



# PYTHON

**ПРОГРАММИРОВАНИЕ  
ДЛЯ НАЧИНАЮЩИХ**

**ПЕРВЫЙ ШАГ НА ПУТИ К УСПЕШНОЙ КАРЬЕРЕ**

➤ Для версий 3.1 - 3.4

УДК 004.43  
ББК 32.973.2-018.2  
М15



Mike McGrath  
Python in Easy Steps  
By Mike McGrath. Copyright ©2013 by In Easy Steps Limited. Translated and reprinted under a licence agreement from the Publisher: In Easy Steps, 16 Hamolton Terrace, Holly Walk, Leamington Spa, Warwickshire, U.K. CV32 4LY.

### МакГрат, Майк.

М15 Программирование на Python для начинающих : [перевод с англ. М.А. Райтмана] / Майк МакГрат. — Москва : Эксмо, 2015. — 192 с. — (Программирование для начинающих).

Книга «Программирование на Python для начинающих» является исчерпывающим руководством для того, чтобы научиться программировать на языке Python.

В этой книге с помощью примеров программ и иллюстраций, показывающих результаты работы кода, разбираются все ключевые аспекты языка. Установив свободно распространяемый интерпретатор Python, вы с первого же дня сможете создавать свои собственные исполняемые программы!

УДК 004.43  
ББК 32.973.2-018.2

Производственно-практическое издание  
ПРОГРАММИРОВАНИЕ ДЛЯ НАЧИНАЮЩИХ

Майк МакГрат

**ПРОГРАММИРОВАНИЕ НА PYTHON  
ДЛЯ НАЧИНАЮЩИХ**  
(орыс тілінде)

Директор редакции *Е. Капьев*  
Ответственный редактор *В. Обручев*  
Художественный редактор *В. Брагина*

В оформлении обложки использована фотография:  
Toria / Shutterstock.com  
Используется по лицензии от Shutterstock.com

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить по адресу: <http://eksmo.ru/certification/>  
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Все права защищены. Книга или любая ее часть не может быть скопирована, воспроизведена в электронной или механической форме, в виде фотокопии, записи в память ЭВМ, репродукции или каким-либо иным способом, а также использована в любой информационной системе без получения разрешения от издателя. Копирование, воспроизведение и иное использование книги или ее части без согласия издателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

ООО «Издательство «Эксмо»  
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86, 8 (495) 956-39-21.  
Home page: [www.eksmo.ru](http://www.eksmo.ru) E-mail: [info@eksmo.ru](mailto:info@eksmo.ru)  
Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.  
Тел. 8 (495) 411-68-86, 8 (495) 956-39-21  
Home page: [www.eksmo.ru](http://www.eksmo.ru) E-mail: [info@eksmo.ru](mailto:info@eksmo.ru)  
Тауар белгісі: «Эксмо»  
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3-а, литер Б, офис 1.  
Тел.: 8 (727) 2 51 59 89, 90, 91, 92, факс: 8 (727) 251 58 12 вн. 107; E-mail: [RDC-Almaty@eksmo.kz](mailto:RDC-Almaty@eksmo.kz)  
Өнімнің жарамдылық мерзімі шектелмеген.  
Сертификация туралы ақпарат сайты: [www.eksmo.ru/certification](http://www.eksmo.ru/certification)

Подписано в печать 08.09.2015. Формат 84x108<sup>1</sup>/<sub>16</sub>.  
Печать офсетная. Усл. печ. л. 20,16.  
Тираж экз. Заказ

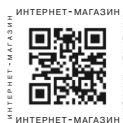


ISBN 978-5-699-81406-0



9 785699 814060 >

ISBN 978-5-699-81406-0



ИНТЕРНЕТ-МАГАЗИН  
ИНТЕРНЕТ-МАГАЗИН

В электронном виде книги издательства Эксмо вы можете купить на [www.litres.ru](http://www.litres.ru)

ЛитРес:  
один клик до книги



© Райтман М.А., перевод на русский язык, 2015  
© Оформление. ООО «Издательство «Эксмо», 2015

