Python – это высокоуровневый язык программирования, который широко применяется в веб-разработке, сценариях, научных вычислениях, программах искусственного интеллекта и других областях.

Язык очень популярен и используется такими организациями, как Google, NASA, CIA и Disney.

Код Python обрабатывается программой-интерпретатором во время выполнения. Нет необходимости проводить компиляцию программы перед её выполнением.

Существуют три основные версии Python: 1.x, 2.x, 3.x. Они с свою очередь разделяются на подверсии, например, 2.7 или 3.3.

Изменения языка, совместимые с более ранними версиями вводятся, но только между основными версиями; код, написанный на Python 3.x, будет всегда совместим с будущими версиями.

В настоящее время используются Python версий 2.x и 3.x. Наиболее широко используется реализация CPython.

Интерпретатор – это программа, которая выполняет сценарии, написанные на интерпретируемых языках, таких как Python.

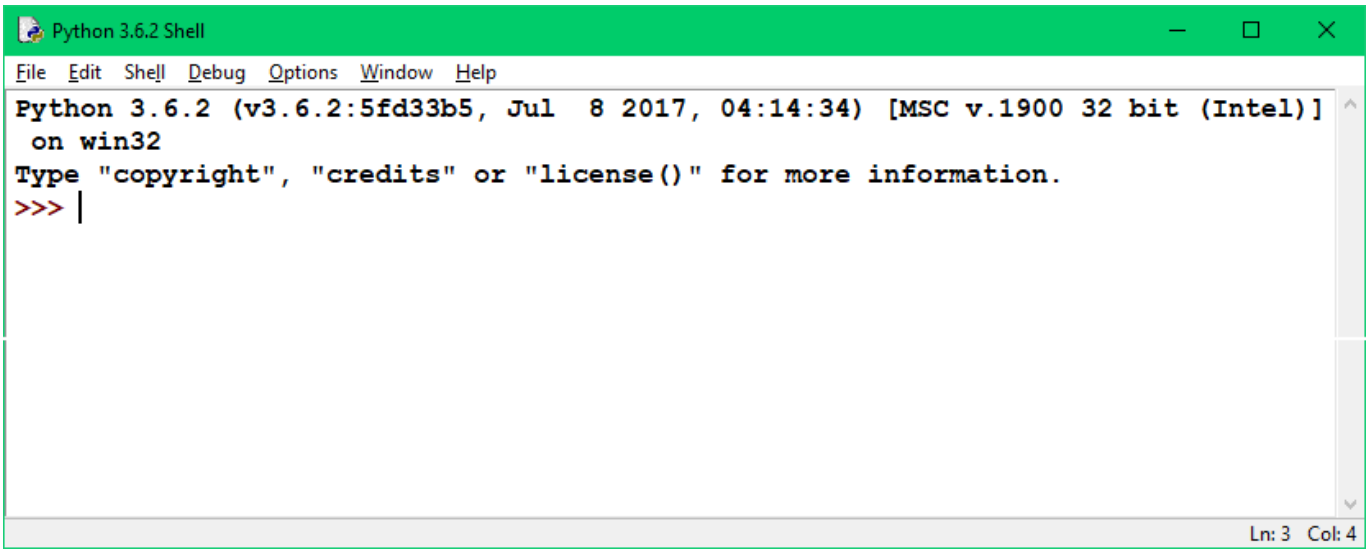

Рис. 1

На рисунке 1 представлен вид консоли (основного окна для ввода команд программы). В заголовке консоли указана версия языка Python и дата релиза, а также разрядность платформы. Обратите внимание на символ >>> в коде выше, на рисунке 1. Это символ пользовательского ввода консоли Python. Python – это интерпретированный язык, из чего следует, что каждая строка (один из основных типов данных в Python для хранения текста) выполняется после её ввода. В Python также присутствует IDLE (интегрированная среда разработки), включающая в себя инструменты для написания и отладки всей программы.

Лабораторная работа № 1.

Выполните операции с числами в консоли по примеру на рисунках со 2 по 15, осуществив простой ввод этих чисел, символов и знаков с клавиатуры в консоль после символа >>>. Чтобы получить результат необходимо в конце строки нажать клавишу Ввод (Enter). Результаты запишите в тетради.

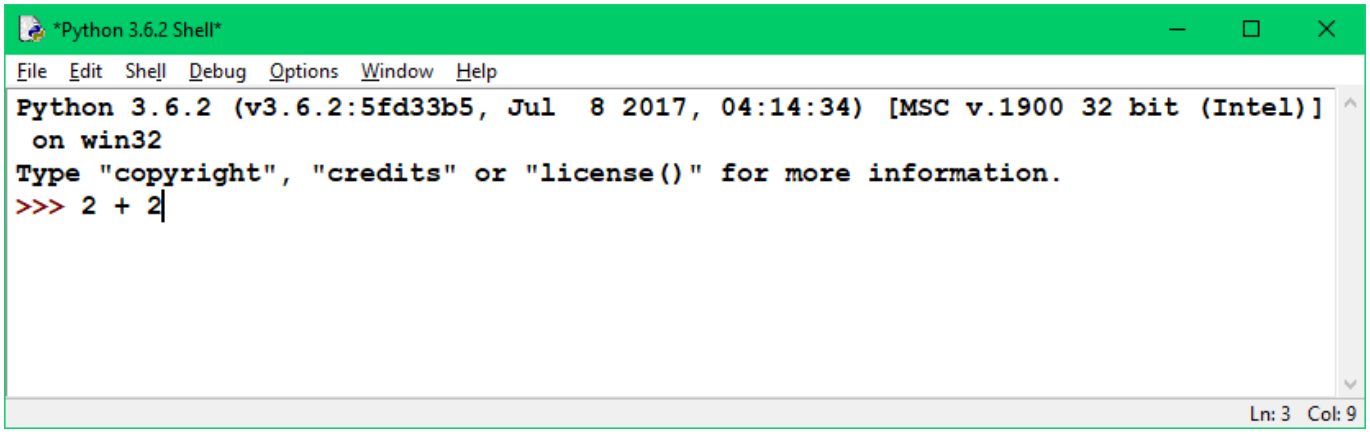

Рис. 2

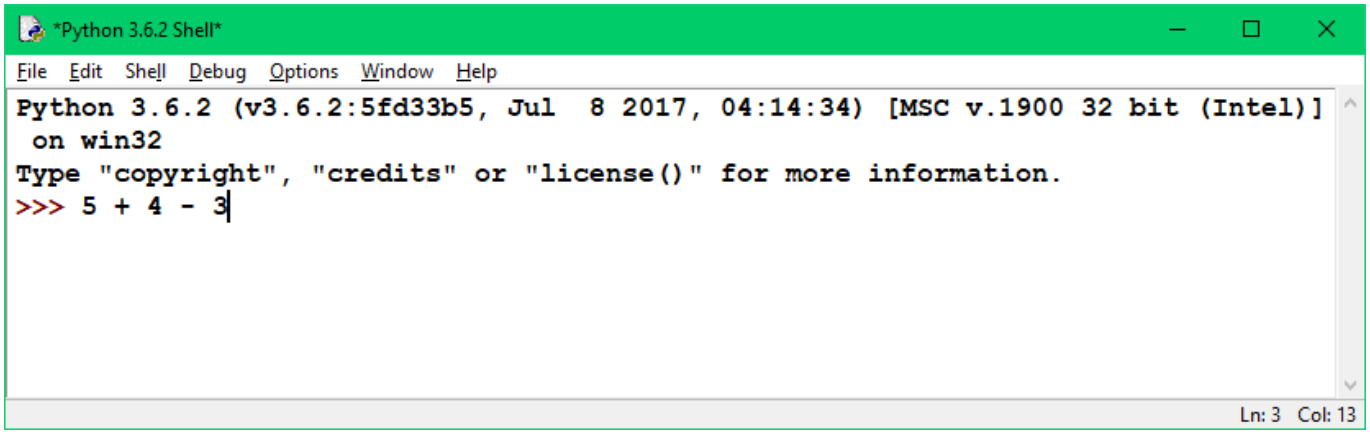

Рис. 3

Пробелы вокруг знаков плюс и минус здесь **не являются обязательными** (код будет правильным и без них, даже более **оптимальным**), но с ними код удобнее читать.

Python также поддерживает операции умножения и деления. Используйте **звёздочку** для обозначения умножения и **косую черту** – для деления. Скобки указывают, какие операции должны выполняться первыми.

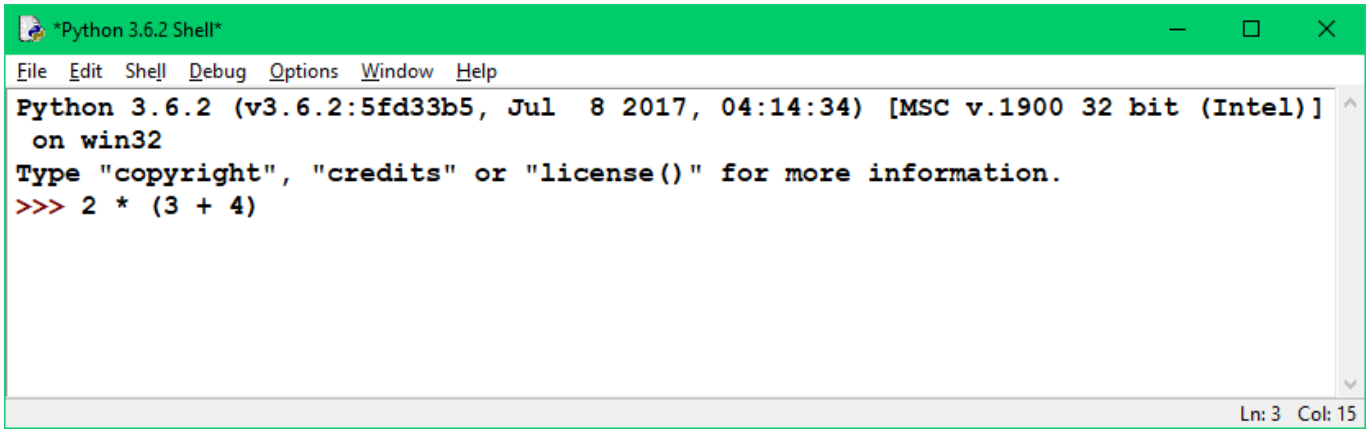

Рис. 4

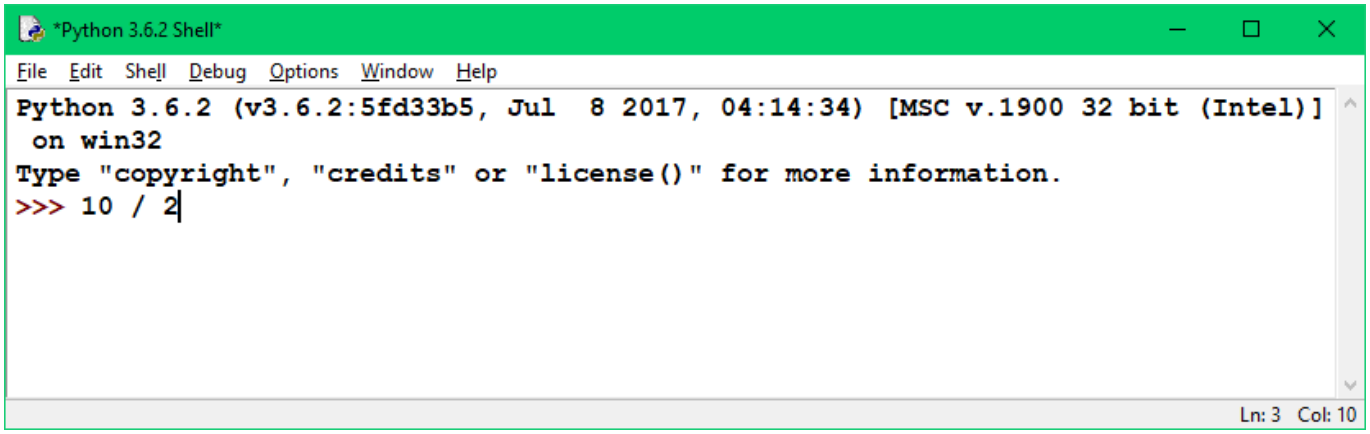

Рис. 5

Одинарная черта используется для деления, в результате чего производится десятичное число (**число с плавающей точкой** или **вещественное число** на языке программистов).

Знак минус указывает на **отрицательное число**. Операции с отрицательными числами аналогичны операциям с положительными.

