Приложение 2.4

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский технологический университет «МИСиС»
(НИТУ «МИСиС»)

УТВЕРЖДАЮ
Проректор по безопасности
и общим вопросам

_____/ И.М. Исаев

«_24_» _12_____ 2021 г.

Сборник методических материалов по программе элективного курса, содержащий основные сведения и иллюстрации по темам программы, 1 п.л.

Москва 2021 г.

ДУМАЙ ЯРКО!

Издательский Дом
НИТУ «МИСиС»

store.misis.ru

№ 4578

МИСиС
Национальный исследовательский
технологический университет

А.И. Широков

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ «ПИТОН» (PYTHON)

Методические указания

№ 4578 МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

ЦЕНТР ПРОФЕССИОНАЛЬНОЙ НАВИГАЦИИ И ПРИЕМА

ПРОЕКТ «ИНЖЕНЕРНЫЙ КЛАСС В МОСКОВСКОЙ ШКОЛЕ»

А.И. Широков

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ «ПИТОН» (PYTHON)

Методические указания

Рекомендовано редакционно-издательским советом университета

МИСиС

Москва 2021

УДК 004.4
Ш64

Рецензент
канд. техн. наук, доц. *Д.В. Калитин*

Широков, Андрей Игоревич.
Ш64 Алгоритмизация и программирование на языке «Питон» (Python) : метод. указания / А.И. Широков. – Москва : Издательский Дом НИТУ «МИСиС», 2021. – 48 с.

В методических указаниях рассматриваются вопросы разработки программ для решения разнообразных задач на популярном алгоритмическом языке «Питон» (Python). Анализируются языковые конструкции языка, различные методы их использования. Для усвоения представленного материала предварительные знания в области программирования не требуются.

Предназначены для учащихся московских школ в рамках городского образовательного проекта «Инженерный класс в московской школе».

УДК 004.4

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Материал, представленный в методических указаниях, способствует формированию начальных навыков разработки программ на языке высокого уровня Python. В дальнейшем вместо этого термина будем использовать его русскоязычное имя – Питон. Целью данного пособия является развитие интереса к современным компьютерным технологиям, и прежде всего – к разработке программ, их практическому применению в проектной деятельности. Все это предполагает развитие у школьников навыков самостоятельного формулирования методов решения поставленных задач, выбора методов их реализации. Изучение программирования также способствует развитию алгоритмического мышления.

Освоившие материал школьники научатся разрабатывать простые программы для решения широкого круга задач. Они смогут ***узнать*** об основных понятиях информатики, методах, средствах и порядке разработки и отладки программ на этом языке, правилах использования языковых конструкций. Также они будут ***уметь*** выбирать подходящий алгоритм для решения разнообразных задач, работать в команде, использовать стандартные средства реализации алгоритмов.

Все примеры программ, приведенные в методических указаниях, выполнены в версии Python 3.9.7, скачанной с официального сайта https://www./downloads/windows/. Информацию о процедуре установки среды разработки программ можно найти на странице с адресом https://pythonworld.ru/osnovy/skachat-python.html.

О ЯЗЫКЕ ПРОГРАММИРОВАНИЯ «ПИТОН»

Разработку языка программирования «Питон» начал голландский программист Гвидо ван Россум во второй половине 1980-х годов [1, 2]. Но только в феврале 1991 г. исходный текст языка был опубликован в группе новостей alt.source (версия 0.9.0). Версия 1.0 была выпущена в 1994 г. Таким образом, этот язык появился позже многих наиболее популярных на сегодняшний день языков программирования. При этом он естественным образом впитывал идеи, заложенные в основу его «предшественников». Как и другие языки программирования, «Питон» постоянно развивается. Наиболее широкое распространение он получил после выхода 3-й версии.

На странице Википедии [2] предлагается такая характеристика: «*Python [7] (МФА: ['pʌɪθ(ə)n]; в русском языке распространено название питóн [8]) – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объем полезных функций*».

Язык программирования высокого уровня «Питон» поддерживает несколько парадигм программирования, в том числе структурную, объектно-ориентированную, функциональную, императивную и аспектно-ориентированную.

«Питон» является интерпретируемым, в противоположность языкам компилируемым, которые превосходят первые в быстродействии. Видимо поэтому существует несколько технологий реализации программ на нем. По одной из технологий исходный код преобразуется в специальный байт-код, который является результатом синтаксического и семантического анализа. Байт-код – это промежуточный вид программы, получаемый из исходного кода, но еще не предназначенный для автономного запуска (вне среды разработки). Имеются реализации с предварительной компиляцией в байт-код MSIL (Microsoft) или Java. Также есть средства, формирующие на основе исходного кода Python исполнимый код, выполняемый вне среды разработки.

Часто среди свойств рассматриваемого языка называют его ориентированность на повышение производительности разработки (не выполнения!) и читаемости кода. Несомненным достоинством этого языка является его многоплатформенность. На странице Википедии [2] приводится такое описание этого факта: «*Python портирован и работает почти на всех известных платформах – от КПК до мейнфреймов. Существуют порты под Microsoft Windows, практически все варианты UNIX (включая FreeBSD и Linux), Plan 9, Mac OS и Mac OS X, iPhone OS 2.0 и выше, Palm OS, OS/2, Amiga, HaikuOS, AS/400 и даже OS/390, Windows Mobile, Symbian и Android*».

Создатели языка программирования «Питон» следуют определенной философской концепции, которую создал Тим Петерс (Tim Peters). Она базируется на The Zen of Python («Дзен Питона», или «Дзен Пайтона»). После выполнения команды import this в оболочке «Питона» получим основные положения этой философской концепции. Приведем эти основные положения, переведенные на русский язык:

Красивое лучше, чем уродливое.

Явное лучше, чем неявное.

Простое лучше, чем сложное.

Сложное лучше, чем запутанное.

Плоское лучше, чем вложенное.

Разреженное лучше, чем плотное.

Читаемость имеет значение.

Особые случаи не настолько особые, чтобы нарушать правила.

При этом практичность важнее безупречности.

Ошибки никогда не должны замалчиваться.

Если не замалчиваются явно.

Встретив двусмысленность, отбрось искушение угадать.

Должен существовать один – и, желательно, только один – очевидный способ сделать это.

Хотя он поначалу может быть и не очевиден, если вы не голландец.

Сейчас лучше, чем никогда.

Хотя никогда зачастую лучше, чем прямо сейчас.

Если реализацию сложно объяснить – идея плоха.

Если реализацию легко объяснить – идея, возможно, хороша.

Пространства имен – отличная вещь! Давайте будем делать их больше!

Эти полушутливые фразы призывают программистов при написании исходного текста придерживаться определенного стиля. Эти положения, выраженные в абстрактной форме, дополняет свод правил написания программ PEP8 [3].

При написании методических указаний были использованы прежде всего источники [1, 4–7].

1. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ «ПИТОН». С ЧЕГО НАЧАТЬ?

Запустив программу, позволяющую использовать конструкции языка «Питон», можно работать в двух режимах: оболочки (интерпретатора) и среды разработки. В первом режиме в строку вводятся одиночные команды языка, которые выполняются сразу после нажатия на клавишу <Enter>, расположенную на клавиатуре компьютера. На рис. 1.1 представлено окно диалога при работе в таком режиме.

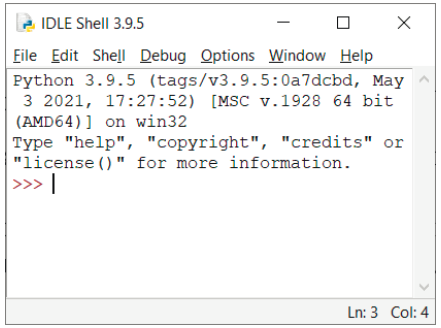


Рис. 1.1. Начальный вид экрана оболочки языка «Питон»

Как видно на рис. 1.1, окно называется IDLE Shell 3.5.9. Три знака «>» определяют строку, в которой можно вводить команды.

Во втором режиме несколько операторов записываются в файл с расширение *.py. Затем по специальной команде (Run – выполнить) все операторы, записанные в файле, исполняются (если синтаксис программы верен). Для выполнения такой команды можно также нажать на клавишу F5.