# Оглавление

## Предисловие

8

## 1

## Приступаем к работе

9

Введение в язык Python . . . . .	10
Установка Python в операционной системе Windows . . . . .	12
Установка Python в операционной системе Linux. . . . .	14
Знакомство с интерпретатором . . . . .	16
Ваша первая программа . . . . .	18
Работа с переменными . . . . .	20
Получение введенных пользователем данных . . . . .	22
Исправление ошибок . . . . .	24
Заключение. . . . .	26

## 2

## Выполнение операций

27

Арифметические действия. . . . .	28
Присваивание значений . . . . .	30
Сравнение величин . . . . .	32
Оценочная логика . . . . .	34
Проверка условий . . . . .	36
Определение приоритетов . . . . .	38
Преобразование типов данных. . . . .	40
Манипуляции с битами . . . . .	42
Заключение. . . . .	44

## 3

## Конструирование инструкций

45

Списки . . . . .	46
Работа со списками. . . . .	48
Неизменяемые списки . . . . .	50
Элементы ассоциативного списка . . . . .	52
Ветвление с помощью условного оператора. . . . .	54
Цикл while. . . . .	56

Обход элементов в цикле. . . . .	58
Выход из цикла . . . . .	60
Заключение. . . . .	62

## 4 Определение функций 63

Область видимости переменных. . . . .	64
Подстановка аргументов . . . . .	66
Возвращение значений . . . . .	68
Использование обратного вызова. . . . .	70
Добавление заполнителей. . . . .	72
Генераторы в Python . . . . .	74
Обработка исключений . . . . .	76
Отладка с помощью инструкции assert. . . . .	78
Заключение. . . . .	80

## 5 Импорт модулей 81

Хранение функций . . . . .	82
Принадлежность имен функций . . . . .	84
Системные запросы . . . . .	86
Математические операции . . . . .	88
Вычисления с десятичными дробями. . . . .	90
Работа со временем . . . . .	92
Запуск таймера . . . . .	94
Шаблоны соответствий . . . . .	96
Заключение. . . . .	98

## 6 Строки и работа с файлами 99

Работа со строками. . . . .	100
Форматирование строк. . . . .	102
Модификация строк . . . . .	104
Преобразование строк . . . . .	106
Доступ к файлам. . . . .	108
Чтение и запись файлов. . . . .	110
Изменение текстового файла . . . . .	112
Консервация данных. . . . .	114
Заключение. . . . .	116

## 7 Объектное программирование 117

Инкапсуляция данных. . . . .	118
Создание экземпляров объектов . . . . .	120
Доступ к атрибутам класса. . . . .	122

Встроенные атрибуты . . . . .	124
Сборка мусора . . . . .	126
Наследование свойств . . . . .	128
Переопределение основных методов . . . . .	130
Реализация полиморфизма . . . . .	132
Заключение. . . . .	134

## 8      **Обработка запросов**      **135**

Отправка ответов . . . . .	136
Обработка данных . . . . .	138
Передача данных через формы . . . . .	140
Использование текстовых областей . . . . .	142
Установка флажков . . . . .	144
Установка переключателя в положение . . . . .	146
Элементы списка . . . . .	148
Выгрузка файлов . . . . .	150
Заключение. . . . .	152

## 9      **Разработка интерфейсов**      **153**

Запуск оконного интерфейса . . . . .	154
Работа с кнопками . . . . .	156
Вывод сообщений . . . . .	158
Прием данных от пользователя . . . . .	160
Выбор из списка. . . . .	162
Использование переключателей. . . . .	164
Флажки. . . . .	166
Добавление изображений . . . . .	168
Заключение. . . . .	170

## 10     **Разработка приложений**      **171**

Генерирование случайных чисел . . . . .	172
Планирование программы. . . . .	174
Построение интерфейса . . . . .	176
Определение постоянных величин . . . . .	178
Инициализация изменяемых значений . . . . .	179
Добавление рабочей функциональности . . . . .	180
Тестирование программы . . . . .	182
Компиляция программы . . . . .	184
Распространение приложения . . . . .	186
Заключение. . . . .	188

