2 Введение в Python

Разработка Python началась в конце 1980-х годов голландцем Гвидо ван Россумом, и первый релиз системы вышел в 1991 году. Python - кроссплатформенный язык программирования, способный работать на аппаратных платформах под управлением Windows, Mae OS, Linux. Разработчики языка отмечают, что Python может быть использован на компьютерах под управлением 21 операционной системы.

Python - объектно-ориентированный язык, но если понимание основ объектно-ориентированного программирования (ООП) в C#, Java затруднительно для начинающего программиста, то реализация ООП в Python элегантна и проста. Немаловажным преимуществом для человека, решившего начать путь в программирование именно с языка Python, является то обстоятельство, что его интерпретатор распространяется бесплатно и доступен для скачивания на сайте по адресу https://www.python.org/.

2.1 Процесс создания проекта в Python

Язык Python - активно развивающийся язык, и несколько раз в год появляются его новые версии, при этом основные возможности языка не зависят от нового релиза программы.

Рассмотрим процесс установки среды программирования и создания первого проекта. Перейдем на web-ресурс www.python.org, при этом откроется окно, представленное на рисунке 4.

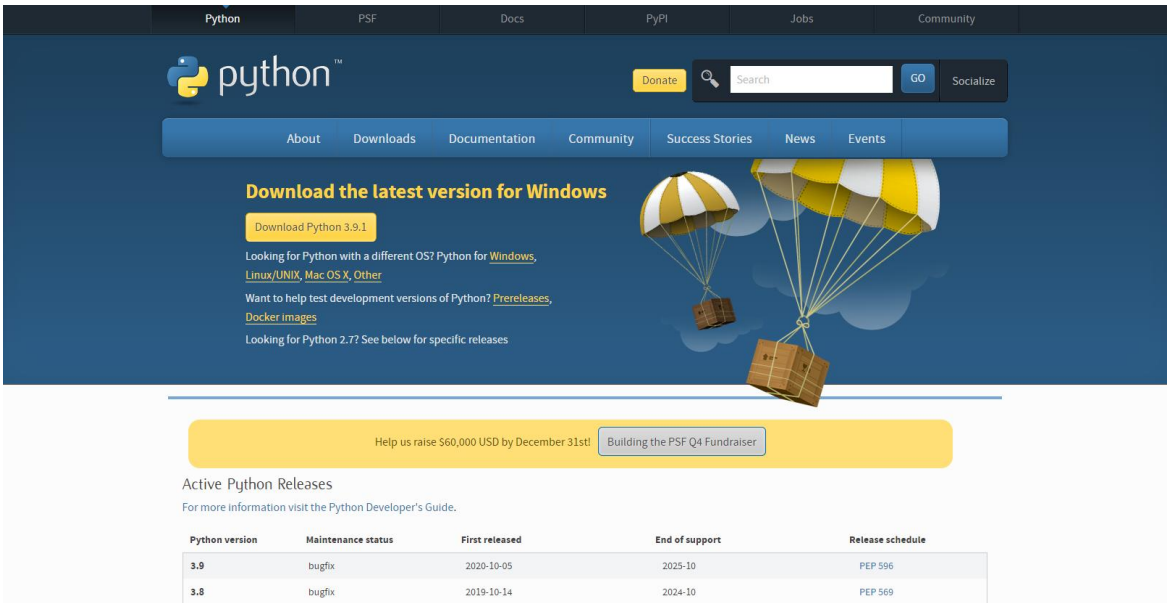
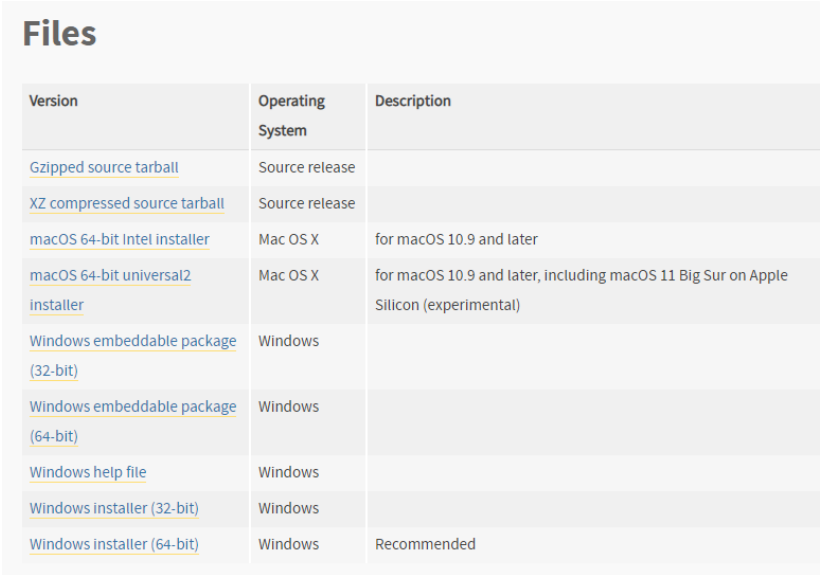


Рисунок 4 – Главная страница сайта www.python.org

Щелкнув по ссылке Downloads, вы сможете не только осуществить выбор операционной системы, под управлением которой работает ваш компьютер, но и версию языка Python (рисунок 5).

Рисунок 5 – Страница выбора версии языка Python

Интерпретатор языка Python работает во многих операционных системах, является кроссплатформенным. Выбор инсталляторов для различных платформ представлен на рисунке 6.



Files

Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)
Windows embeddable package (32-bit)	Windows	
Windows embeddable package (64-bit)	Windows	
Windows help file	Windows	
Windows installer (32-bit)	Windows	
Windows installer (64-bit)	Windows	Recommended

Рисунок 6 – Список инсталляторов Python для различных платформ

В операционную систему Linux интерпретатор Python встроен по умолчанию. Он может быть обновлён из встроенных депозитариев операционной системы.

Адаптация Python под ОС Android называется QPython и может быть скачена с PlayMarket. Для IOS интерпретатор Python можно скачать с AppStore.