Язык программирования предназначен для записи алгоритма решения задачи. В ней используется информация, хранимая в памяти компьютера.

Для общения с пользователем программы в «Питоне», как и в других языках программирования, существуют операторы ввода и вывода. Действие по выводу информации на экран компьютера выполняет оператор print (рис. 1.2).

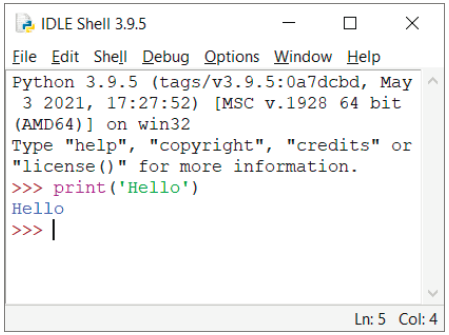


Рис. 1.2. Оператор print

В этом примере, выполненном в режиме оболочки, выводится строка. Приведем пример ввода информации с клавиатуры и вывода информации на экран. Для того чтобы перейти в режим подготовки программ и сохранения исходного текста программы в файле, надо в режиме оболочки исполнить команду *File* → *New File* (рис. 1.3).

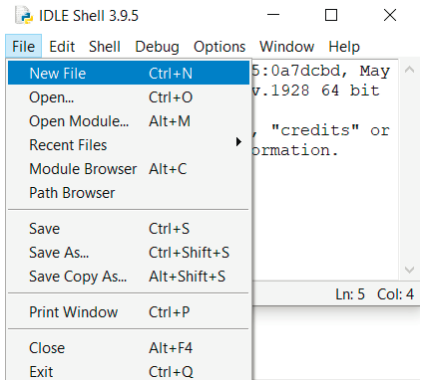


Рис. 1.3. Команда для перехода в режим отладчика программ на языке «Питон»

После выполнения приведенной выше команды открывается окно, вид которого приведен на рис. 1.4. Наберем в нем текст.

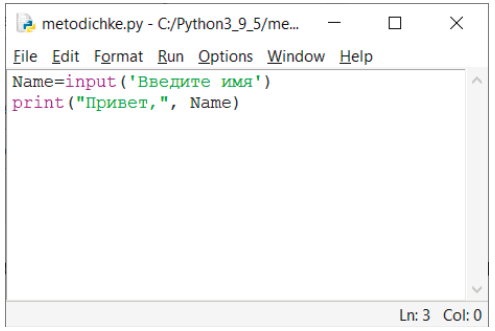


Рис. 1.4. Окно среды языка «Питон» для подготовки текста программы

Перед исполнением команды набранный текст надо сохранить в файле. Это достигается с помощью команды Сохранить (Save, <Ctrl+S>), представленной на рис. 1.5.

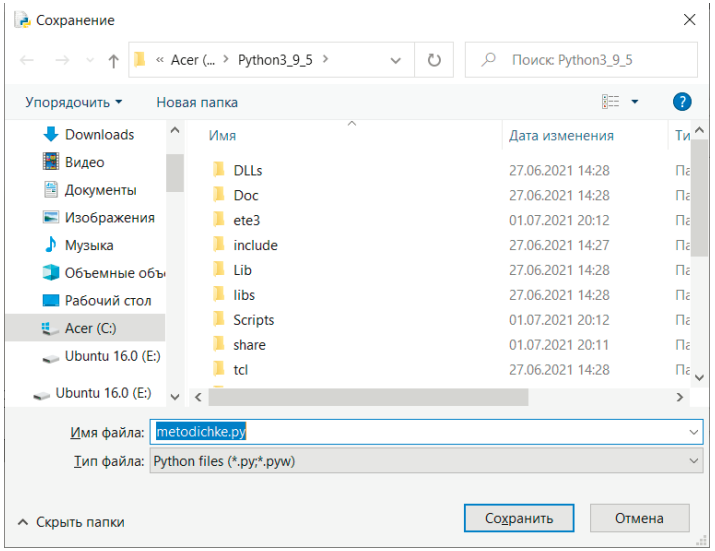


Рис. 1.5. Диалоговое окно команды сохранения подготовленного файла программы

Теперь программу можно запустить. Надо нажать на клавишу F5 или выполнить команду *Run → Run Module*. Результат выполнения программы отобразится в окне оболочки (рис. 1.6).

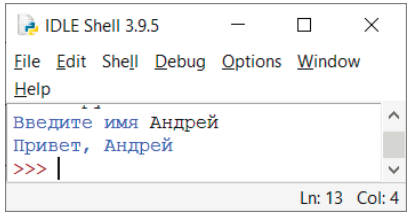


Рис. 1.6. Результат выполнения программы с вводом и выводом информации

Отметим, что разная информация, выводимая в диалоговом окне, выделяется различными цветами (рис. 1.7).

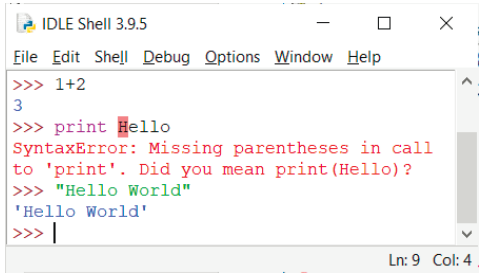


Рис. 1.7. Пример выделения цветом информации в диалоговом окне

Различные элементы диалогового окна выделяются такими цветами:

Ключевые слова – оранжевым;

Символьные данные – зеленым;

Вывод – синим;

Комментарий, ошибки – красным.

Настройки, связанные с выделением цветом элементов диалогового окна, можно изменить с помощью команды *Option → Configure IDLE* (рис. 1.8).

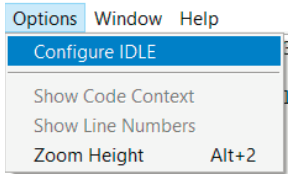


Рис. 1.8. Команда настройки параметров среды IDLE

Далее надо выбрать вкладку *Highlights* (рис. 1.9).

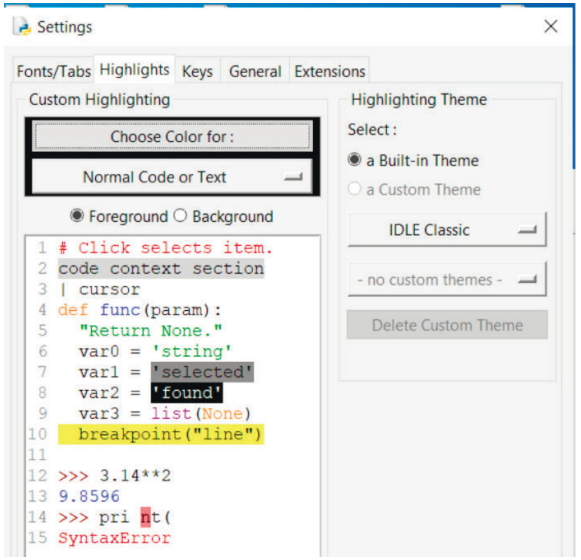


Рис. 1.9. Окно определения цветов диалога

Перейдем к описанию основных конструкций языка программирования «Питон».

2. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА ПРОГРАММИРОВАНИЯ «ПИТОН»

2.1. Константы, переменные

Программы, написанные на языке «Питон», как и в случае реализации многих других языков программирования, могут обрабатывать данные разных типов – символьные, логические и числовые. Последние в свою очередь могут быть: целыми, вещественными и комплексными.

Для записи целых чисел используются цифры и знаки «+» и «–». Вещественные числа могут быть записаны двумя способами – с фиксированной точкой и в экспоненциальной форме. Приведенный на рис. 2.1 скриншот иллюстрирует вышесказанное.

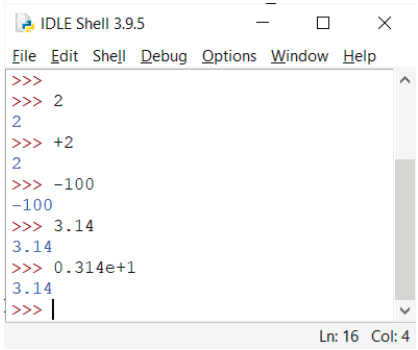


Рис. 2.1. Форма представления чисел в «Питоне»

При записи целого положительного числа со знаком «+» он не выводится. Далее число Пи записывается двумя способами – сначала с фиксированной точкой, а затем в экспоненциальной форме.