## **Предметный указатель**      **189**

# 2

## Выполнение операций

*Эта глава знакомит нас с операторами языка Python и демонстрирует их работу.*

- Арифметические действия
- Присваивание значений
- Сравнение величин
- Оценочная логика
- Проверка условий
- Определение приоритетов
- Преобразование типов данных
- Манипуляции с битами
- Заключение

Совет



Значения, используемые вместе с операторами для формирования выражений, называются операндами — например, операндами в выражении `2 + 3` являются числа 2 и 3.

# Арифметические действия

Основные операторы, используемые при программировании на языке Python, а также выполняемые ими операции, представлены в таблице ниже.

Оператор	Операция
<code>+</code>	Сложение
<code>-</code>	Вычитание
<code>*</code>	Умножение
<code>/</code>	Деление
<code>%</code>	Деление по модулю
<code>//</code>	Целочисленное деление
<code>**</code>	Возведение в степень

Операторы сложения, вычитания, умножения и деления не должны вызывать каких-либо сложностей. Они работают так, как вы, наверное, и ожидали. Стоит, однако, обращать внимание, что при использовании нескольких операторов рекомендуется группировать выражения с помощью скобок — операции внутри скобок выполняются первыми. Например, выражение

```
a = b * c - d % e / f
```

может быть не совсем понятно с точки зрения порядка вычислений. Но его разрешается уточнить, записав в виде:

```
a = ( b * c ) - ( ( d % e ) / f )
```

Оператор `%` (деление по модулю) делит одно число на другое и возвращает остаток от деления. Он очень полезен для определения четности или нечетности числа.

Оператор `//` (целочисленное деление) работает так же, как и обычное деление, `/`, но отбрасывает результат после десятичной точки.

Оператор `**` (возведение в степень) возводит первый операнд в степень второго операнда.

1. Начните новую программу с инициализации двух переменных с целочисленными значениями.

```
a = 8
```

```
b = 2
```

2. Затем выведите результат сложения переменных.

```
print( 'Addition:\t' , a , '+' , b , '=' , a + b )
```

3. Теперь отобразите результат вычитания переменных.

```
print( 'Subtraction:\t' , a , '-' , b , '=' , a - b )
```

4. Затем отобразите результат умножения переменных.

```
print( 'Multiplication:\t' , a , 'x' , b , '=' , a * b )
```

5. Отобразите результат деления переменных как с плавающей точкой, так и целочисленного.

```
print( 'Division:\t' , a , '÷' , b , '=' , a / b )
```

```
print( 'Floor Division:\t' , a , '÷' , b , '=' , a // b )
```

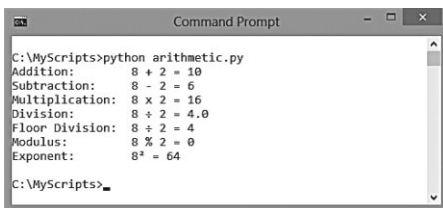
6. Затем выведите остаток от деления одной величины на другую.

```
print( 'Modulus:\t' , a , '%' , b , '=' , a % b )
```

7. Наконец отобразите результат возведения первого операнда в степень второго.

```
print( 'Exponent:\t' , a , '^2 = ' , a ** b , sep = ' ' )
```

8. Сохраните файл в вашем рабочем каталоге, откройте командную строку и запустите программу — вы увидите результат арифметических операций.



```
Command Prompt
C:\MyScripts>python arithmetic.py
Addition:      8 + 2 = 10
Subtraction:   8 - 2 = 6
Multiplication: 8 x 2 = 16
Division:      8 ÷ 2 = 4.0
Floor Division: 8 ÷ 2 = 4
Modulus:       8 % 2 = 0
Exponent:      8^2 = 64
C:\MyScripts>
```



arithmetic.py

#### Совет



`\t` — это так называемая управляющая последовательность, которая добавляет невидимый символ табуляции для форматирования вывода.