В операционной системе Windows, щелкнув на скачанном файле python-3.x.x.exe, вам придется подождать несколько минут, пока не произойдет инсталляция Python на ваш компьютер (рисунок 7).

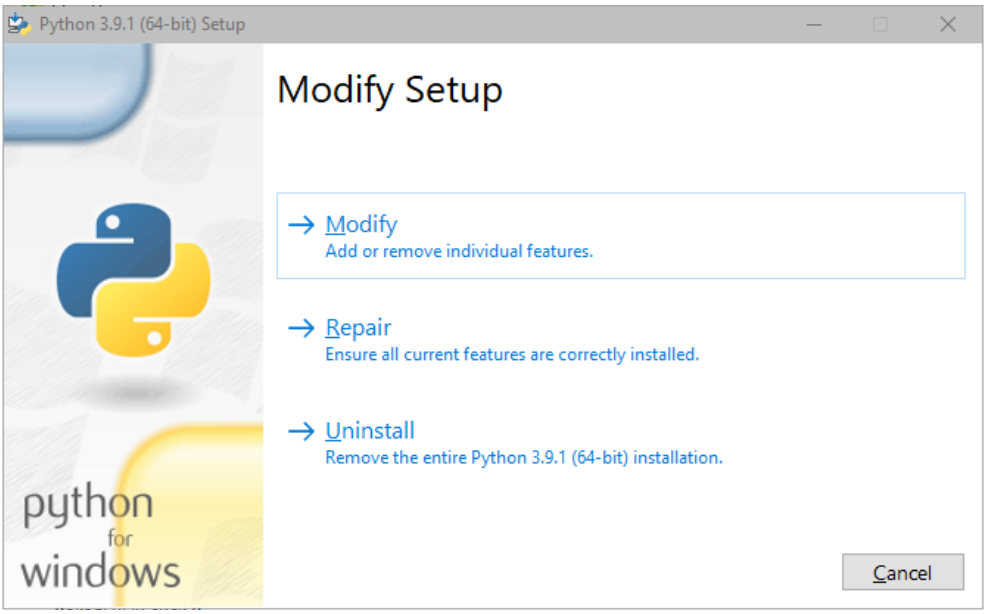


Рисунок 7 – Установка Python

Затем выполнив команду **Пуск/Python 3.x,** следует выполнить щелчок на ярлыке **IDLE (Python 3.x 32-bit),** что, и показано на рисунке 8.

Рисунок 8 – Ярлыки установленной программы

Откроется окно, представленное на рисунке 9.

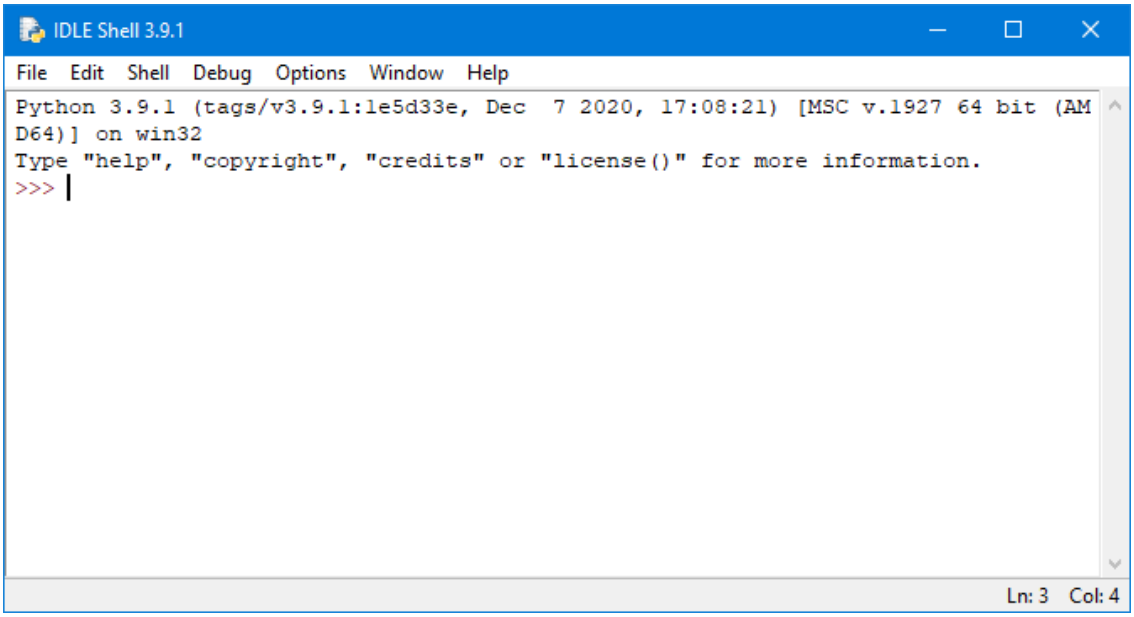


Рисунок 9 – Окно среды IDLE

IDLE (Integrated Development Environment) - это интегрированная среда разработки на языке Python, с ее помощью мы будем просматривать, редактировать, запускать и отлаживать программы написанные на Python. IDLE создана с помощью графической библиотеки под названием **Tkinter** (англ. Tk interface), которая широко распространена в операционных системах Linux и UNIX-подобных системах. IDLE является активным интерпретатором (программа-переводчик с языка высокого уровня в машинный код), поддерживающим функции многооконного редактора с функцией отмены, подсветкой синтаксиса, отладчика, и написана на Python.

Пункты меню оболочки - стандартные для подобных программ, поддерживающие стандартные функции сохранения, открытия, редактирования и отладки созданных программ, поэтому мы не будем приводить их отдельное описание, а познакомимся с ними непосредственно в ходе работы с IDLE.

Напишем нашу первую программу на языке Python, а именно - программу, которая выводит сообщение **"Привет, мир!",** представляющее собой неформальное приветствие другим людям от человека, делающего первый шаг в программировании на любом языке.

Ее мы напишем, используя оператор **print** и синтаксис данного оператора. **Оператор** (инструкция) в программировании - это действие, которое выполняет компьютер в ответ на запуск программы на выполнение. Каждый оператор имеет свой **синтаксис,** т. е. правила записи, согласно которым данный оператор может быть выполнен в той или иной среде программирования.