Интересным фактом является то, что в «Питоне» можно получить очень большие значения целых чисел. Например, при объявлении целой переменной в языке С в качестве длинного целого (8 байт) диапазон ее представления такой: от (–9 223 372 036 854 775 808) до (+9 223 372 036 854 775 807), т.е. он содержит 18 знаков [8]. В «Питоне» можно получить

целое число с гораздо бóльшим количеством знаков. Следующий пример (рис. 2.2) демонстрирует этот факт. Переменной x присваивается значение числа 2 147 483 647 в степени 2000. Как видно, в записи этого числа содержится 5490 строк.

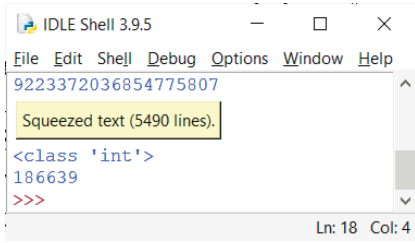


Рис. 2.2. Форма представления целых чисел в «Питоне»

Для вещественных чисел применяется ограничение на допустимое максимальное значение [9]. Если в оболочке ввести сначала команду `import sys`, а затем `sys.float_info.max`, то поучим максимальное вещественное число `1.7976931348623157e+308`. Таким образом, в записи числа может быть 17 верных знаков, умноженных на 10 в степени +308.

Заметим, что если в оболочке ввести 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 (6 слагаемых), то получим 0,6. А если 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 + 0,1 (10 слагаемых), то получим 0,9999999999999999. Таким образом, приближенное значение равно 1,0.

Теперь рассмотрим, как в «Питоне» следует использовать комплексные числа. Питон поддерживает комплексные числа и операции над ними. Они выглядят так: a + jb. Здесь a и b – целые или вещественные числа. Сама по себе мнимая единица не может быть использована, но можно записать ее в виде 1j. В приведенном ниже фрагменте кода «Питона» (результат представлен на рис. 2.3) приводятся примеры способов задания и выполнения математических операций (сложения, вычитания, умножения, деления) над комплексными числами. Показано, как получить действительную и мнимую части таких чисел, а также сопряженное число. Продемонстрировано, как комплексные числа можно сравнивать, а как нельзя.

```
x = complex(1, 2)
print('x  ',x)
y = complex(3.0, 4)
print('y  ',y)
z = x + y
print(,z (x+y}:  ,, z)
zm = x * y
print('  (x*y):  ',zm, ' )
zd = x / y
print('  (x/y):  ',zd)
print(' Сопряженное z:', z.conjugate()) # Сопря-
женное число
print(' real z: ',z.real) # Действительная часть
print(' imag z: ',z.imag) # Мнимая часть
print(, abs(z): ,, abs(z)') # Модуль числа
print(' x==y:  ', x == y) # Комплексные числа
можно сравнивать так
print('NO x > y ') # Комплексные числа НЕЛЬЗЯ
сравнивать так
```

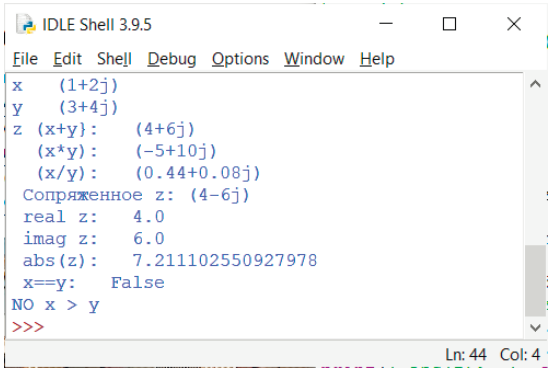


Рис. 2.3. Форма представления комплексных чисел в «Питоне»

Замечание. В «Питоне», как и в других языках программирования, можно в исходный текст добавлять комментарии. Они бывают двух типов. Строчный комментарий фиксируется одним знаком #, а многострочные комментарии начинаются тремя одинарными кавычками и заканчиваются ими же, например:


```
# Это комментарий, записанный до конца строки
''' Многострочный комментарий
может занимать несколько строк
'''
```

В языках программирования традиционно символьные строки заключаются в двойные кавычки. В «Питоне» для этого используются как двойные, так и одинарные кавычки (рис. 2.4). Выведем сообщение, предварительно соединив две заданные заранее символьные строки. В рассмотренных далее примерах показано равноправие двойных и одинарных кавычек при формировании символьных констант. В первом операторе переменная инициируется одинарными кавычками, а во втором – двойными:

```
s1 = 'Hello, '
s2 = "world!"
```

Далее переменные объединяются операцией «+» и полученное значение выводится на экран:

```
S12 = s1 + s2
print(S12)
```

В двух следующих примерах показано, что кавычки одного типа могут располагаться внутри других:

```
S0 = 'Hello, "world"'
print(S0)
S0 = "Hello, 'world'"
print(S0)
```

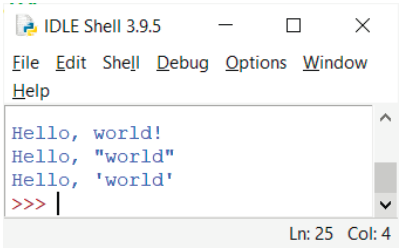


Рис. 2.4. Использование двойных и одинарных кавычек

В «Питоне» нет операторов определения типа переменной. Эту функцию выполняет оператор присваивания (помимо обычного действия – смены значения переменной). Также тип переменной определяется при вводе значения, например, с клавиатуры.

«Питон» обладает свойством динамического переопределения типа переменной. Это можно продемонстрировать с помощью фрагмента программы и полученных результатов (рис. 2.5):

```
v=1
print(v, '          ',type(v) )
v=1.41
print(v, '      ', type(v) )
v=0.141e+1
print(v, '      ',type(v) )
v=complex(v)
print(v, '', type(v) )
v="0.141e+1"
print(v, ' ',type(v) )
```

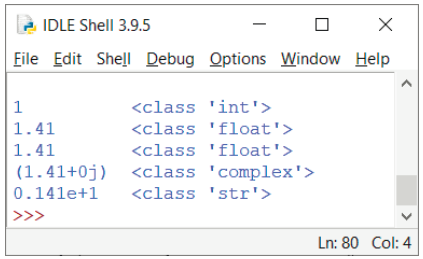


Рис. 2.5. Динамическое переопределение типа переменной

Теперь расширим представление об операциях в «Питоне».

2.2. Операции в «Питоне»

При программировании на языке «Питон» следует знать правила использования выражений. Бóльшая часть кода содержит именно их. Например, выражениями являются 1 + 2,3 – x; 4 – sin(x), x > y. Как видно, в них константы, имена переменных или функций соединяются знаками операций. Можно выделить следующие группы операций:

1) арифметические;
2) присваивание;
3) сравнения;
4) логические.

Из-за ограниченного объема методических указаний, в них рассмотрены не все группы операций. Полный состав операций представлен в источниках [10, 11].

В группе арифметических операций содержится семь операций (табл. 2.1).

Таблица 2.1

Арифметические операции

Обозначение арифметической операции	Описание операции
+	Сложение
–	Вычитание
*	Умножение
/	Деление
**	Возведение в степень
//	Деление (целочисленное)
%	Вычисление остатка

В качестве примера использования арифметических операций приведем следующий фрагмент программы:

```
'''
```

Далее показано, что два оператора можно записать в одной строке используя символ «;»:

```
'''
c1=12; c2=5
print('числа', c1, '  и ', c2)
r1=c1+c2
print(«сложение: «, r1)
r2=c1-c2
print(«вычитание: «, r2)
r3=c1*c2
print(«умножение: «, r3)
r4=c1/c2
print(«деление1: «, r4)
```

```
r5=c1**c2
print(«возведение в степень: «, r5)
r6=c1//c2
print(«деление2: «, r6)
r7=c1%c2
print(«вычисление остатка: «, r7)
```

После выполнения приведенного выше фрагмента программы получаются следующие результаты (рис. 2.6).

