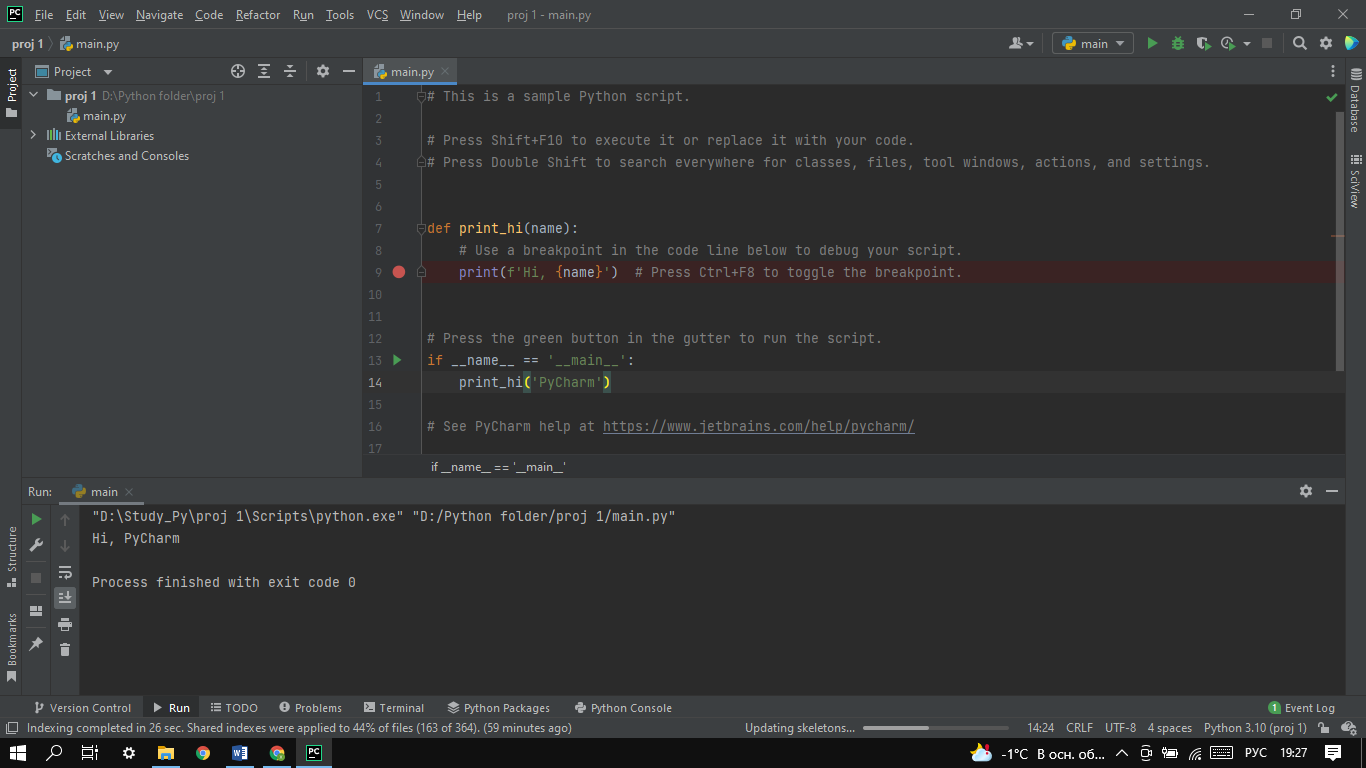
Следующие пункты создания файла это выбор ***интерпретатора*** и ***виртуального окружения***. Они установлены по-умолчанию, не будем ничего менять.

### «Hello, PyCharm!»

Проект создан. И первое, что мы видим, готовый скрипт. Запустим его – *Shift+F10* (Рис. 3.2.3 – 3.2.4).