Оператор **print** имеет следующий синтаксис:

print("Сообщение").

Следует отметить, что оператор **print** необходимо записывать строчными буквами, поскольку язык Python чувствителен к регистру. Таким образом, оператор **Print** или **PRINT** - неправильно записанные операторы.

Итак, вы записываете оператор **print("IIривет, мир!**") сразу после так называемого приглашения, которое в IDLE выглядит как тройное перечисление символа >, то есть >>>. Для того чтобы увидеть результат, вам следует нажать клавишу **Enter.** Результатом инструкции, естественно, станет вывод на экран сообщения "Привет, мир!" (рисунок 10). Первый шаг в программирование на языке Python сделан.

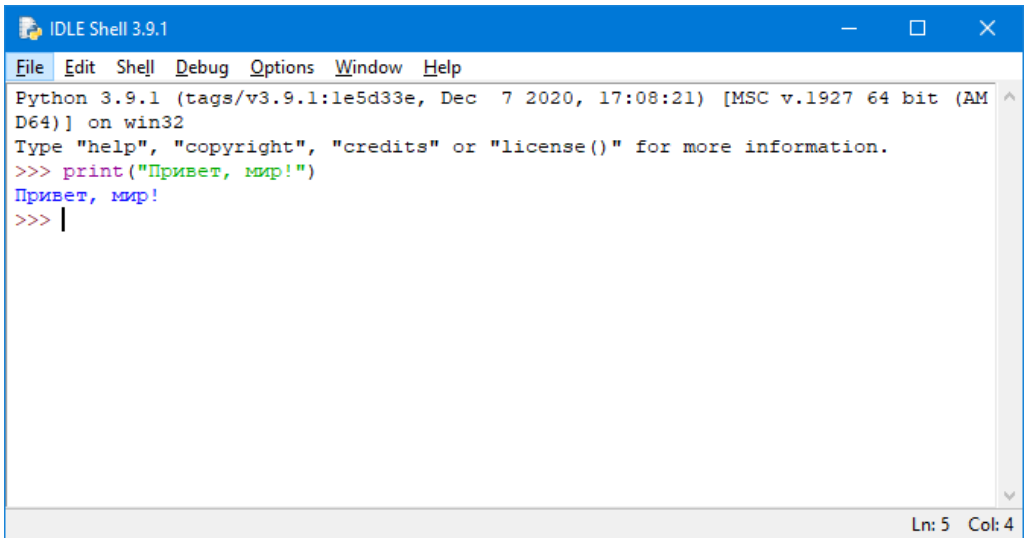


Рисунок 10 – Результат первой написанной программы

Сохраним нашу первую программу. Для этого выполним команду **File/Save.** Откроется окно, представленное на рисунке 11. Для корректного сохранения своих программ лучше всего создать отдельную папку, в данном случае она имеет название **Мои проекты.** По умолчанию, если вы работаете в операционной системе Windows, программа Python будет установлена по следующему пути: **C:\Users*Имя пользователя на компьютере*\AppData\Local\Programs\Python\Python39\.**

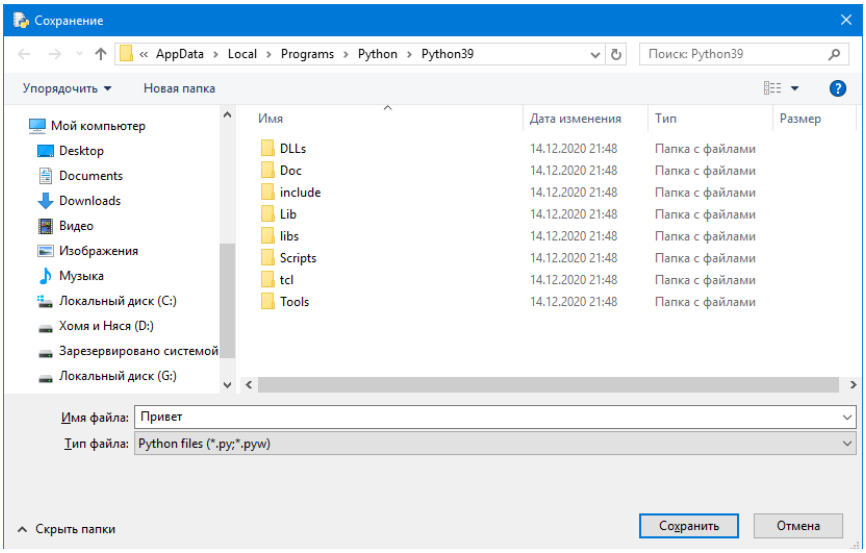


Рисунок 11 – Окно «Сохранить как»

Таким образом, если в дальнейшем вы хотите найти свои программы, непосредственно указывая к ним путь, вы будете следовать по вышеуказанному адресу.

Теперь рассмотрим другой способ создания наших программ в IDLE Python. Первый способ позволил сразу увидеть результаты написанной нами программы. Однако такой интерактивный режим, при котором в ответ на запись оператора пользователем происходит мгновенная реакция системы, будет совсем не удобен при разработке более серьезных программ, поскольку практически любая написанная в будущем программа, потребует выполнения такого этапа, как ее отладка. Следовательно, было бы очень удобно сначала написать предполагаемые действия в виде операторов программы, а затем написанный текст «запустить», как принято выражаться среди программистов, на выполнение.

Для этого, открыв ранее описанным способом IDLE, следует выполнить команду File/New. Откроется окно, представленное на рисунке 12.

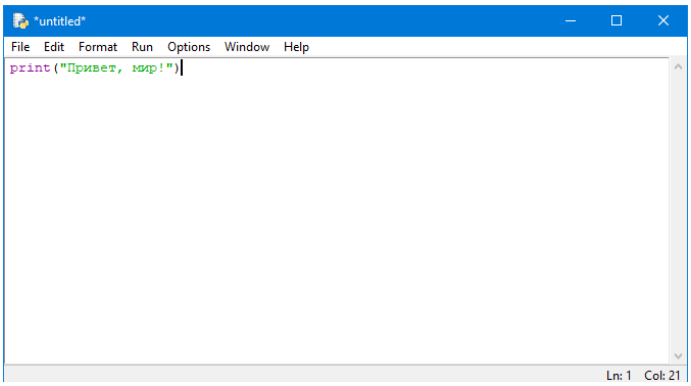


Рисунок 12 – Окно с программой

Именно в нем мы вновь запишем оператор print("Привет, мир!"). Замечание. Не делайте отступы от левой границы окна при записи оператора. С чем это связано, будет объяснено позже.