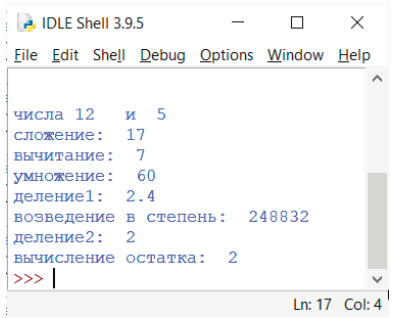


Рис. 2.6. Результат использования арифметических операций

Обратим внимание на то, что выполнение операции «деление1» дает вещественное число, в отличие от операции «деление2» – целочисленного деления. В «Питоне» узнать тип полученного значения можно путем использования функции `type`. Продолжим пример с полученными значениями переменных (рис. 2.7).

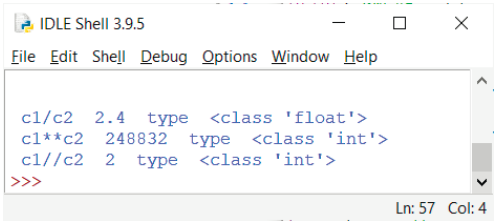


Рис. 2.7. Результат вывода типов переменных

В «Питоне» можно использовать несколько операций присваивания, которые представлены в табл. 2.2 (в таблице использована терминология, приведенная в источнике [12]).

Таблица 2.2

Операции присваивания

Обозначение операции	Описание операции
=	Простое присваивание
+=	Оператор присваивания «сложение И» (складывает оба операнда, а затем присваивает значение правому)
−=	Оператор присваивания «вычитание И» (вычитает правый операнд из левого и затем присваивает значение левому)
*=	Оператор присваивания «умножение И» (умножает оба операнда, а затем присваивает левому)
/=	Оператор присваивания «деление И» (делит левый операнд на правый, а затем присваивает значение левому)
**=	Оператор присваивания «степень И» (вычисляет значение левого операнда в степени правого, а затем присваивает левому)
//=	Оператор присваивания «целочисленное деление И» (выполняет целочисленное деление левого и правого операндов, а затем присваивает левому)
%=	Оператор присваивания «модуль И» (вычисляет модуль деления левого операнда на правый, а затем присваивает левому)

Приведем примеры использования операций группы присваивания:

```
A=13; B=5
print('числа ', A, '  ', B)
A+=B
print('A+=B ', A, ' B ', B)
A-=B
print('A-=B ', A, ' B ', B)
A*=B
print('A*=B ', A, ' B ', B)
A//=B
print('A//=B ', A, ' B ', B)
A%=B
print('A%=B ', A, ' B ', B)
```

```
A**=B
print('A**=B ', A, ' B ', B)
A/=B
print('A/=B ', A, ' B ', B)
```

На рис. 2.8 представлен результат выполнения этого фрагмента программы.

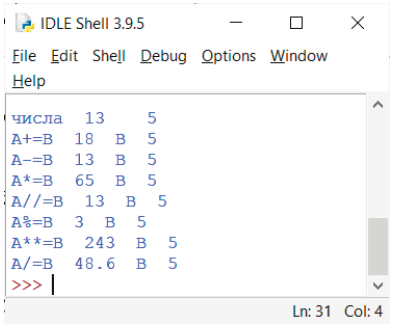


Рис. 2.8. Результат выполнения операции присваивания

В операторе цикла и условном операторе используются логические выражения, построенные с использованием операций сравнения. Их результатом всегда является логическое значение – True или False. Список операций сравнения приведен в табл. 2.3.

Таблица 2.3

Операции сравнения

Обозначение операции	Описание операции
==	Возвращает True, если оба значения равны, иначе False
!=	Возвращает True, если оба операнда не равны, иначе False
>	Возвращает True, если левый операнд больше правого, иначе False
<	Возвращает True, если левый операнд меньше правого, иначе False
>=	Возвращает True, если левое значение больше или равно правому, иначе False
<=	Возвращает True, если левое значение меньше или равно правому значению, иначе False

Ниже представлен фрагмент программы, поясняющий использование операндов сравнения:

```
t=10; f=6;
print('t ',t, " t==t ", (t==t))
print('t ',t, '  f ',f, " t==f ", (t==f))
t=10; f=6;
print('t ',t, '  f ',f, " t!=f ", (t!=f))
t=10; f=6;
print('t ',t, '  f ',f, " t>f ", (t>f))
```

На рис. 2.9 представлен результат выполнения этого фрагмента программы.

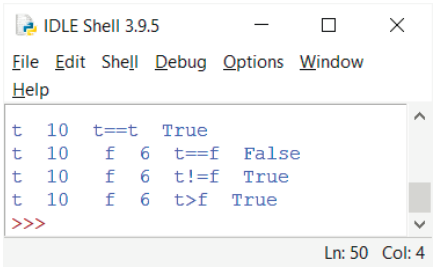


Рис. 2.9. Результат выполнения операции сравнения

И наконец, последняя группа операндов, рассматриваемых в этом разделе, – логические операции (табл. 2.4). Они строятся на основе логических выражений или операций сравнений, имеющих значение True или False.

Таблица 2.4

Логические операции

Обозначение операции	Описание операции
not	Логическое НЕ (изменяет логическое значение на обратное, иначе False)
and	Логическое И (получает значение True только если все значения True, иначе False)
or	Логическое ИЛИ (получает значение True только если хотя бы одно значение True, иначе False)

Операторы and и or могут содержать два и более простых логических выражения. Ниже приведен пример использования этих операторов:

```
Str1='Hello' ; Str2='hello'
print('Str1 ', Str1, '  Str2 ', Str2)

tf1=(Str1 == Str2)
print('tf1: Str1 == Str2                    ',tf1)

tf2=(Str1[2:3] == Str2[2:3])
print('tf2: Str1[2:3] == Str2[2:3]          ',tf2)

tf3=not(Str1[2:3] == Str2[2:3])
print('tf3: not(Str1[2:3] == Str2[2:3])     ',tf3)

a=1; b=2; c=3
tf4 = (a==1 and b==2 and c==3)
print('tf4: a==1 and b==2 and c==3          ',tf4)

tf5 = (a==2 or b==2 or c==2)
print('tf5: a==2 or b==2 or c==2            ',tf5)
```

Результат применения операторов and и or представлен на рис. 2.10.

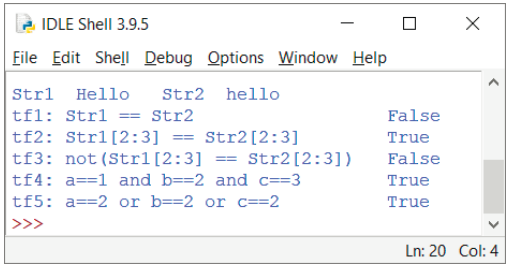


Рис. 2.10. Результат применения логических операций

Если выражение содержит несколько операций, то они выполняются в соответствии с установленными приоритетами.

Мы можем создать группу выражений, используя круглые скобки. В этом случае сначала вычисляются действия в группе, затем они участвуют в последующих вычислениях. Если операции имеют одинаковый уровень приоритета, действия выполняются слева направо.

В табл. 2.5 приведены приоритеты операций в порядке убывания приоритета.

Таблица 2.5

Приоритет операций

Обозначение операции	Описание операции
**	Возведение в степень
*, /, //, %	Умножение, деление, взятие модуля
+, –	Сложение, вычитание
==, !=, >, <, >=, <=	Сравнение
=, +=, -=, *=, /=, //=, %=	Присваивание
not, and, or	Логические

Приведем пример, в котором показано, что операции одного приоритета выполняются слева направо, однако скобки отменяют такой порядок (рис. 2.11):

```
a=15; b=2
print('  a=15; b=2  ')
print('a) a*b+b                ', a*b+b)
print('b) a*(b+b)              ', a*(b+b))
print('c) a/b   ', a/b)
print('d) 1-/  2-// a/b//b     ', a/b//b)
print('e) 1-/  2-// a/(b//b)   ', a/(b//b))
print('f) 1-// 2-/  a//b/b     ', a//b/b)

print('g)  5+4*3**2  ',  5+4*3**2)
print('h) (5+4)*3**4 ', (5+4)*3**4)
```

В строках a) и b) показано, что операции в скобках выполняются в первую очередь. Далее в программе над этими же двумя целыми числами выполняется две операции деления (обычное и целочисленное) в разном порядке. В случае d) сначала записано обычное деление (результат 7,5), а затем целочисленное (результат – 3,0, так как дробная часть отбрасывается). В стро-

ке e) порядок такой же, но целочисленное деление записано в скобках, оно выполняется первым (результат 15,0). А в случае f) сначала записано целочисленное деление (результат 7) и далее обычное деление (результат 3,5). В строках g) и h) еще раз показано, как скобки изменяют порядок вычислений.