#### На заметку



Вы можете использовать параметр `sep` для явного указания разделения между выводами. В этом примере нет разделения, так как указано два символа апострофа без пробела.



# Присваивание значений

Операторы, используемые при программировании на языке Python для присваивания значений переменным, перечислены в таблице ниже. Все они, за исключением простого присваивания, `=`, являются сокращенными формами от более длинных выражений. Для ясности в таблице представлены их эквиваленты.

Оператор	Пример	Эквивалентная операция
<code>=</code>	<code>a = b</code>	<code>a = b</code>
<code>+=</code>	<code>a += b</code>	<code>a = ( a + b )</code>
<code>-=</code>	<code>a -= b</code>	<code>a = ( a - b )</code>
<code>*=</code>	<code>a *= b</code>	<code>a = ( a * b )</code>
<code>/=</code>	<code>a /= b</code>	<code>a = ( a / b )</code>
<code>%=</code>	<code>a %= b</code>	<code>a = ( a % b )</code>
<code>//=</code>	<code>a //= b</code>	<code>a = ( a // b )</code>
<code>**=</code>	<code>a **= b</code>	<code>a = ( a ** b )</code>

В таблице выше переменной с именем `a` присваивается значение, которое содержится в переменной с именем `b`, и, таким образом, в переменной `a` будет храниться новое значение.

Оператор `+=` полезно использовать для добавления какого-то значения к существующему, хранящемуся в переменной `a`.

Оператор `+=` сначала добавляет к значению, содержащемуся в переменной, `b` значение, содержащееся в переменной `a`, а потом результат присваивает значению переменной `a`.

Все другие операторы работают по тому же принципу — сначала выполняют арифметическую операцию между двумя операндами, а затем присваивают первой переменной значение этого результата.

При использовании оператора `%=` первый операнд `a` делится на второй операнд `b`, и затем остаток от этого деления присваивается переменной `a`.

## На заметку



Важно различать операции присваивания, `=`, и равенства, `==`, чтобы избежать недоразумений с логическим оператором равенства `==`.

1. Начните новую программу на Python, которая проинициализирует две переменные с помощью присваиваний им целых значений, и выведите эти присвоенные значения.

```
a = 8
b = 4

print( 'Assign Values:\t\t' , 'a =' , a , '\tb =' , b )
```

2. Теперь присвойте новое значение первой переменной и выведите его на экран.

```
a += b

print( 'Add & Assign:\t\t' , 'a =' , a , '(8 += 4)' )
```

3. Прodelайте то же самое, что и в предыдущем шаге, но теперь с использованием вычитания и умножения.

```
a -= b

print( 'Subtract & Assign:\t' , 'a =' , a , '(12 - 4)' )
```

```
a *= b

print( 'Multiply & Assign:\t' , 'a =' , a , '(8 x 4)' )
```

4. Наконец с помощью деления и присваивания получите новое значение для первой переменной и выведите на экран результат, после чего используйте деление по модулю и присваивание с выводом результата.

```
a /= b

print( 'Divide & Assign:\t' , 'a =' , a , '(32 ÷ 4)' )

a %= b

print( 'Modulus & Assign:\t' , 'a =' , a , '(8 % 4)' )
```

5. Сохраните файл в вашем рабочем каталоге, затем откройте командную строку и запустите программу — вы увидите результат работы операции присваивания.

```

C:\MyScripts>python assign.py
Assign Values:      a = 8  b = 4
Add & Assign:       a = 12 (8 += 4)
Subtract & Assign:  a = 8  (12 - 4)
Multiply & Assign:  a = 32 (8 x 4)
Divide & Assign:    a = 8.0 (32 ÷ 4)
Modulus & Assign:   a = 0.0 (8 % 4)
C:\MyScripts>

```



assign.py

### Внимание



В отличие от оператора присваивания, `=`, оператор равенства, `==`, сравнивает операнды. Он описывается в следующем разделе.