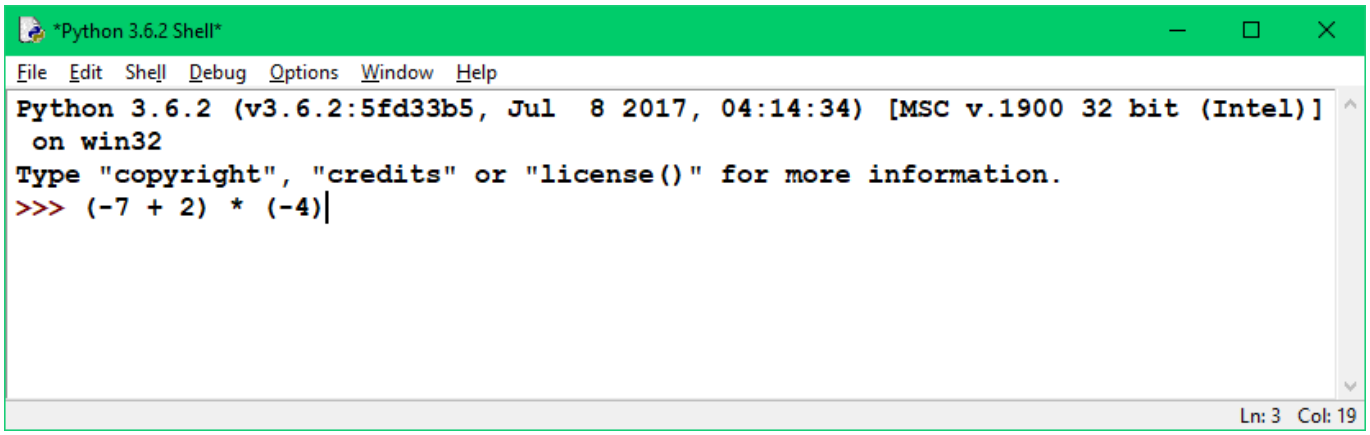

Рис. 6

Деление на ноль выдаст **ошибку**, так как результат невозможно рассчитать.

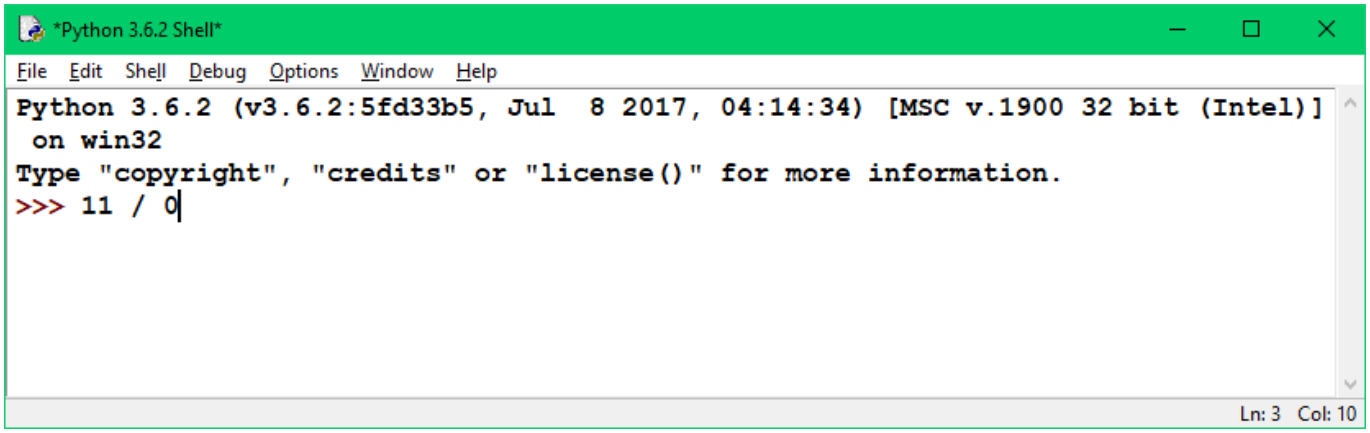

Рис. 7

Числами с плавающей точкой в Python называются нецелые числа. Например это такие числа, как 0.5 и -7.8237591.

Они могут быть созданы непосредственно путём ввода десятичных чисел, либо с помощью таких операций, как деление целых чисел. Нули в конце чисел программой игнорируются.

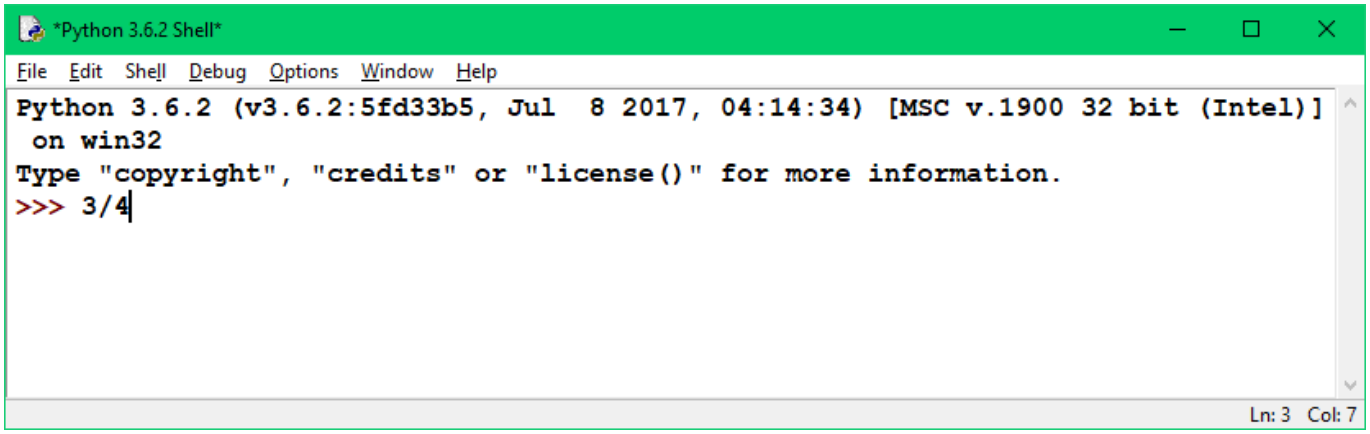

Рис. 8

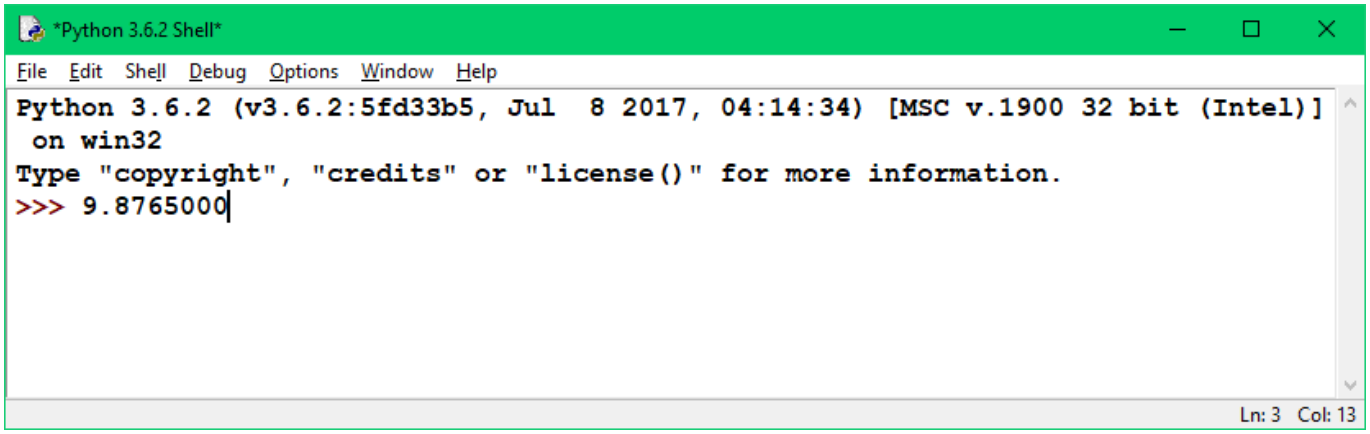

Рис. 9

Помните о том, что компьютеры не могут с точностью хранить вещественные числа, так же, как и мы не можем записать все числа после запятой для уравнения 1/3 (0,33333333333333333333…), и это может привести к ошибке.

В результате деления целого числа на целое число мы получаем **вещественные числа**. Также такие числа производятся путём операции с двумя вещественными числами или с вещественным и целым числом.

Кроме сложения, вычитания, умножения и деления Python также поддерживает **возведение в степень**, то есть многократное умножение числа на себя. Эта операция записывается с помощью двух звёздочек.

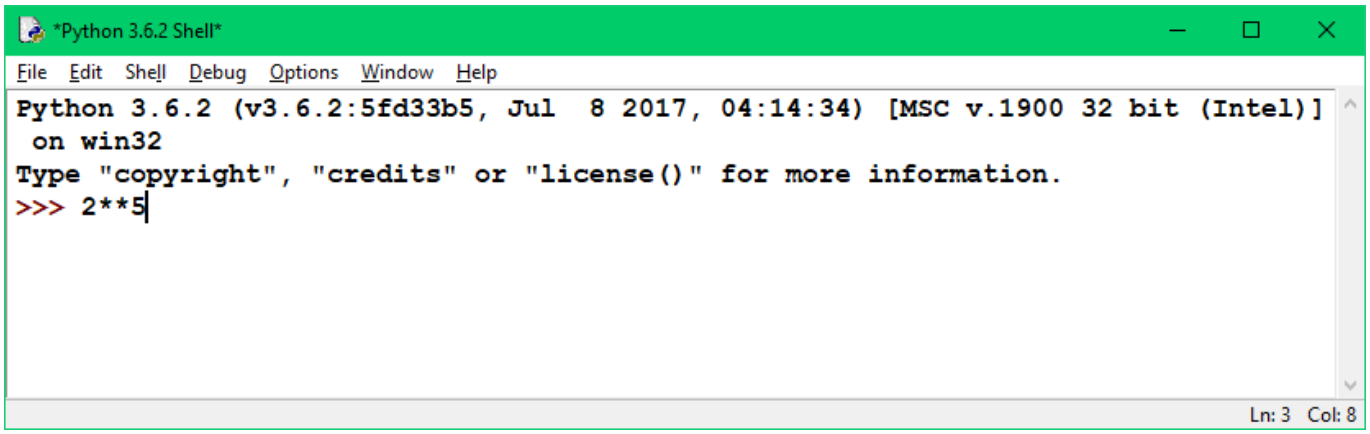

Рис. 10

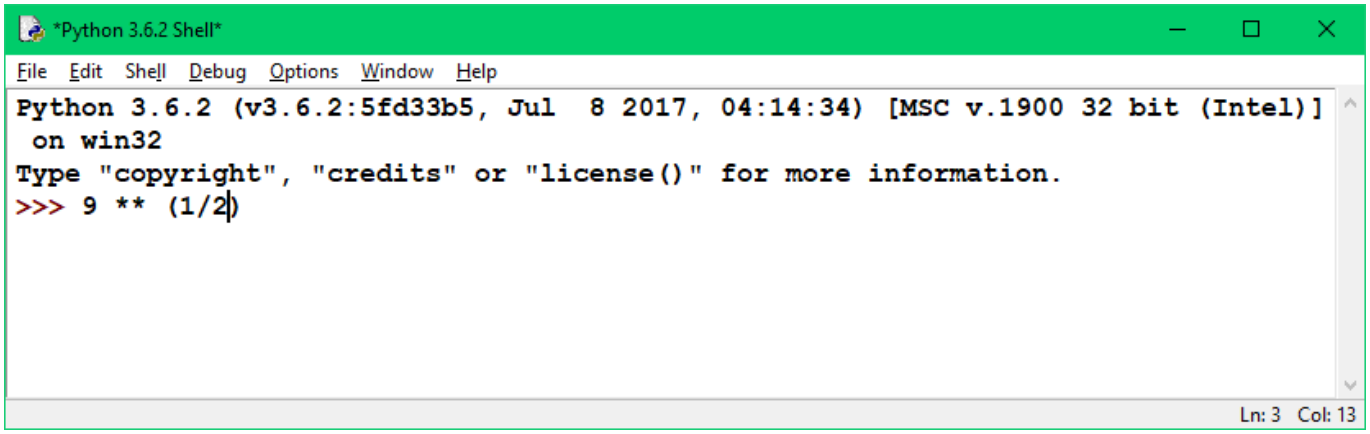

Рис. 11

Для определения **частного** и **остатка от деления** используются соответственно операции **целочисленного деления** и **деление по модулю**.

Целочисленное деление осуществляется с помощью двух косых черт (/).

Операция **деление по модулю** осуществляется с помощью символа процента (%).

Эти операции могут проводится как с дробными числами, так и с целыми.