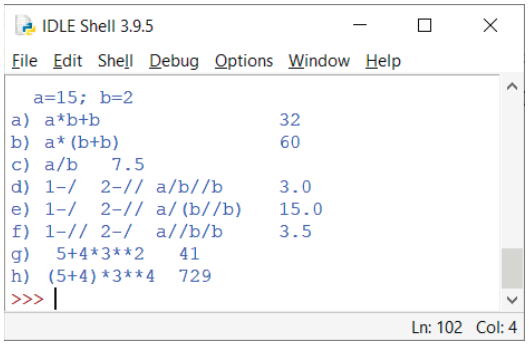


Рис. 2.11. Приоритет операций

В следующем разделе опишем операторы языка программирования «Питон».

2.3. Две основные алгоритмические конструкции в программировании

2.3.1. Условный оператор

Перейдем к анализу возможностей условной конструкции. Она начинается с ключевого слова `if`, после которого записывается условие. Как было сказано выше, оно является выражением, которое приобретает только два возможных значения – истина (ключевое слово `True`) или ложь (ключевое слово `False`).

В примере, рассмотренном далее, эти значения выводятся (рис. 2.12) в зависимости от того, которая из двух переменных имеет большее значение. Точнее, если большее значение имеет первая переменная, то выводится «1e>2го», в противоположном случае – «1e<=2го». И затем выводится `True` или `False`.


```
f=2; t=1
if f>t :
    TF=True
    print('1e>2ro\n', )
else:
    TF=True
    print('1e<=2ro\n', )
print(' TF ',TF, '\n')
```

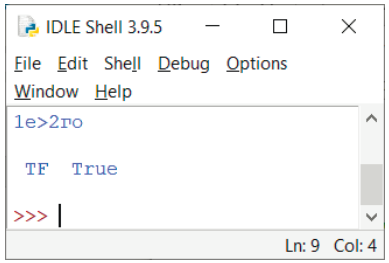


Рис. 2.12. Пример с ключевым словом `if`

Расширим представление о логических константах. Если выполнить представленный ниже фрагмент программы, то получим результат, показанный на рис. 2.13:

```
if ( 1 ): # Можно использовать скобки
    print(' 1: условие не 0')
else:
    print(' 2: условие 0')
```

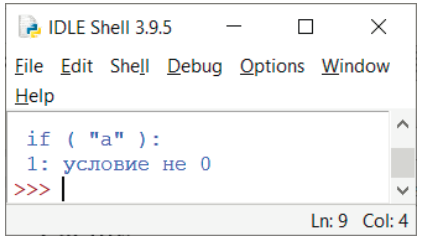


Рис. 2.13. Пример с условием «`1`»

Но если в условии записать 0, то будет выведено 2: условие 0. Это означает, что после ключевого слова if любое число, не равное 0, равносильно константе True, а 0 – False.

Оператор if реализует конструкцию, называемую условной. При этом есть две ее разновидности – разветвление и обход. При реализации конструкции первой разновидности – разветвлении – если значение условия истинно, выполняется одна группа действий, а если оно ложно – другая. При реализации конструкции второй разновидности – обходе – ситуация следующая: если условие истинно, выполняется группа действий, если оно ложно – не выполняется ничего. Приведем фрагмент программы (результат ее выполнения представлен на рис. 2.14), в котором задача нахождения максимального значения из двух чисел реализована с помощью конструкции «обход». При этом первое из двух чисел сначала присваивается максимальному значению. А затем оно сравнивается со вторым значением и, если нужно, заменяется на его значение.

```
print('Input two value')
c1=int(input() )
# Оператор input вводит символьное значение,
# затем оно преобразуется в целый тип
c2=int(input() )
max = c1
if c2>max:
    max = c2
print(c1, c2, '  max = ',max)
```

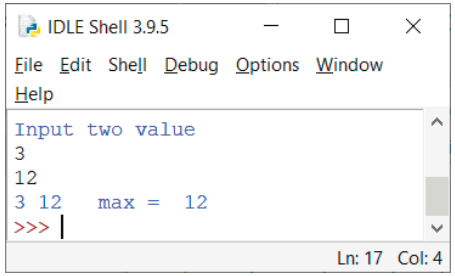


Рис. 2.14. Пример вычисления максимума из двух чисел с обходом

Если использовать условную конструкцию с оператором if (разветвление), то в целом, при многократном выполнении программы с разными исходными данными, действий будет меньше (при обходе иногда будут выполняться два присваивания, а при разветвлении всегда только одно):

```
print(Input two value')
c1=int(input() )
# Оператор input вводит символьное значение,
# затем оно преобразуется в целый тип
c2=int(input() )
if c2>c2:
    max = c2
 else
    max = c1
print(c1, c2, '  max = ',max)
```

В «Питоне», как и в других языках программирования, предлагается более сложная условная конструкция. Например, в задаче сравнения двух чисел выделяется не два случая, в три:

1) первое число больше второго;
2) оба числа равны;
3) второе число больше первого.

Такую задачу можно реализовать разными способами. При первом способе используется вложенное разветвление:

```
a=2; b=2
print('a= ',a, '  b= ',b)
if a == b:
    print("a=b")
else:
    if a>b:
        print('a>b')
    else:
        print('a<b')
```

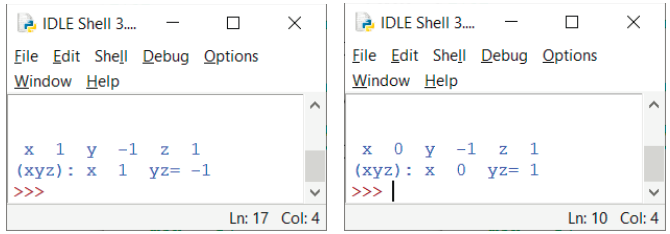
Второй способ предполагает использование конструкции if-elif-else:

```
a=3; b=2
if a == b:
    print('a=b')
elif a>b:
        print('a>b')
else:
    print('a<b')
```

Для присваивания переменной двух различных значений, в зависимости от некоторых условий в «Питоне» имеется специальная конструкция. В приведенном далее примере переменная `YZ` получает значение в зависимости от значения `X`. Используется трехместная конструкция `if/else`.

```
y=-1; z=1
print('\n y ',y, ' z ',z)
x=0
yz = y if x else z
print('2(xyz): x ',x, ' yz=', yz)
```

На рис. 2.15 приведен результат выполнения приведенного выше фрагмента программы.



а *б*

Рис. 2.15. Трехместная конструкция с оператором `if`:
а – при x = 1; *б* – при x = 0

Используемые ранее условия являются простыми. Из них могут образовываться сложные условия. Для этого используются логические операции `not`, `and` и `or`, о которых уже упоминалось ранее. Первая операция изменяет истинностное

значение на противоположное. Логическая операция `and` связывает два условия и будет истинной, если истинны оба условия. Логическая операция `or` связывает два условия и будет истинной, если истинно хотя бы одно из них. Приведем следующий пример:

```
a=1
print('a=',a)
if a==1:
    TF= (a==1)
    print('  1      : TF ', TF)
a=2
print('a=',a)
if not a==1:
    TF= not(a==1)
    print('  2(not): TF ', TF)
```

В приведенном выше фрагменте программы переменная сначала получает значение 1, а после проверки (`a` равно 1) выводится значение `True`. Далее a присваивается значение 2 и проверяется противоположное условие (`a` не равно 1). И снова будет выведено значение `True`. Результат выполнения этого фрагмента программы представлен на рис. 2.16.