# Сравнение величин

Операторы, которые обычно используются при программировании на языке Python для сравнения двух операндов, представлены в таблице ниже.

Оператор	Проверяемое условие
<code>==</code>	Равенство
<code>!=</code>	Неравенство
<code>&gt;</code>	Больше
<code>&lt;</code>	Меньше
<code>&gt;=</code>	Больше либо равно
<code>&lt;=</code>	Меньше либо равно

Оператор равенства, `==`, производит сравнение двух операндов и возвращает **True** (Истина), если их значения равны, в противном случае возвращает **False** (Ложь). При этом если операндами являются числовые значения, и они одинаковые, то они равны, а если символы, то сравниваются их ASCII-коды.

Оператор неравенства, `!=`, наоборот, возвращает значение **True**, если оба операнда не равны, используя то же самое правило, как и оператор равенства, в противном случае возвращая **False**. Операторы равенства и неравенства полезны для выполнения условного ветвления в программе по результатам сравнения значений двух переменных.

Оператор «больше», `>`, сравнивает два операнда и возвращает **True**, если значение первого больше, чем значение второго, и наоборот, возвращает **False**, если значение первого равно значению второго или меньше его. Оператор «меньше», `<`, делает то же самое, но возвращает **True**, если первый оператор меньше. Эти два оператора часто используются для проверки значения счетчика операций в цикле.

Добавление оператора «равно», `=`, после оператора «больше», `>`, или «меньше», `<`, заставляет их возвращать значение **True**, если значения операндов совпадают.

Совет



Символы верхнего регистра от A до Z имеют коды ASCII 65–90, а символы нижнего регистра a–z — от 97 до 122.

1. Начните новую программу в Python с инициализации пяти переменных, которые будут использоваться для сравнения.

```
nil = 0

num = 0

max = 1

cap = 'A'

low = 'a'
```



comparison.py

2. Теперь добавьте инструкции для вывода результатов числового и символического сравнения с помощью оператора равенства.

```
print( 'Equality :\\t' , nil , '==' , num , nil == num )

print( 'Equality :\\t' , cap , '==' , low , cap == low )
```

3. Теперь добавьте инструкцию для вывода результата сравнения с помощью оператора неравенства.

```
print( 'Inequality :\\t' , nil , '!=' , max , nil != max )
```

4. Теперь добавим инструкции, чтобы вывести результаты сравнения, выполненного операторами «больше» и «меньше».

```
print( 'Greater :\\t' , nil , '>' , max , nil > max )

print( 'Lesser :\\t' , nil , '<' , max , nil < max )
```

5. Наконец добавьте инструкцию для вывода результатов работы операторов сравнения, «больше либо равно», «меньше либо равно».

```
print( 'More Or Equal :\\t' , nil , '>=' , num , nil >= num )

print( 'Less or Equal :\\t' , max , '<=' , num , max <= num )
```

6. Сохраните файл в вашем рабочем каталоге, откройте командную строку и запустите программу — вы увидите результаты работы операторов сравнения.

```
Command Prompt
C:\\MyScripts>python comparison.py
Equality :      0 == 0 True
Equality :      A == a False
Inequality :    0 != 1 True
Greater :       0 > 1 False
Lesser :        0 < 1 True
More Or Equal : 0 >= 0 True
Less or Equal : 1 <= 0 False
C:\\MyScripts>
```

### На заметку



Управляющая последовательность `\\t`, представленная здесь, добавляет невидимый символ табуляции для форматирования вывода.

### На заметку



Коды ASCII для символов `A` (65) и `a` (97) не равны — так что сравнение этих значений возвращает значение `False`.

# Оценочная логика

Логические операторы, используемые в программировании на Python, представлены в таблице ниже.

Оператор	Операция
and	Логическое «И»
or	Логическое «ИЛИ»
not	Логическое «НЕ»

Логические операторы работают с операндами, которые имеют значение логического (булева) типа, то есть **True** или **False**, либо со значениями, которые преобразуются в **True** или **False**.