Для того чтобы выполнить нашу программу, придется выбрать пункт меню Run, а в нем пункт Run module. Однако после этого появится не результат выполнения программы, а просьба о сохранении проекта (рисунок 13). Заметим, что как только в дальнейшем вы сделаете любые изменения в вашей программе, даже если до этого она была сохранена, будет появляться окно с просьбой сохранить сделанные изменения и вам остается только нажать на кнопку Ok.

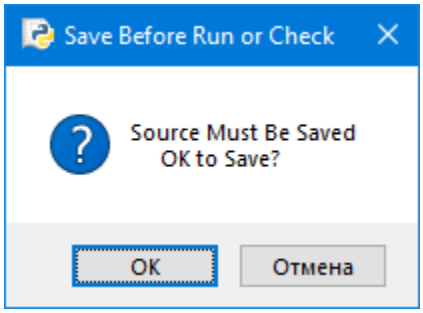


Рисунок 13 – Вопрос о сохранении программы

Нажав на кнопку Ok, следует указать путь к папке, которую вы создали ранее для своих будущих проектов. Поскольку вы уже сохраняли предыдущий проект, вы можете указать то же самое имя проекта, что и ранее. Появится окно информирующее о том что подобный проект уже существует и вам необходимо подтвердить свои действия. Теперь открывается среда (оболочка) Python, в которой вы увидите результаты программы, а именно, сообщение "Привет, мир!" (рисунок 14).

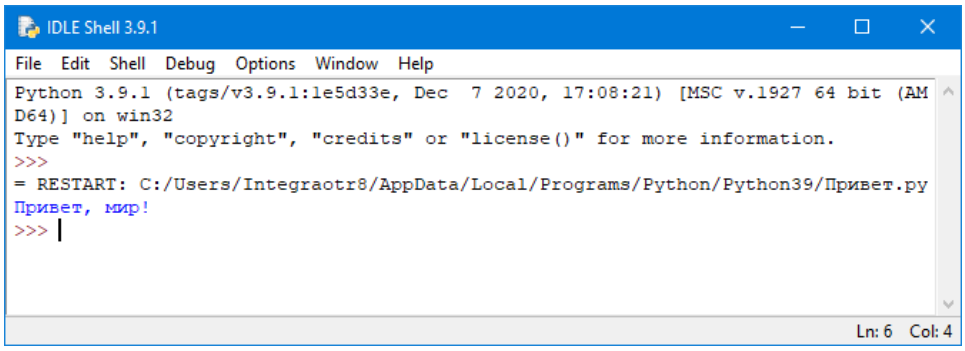


Рисунок 14 - Результат работы программы

Рассмотрим процесс открытия уже созданной программы. Выполним команду Пуск/Python 3.x и сделаем щелчок на ярлыке IDLE (Python 3.x 32-bit). В появившемся окне выполним команду **File/Open**, укажем путь к папке, в которой

будут храниться проекты, написанные на языке Python, и выберем файл созданной нами программы "Привет, мир!" Подобная ситуация представлена на рисунке 15. Перед нами вновь откроется окно так называемого сценарного режима, в котором мы будем писать программы на Python (рисунок 16). В нем вы увидите текст ранее созданной программы.

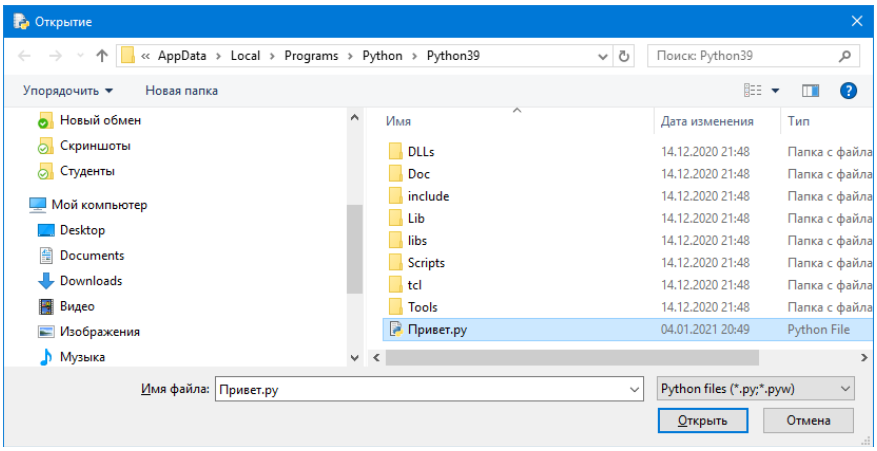


Рисунок 15 – Окно «Открыть»

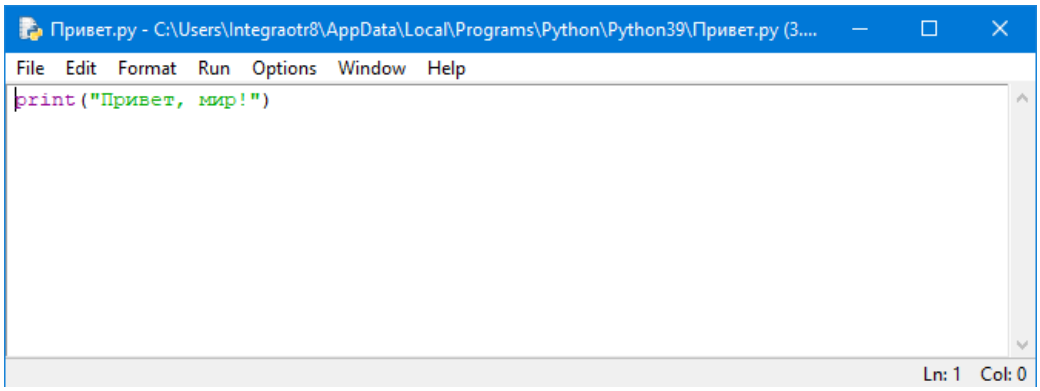


Рисунок 16 – Окно сценарного режима

Что делать дальше, вы уже знаете, однако напомним, что для запуска программы на выполнение следует выбрать пункт меню **Run**, а в нем пункт **Run module**. Согласно этой команде программа выполнится, и вы сможете увидеть ее результат.