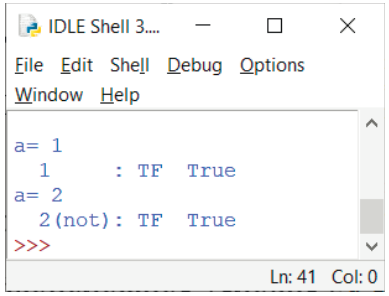


Рис. 2.16. Пример использования оператора `if`
с логической операцией `not`

Следующие два примера повторяют рассмотренное в разделе «Операции».

```
a=1; b=2
print('a=',a, ' b=',b)
if a==1 and b==2:
    TF=(a==1) and (b==2)
    print('1(and): TF ', TF)
```

В этом фрагменте будет выведено значение `True` после проверки условия двух равенств. Первая переменная должна быть равна числу 1, а вторая – числу 2.

Ниже представлен еще один фрагмент программы, в котором используется логический оператор `or`:

```
a=1; b=1
print('\na=',a, ' b=',b)
if a==1 or b==2:
    TF=(a==1) and (b==2)
    print('2(or) : TF ', TF)
```

В «Питоне» сложные условия могут связывать более двух простых условий:

```
a=1; b=2; c=3; d=4
if a==1 and b==2 and c==3 and d==4:
    TF = a==1 and b==2 and c==3 and d==4
    print('3(
```

Еще в одном примере условная конструкция используется для определения принадлежности заданной точки одному из двух заданных интервалов (рис. 2.17). Значения чисел, определяющие интервалы, следующие: a = –2,0; b = 0,0; c = 2,0 и d = 5,0.

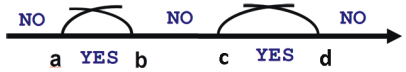


Рис. 2.17. Принадлежность точки любому из двух заданных интервалов

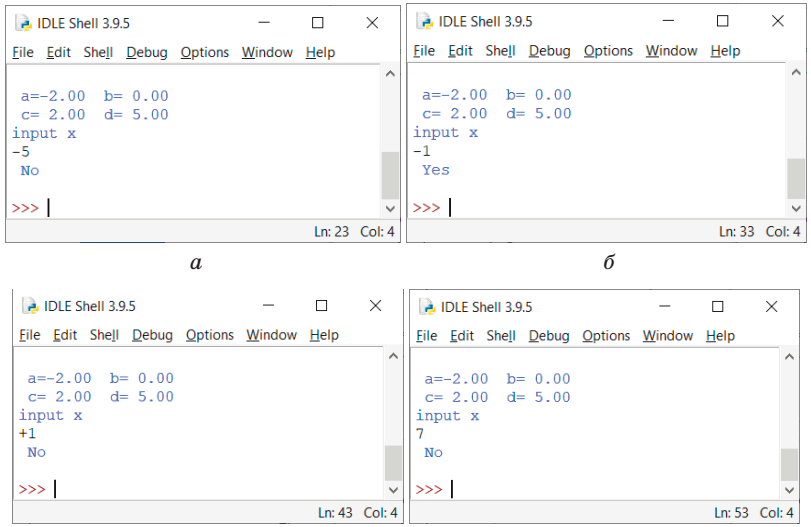
Программа, решающая поставленную задачу, может быть такой. Для вывода значений переменных, используемых в программе, применяются средства форматирования в стиле C#. Например, в первом операторе выводятся две переменные. Их номера в списке вывода (внутри ключевого слова `format`) 0

и 1. Оба значения выводятся в формате `5.2f`, что предполагает 5 знаков (с учетом десятичной точки и знака «–») и с двумя знаками в дробной части.

```
a = -2.0;  b = 0.0;  c = 2.0;  d = 5.0;
print(" a={0:5.2f}  b={1:5.2f}".format(a, b))
print(" c={0:5.2f}  d={1:5.2f}".format(c, d))

print('input x')
x = float(input())
TF1 = (x >= a) and (x <= b)
TF2 = (x >= c) and (x <= d)
if (TF1 or TF2):
    print(" Yes\n")
else:
    print(« No\n»)
```

Приведем результаты выполнения программы при разных значениях координаты точки по оси x (рис. 2.18).



а *б*

в *г*

Рис. 2.18. Решение задачи принадлежности двум интервалам при разных значениях координаты точки по оси x:
а – при x = –5; *б* – при x = –1; *в* – при x = +1; *г* – при x = 7

2.3.2. Циклические конструкции while и for. Конструкция range

Рассмотрим, как в «Питоне» реализуется циклическая конструкция. В отличие от других языков программирования, в «Питоне» циклическая конструкция имеет две формы – `while` и `for`. В приведенном ниже примере представлена первая из них. Решается задача вычисления суммы пяти членов натурального ряда.

```
# 1+2+3+4+5
a=1; s=0
while a<=5:
    s=s+a
    print(' i ',a, '   S ',s)
    a=a+1
print(' all 1 s= ',s)
```

Используя предыдущий алгоритм, можно решить другую задачу, заменив лишь способ вычисления очередного слагаемого. Вычисляется сумма пяти *четных* значений натурального ряда. Ниже показано, как это можно сделать с использованием циклической конструкции `for` и конструкции `range`:

```
# 2+4+6+8+10
i=1; s=0
# while e<=5:
for i in range(1,6):
    e=i*2
    s=s+e
    print(' i ',i, '   e ',e, '   s ',s)
print(' all  s= ',s)
```

Результат выполнения приведенной выше программы представлен на рис. 2.19. Выводится не только окончательный результат, но и значения переменных, изменяемых в цикле, что облегчает анализ правильности выполнения программы.

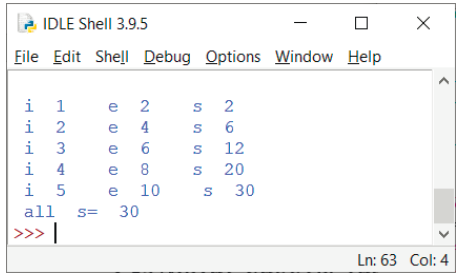


Рис. 2.19. Пример цикла `for` с выводом промежуточных результатов

Подробно оператор цикла `for` будет рассмотрен далее. А сейчас вернемся к оператору цикла `while`. После ключевого слова записывается условие, рассмотренное ранее при изучении условной конструкции. Напомним, что оно имеет два значения – `True` (истина) и `False` (ложь), но имеет числовые эквиваленты. Оператор выполняется так. Если условие истинно, то выполняются действия, записанные в нем (тело цикла). Как только условие становиться ложным, цикл прекращается. Напомним, что любое число, не равное 0, – это ложное условие. Если условие не равно 0 и не меняется, цикл будет выполняться бесконечно. В случае если условие сразу равно 0, цикл не выполнится ни разу. А вот если начальное значение условия равно –2 и изменяется на +1, цикл выполнится два раза (пока начальное значение условия не станет равным 0). Сказанное выше иллюстрируется следующим фрагментом программы:

```
# Бесконечный цикл
#while 1:
#    print('1')
#print('all')
print('all1: Бесконечный цикл\n')

# Нет цикла
a=0
while a:
    print('a ', a)
print('all2: Нет итераций\n')
```

```
  # Цикл выполняется два раза для условия -2 и -1.
  a=-2
  while a:
      print('a ', a)
      a=a+1
  print('all3: Две итерации a=',a,'\n')

  # Пример программы с циклом и выводом текста самой
программы
  print('Текст программы')
  print("tf=True  \nwhile  tf:\n       print('tf  ',
tf)\n    tf=False\
  print('all4: Одна итерация tf=',tf)  \nprint('tf
', tf)        ")
  tf=True
  while tf:
      print('\n Результат:\ntf ', tf)
      tf=False
  print('all4: Одна итерация tf=',tf)
  print('tf ', tf)
```

Такой цикл выполнится один раз, пока условие не станет ложным (`False`), что представлено на рис. 2.20. В тексте программы показано, как записать длинное символьное данное в исходном коде с разбиением на строки. Для этого используется символ «обратная косая черта» – «`'\'`». А для переноса текста при выводе используется пара символов «`\n'`».