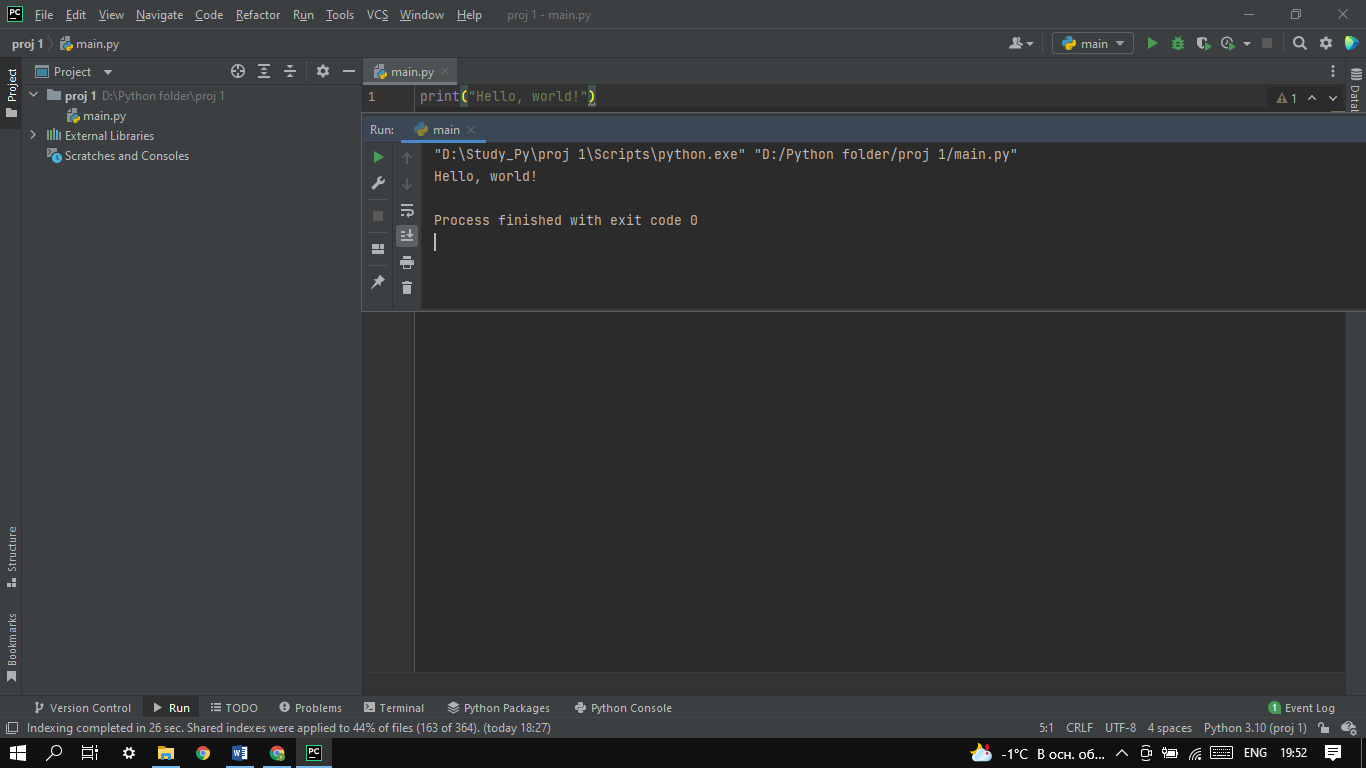


*(Рис. 3.2.3)*

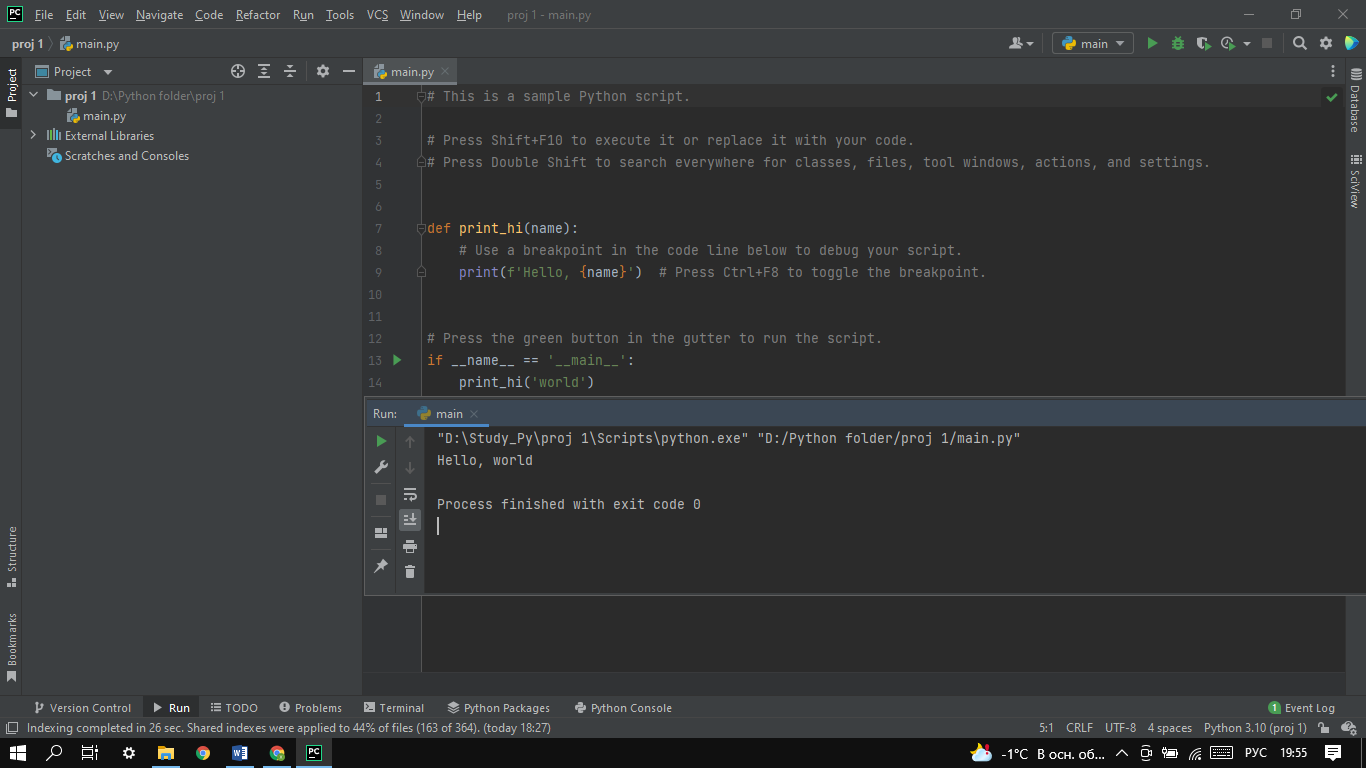


*(Рис. 3.2.4)*

Можем стереть скрипт и написать стандартный ***«Hello, world!»***, либо изменить готовый код (Рис. 3.2.5 – 3.2.6).