2.2. Методы ввода и вывода данных и обработка исключений

Напишем более сложную программу, результатом выполнения которой станет сумма двух чисел, вводимых пользователем. На ее примере мы сможем рассмотреть новые операторы, позволяющие осуществить ввод данных и присваивание переменной какого-либо значения.

С действием, которое осуществляет оператор присваивания, мы познакомились ранее при чтении параграфа 1.6. Напомним, что в языке Python он имеет вид знака равенства (=).

Функция **input** будет использоваться в кодах программ для получения значений (ввода данных), которые будет вводить пользователь с клавиатуры. Она имеет следующий синтаксис:

имя переменной = **input**("Приглашение").

Такая запись очень похожа на синтаксис оператора **print.** Однако отличия все же имеются. Прежде всего, расскажем о самом понятии **«функция».** В ходе изучения языка программирования Python мы должны в совершенстве овладеть программированием на основе использования функций, но сейчас, в качестве небольшого знакомства, следует отметить: никого не удивляет тот факт, что при вычислении функции **sin(x)** пользователь, работая, например, с программой Microsoft Excel, выбирает из категории **Формулы/Вставка/Математика и тригонометрия** функцию **SIN** и начинает с ней работать, вычисляя значение этой функции от какого-то аргумента.

Между тем, функция SIN - это сложный знакопеременный математический ряд, который может быть вычислен по формуле:

$$sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \cdots$$

Для вычисления суммы такого ряда в среде программирования необходимо написать достаточно сложную программу. Безусловно, легче вызвать уже написанную программистами функцию SIN, которая размещена в библиотеке математических подпрограмм, и работать с ней, используя ее имя - SIN.

Однако следует уяснить, что для получения результата от ее использования в среде программирования вы должны не просто указать ее имя sin и значение аргумента в скобках (sin(x)), но слева от имени функции указать имя переменной, а также оператор присваивания, т. е., вызов функции будет осуществлен в виде оператора y=sin(x).

Теперь надо запомнить, что понятие «функция» относится не только к категории математических действий, но и распространяется на другие. Когда мы будем использовать функцию **input** для того, чтобы разместить в ячейке оперативной памяти какое-либо значение, полученное от пользователя (в этом, напомним, заключается действие оператора присваивания), слева от этого оператора, мы должны будем указать имя переменной.

Учитывая вышесказанное, синтаксис функции **input**, записываемой как *имя переменной* = **input**(«Приглашение»), не должен вызывать какие либо вопросы.

В редакторе сценариев напишем программный код, приведённый на рисунке 17.

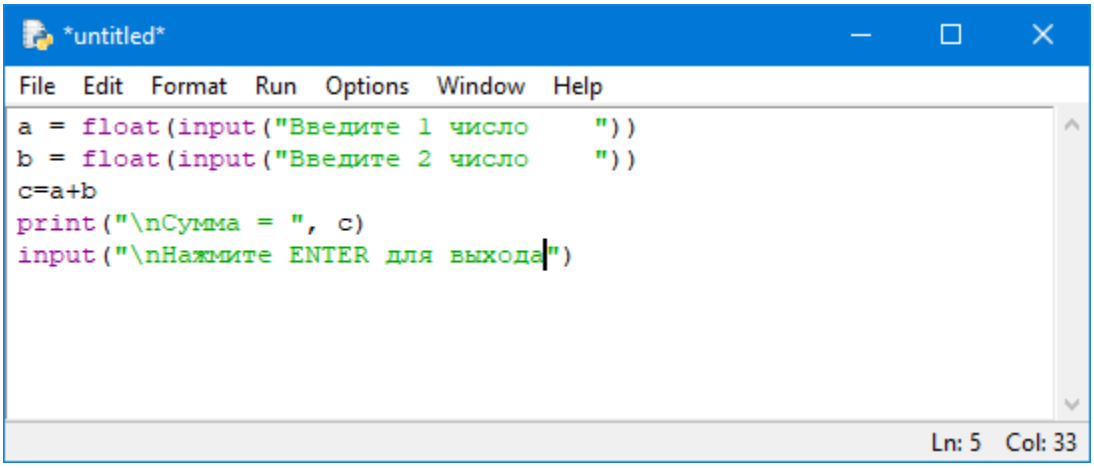


Рисунок 17 – Текст программы «Сложить два числа»

Вывод результата в программе производит оператор **print**, синтаксис которого несколько иной, чем в предыдущем случае вывода на экран просто строки, а именно,

print("Приглашение", идентификатор), где

1. *Приглашение* - строка, содержащая информацию о характере вывода;
2. *идентификатор* - имя переменной для вывода результата на печать.

Можно еще раз отметить, что Escape-последовательность \n, содержащаяся в двух последних строках программы, дает указание интерпретатору перевести курсор на новую строку.

В коде программы **Сложить два числа** в может быть непонятно на этом этапе только применение функции **float** и то, что, несмотря на вышеприведенные рассуждения, функция **input** вызывается без упоминания имени переменной слева от оператора присваивания, который тоже отсутствует в строке программного кода **input("\nНажмите ENTER для выхода")**.

При объяснении типов данных, используемых в языке Python (см. параграф 1.7), упоминался вещественный тип данных (**float**), используемый, когда программа должна работать с дробными числами. Так вот, функция **input** возвращает строковый тип данных. То есть то число, которое будет принято от пользователя, будет восприниматься компьютером не как число, а как строковое значение, а следовательно, никакие математические операции над строками выполнить мы не сможем. Таким образом, функция **float** играет роль функции приведения одного типа данных к другому (в данном случае конвертирует строковый тип в вещественный).