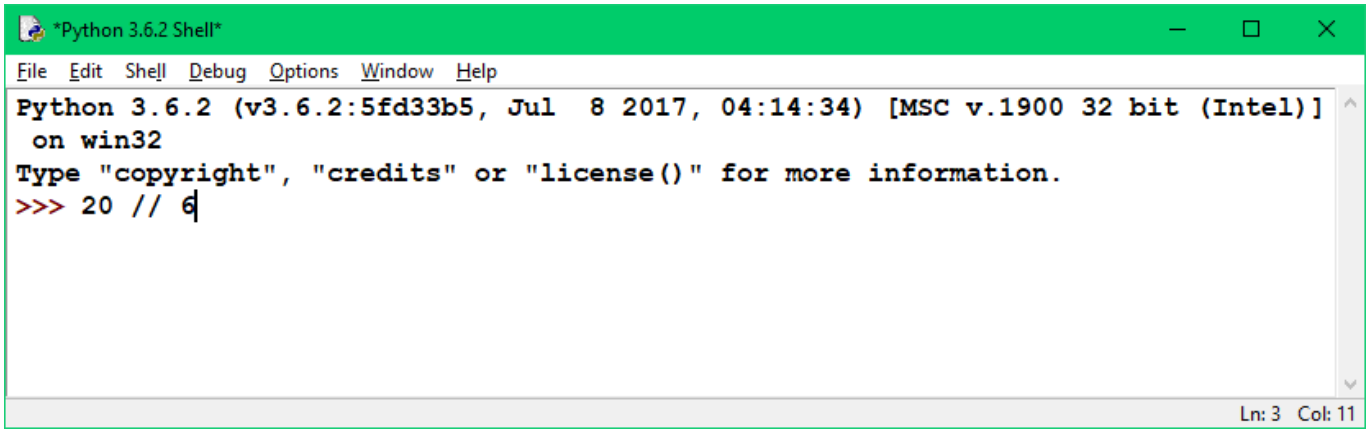

Рис. 12

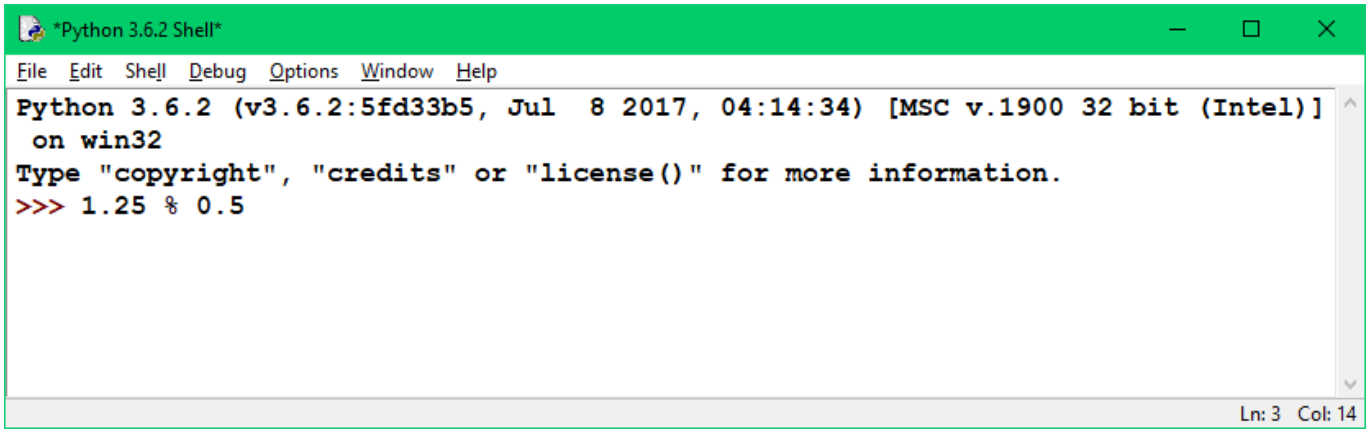

Рис. 13

Если вы хотите использовать текст в Python, нужно создать **строку**. **Строка** создается путём ввода текста между парой **одинарных** или **двойных кавычек**.

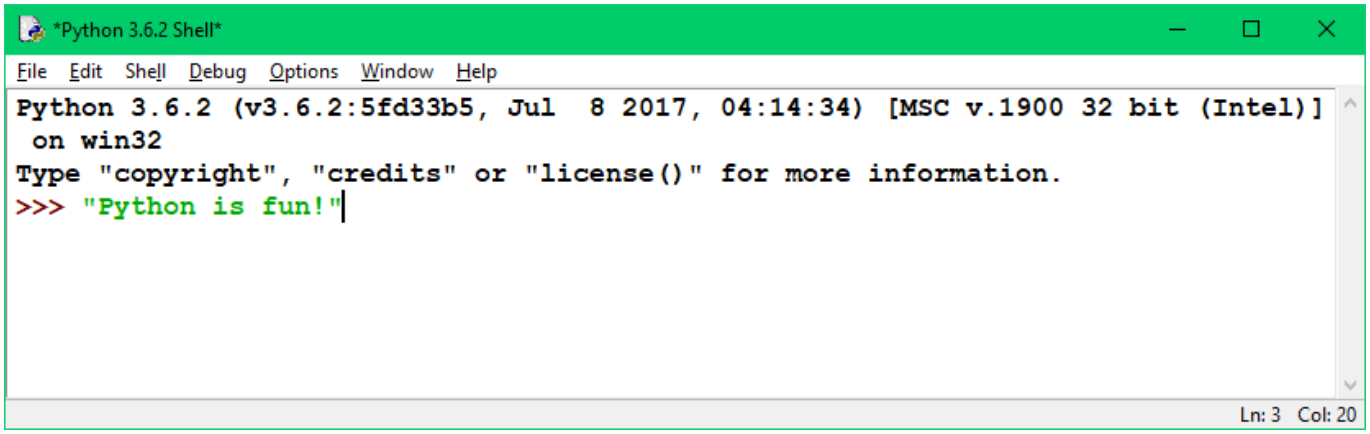

Рис. 14

Некоторые символы нельзя просто так писать в строке. Например, двойные кавычки нельзя заключать в другие двойные кавычки, оформляющие строку; это приведёт к тому, что программа будет преждевременно прервана.

Чтобы использовать такие символы необходимо создавать для них исключение: ставить перед ними **обратную косую черту (бэкслэш)**.

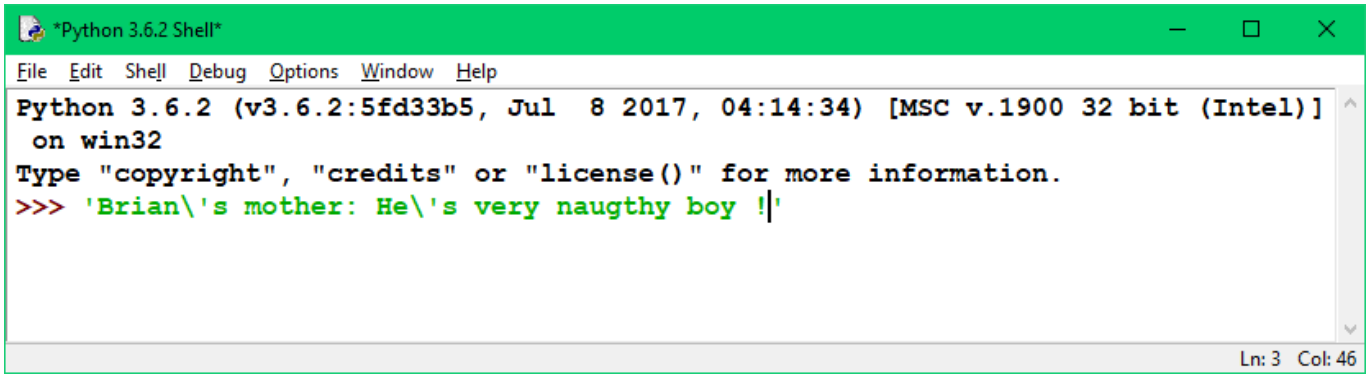

Рис. 15

Лабораторная работа № 2.

Выполните простые функции ввода и вывода, операции с переменными в консоли по примеру на рисунках с 16 по 35, осуществив простой ввод этих чисел, символов и знаков с клавиатуры после символа >>>. Чтобы получить результат необходимо в конце строки нажать клавишу Ввод (Enter). Результаты запишите в тетради.

Как правило, программы принимают некоторый пользовательский **ввод**, обрабатывают его и отображают **вывод**. В Python для вывода результата на экран используется функция **print**. В результате на экран будет выведен некоторый текст.

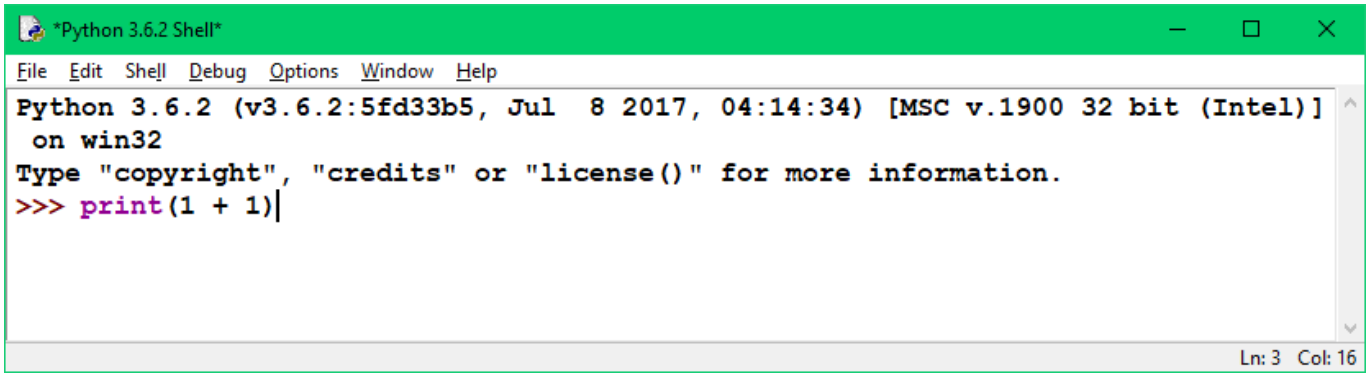

Рис. 16

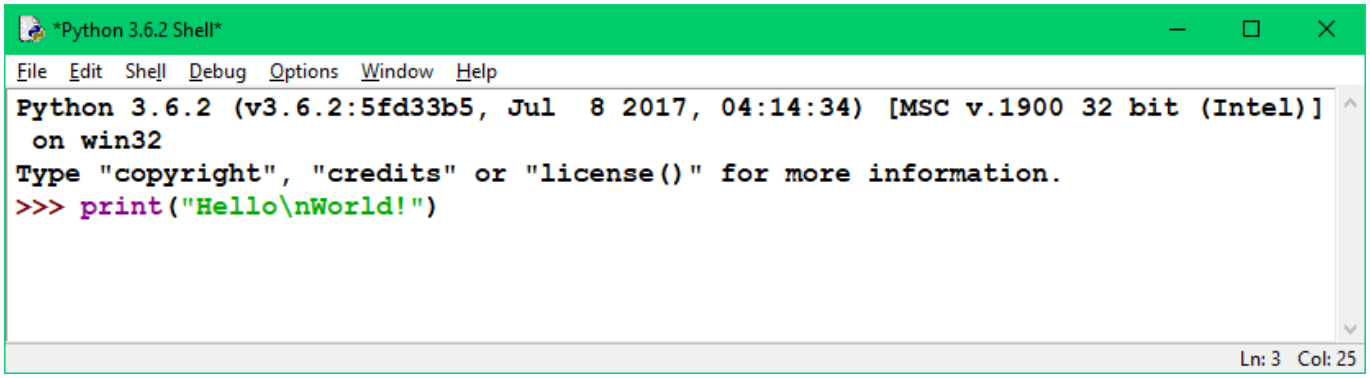

Рис. 17

Чтобы получить данные от пользователя, в Python используется функция с интуитивным названием **input** (англ. ввод).

Функция приглашает пользователя ввести данные, после чего введённый текст возвращается в виде строки (символы автоматически экранируются)

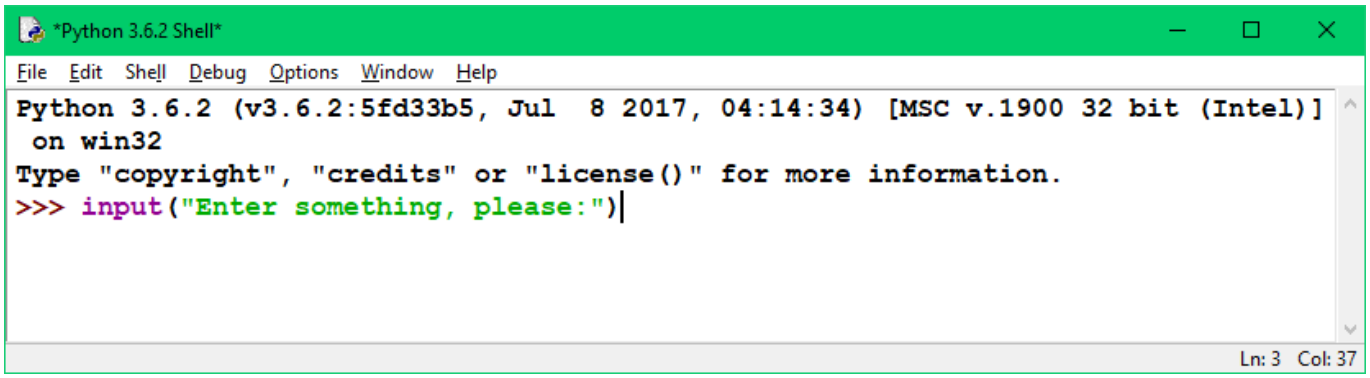

Рис. 18

Подобно целым и дробным числам, строки в Python можно сложить с помощью операции **объединения** (**конкатенации**).