Оператор «логическое И», **and**, оценивает два операнда и возвращает значение **True**, только если оба операнда сами имеют значение **True**, в противном случае возвращает значение **False**. Данный оператор обычно используется при условном ветвлении, когда направление работы программы определяется проверкой двух условий — если оба они верны, программа идет в определенном направлении, в противном случае — в другом.

В отличие от оператора **and**, которому необходимо, чтобы оба операнда имели значение **True**, оператор «логическое ИЛИ», **or**, оценивает два операнда и возвращает **True**, если хотя бы один из них сам возвращает значение **True**. В противном случае оператор **or** возвратит значение **False**. Это полезно использовать при программировании определенных действий в случае выполнения одного из двух проверяемых условий.

Оператор «логическое НЕ», **not**, является унарным и используется с одним операндом. Он возвращает противоположное значение от того, какое имел операнд. Так, если переменная **a** имела значение **True**, то **not a** возвратит значение **False**. Он может использоваться, например, для переключения значения переменной в последовательных итерациях цикла при помощи инструкции **a = not a**. Это значит, что на каждой итерации цикла логическое значение меняется на противоположное, подобно выключению и включению лампочки.

## На заметку



Термин «булев» назван в честь английского математика Джорджа Буля, одного из основателей математической логики.

1. Начните новую программу в Python, проинициализировав две переменные с булевыми значениями для последующей их оценки.

```
a = True
```

```
b = False
```



logic.py

2. Теперь добавьте инструкции для того, чтобы вывести результаты работы оператора «логическое И».

```
print( 'AND Logic:' )
```

```
print( 'a and a =' , a and a )
```

```
print( 'a and b =' , a and b )
```

```
print( 'b and b =' , b and b )
```

3. Теперь добавьте инструкции, чтобы отобразить вывод результатов работы оператора «логического ИЛИ».

```
print( '\nOR Logic:' )
```

```
print( 'a or a =' , a or a )
```

```
print( 'a or b =' , a or b )
```

```
print( 'b or b =' , b or b )
```

4. Наконец добавляем инструкции для вывода результатов работы «логического НЕ».

```
print( '\nNOT Logic:' )
```

```
print( 'a =' , a , '\tnot a =' , not a )
```

```
print( 'b =' , b , '\tnot b =' , not b )
```

5. Сохраните файл в рабочем каталоге, откройте командную строку и запустите программу. Вы увидите результаты логических операций.

```
Command Prompt
C:\MyScripts>python logic.py
AND Logic:
a and a = True
a and b = False
b and b = False

OR Logic:
a or a = True
a or b = True
b or b = False

NOT Logic:
a = True      not a = False
b = False     not b = True

C:\MyScripts>
```

### Совет



При программировании на Python логические значения могут быть представлены числами: True — это 1, False — это 0.

### На заметку



Обратите внимание, что выражение False and False возвращает False, а не True, наглядно демонстрируя непреложную истину «две лжи не станут правдой».

# Проверка условий

Во многих языках программирования, таких как, например, C++ или Java, существует так называемый **тернарный** оператор `?:`, который оценивает выражение на условие истинности, возвращая, в зависимости от результата оценки, одно из двух определенных значений. Тернарный оператор `?:` имеет следующий синтаксис:

```
( проверочное выражение ) ? если-истина-возвращаем-это : если-ложь-возвращаем-это
```

В языке Python роль тернарного оператора выполняет **условное выражение**, которое работает аналогичным способом и использует в своем синтаксисе ключевые слова `if` и `else`:

```
если-истина-возвращаем-это if ( проверочное-выражение ) else если-ложь-возвращаем-это
```

Несмотря на то, что синтаксис условного выражения может поначалу показаться непонятным, с ним стоит познакомиться, поскольку он является достаточно мощным инструментом для организации ветвлений в программах, используя минимальное количество кода. Например, чтобы организовать ветвление при условии, если значение переменной не равно единице, мы пишем следующее:

```
если-истина-выполняем-это if ( var != 1 ) else если-ложь-выполняем-это
```

Условное выражение может быть использовано, например, для присваивания максимального или минимального из значений двух переменных третьей. Например, для случая с минимальным значением пишем таким образом:

```
c = a if ( a < b ) else b
```

Выражение в скобках возвращает `True`, когда значение переменной `a` меньше, чем `b` — так что в этом случае меньшее значение присваивается переменной `c`.