Оператор **input("\nНажмите ENTER для выхода")**, в целом, необязателен. Если его не будет в программе, то, конечно, не изменится ее результат с точки зрения математики, и, кроме того, после получения результата и нажатия ENTER, как такого выхода из программы не произойдет. Осуществится лишь переход к приглашению >>>. Тем не менее эта строка говорит пользователю, что его

программа завершена и надо принять какое-то решение: либо осуществить выход из среды программирования, либо возобновить действия с программой.

К тому же использование функции **input** в данном операторе не приводит к получению числового значения от пользователя, а только лишь к считыванию внутреннего кода клавиши ENTER, который непосредственно не может изменить результат математического действия сложения чисел. Следовательно, возможен и другой синтаксис функции **input**, который и продемонстрирован в примере. Результат ввода данных и получения результата представлен на рис. 18.

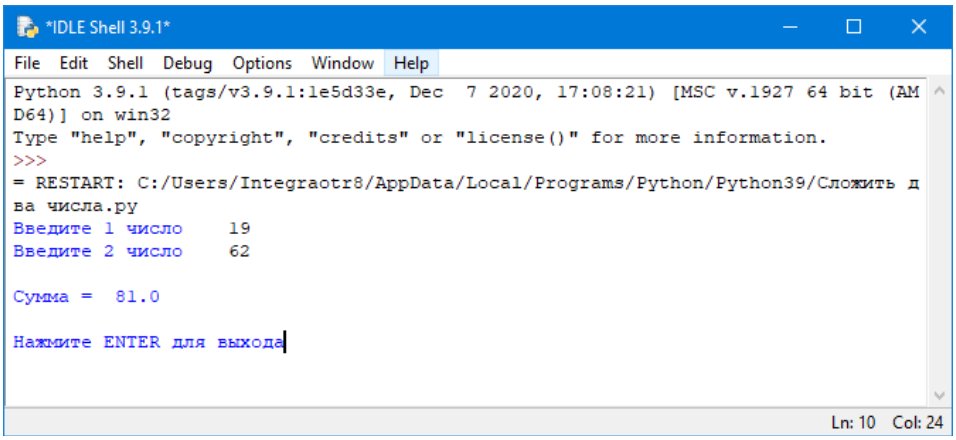


Рисунок 18 – Результат работы программы

Прерывания, которые далее будут рассматриваться в данном параграфе, относятся к классу внутренних прерываний и называются **исключениями** (exceptions). Они происходят синхронно выполнению программы и возникают при появлении аварийной ситуации в ходе исполнения некоторой инструкции. Примерами исключений являются деление на нуль, переполнение, обращение к несуществующему файлу и т. д.

Обработчик ошибок в Python использует блок **try...except...finally.** Блок **try...except** должен окружать ту часть кода, где может возникнуть исключительная ситуация. Блок **finally** всегда исполняется, поэтому в него помещают те инструкции, которые должны выполниться независимо от того, произошло ли исключение.

Программа может прервать свою работу по разным причинам, поэтому типов исключений существует довольно много. В Python3 иерархия исключений выглядит, так как на рисунке 19.

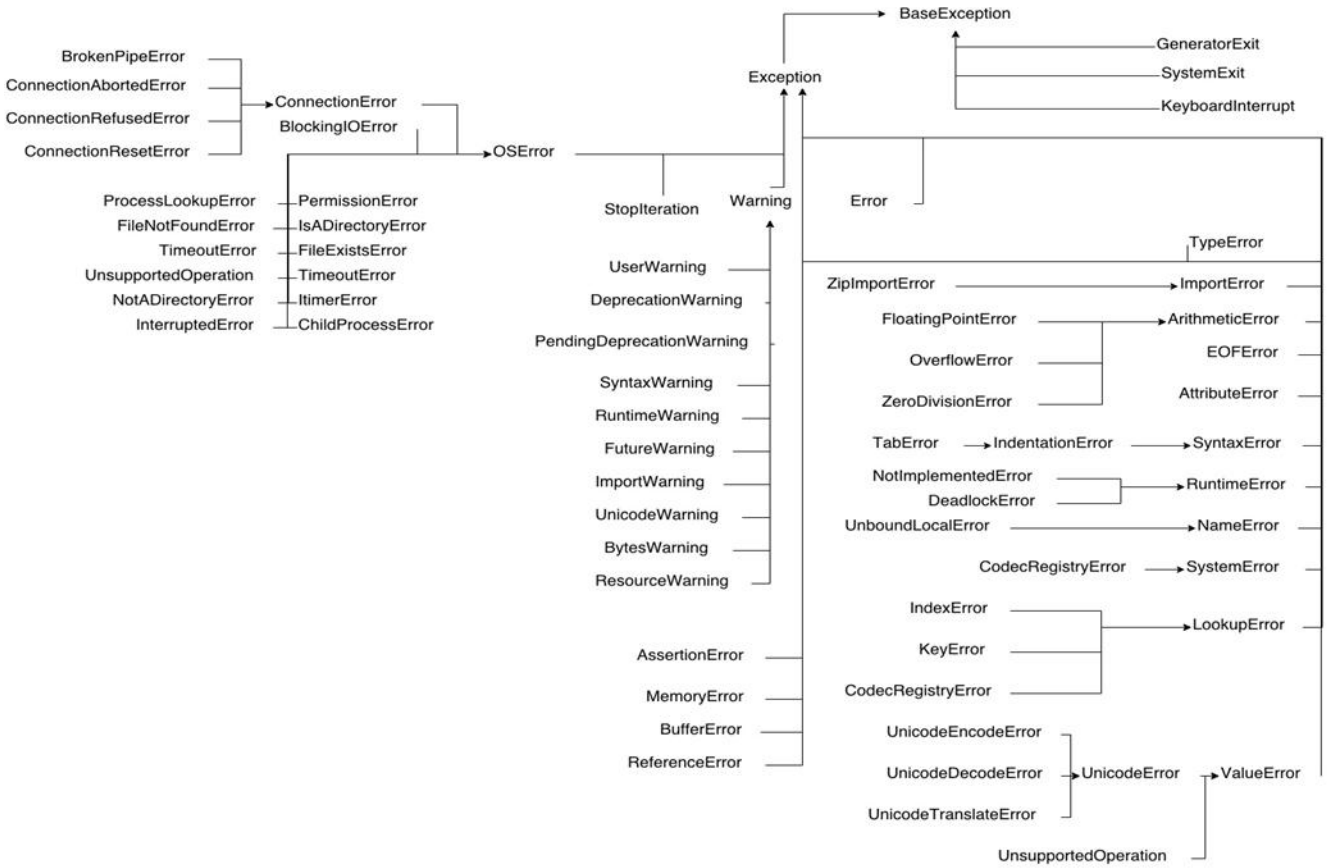


Рисунок 19 – Иерархия исключений в Python3

В таблице 5 приведем наиболее распространенные среди них исключения.