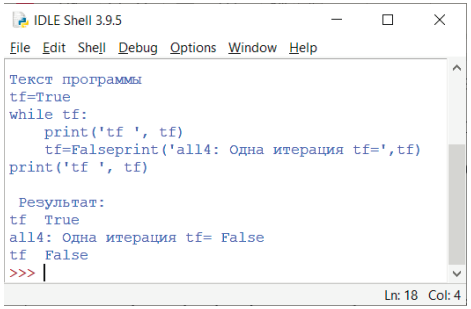


Рис. 2.20. Пример программы с циклом и выводом текста самой программы

Приведем еще один пример, в котором вычисляется сумма последовательности чисел, каждый элемент которой вычисляется так: a(i) = 1/i, где i – номер элемента в последовательности. Количество слагаемых определяется следующим условием: «пока очередное слагаемое по модулю не станет меньше или равно заданному числу». В нашем примере это 0,1. Отметим, что текст сформирован на основе программы, вычисляющей сумму 5 четных целых чисел. В ней изменен только один оператор в цикле – он вычисляет очередное слагаемое. Используя этот «шаблон», можно решать много задач.

```
import math
i=1; s=0; e=1
while math.fabs(e)>0.1:
    e=1/i # вычисление очередного слагаемого
    s=s+e
#     print(' i ',i, '   e ',e, '   s ',s)
    i=i+1
print(' all  s= ',s)
```

Заметим, что в программе используется функция «модуль числа». Она расположена в модуле (библиотеке) `math`. Поэтому ее надо подключить (`import math`) и обратиться к функции этой библиотеки так: `math.fabs`.

Продолжим рассматривать циклические структуры. Узнаем, как они реализуются оператором `for`. В первом примере создается список из трех символьных значений. Далее в цикле выводятся как значения всех элементов списка, так и их номера. Для этого используется функция `enumerate()`. Она применяется для объектов с множественными значениями (коллекциями) и генерирует два числа – номер элемента и его значение.

```
sub=('1','2','3')
for i,sub in enumerate(sub):
    print(i,sub)
```

После выполнения этого фрагмента программы будет выведена следующая информация:

```
0 1
1 2
2 3
```

Рассмотрим еще две языковые конструкции, которые применяются в циклах, – `continue` и `break`. Первая переходит к следующей итерации цикла, без выполнения операторов, записанных после `continue`. Но цикл будет продолжен, пока не наступит условие окончания цикла. В отличие от `continue`, `break` завершает цикл сразу. Следующий пример иллюстрирует сказанное выше. В нем в цикле анализируется содержимое текстовой строки – является ли очередной ее элемент символом «c»:

```
sub1='abccccc'
#for i,sub1 in sub1: Показано, что нужно получать два элемента.
print('continue')
for i in sub1:
    if i == 'c':
        continue
    print(i)

#break
print('break')
sub2='ccab'
for i in sub2:
    if i !='c':
        break
    print(i)
print('all')
```

В скриншоте (рис. 2.21) показаны результаты выполнения последнего исходного текста.

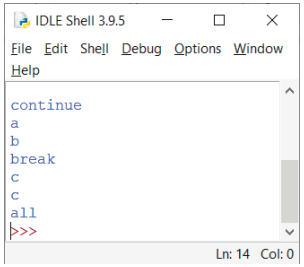


Рис. 2.21. Результат применения языковых конструкций `continue` и `break`

В заключение раздела покажем, как следует использовать конструкцию `range`, часто применяемую вместе с `for`. Эта конструкция генерирует множество целых чисел, вычисляемых на основе начального и предельного значений, а также шага приращения значений. В общем виде конструкция range имеет следующий вид:

```
range(start, stop, step)
```

При использовании `range` целые числа генерируется так, чтобы значение `stop` не достигалось. Из трех параметров рассматриваемой функции обязательным является только `stop`. В качестве примера приведем фрагмент программы (выполненный в оболочке):

```
>>> for i in range(5):
    print(i)
0
1
2
3
4
```

В следующем примере заданы только начальное и конечное значения:

```
>>> range(1,5)
range(1, 5)
```

```
>>> for i in range(1,5):
    print(i)
1
2
3
4
```

Из этих примеров следует, что значение `step` по умолчанию равно +1. Но можно задавать и другие значения `step`, например +2 – как в следующем примере:

```
>>> for i in range(1,7,2):
    print(i)
1
3
5
```

Рассмотрим пример, в котором инструмент `range` использован с отрицательным шагом:

```
for i in range(7,1,-1):
    print(i)

7
6
5
4
3
2
```

Приведем еще два примера использования функции `range` с разным последним значением и шагом:

```
for i in range(5,1,1):
    print(i)

for i in range(1,5,-1):
    print(i)
>>>
```

Как видим, в обоих примерах цикл не сработает ни разу – вывод пустой.

2.4. Модули в «Питоне»

Удобство использования «Питона» определяется возможностью применения модулей. Они представляют собой готовые для выполнения, часто используемые алгоритмы, к которым можно обратиться в своей программе. Выше уже говорилось о модуле `math`. Без него в программе нет доступа к элементарным математическим функциям. Для подключения модулей необходим оператор `import`. А для использование элементов модулей (функций) сначала надо указать имя модуля, точку и имя функции (ранее было записано обращение к функции `math.fabs`). После выполнения команды `help(«math»)` выводится информация о функциях этого модуля (285 строк), как показано на рис. 2.22. После появления сообщения, представленного на рис. 2.22, надо два раза щелкнуть левой клавишей мыши, после чего все строки раскроются.

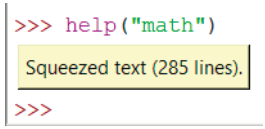


Рис. 2.22. Информация о модуле `math`

По адресу, указанному в источнике [13], представлена информация о 44 функциях этого модуля. Также список доступных функций этого модуля можно получить как в режиме интерпретатора, так и в среде разработки (рис. 2.23).

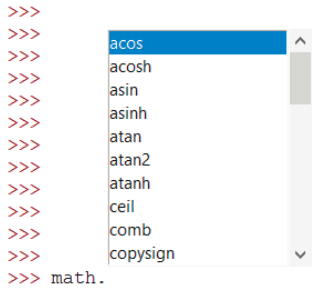


Рис. 2.23. Список функций модуля `math`

Приведем пример вычисления площади треугольника по формуле Герона с использованием модуля `math`:

```
import math
a=3; b=4; c=5;
p=(a+b+c)/2
s=math.sqrt(p*(p-a)*(p-b)*(p-c))
print(«Стороны треугольника: «,a,b,c)
print(«Площадь:», s)
```

После выполнения приведенного выше фрагмента программы будет выведена следующая информация (рис. 2.24).

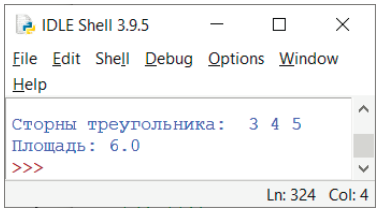


Рис. 2.24. Результат вычисления площади треугольника

2.5. Ввод значений переменных и вывод результатов выполнения программ

В заключение приведем несколько примеров ввода значений переменных и форматированного вывода получаемых результатов.

Ввод исходных данных в «Питоне» осуществляется с помощью оператора `input`. Как было сказано ранее, он вводит символьную строку. Если переменной при вводе надо присвоить значение другого типа, то оператор `input` соединяют с оператором преобразования типа. Или сначала вводят данные в промежуточную переменную, а затем изменяют ее тип. Это можно проиллюстрировать следующим примером:

```
a1=input()
a2=input()
print('a1+a2 ',(a1+a2))
```



```
i1=int(input())
print('i1 ',i1, type(i1))
s=float(input())
f1=float(s)
print('f1 ',f1, type(f1))
print('i1+f1 ',(i1+f1))
```

Далее представлен результат выполнения приведенного выше фрагмента программы (рис. 2.25).

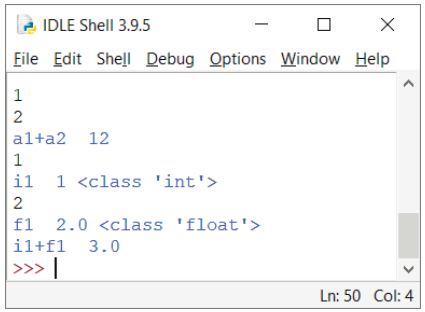


Рис. 2.25. Ввод данных в программу

Оформить вывод результатов работы программы на экране можно с использованием форматирования. Есть две конструкции, позволяющие сделать это. В источнике [14] они называются следующим образом: «по старинке (оператор %)» и «по новому (str.format)».