Аналогично, заменив в проверочном выражении оператор «меньше» на оператор «больше», мы можем присвоить переменной `c` значение большей переменной `b`. Еще одна распространенная область применения условного оператора включает использование оператора деления по модулю, `%`, для определения того, является число четным или нечетным:

```
если-истина(нечетное)-выполняем-это if ( var % 2 != 0 ) else если-ложь(четное)-  
выполняем-это
```

## На заметку



Условное выражение имеет три операнда — проверочное выражение и два возможных возвращаемых значения.

Таким образом, если результат от деления значения переменной на два дает остаток, то число нечетное, а если остатка нет, то четное. Того же эффекта можно достичь, используя проверочное выражение ( `var % 2 == 1` ), но предпочтительней все же проверять на неравенство — так как два отличающихся значения найти легче, чем два идентичных.

1. Начните новую программу на Python с инициализации двух переменных, которые будем оценивать, целочисленными значениями.

```
a = 1
```

```
b = 2
```

2. Теперь добавьте инструкции, чтобы вывести результаты проверки условий, описывающие первую переменную.

```
print( '\nVariable a Is : ' , 'One' if ( a == 1 ) else 'Not One' )
```

```
print( 'Variable a Is : ' , 'Even' if ( a % 2 == 0 ) else 'Odd' )
```

3. Затем добавьте инструкции для вывода результатов проверки логического выражения, описывающего значение второй переменной.

```
print( '\nVariable b Is : ' , 'One' if ( b == 1 ) else 'Not One' )
```

```
print( 'Variable b Is : ' , 'Even' if ( b % 2 == 0 ) else 'Odd' )
```

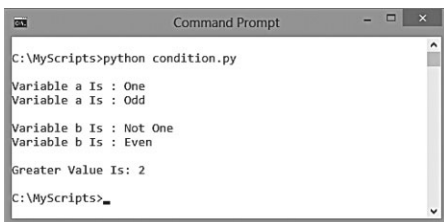
4. Добавьте инструкцию для присваивания его результата новой переменной.

```
max = a if ( a > b ) else b
```

5. Наконец, добавьте инструкцию для вывода присвоенного результата, определяющего большую из двух величин.

```
print( '\nGreater Value Is:' , max )
```

6. Сохраните файл в вашем рабочем каталоге, откройте командную строку и запустите программу — вы увидите результаты операций с условными выражениями.



```
Command Prompt
C:\MyScripts>python condition.py
Variable a Is : One
Variable a Is : Odd
Variable b Is : Not One
Variable b Is : Even
Greater Value Is: 2
C:\MyScripts>
```



condition.py



#### Внимание

Вы можете обнаружить, что некоторые программисты Python не любят условные выражения, поскольку их синтаксис противоречит принципам читаемости кода.



На заметку



Оператор умножения, \*, выше в таблице, чем оператор сложения, поэтому умножение вычисляется перед сложением.

Совет



Операторы идентичности, побитовые операторы, а также операторы вхождения будут представлены дальше — здесь они включены в таблицу для полноты сведений.

# Определение приоритетов

Приоритет операторов определяет порядок, которому интерпретатор Python следует при оценке выражений. Например, в выражении `3 * 8 + 4` порядок действий по умолчанию определяет, что умножение выполняется первым, так что результат равен 28 (24 + 4).

В таблице ниже перечисляются операторы в порядке убывания приоритета. Те из них, что находятся в строках выше, имеют более высокий приоритет. Приоритет операторов, находящихся на одной строке в таблице, определяется правилом «слева направо».