*(Рис. 3.2.5) «Hello, world!» PyCharm*



*(Рис. 3.2.6) «Hello, world» PyCharm*

## Настройки

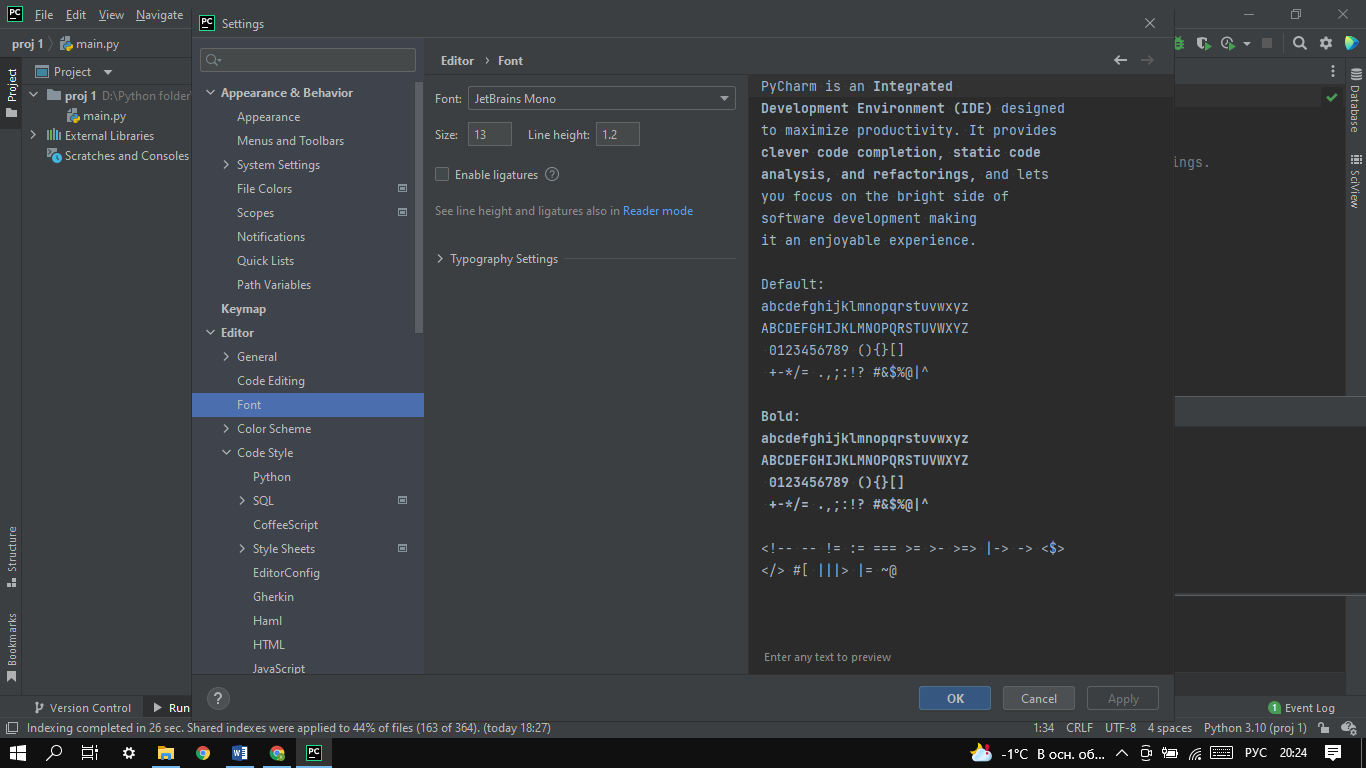
### Шрифт и подсветка

Шрифт:

***File >> Settings... >> Editor >> Font*** (Рис. 3.2 – 3.3)



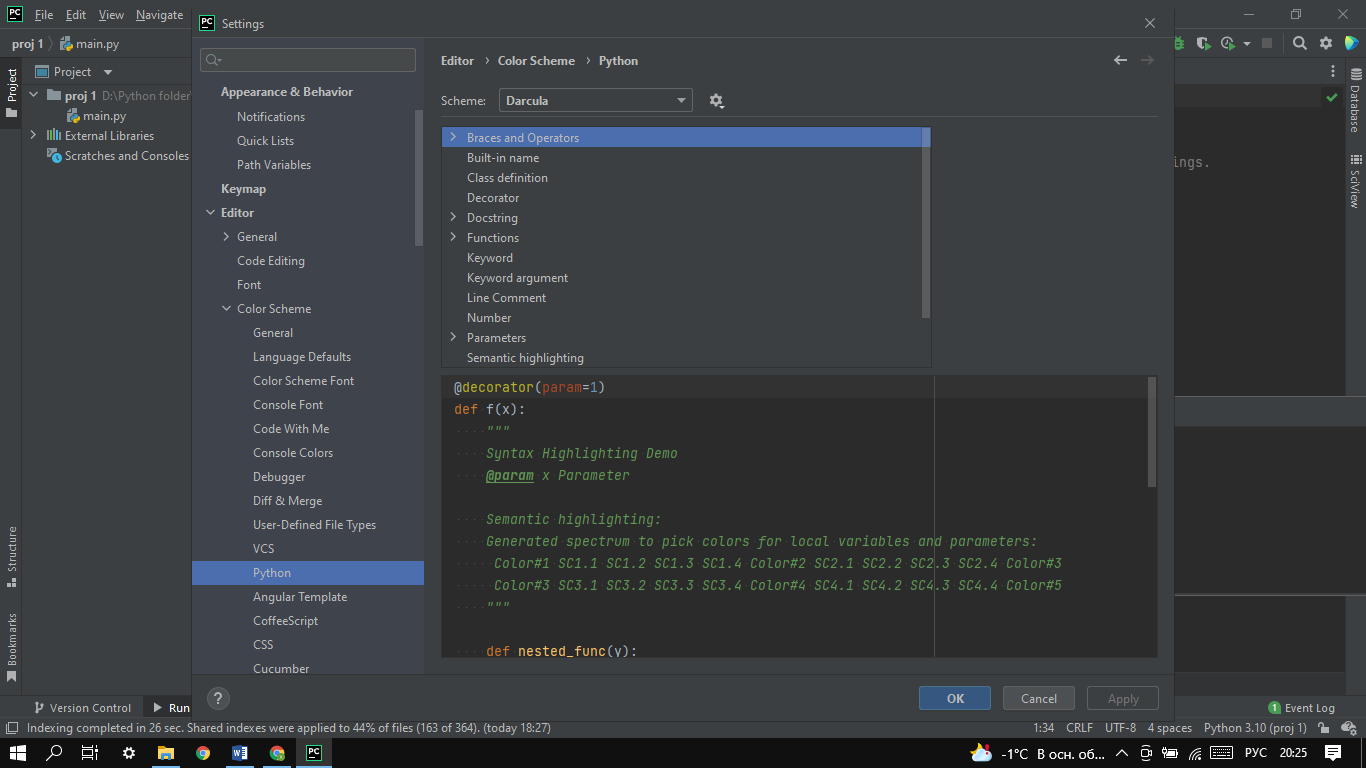
*(Рис. 3.2) Settings...*



*(Рис. 3.3) Font*

Подсветка:

***File >> Settings... >> Editor >> Color Scheme >> Python*** (Рис. 3.4)

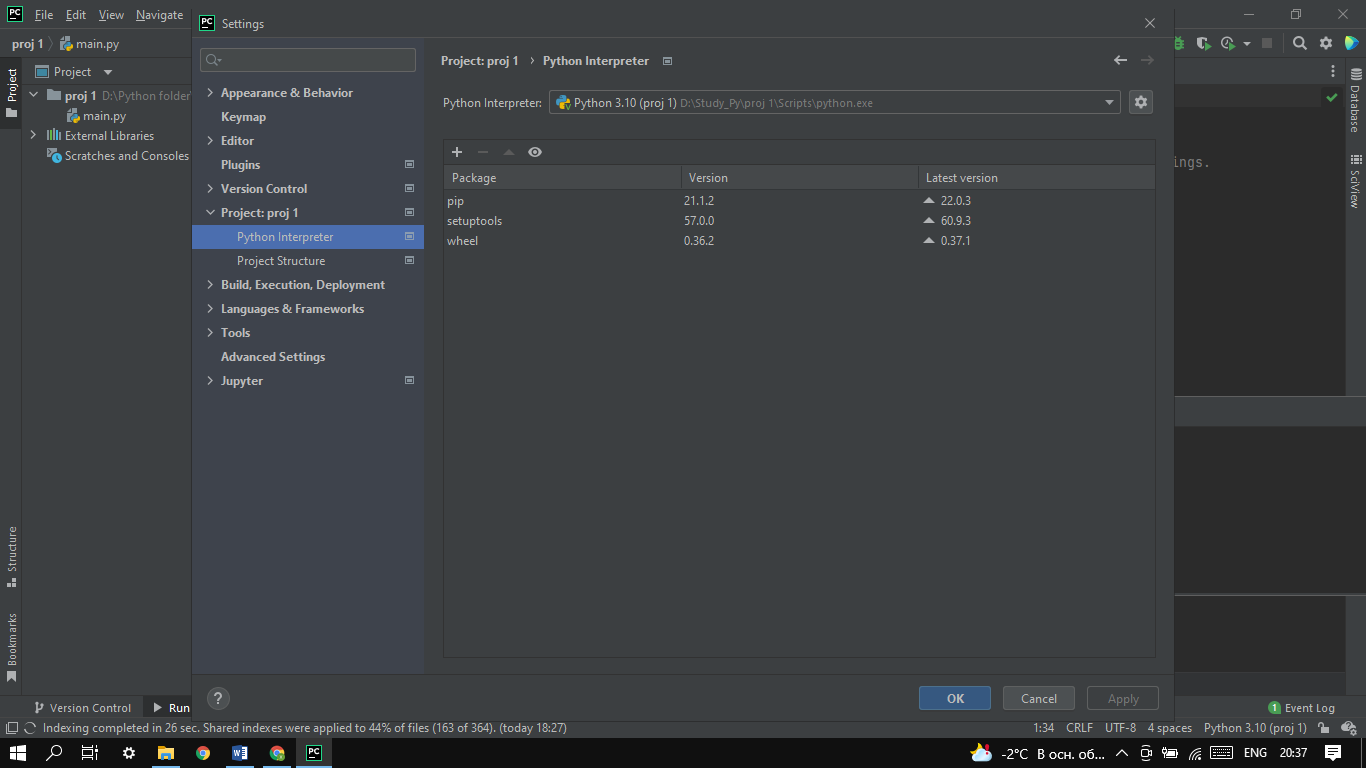


*(Рис. 3.4) Color Scheme*

### Интерпретатор и установка библиотек

Изменить ***интерпретатор*** и установить ***библиотеки*** очень просто.

Заходим снова в настройки и находим раздел ***Project: <название проекта>***, затем выбираем ***Python Interpreter*** (Рис. 3.4).

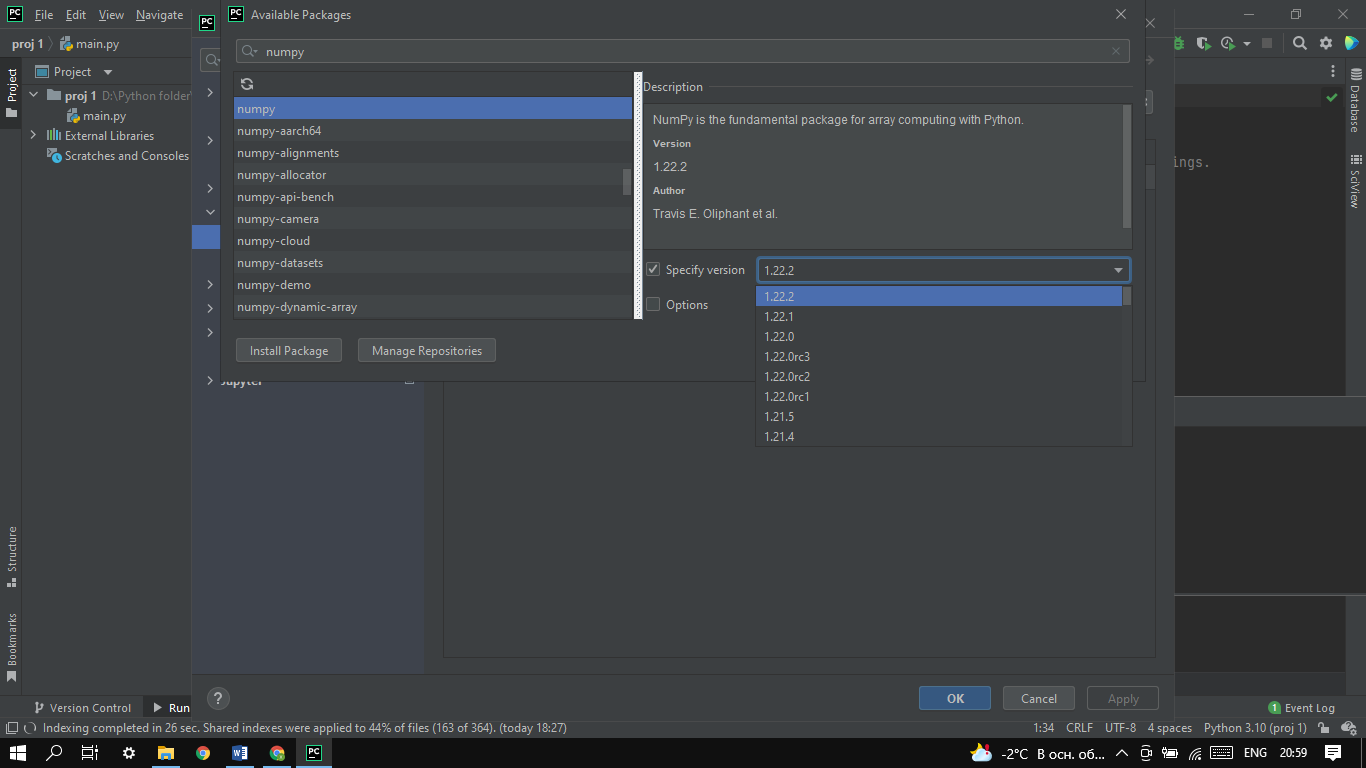


*(Рис. 3.4) Interpreter*

Для выбора ***интерпретатора*** нажимаем на ***шестеренку*** и находим к нему путь.