Таблица 5 – Типы исключений

Тип исключений	Описание
IOError	Возникает при появлении ошибок, связанных с операциями ввода/вывода
IndexError	Возникает, если в последовательности не найден элемент с указанным индексом
NameError	Возникает, если не найдено имя переменной, имя функции
SyntaxError	Возникает в случае синтаксической ошибки в программе
TypeError	Возникает, если стандартная операция применяется к объекту несоответствующего типа
ValueError	Возникает, если стандартная операция применяется к объекту соответствующего типа, но с неподходящим значением
ZeroDivisionError	Возникает в случае выполнения операции деления на нуль

Приведем пример использования конструкции **try...except...finally** и напишем программу для получения частного от деления ДВУХ чисел. Её код приведён в листинге ниже:

```
a=float(input("Введите 1 число  "))
b=float(input("Введите 2 число  "))
try:
    c=a/b
    print("\n Частное от деления = ", c)
except ZeroDivisionError:
    print("Вы делите на нуль!")
finally:
    print("Давайте запустим программу еще раз или \n Нажмите ENTER для выхода")
```

Обратите внимание на отступы в данном коде. Дело в том, что в отличие от других популярных языков программирования, таких как C#, Microsoft Visual Basic, Pascal-ориентированных языков, где отступы в блоках кода советуют делать для того, чтобы программу было легче читать и отлаживать, но сам принцип их простановки необязателен, в Python **отступы** являются частью синтаксиса программных конструкций. Для того чтобы сделать отступ в строке кода, достаточно нажать клавишу **Tab** или нажать четыре раза клавишу **Пробел.**

Базовой программной конструкцией в вышеприведенном коде является конструкция **try...except...finally,** поэтому стоит не сделать отступ хотя бы в одной программной строке этого блока, так, например, как это показано на рисунке 20 (не сделан отступ в операторе c=a/b), и программа перестанет работать, выдавая ошибку, показанную на рисунке 21.

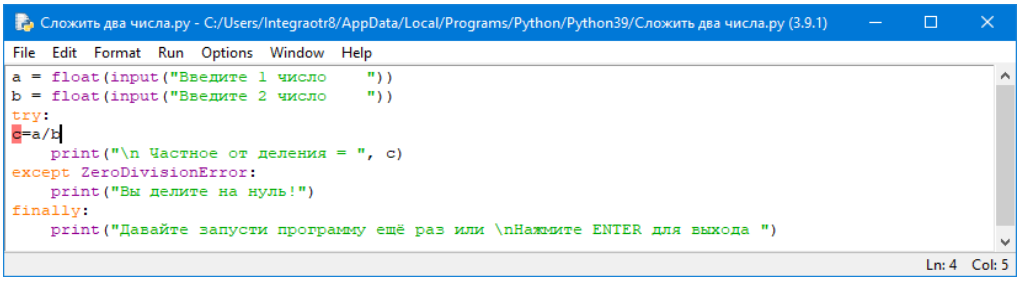


Рисунок 20 – Не сделан отступ перед операторами c=a/b

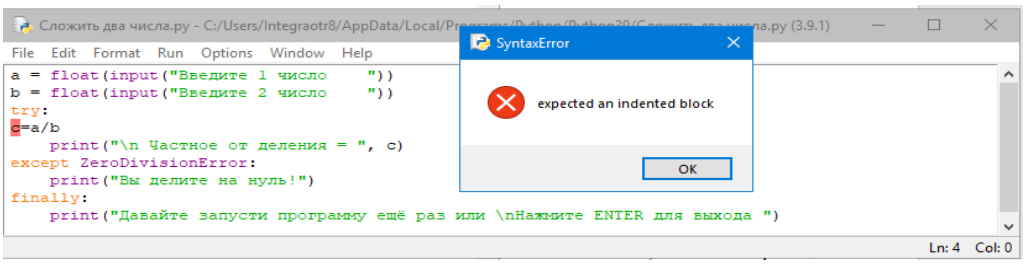


Рисунок 21 – Ошибка получения из – за отсутствия отступа

Также стоит обратить внимание на то, что если сделаете отступ в строке программного кода, которая не относится к блоку операторов, например, в **a=float(input("Введите 1 число "))**, то программа также перестает функционировать и интерпретатор выдаст ошибку.

Значит, если не соблюдать такие правила, начинающему будет очень трудно без посторонней помощи разобраться в сложившейся ситуации. В дальнейшем мы будем давать специальные комментарии по поводу необходимости делать отступы в тех или иных программных конструкциях.

Вернемся к комментарию кода программы. Мы окружили код, где возможно возникновение ошибки деления на нуль, блоком **try...except...finally,** при этом использовали ожидаемый тип исключения **ZeroDivisionError** (см. табл. 5). Приведенный блок перехватывает конкретную ошибку переполнения, возникающую вследствие деления на нуль.

Предположим, мы введем число 5, а второе число - равное 0. При выполнении этого кода появится сообщение: «Вы делите на нуль!». Результат работы программы показан на рисунке 22.