В первом случае это выглядит следующим образом (во фрагменте программы использованы три форматных кода (`f`, `e`, `g`), применимых для вещественных значений):

```
import math

c1=1/3
c2=math.pi
c3=(1/7)**4
print('Без форматирования')
print('',c1,'\n',c2,'\n',c3)
print("\nТри формата f e g")
```

```
print("c1 %6.3f %6.3e %6.3g" %(c1,c1,c1))
print("c1 %6.3f %6.3e %6.3g" %(c2,c2,c2))
print("c1 %6.3f %6.3e %6.3g" %(c3,c3,c3))
```

Результат применения этих операторов представлен на рис. 2.26.

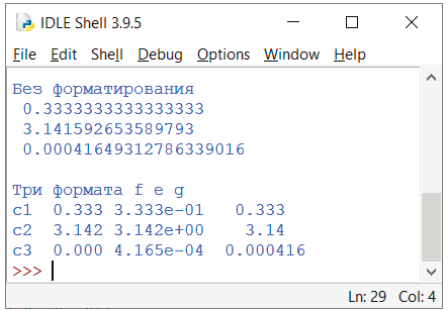


Рис. 2.26. Форматирование вывода

Этот же фрагмент программы с использованием оператора форматирования будет иметь такой вид:

```
print('Без форматирования')
print('',c1,'\n',c2,'\n',c3)
print(«\nТри формата f e g»)
print(«c1    {0:6.3f}    {1:6.3e}    {2:6.3g}».
format(c1,c1,c1))
print("c2    {0:6.3f}    {1:6.3e}    {2:6.3g}".
format(c2,c2,c2))
print("c3    {0:6.3f}    {1:6.3e}    {2:6.3g}".
format(c3,c3,c3))
```

При использовании форматов следует соблюдать соответствие «тип переменной – допустимый форматный код». В примере, приведенном ниже, значение переменной c2 преобразуется к строковому и целому типу, а затем выводится с допустимым для этого типа переменной форматом (s и d). Если применить такие два формата к вещественной переменной c2, то будет выведено сообщение об ошибке.


```
c2=math.pi
sc2=str(c2)
ic2=int(c2)
print("c2      {0:6.3f}      {1:s}      {2:d}".
format(c2,sc2,ic2))
print("c2 {0:6.3f} {1:s} {2:d}".format(c2,c2,c2))
```

На рис. 2.27 представлен результат выполнения приведенной выше программы. Выводится сообщение о том, что форматный код s не применим к объекту типа float.

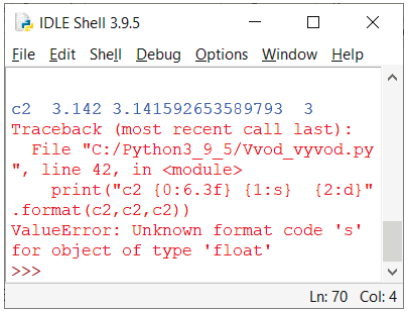


Рис. 2.27. Ошибка применения формата

Ранее кратко упоминалось о небольшом количестве форматных кодов. Получить помощь в использовании операций форматирования можно с использованием команды help('FORMATTING'), или на страницах Интернета, например по адресу https://pyprog.pro/python/py/str/str_format_method.html. Кроме форматных кодов используется еще много параметров форматирования. Например, это длина поля вывода и выравнивание. В следующем примере приводится фрагмент кода, в котором выводится строка из 5 символов в поле размером 20 и тремя способами выравниваниями (по левому краю, по центру, по правому краю):

```
s="Hello"
print('{0:<20s}'.format(s))
print('{0:^20s}'.format(s))
print('{0:>20s}'.format(s))
```

Выполнение этого фрагмента программы приведет к результату, представленному на рис. 2.28.

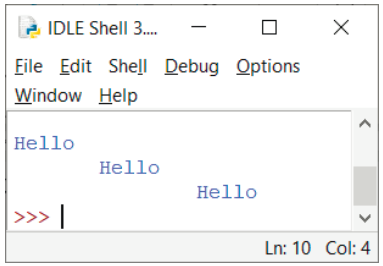


Рис. 2.28. Форматирование с заданием длины поля и выравниванием

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Широков, А.И. Информатика: основы разработки программ на языке программирования Питон [Текст] : учебник / А.И. Широков, М.О. Пышняк. – М. : Изд. Дом НИТУ «МИСиС», 2020. – Ч. 1. – 146 с.

2. Python [Электронный ресурс] : материал из Википедии – свободной энциклопедии. © Wikimedia Foundation, Inc. – Режим доступа: https://ru.wikipedia.org/wiki/Python (дата обращения: 10.10.2021).

3. PEP 8 [Электронный ресурс] : руководство по написанию кода на Python. © 2012–2017 Python 3 для начинающих. – Режим доступа: pythonworld.ru/osnovy/pep-8-rukovodstvo-ponapisaniyu-koda-na-python.html (дата обращения: 10.08.2019).

4. Мусин, Дмитрий. Самоучитель Python [Электронный ресурс] / Дмитрий Мусин. – Вып. 0.2. pythonworld.ru. – Режим доступа: https://pythonworld.ru/uploads/pythonworldru.pdf (дата обращения: 10.10.2021).

5. Любанович, Билл. Простой Python. Современный стиль программирования [Текст] / Билл Лобанович. – СПб. : Питер, 2016. – 480 с.

6. Сысоева, М.В. Программирование для нормальных с нуля на языке Python [Текст] : учебник : в 2 ч. / М.В. Сысоева, И.В. Сысоев ; отв. ред. В.Л. Черный. – М. : Базальт СПО : МАКС Пресс, 2018. – Ч. 1. – 176 с.

7. Бриггс, Джейсон. Python для детей [Текст] : самоучитель по программированию / Джейсон Бриггс ; пер. с англ. Станислава Ломакина ; науч. ред. Д. Абрамова. – М. : Манн, Иванов и Фербер, 2017. – 320 с.

8. Диапазоны типов данных [Электронный ресурс]. © Microsoft. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/data-type-ranges?view=msvc-160&viewFallbackFrom=vs-2019 (дата обращения: 10.10.2021).

9. Вещественные числа – float [Электронный ресурс]. © Семен Лукшевский. – Режим доступа: https://pyprog.pro/python/py/nums/float.html (дата обращения: 10.10.2021).

10. Операторы в Python [Электронный ресурс] : сайт Дмитрия Красоты. – Режим доступа: http://pythonicway.com/python-operators (дата обращения 10.10.2021).

11. Основные операторы Python [Электронный ресурс] : сайт Федорова. – Режим доступа: https://pythononline.ru/osnovy/operatory-python (дата обращения 10.10.2021).

12. Python с нуля – часть 3: базовые операторы [Электронный ресурс]. – Режим доступа: https://rtfm.co.ua/books-translations/python_s_nulya/python-s-nulya-chast-3-bazovye-operatory/#Операторы_присваивания_в_Python (дата обращения: 10.10.2021).

13. Модуль math [Электронный ресурс]. © 2012–2021 Python 3 для начинающих. –Режим доступа: https://pythonworld.ru/moduli/modul-math.html (дата обращения: 10.10.2021).

14. Лучшие примеры форматирования строк в Python [Электронный ресурс]. – Режим доступа: https://python-scripts.com/string-formatting (дата обращения: 10.10.2021).

Учебное издание

Широков Андрей Игоревич

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ «ПИТОН» (PYTHON)

Методические указания

Научный редактор *Т.А. Кравченко*
Корректор *Е.Н. Леонова*
Компьютерная верстка *А.Л. Бабабекова*

Подписано в печать 30.11.21 Формат 60 × 90 $^{1}/_{16}$ Уч.-изд. л. 3

Национальный исследовательский
технологический университет «МИСиС»,
119049, Москва, Ленинский пр-т, 4

Издательский Дом НИТУ «МИСиС»,
119049, Москва, Ленинский пр-т, 4
Тел. 8 (495) 638-44-06

Отпечатано в типографии
Издательского Дома НИТУ «МИСиС»,
119049, Москва, Ленинский пр-т, 4
Тел. 8 (495) 638-44-16, 8 (495) 638-44-43