***Библиотека*** - это просто набор модулей и функций, которые облегчают некоторые специфические операции.

Для установки ***библиотек*** находим значек ***плюс***. Находим нужную нам ***библиотеку*** и устанавливаем (Рис. 3.5).



*(Рис. 3.5)*

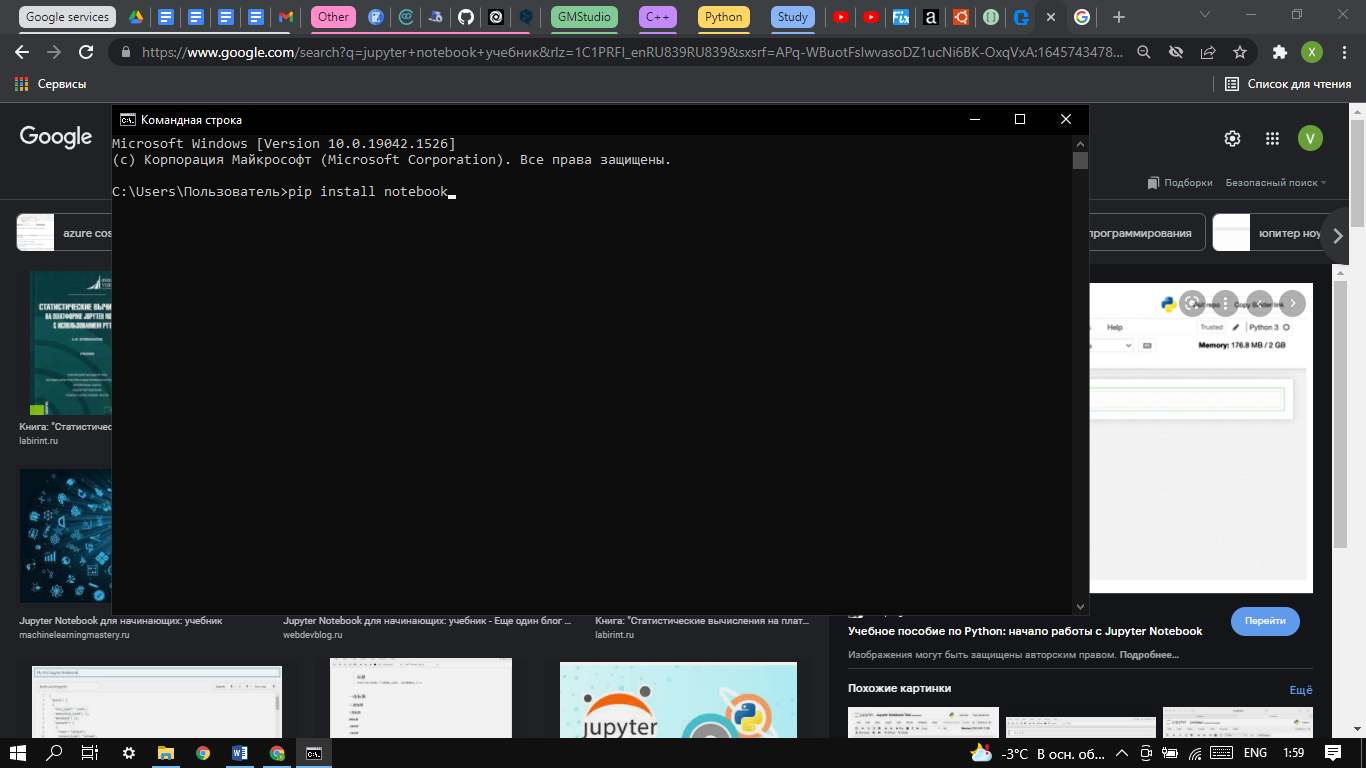
Нам предоставлют некоторую информацию о ***библиотеке*** и возможность выбрать версию.

# **Jupyter notebook**

***Jupyter-notebook - это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. Отличие от традиционной среды разработки в том, что код можно разбить на куски и выполнять их в произвольном порядке.***