При объединении строк не имеет значения, созданы ли они с одинарными или с двойными кавычками.

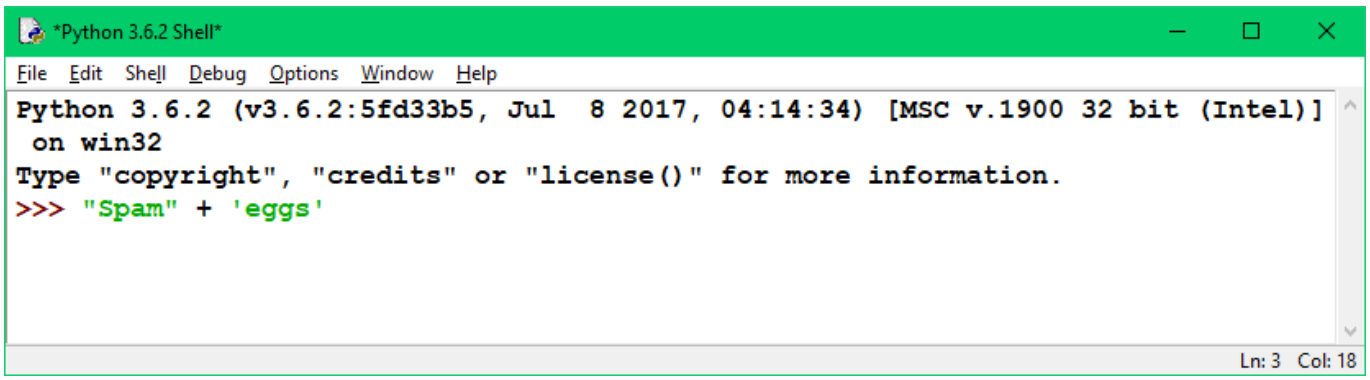

Рис. 19

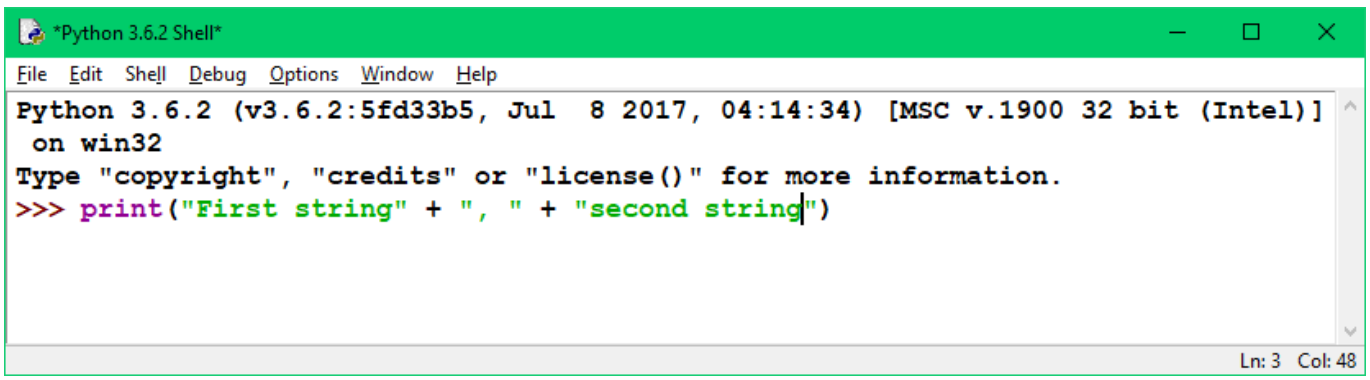

Рис. 20

Даже если ваши строки содержат числа, они будут объединяться, как строки, а не числа. Сложение строки с числом выдаст ошибку: несмотря на сходство, это два разных объекта.

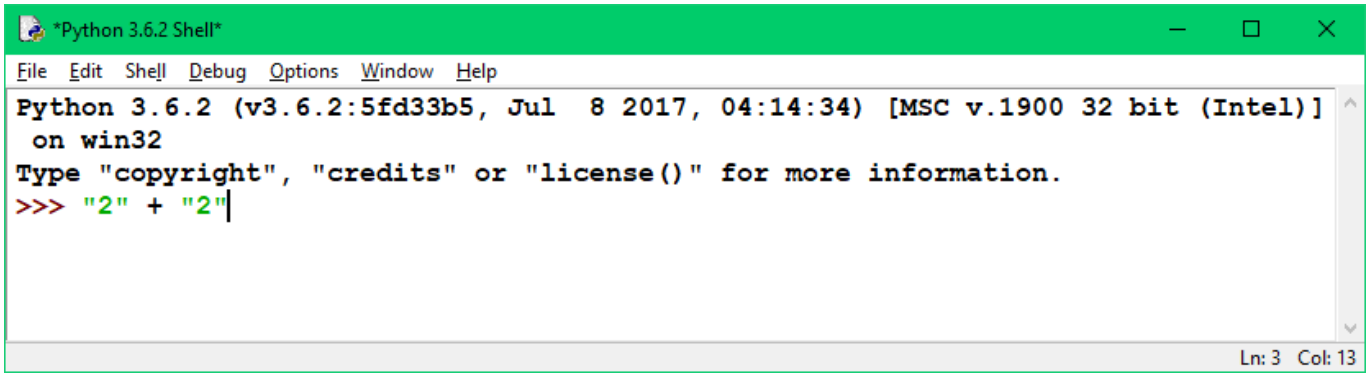

Рис. 21

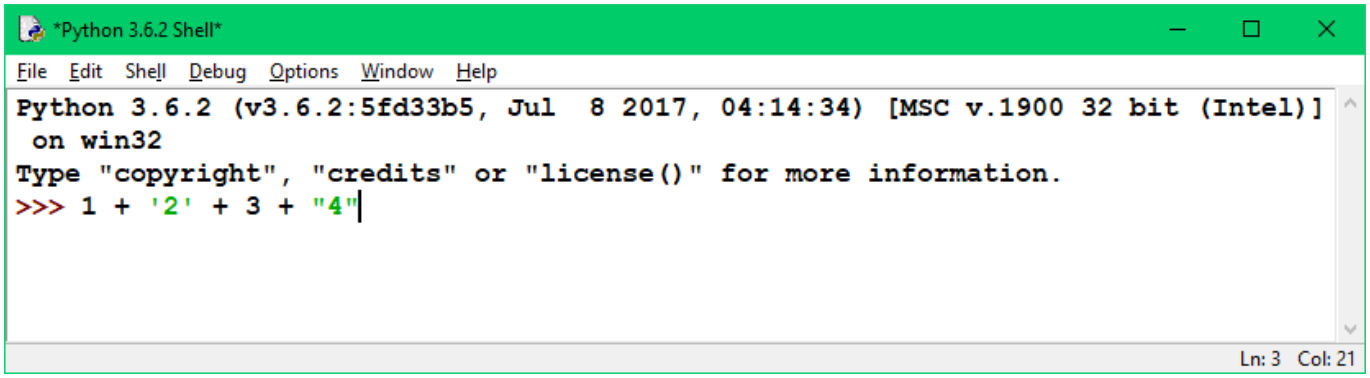

Рис. 22

Строки также можно умножать на целые числа. Это приводит к повторению начальной строки. Строки нельзя умножать на другие строки и на числа с плавающей запятой, даже если это целые числа.

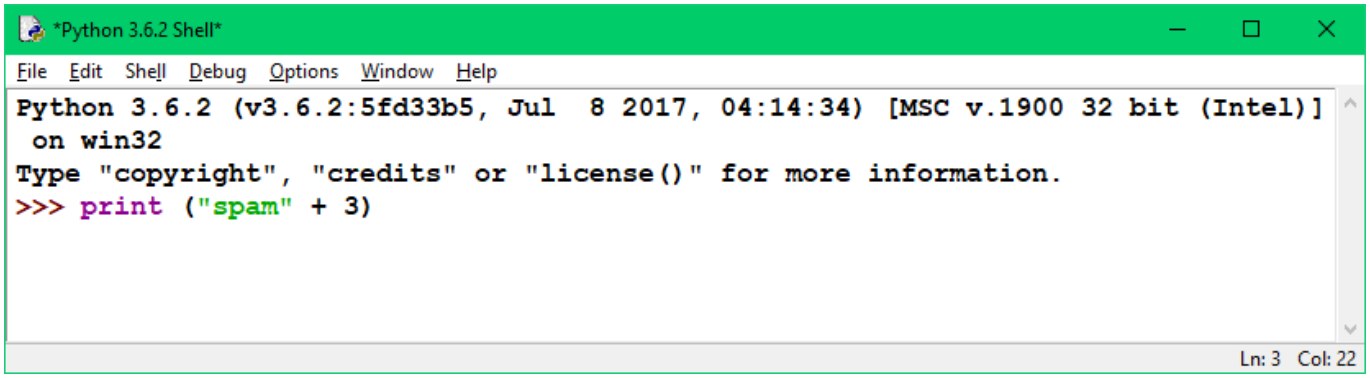

Рис. 23

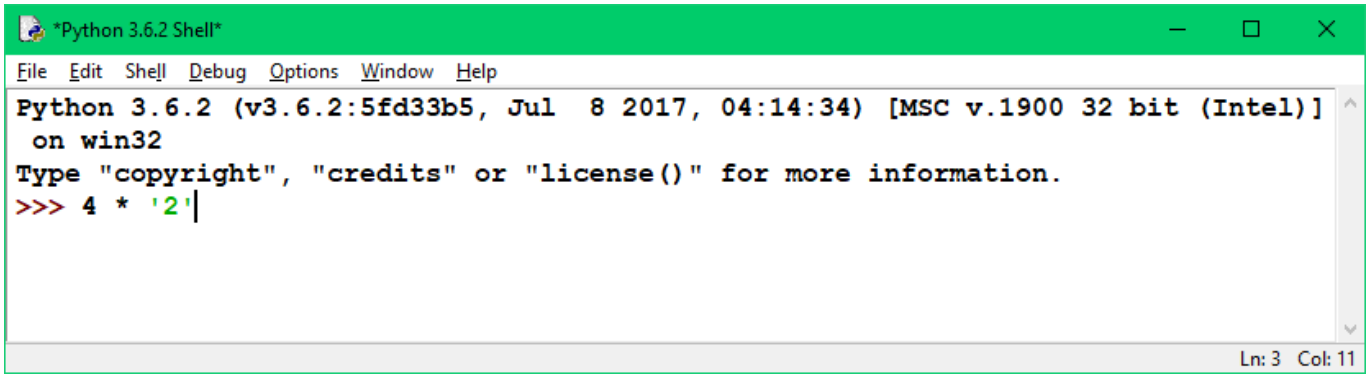

Рис. 24

В Python возможность выполнения операции зависит от типов данных. Например, нельзя сложить две строки, содержащие числа, чтобы получить число.

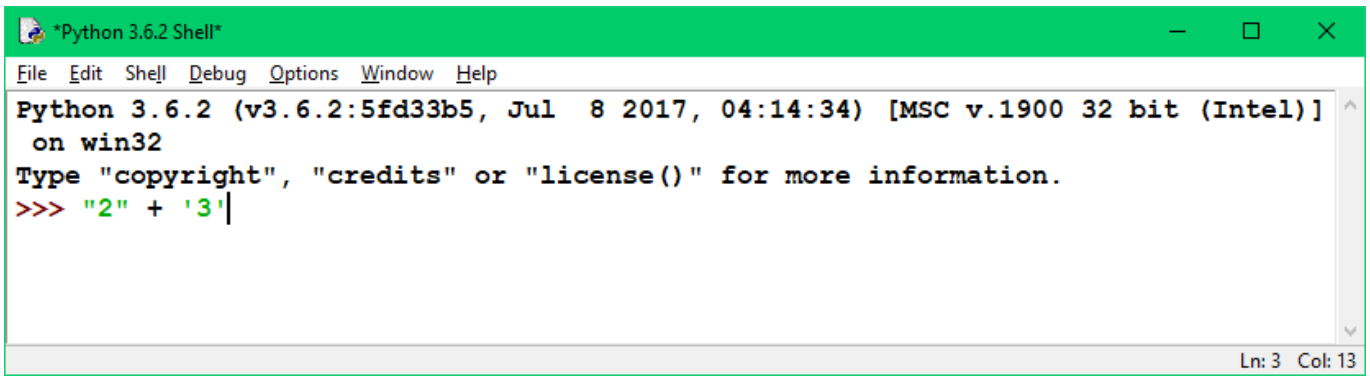

Рис. 25

В этом случае нужно **преобразовать тип данных**. В предыдущем примере необходимо использовать функцию **int**.