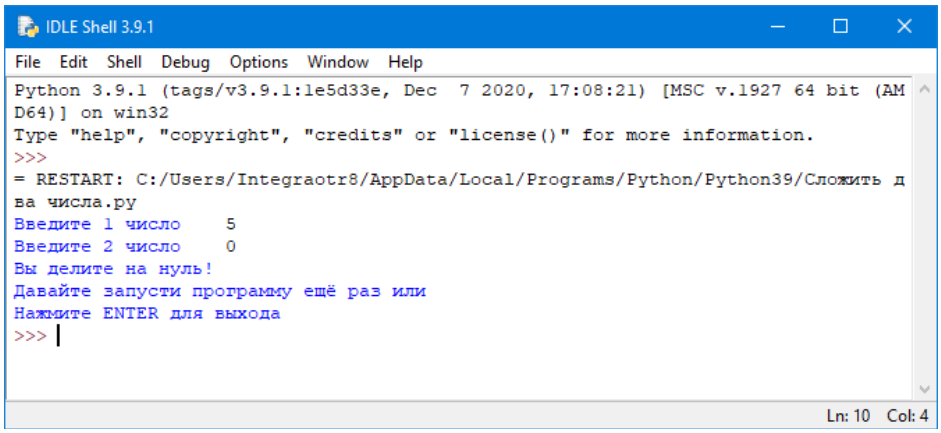


Рисунок 22 – Программа выдает сообщение «Вы делите на нуль!»

Попробуем усовершенствовать программу. Дело в том, что наша программа не совсем корректна: ведь пользователь по ошибке может вместо чисел ввести обычные символы, расположенные на клавиатуре. Нижеприведенный код, представленный в листинге, перехватит другую ошибку, связанную как раз с неверным форматом ввода. В нем, во-первых, оператор **try** разместим в том месте, где возможно возникновение подобной ошибки - это инструкции ввода данных. Во-вторых, воспользуемся тем, что обработка нескольких исключений может быть перехвачена с помощью нескольких вложений конструкции **except,** и включим в наш код исключение **ValueError** (см. табл. 5), которое возникает, если стандартная операция применяется к объекту соответствующего типа, но с неподходящим значением. Таким образом, получим следующий код программы:

try:

```
    a=float(input("Введите 1 число  "))
    b=float(input("Введите 2 число  "))
    c=a/b
    print("\n Частное от деления = ", c)
except ZeroDivisionError:
    print("Вы делите на нуль!")
except ValueError:
    print("Вы ввели не числовое значение!")
finally:
    print("Давайте запустим программу еще раз")
    input("\n Нажмите ENTER для выхода")
```

Теперь программа надежно защищена от неправильного ввода. Предположим, значение первого числа пользователь вводит правильно 5.2, а при вводе второго числа пользователь ошибается и вводит в качестве разделителя не точку, а символ **ю** (поскольку и точка, и символ **ю** расположены на одной клавише). Однако такая ошибка не приведет к исключительной ситуации с точки зрения остановки программы. При ее выполнении появится сообщение «Вы ввели не числовое значение!» (рисунок 23).

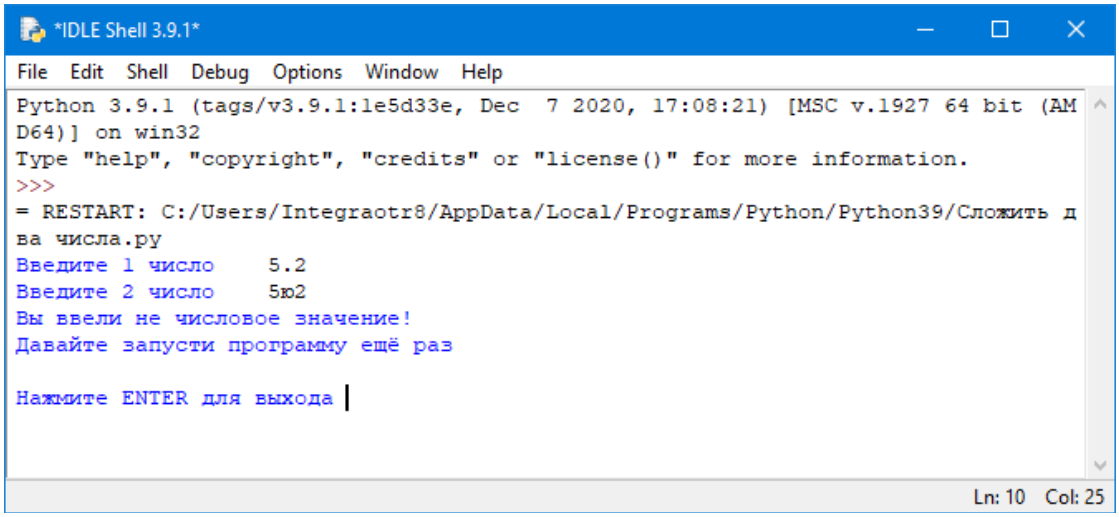


Рисунок 23 – Программа выводит сообщение «Вы ввели не числовое значение!»

Итак, обработка нескольких исключений может быть перехвачена с помощью нескольких вложений конструкции **except**. Такая ситуация была показана в предыдущем коде. Другой прием заключается в перечислении через запятую типов исключений - так, как это показано в следующей программе.

```
try:
```

```
    a=float(input("Введите 1 число  "))
    b=float(input("Введите 2 число  "))
    c=a/b
    print("\n Частное от деления = ", c)
except (ZeroDivisionError, ValueError):
    print("Вы делите на нуль или вы ввели не числовое значение!")
finally:
    print("Давайте запустим программу еще раз")
    input("\n Нажмите ENTER для выхода")
```

Результат работы программы будет аналогичен представленному на рисунке 23, однако код программы стал короче.

2.3 Контрольные вопросы

1. Расскажите о назначении IDLE. С какими способами создания программ в IDLE вы познакомились?
2. Какие операторы ввода и вывода данных используются для приложений, разрабатываемых на языке Python? Напишите синтаксис используемых операторов.
3. В каких случаях при разработке концепции глобальной обработки ошибок применяется конструкция try...except...finally? Поясните работу обработчиков исключений на примерах.
4. Назовите основные типы исключений и укажите причины их возникновения.
5. Какова роль отступов в программах, написанных на языке Python?