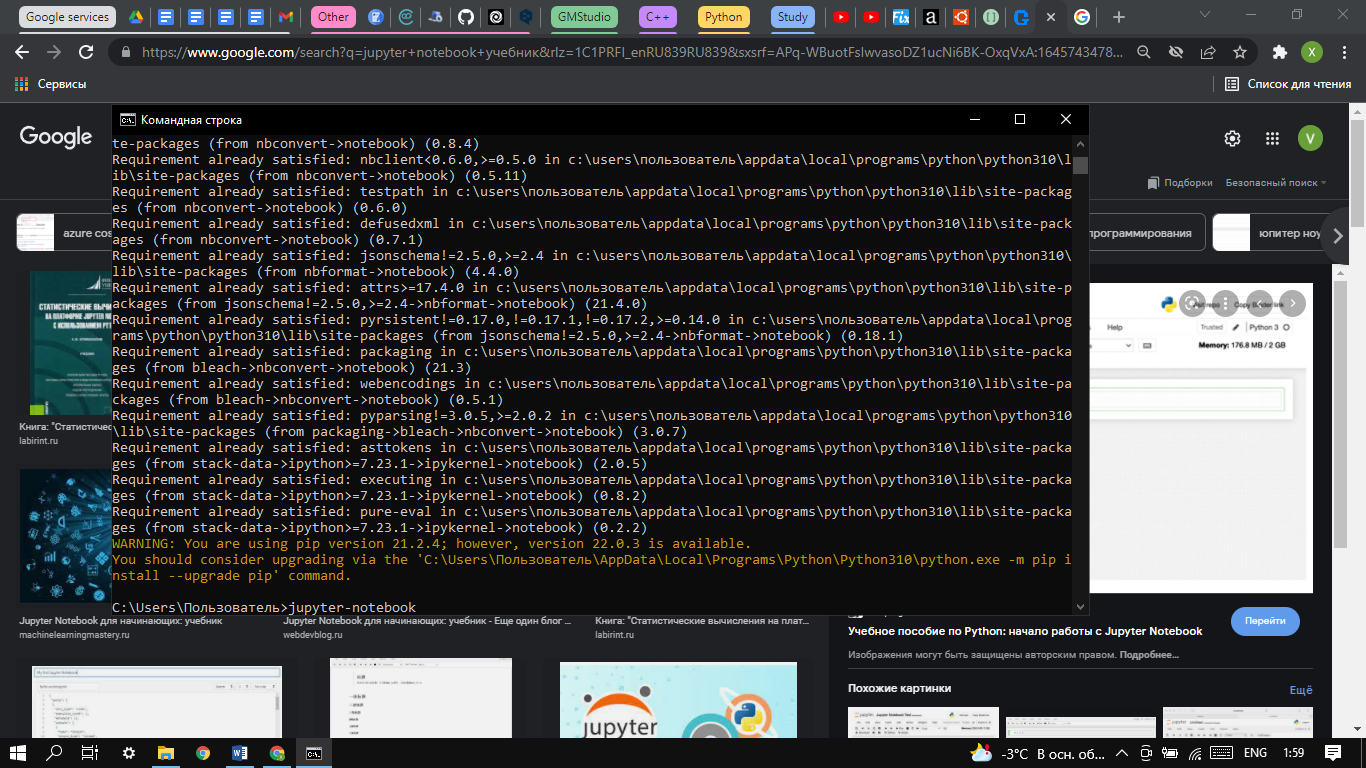
## «Hello, Jupyter!»

Чтобы войти в ***Jupyter***, для начала установим его через консоль (Рис. 4.1).