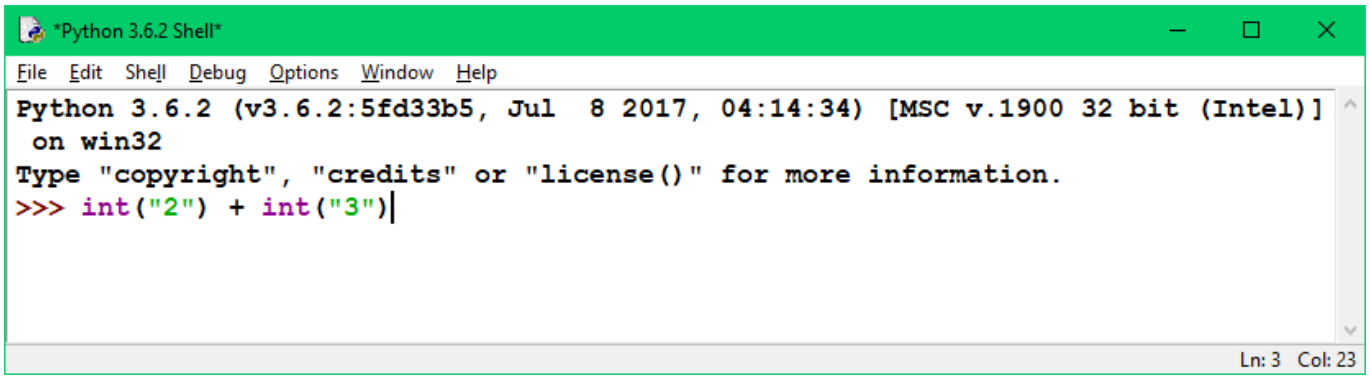

Рис. 26

Для преобразования в целые, дробные числа и строки используются соответственно функции: **int**, **float** и **str**.

В примере ниже выполняется конвертация пользовательского ввода (строки) в числа (целые и дробные) с целью выполнения вычислений.

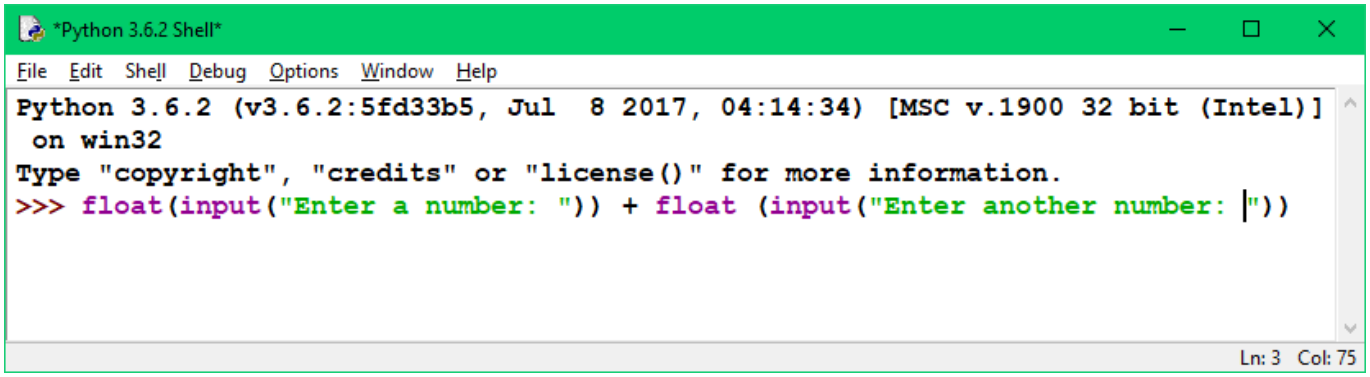

Рис. 27

Переменные играют очень важную роль в Python. Переменная позволяет сохранить значение, присвоив ему имя, по которому позже в программе можно использовать это значение.

Чтобы назначить переменную, используют **один знак равенства**. При присвоении в консоли Python ничего не выводится. Переменные можно использовать для выполнения операций как с числами и строками. В качестве имён

переменных разрешается использовать только буквы, цифры и нижнее подчеркивание.

Кроме того, имена не могут начинаться с цифр и **Name** и **name** будут считаться **разными** именами переменных.

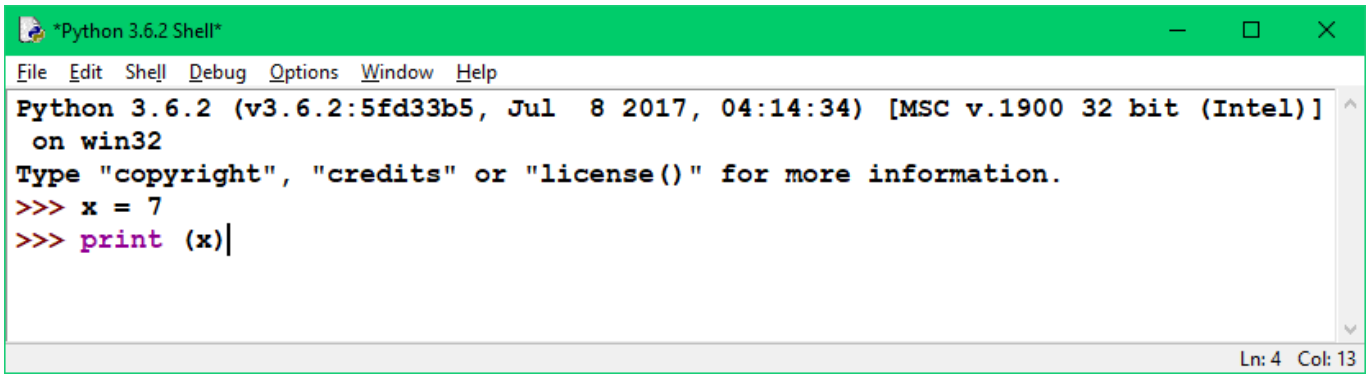

Рис. 28

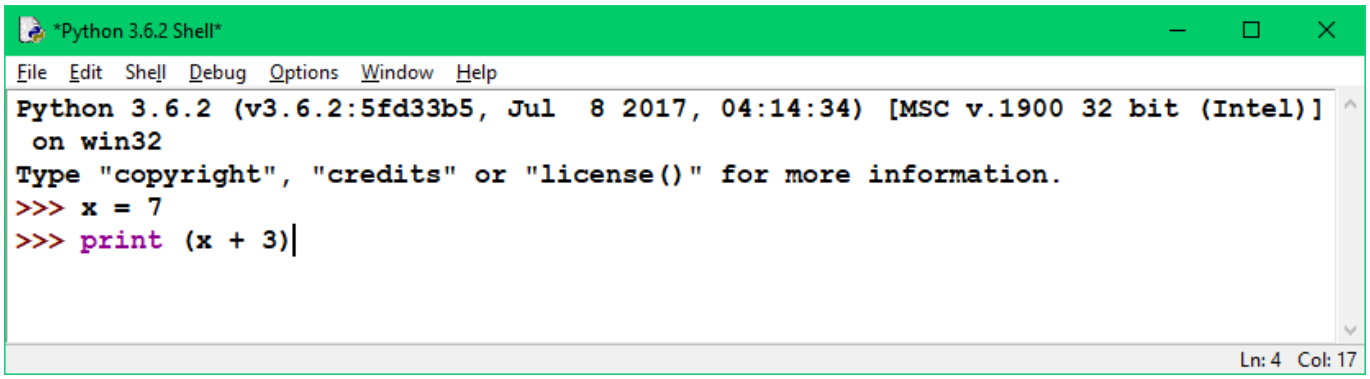

Рис. 29

Переменным могут присваиваться новые значения неограниченное количество раз. В Python переменные не имеют типа, поэтому вы можете переменной присвоить строку, а затем той же переменной целое число. Можно также взять значение переменной из пользовательского ввода.

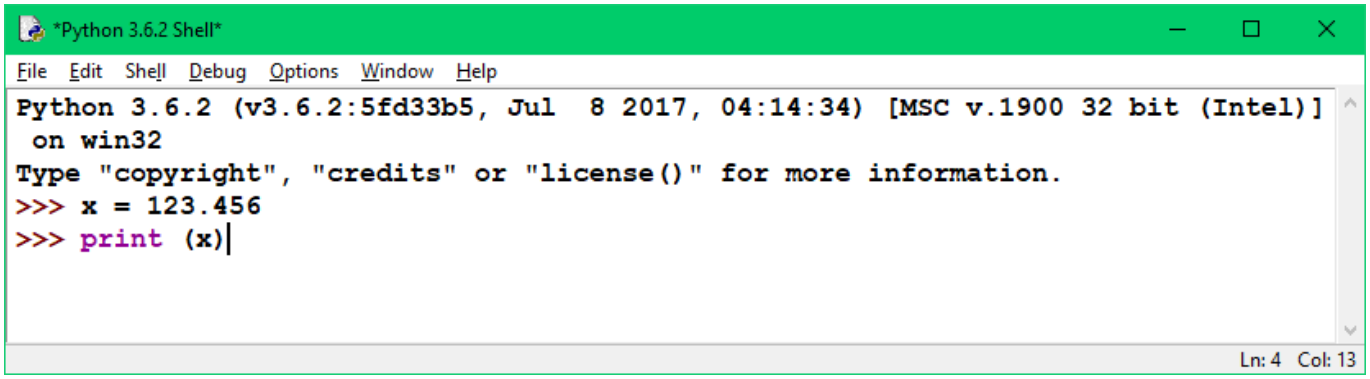

Рис. 30

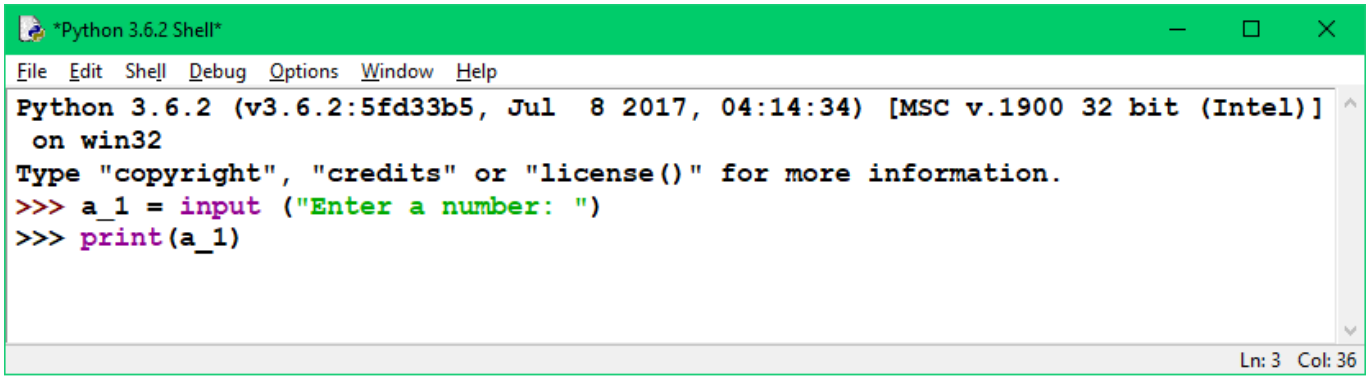

Рис. 31

Операции «на месте» (in-place) позволяют записать такую команду, как ‘x= x+3’, более коротко: ‘x+=3’.

То же возможно и с другими операциями.

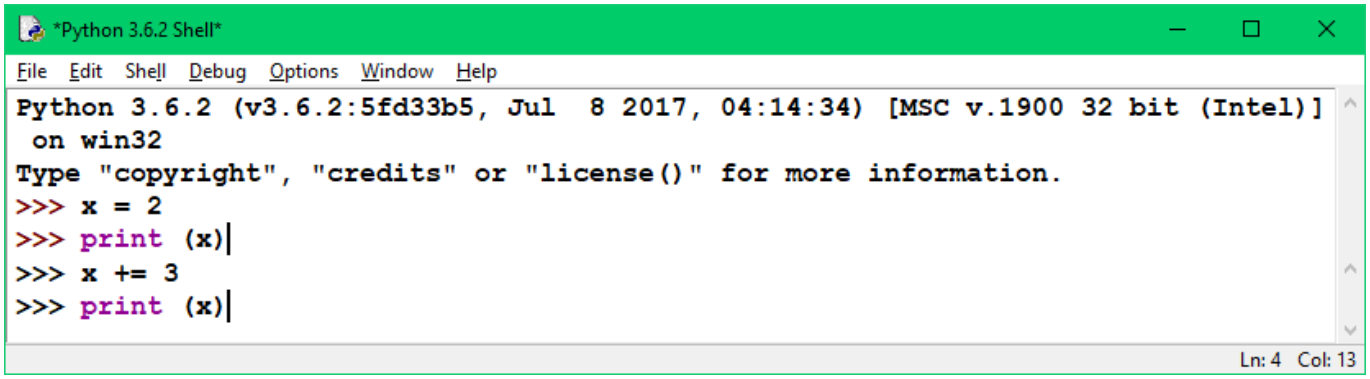

Рис. 32

Эти операции могут применяться и к другим типам данных, например, к строкам.

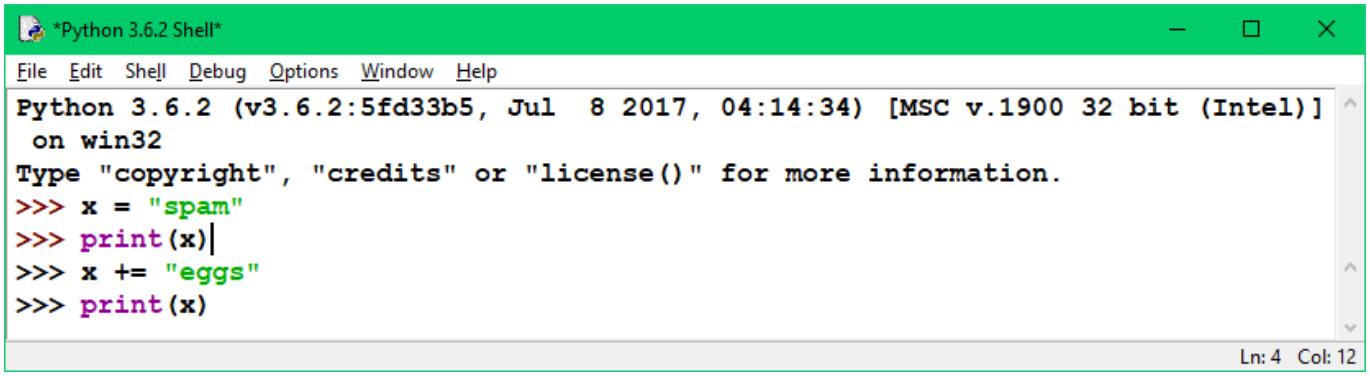

Рис. 33

До сих пор мы работали с Python только с консоли: писали и выполняли одну строку кода один раз.

Настоящие программы пишутся несколько по-другому: сначала пишутся много программных строк в текстовом файле, затем они выполняются с помощью интерпретатора Python.

В IDLE это делается следующим образом: создаётся новый файл, пишется код, затем файл сохраняется и запускается. Это можно сделать с помощью команд меню, либо сочетанием клавиш Ctrl+N, Ctrl+S и F5.

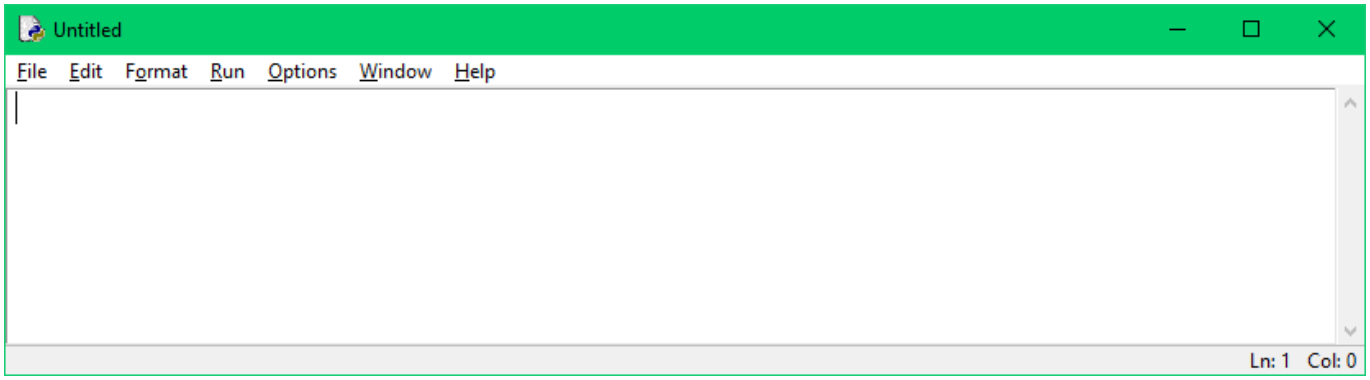

Рис. 34

Строки кода в файле интерпретируются так, как будто вы их вводили в консоли по отдельности.

Программа «Знакомство» написана в IDLE и представлена на рисунке 35.

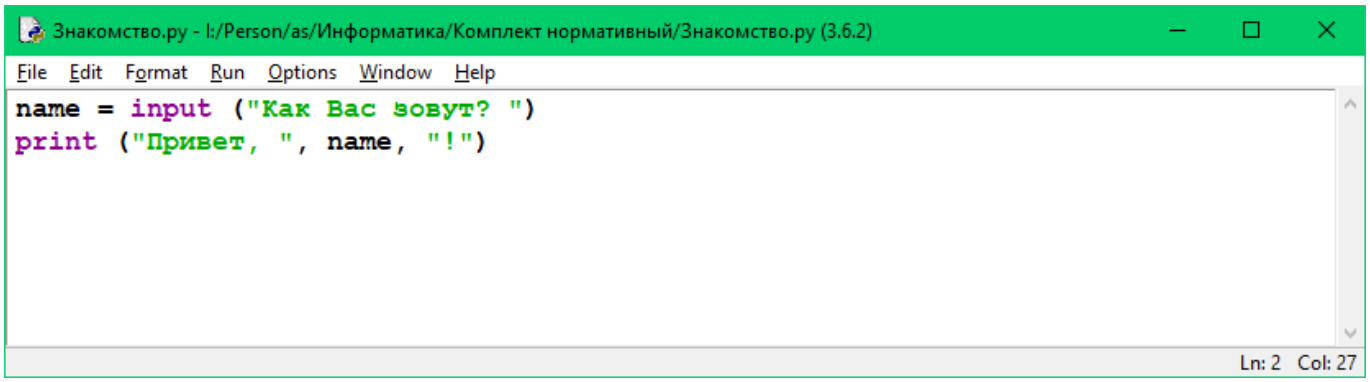

Рис. 35

Самостоятельная работа: создать программу по выполнению условий, заданных формулой, в соответствии с номером по журналу, с учётом организации дружелюбного интерфейса.

Лабораторная работа № 3

Выполните простые функции ввода и вывода, операции с переменными в консоли или текстовом файле с расширением .py по примеру на рисунках с 36 по 35, осуществив простой ввод этих чисел, символов и знаков с клавиатуры. Чтобы получить результат необходимо в конце строки нажать клавишу Ввод (Enter). Результаты запишите в тетради.

Еще одним типом данных в Python является **логический тип (boolean)**. Логические выражения принимают только два значения: **True** (истина) и **False** (ложь). Они используются для сравнения значений с помощью операторов равенства ==.

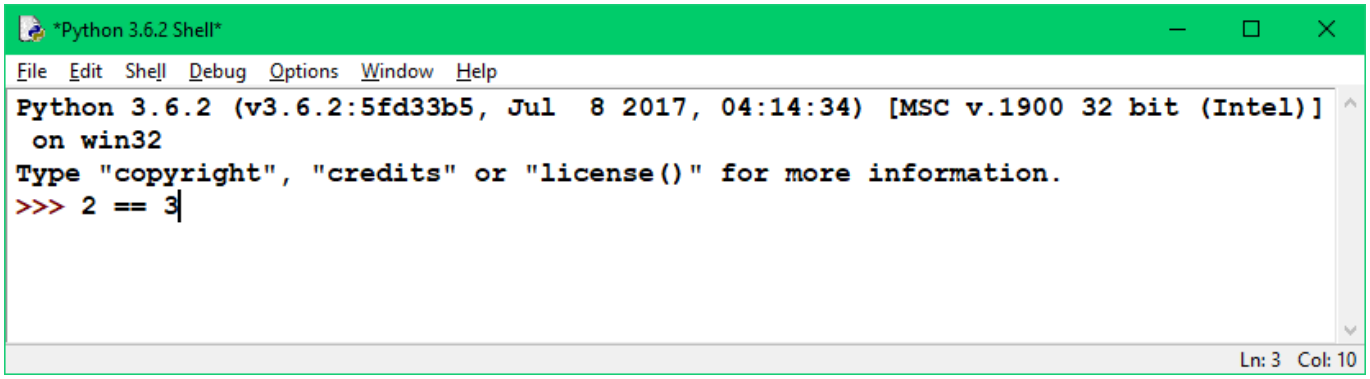

Рис. 36

Еще одна операция сравнения (оператор **неравенства, !=**) возвращает значение **True** (истина), если сравниваемые аргументы неравные и **False** (ложь), если они равные.

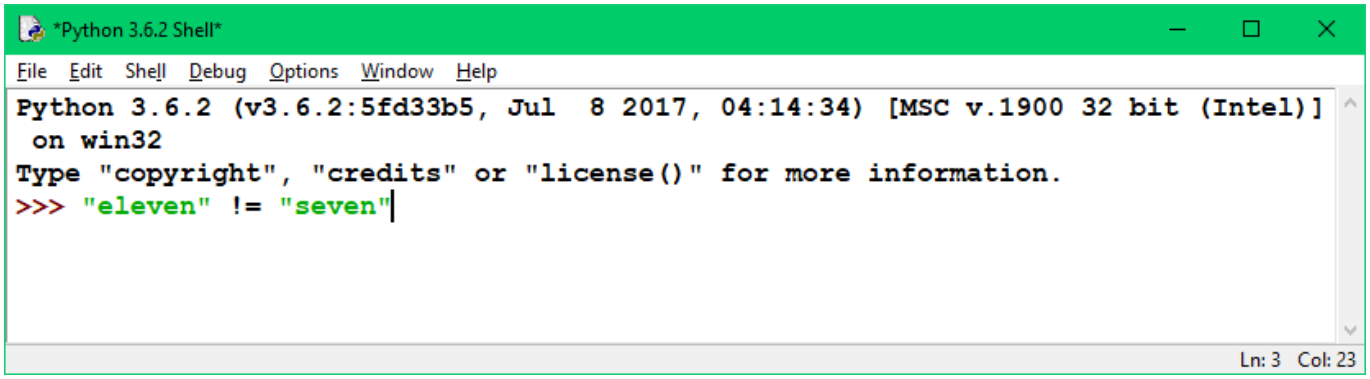

Рис. 37

В Python есть также операции, которые определяют, является ли число (целое или дробное) больше или меньше другого числа. Им соответствуют операторы > и <.

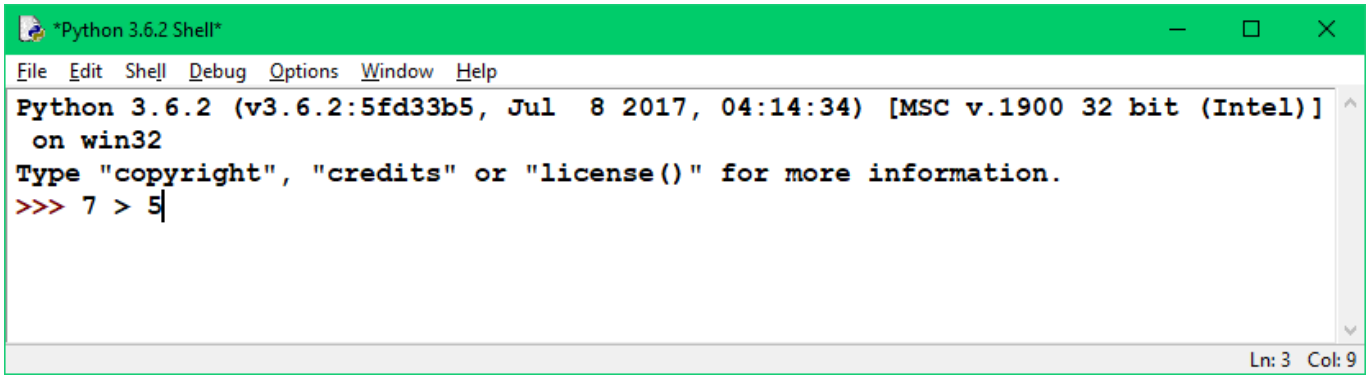

Рис. 38

Используются также операции больше или равно и меньше или равно: >= и <=. Они похожи ан операторы больше и меньше с той разницей, что возвращают значение **True** (истина) при сравнении равных чисел.

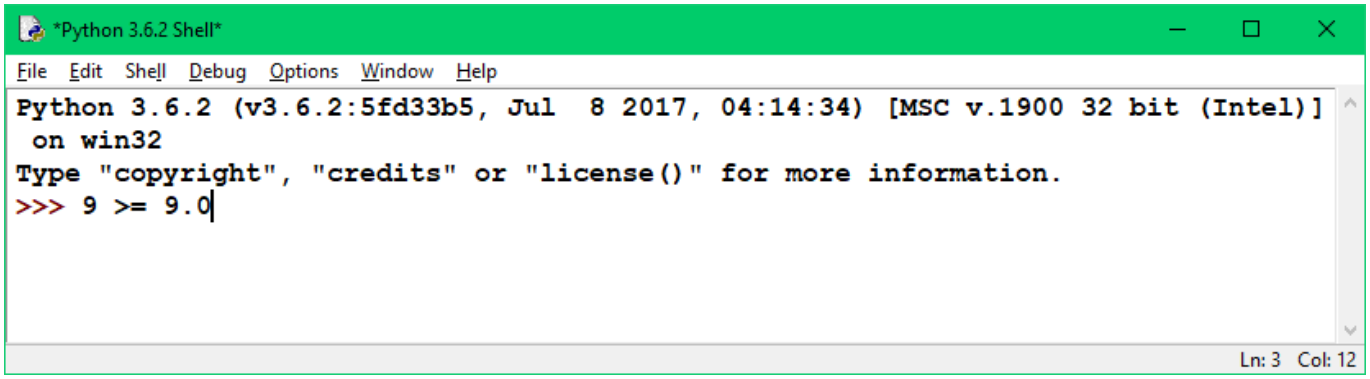

Рис. 39

Операторы больше или меньше также используются для сравнения строк в **лексикографическом порядке** (порядок слов основан на алфавитном порядке их букв).