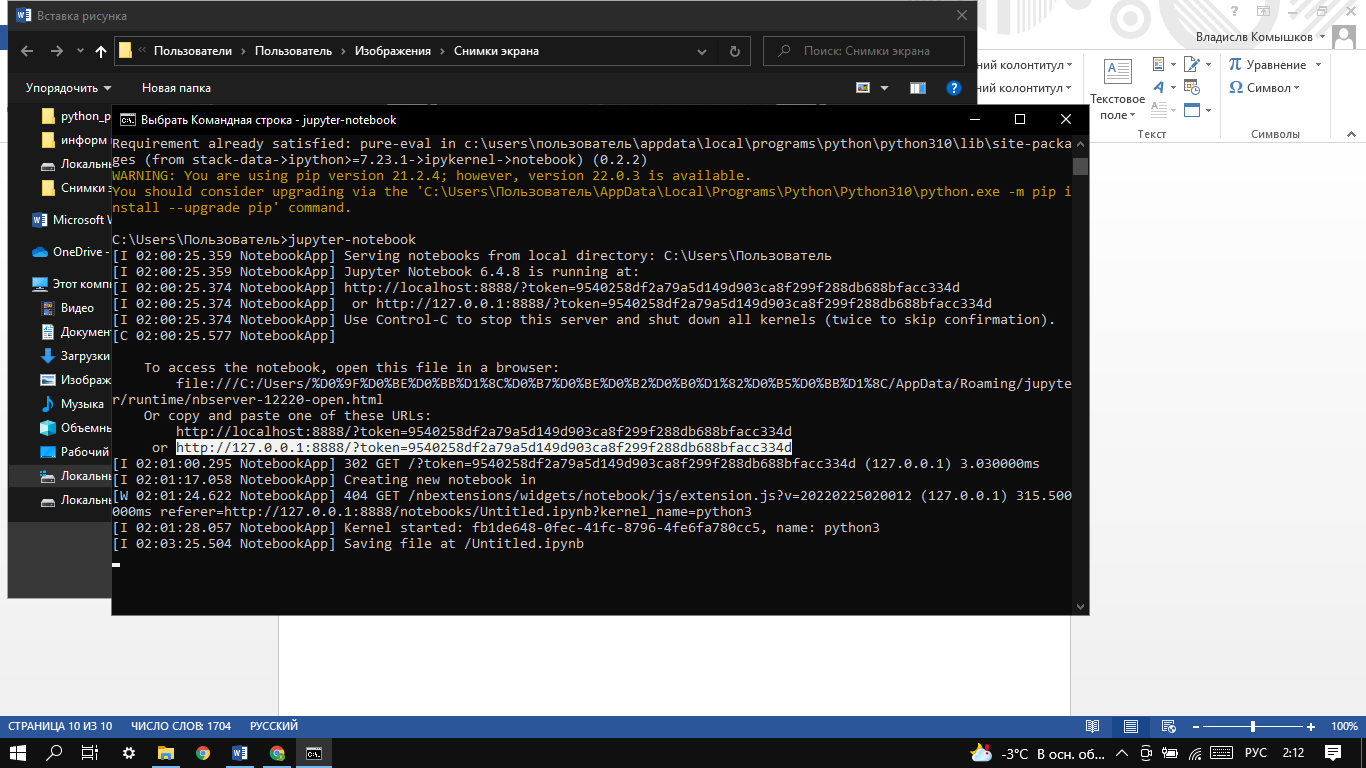


*(Рис. 4.1)*

Загрузка завершена. Теперь нам нужно получить ссылку, а затем вбить ее в браузер (Рис. 4.2 - 4.4). Пишем в командной строке ***“jupiter-notebook”***. И ждем пока наша ссылка появится.



*(Рис. 4.2) “jupiter-notebook”*



*(Рис. 4.3)*

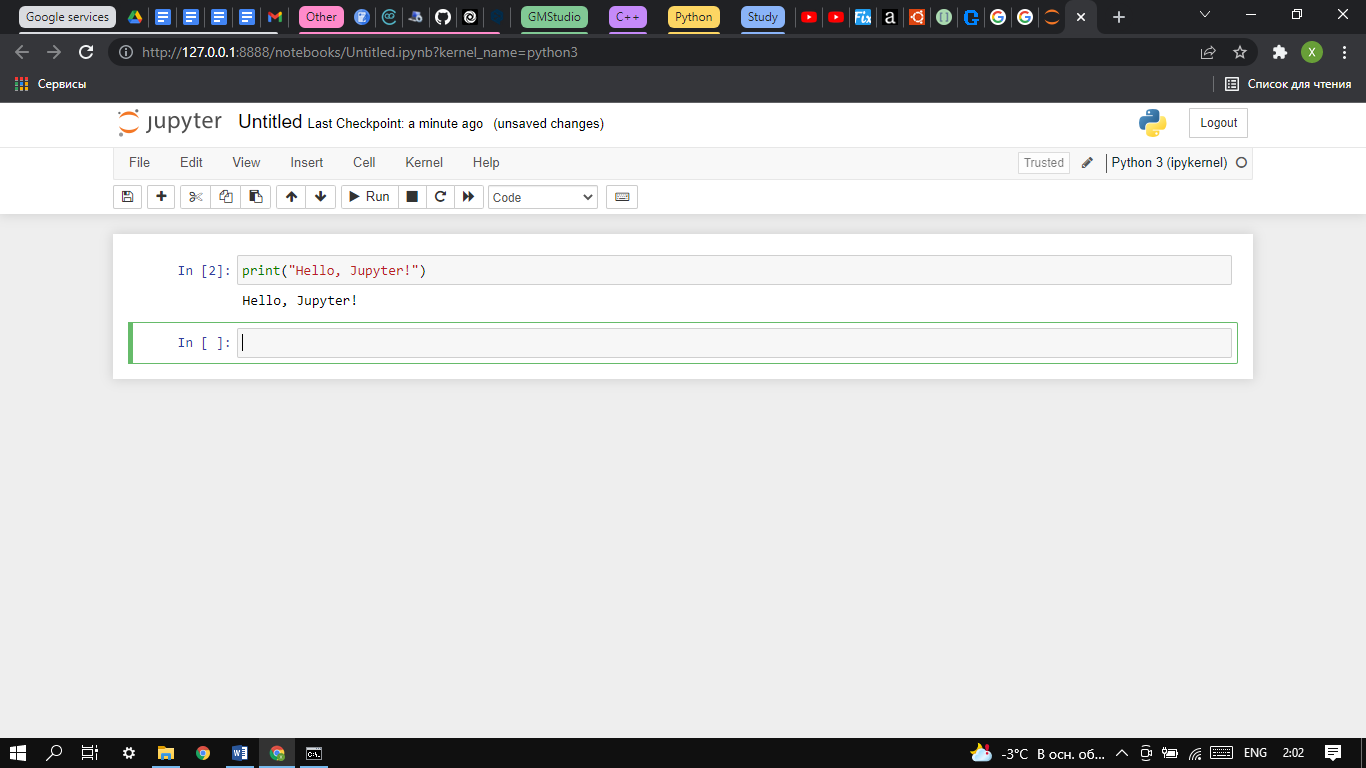


*(Рис. 4.3)*

Мы попадаем в наше виртуальное пространоство. Здесь находятся наши папки с проектами (Рис. 4.4).

Нажимаем ***New >> Python 3***

В новой вкладке появляется наш новый проект. Пишем ***«Hello, Jupyter!»*** и нажимаем кнопку ***Run*** (Рис. 4.4). Готово!



*(Рис. 4.4) «Hello, Jupyter!»*