Оператор **if** (если) указывает, что код должен выполняться только при определённом условии. Если выражение истинно (**True**), то выполняются некоторые инструкции. В противном случае эти инструкции не выполняются.

Для разделения блоков кода в Python используются **отступы** (пустое место в начале строки). Они являются обязательными: программа без них не будет работать.

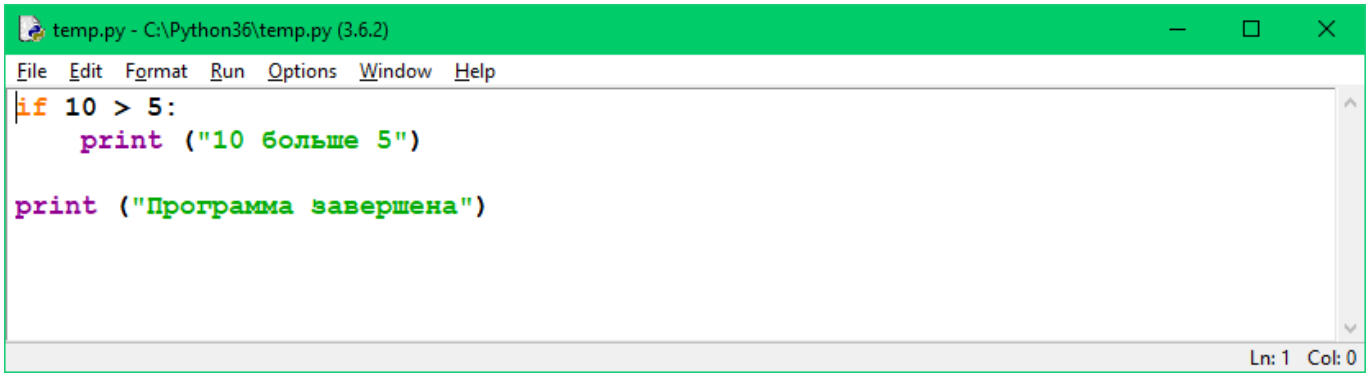

Рис. 40

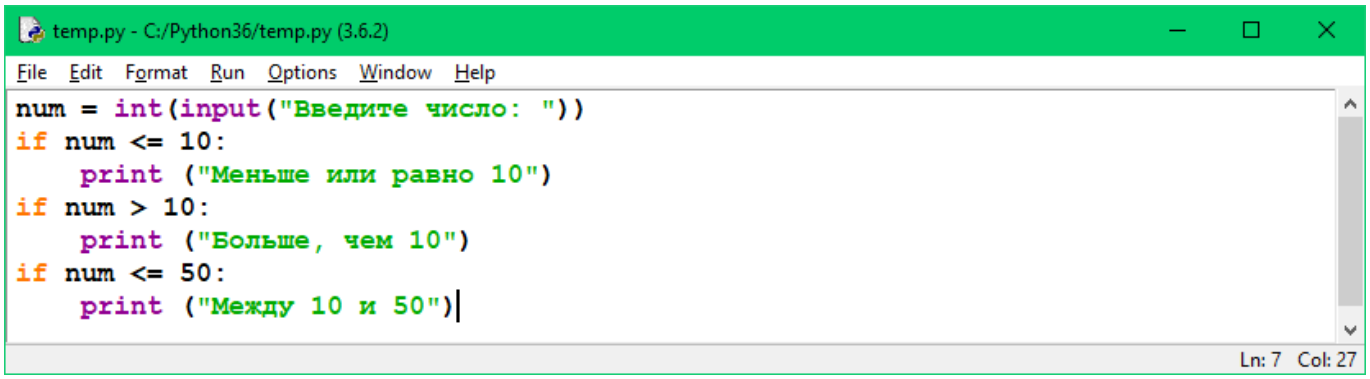

Рис. 41

Инструкция **else** следует за инструкцией **if** и содержит код, который выполняется, когда возвращается **False** (ложь). Как и в случае с инструкциями if, строки внутри ветви else должны иметь отступы.

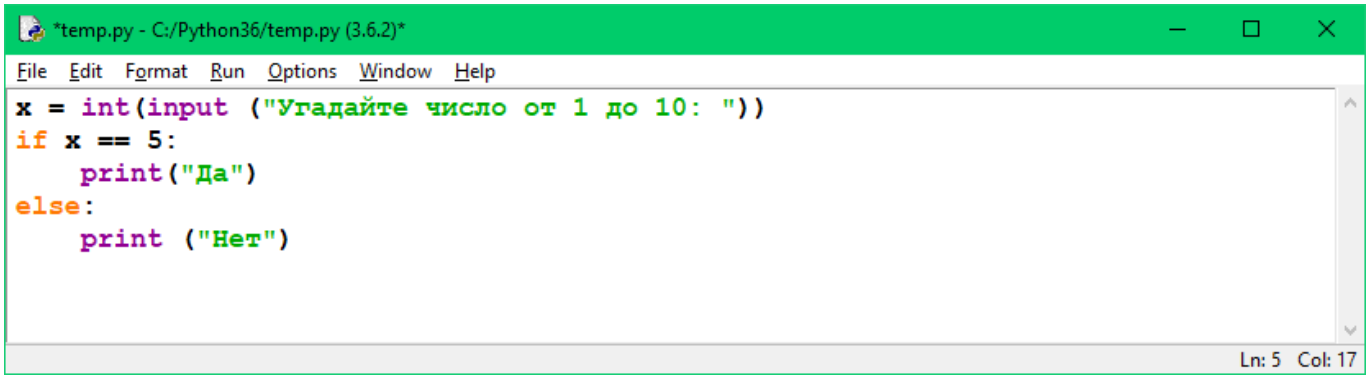

Рис. 42

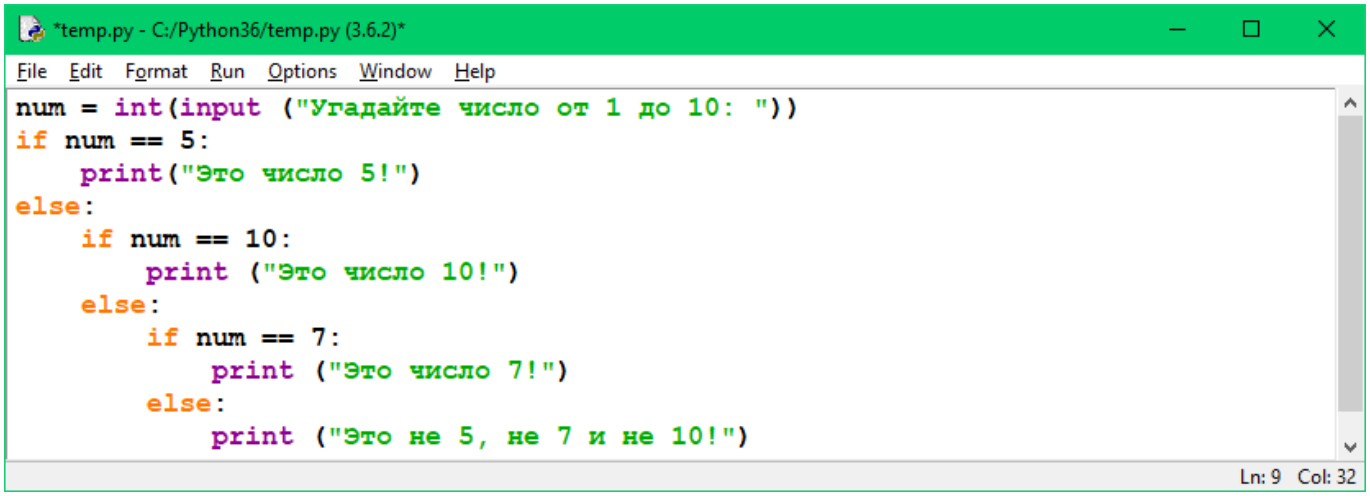

Рис. 43

Инструкция **elif** (сокр. от англ. else if) – это более короткий способ создания цепочек инструкций **if** и **else**.

Цепочка инструкций **if** / **elif** могут иметь конечную ветвь else, которая выполняется при неистинности всех условий **if** и **elif**.

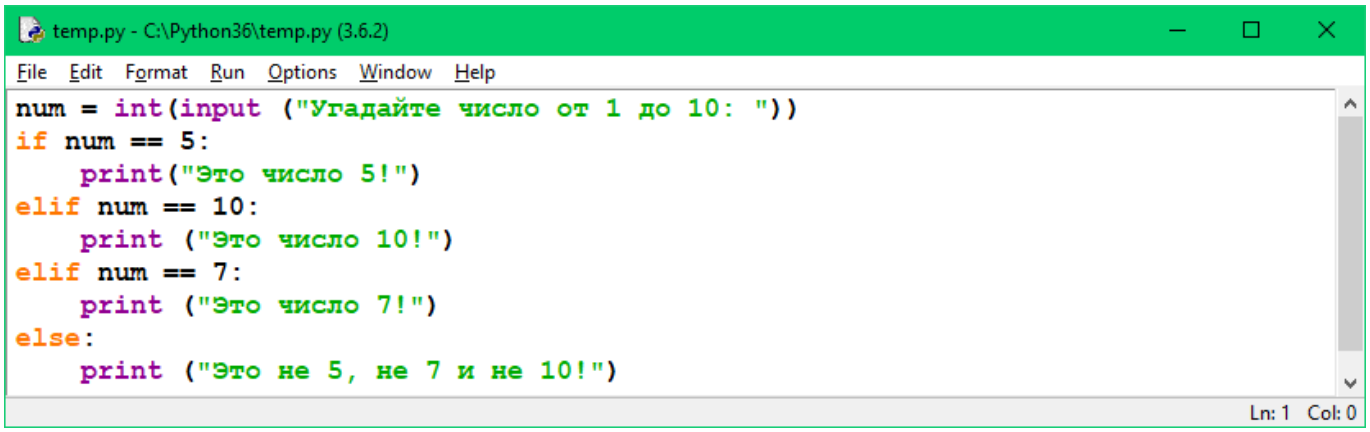

Рис. 44

Также в Python используется **булева логика** для создания более сложных условий в инструкциях **if**, которые имеют больше одного условия. Это такие операторы, как **and**, **or** и **not**.

Оператор and оценивает два аргумента; значение **True** (истина) возвращается только в том случае, если оба аргумента **истинны** (**True**). В противном случае возвращается **False** (ложь). Выполните сложные условия на рис. 46.

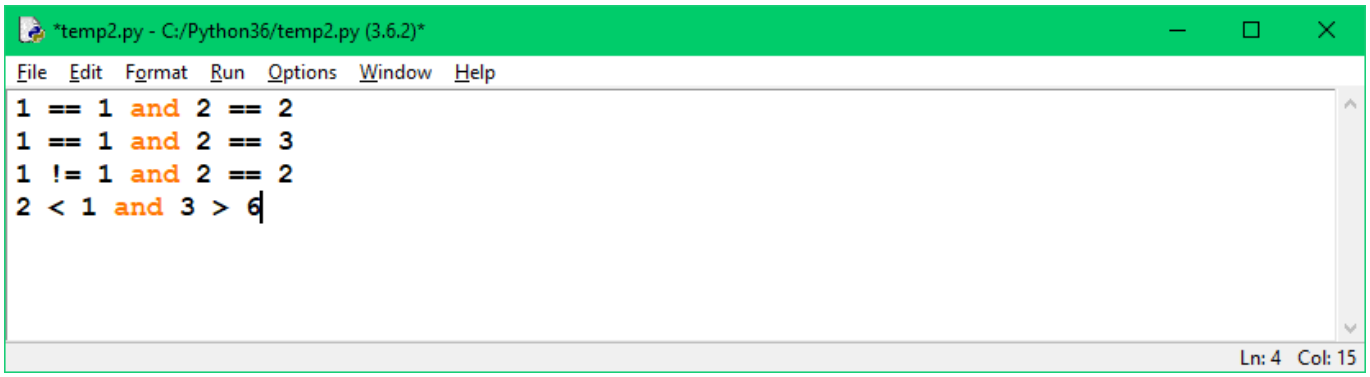

Рис. 45

Логический оператор or также имеет два аргумента. Значение **True** возвращается, когда один его аргумент (или оба) **истинный**, и значение **False**, когда оба аргумента **ложные**. Выполните сложные условия на рис. 47.

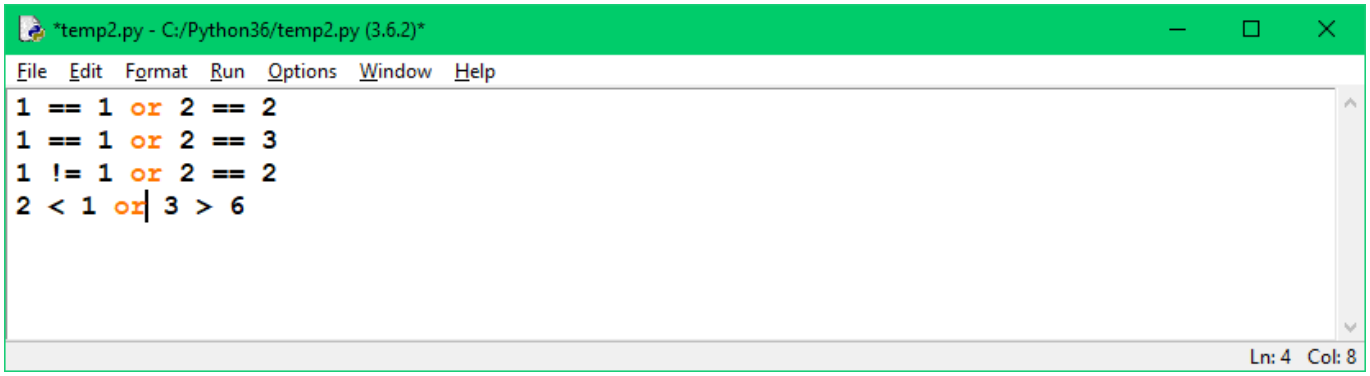

Рис. 46

В отличии от других рассмотренных нами операторов, оператор not может иметь только один аргумент, но может его инвертировать. В случае **not True** возвращается значение **False**, в случае **not False** – **True**.

Приоритет операторов – очень важное понятие в программировании. Оно аналогично порядку операций в математике, но касается операций, использующихся в булевой логике.

В таблице 1 описан приоритет выполнения операторов в Python от наивысшего (выполняется в первую очередь) до наинизшего. Операторы, находящиеся в одной ячейке, имеют равный приоритет. Логические операторы Not, And и Or выполняются именно в такой последовательности.

Таблица 1

Оператор	Описание
**	Возведение в степень
~ + -	Комплиментарный оператор
* / % //	Умножение, деление, деление по модулю, целочисленное деление.
+ -	Сложение и вычитание.
>> <<	Побитовый сдвиг вправо и побитовый сдвиг влево.
&	Бинарный "И".
^ |	Бинарный "Исключительное ИЛИ" и бинарный "ИЛИ"
<= < > >=	Операторы сравнения
<> == !=	Операторы равенства
= %= /= //= -= += *= **=	Операторы присваивания
is is not	Тождественные операторы
in not in	Операторы членства
not or and	Логические операторы

Инструкции **While** выполняются аналогично **if** (выполняются, когда условие истинно и не выполняются, когда условие ложно) с той лишь разницей, что они могут запускаться более одного раза. Когда возвращается **False,** программа переходит к выполнению следующего блока кода.

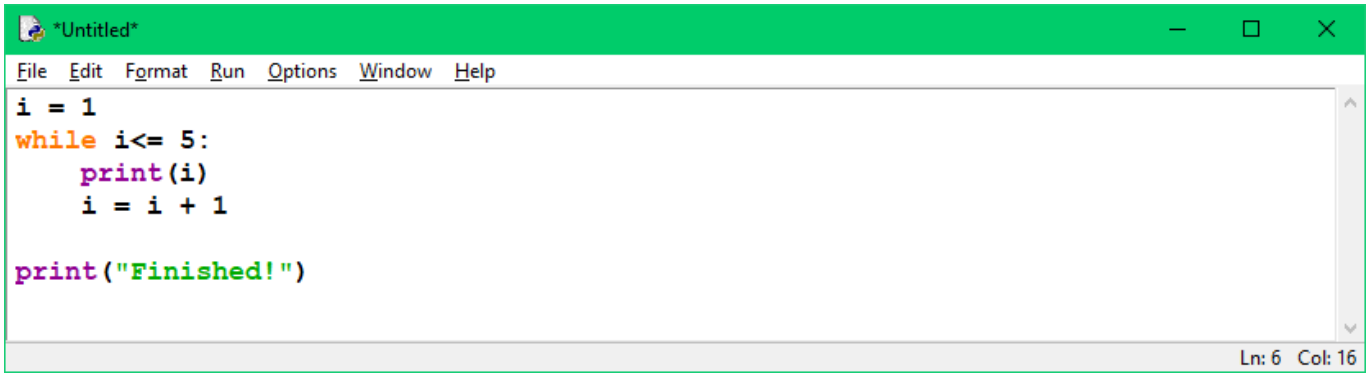

Рис. 47

Иногда нужно пропускать код по каждому пункту в списке. Это называется итерацией, в которой используются цикл **while** и переменная счётчик.

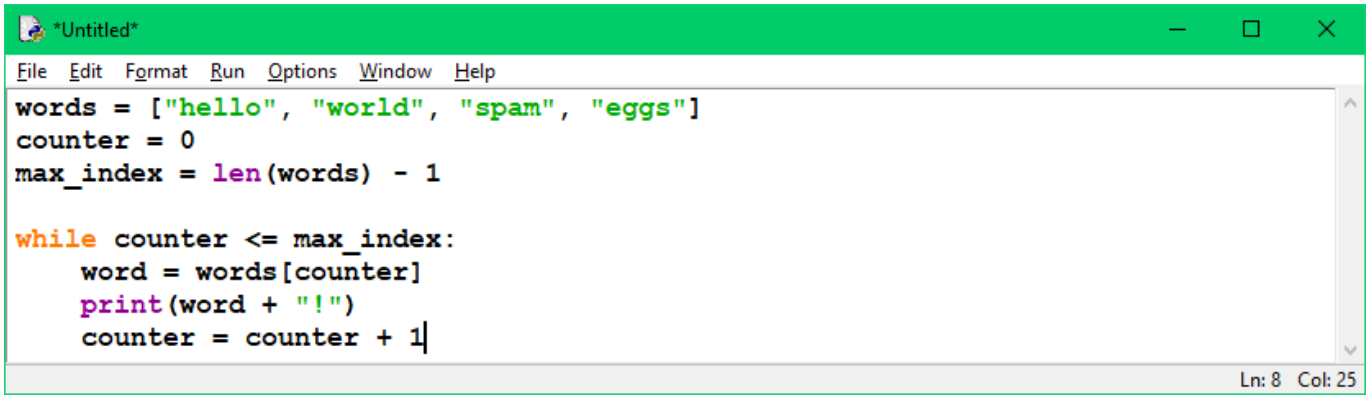

Рис. 48

Цикл for позволяет записать то же самое более кратко.

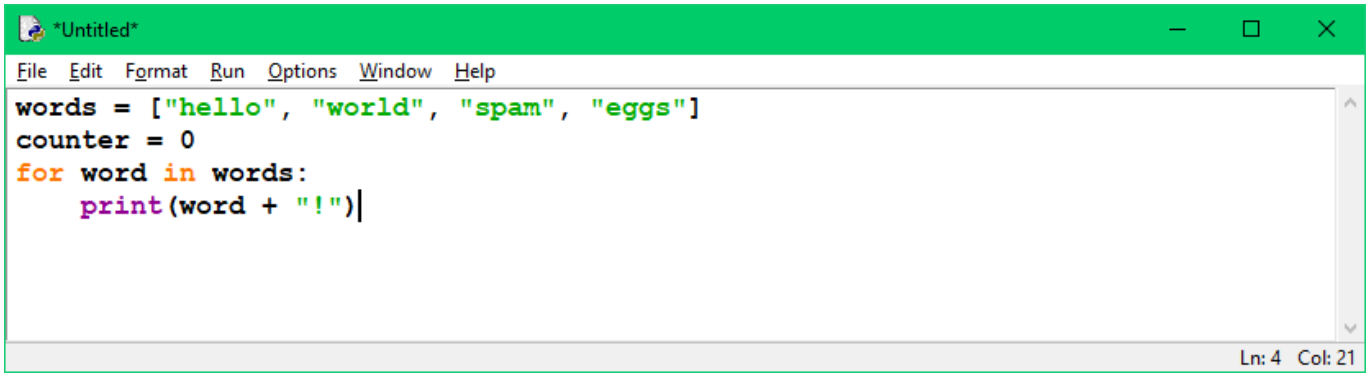

Рис. 49

Создаём простой калькулятор.

Написать программу, которая выполняет над двумя вещественными числами одну из четырех арифметических операций (сложение, вычитание, умножение или деление). Программа должна завершаться только по желанию пользователя.

Чтобы программа самопроизвольно не завершалась, в ней надо запустить бесконечный цикл. Выход из него будем осуществлять с помощью оператора break, если пользователь вводит определенный символ вместо знака арифметической операции.

Если пользователь ввел знак, который не является ни знаком арифметической операции, ни символом-"прерывателем" работы программы, то вывести сообщение о некорректном вводе.

Если был введен один из четырех знаков операции, запросить ввод двух чисел.

В зависимости от знака операции выполнить соответствующее арифметическое действие.

Если было выбрано деление, необходимо проверить не является ли нулем второе число. Если это так, то сообщить о невозможности деления.

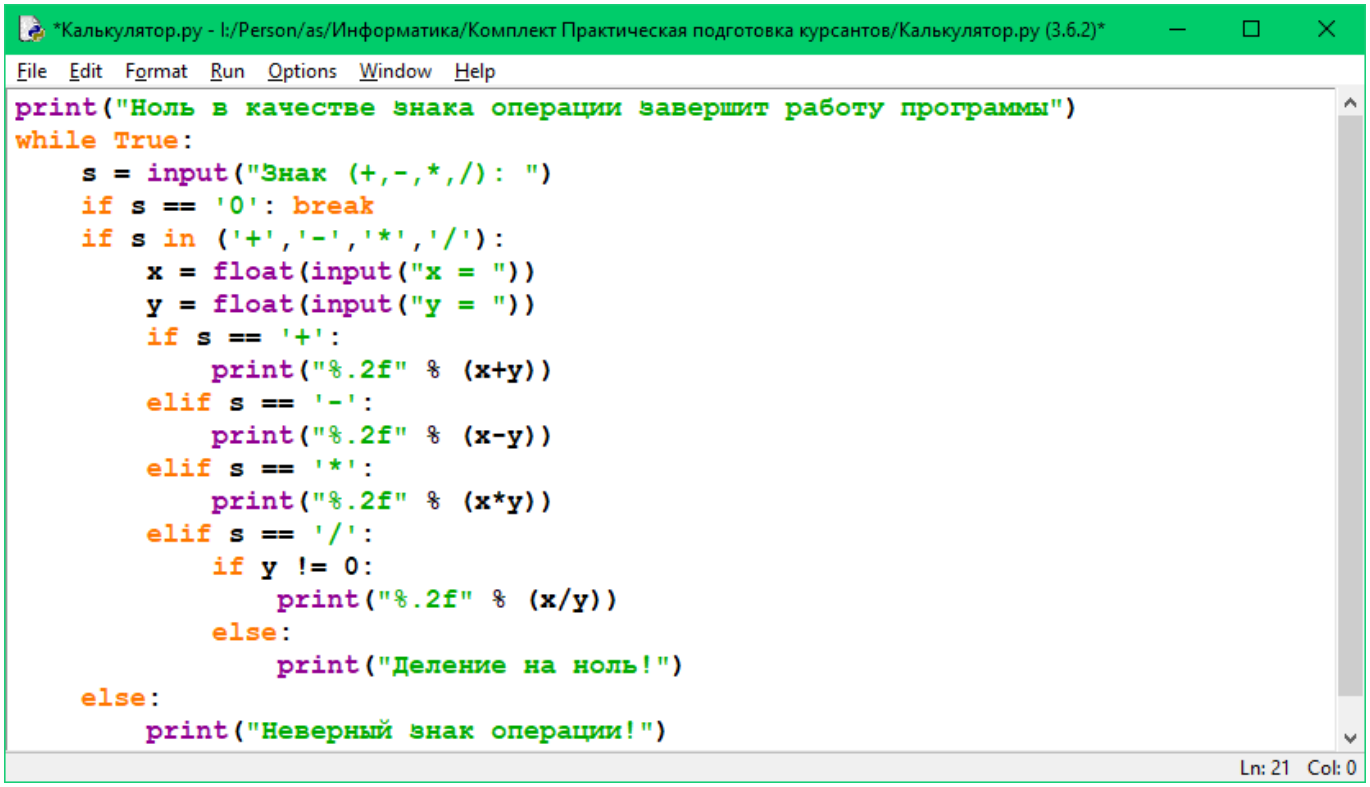

Рис. 50