Оператор	Описание
**	Возведение в степень
+	Положительное значение
-	Отрицательное значение
~	Побитовое отрицание
*	Умножение
/	Деление
//	Целочисленное деление
%	Деление по модулю
+	Сложение
-	Вычитание
	Побитовое ИЛИ
^	Побитовое исключающее ИЛИ
&	Побитовое И
>>	Побитовый сдвиг вправо
<<	Побитовый сдвиг влево
>, < =, <, < =, =, !=	Сравнение
=, %=, /=, //=, -=, +=, *=, **=	Присваивание
is, is not	Идентичность
in, not in	Вхождение
not	Логическое отрицание
and	Логическое И
or	Логическое ИЛИ

1. Начните новую программу на Python, проинициализировав три переменные целочисленными значениями для последующего сравнения приоритетов.

```
a = 2
```

```
b = 4
```

```
c = 8
```

2. Теперь добавьте инструкции для вывода результатов вычислений с порядком действий по умолчанию и явным указанием приоритета операции сложения.

```
print( '\nDefault Order:\t', a, '*', c, '+', b, '=', a * c + b )
```

```
print( 'Forced Order:\t', a, '*' (', c, '+', b, ')', '=', a * ( c + b ) )
```

3. Затем добавьте инструкции для вывода результата вычислений с порядком действий по умолчанию и явным указанием приоритета операции вычитания.

```
print( '\nDefault Order:\t', c, '//', b, '-', a, '=', c // b - a )
```

```
print( 'Forced Order:\t', c, '// (', b, '-', a, ')', '=', c // ( b - a ) )
```

4. Наконец добавьте инструкции для вывода результатов вычислений с порядком операций по умолчанию, а затем с добавлением приоритета операции сложения перед делением по модулю, а также перед вычислением степени.

```
print( '\nDefault Order:\t', c, '%', a, '+', b, '=', c % a + b )
```

```
print( 'Forced Order:\t', c, '% (', a, '+', b, ')', '=', c % ( a + b ) )
```

```
print( '\nDefault Order:\t', c, '**', a, '+', b, '=', c ** a + b )
```

```
print( 'Forced Order:\t', c, '** (', a, '+', b, ')', '=', c ** ( a + b ) )
```

5. Сохраните файл в рабочем каталоге, откройте командную строку и запустите программу — вы увидите результаты работы операторов с порядком по умолчанию и с измененным порядком приоритетов.

```
Command Prompt
C:\MyScripts>python precedence.py

Default Order:  2 * 8 + 4 = 20
Forced Order:   2 * ( 8 + 4 ) = 24

Default Order:  8 // 4 - 2 = 0
Forced Order:   8 // ( 4 - 2 ) = 4

Default Order:  8 % 2 + 4 = 4
Forced Order:   8 % ( 2 + 4 ) = 2

Default Order:  8 ** 2 + 4 = 68
Forced Order:   8 ** ( 2 + 4 ) = 262144

C:\MyScripts>
```



precedence.py

### На заметку



В отличие от оператора деления (/) оператор деления нацело (//) отбрасывает дробную часть числа.

### Внимание



Никогда не полагайтесь на порядок операций по умолчанию — старайтесь всегда использовать для ясности скобки в ваших выражениях.

# Преобразование типов данных

Переменные в языке Python могут хранить данные любого типа, и очень важно различать эти типы для того, чтобы избежать ошибок при обработке данных в программе. Типов данных несколько, но пока мы рассмотрим основные из них: строковые (`str` (`string`)), целочисленные (`int` (`integer`)), с плавающей точкой (`float` (`floating-point`)).

Очень важно различать типы данных, особенно при присваивании переменным значений, используя пользовательский ввод, так как по умолчанию в нем хранится строковый тип данных. Строковые величины не могут быть использованы для арифметических выражений, и попытка сложить два строковых значения просто объединяет эти строки, а не использует операции над числами. Например, `'8' + '4' = '84'`.

Но, к счастью, любые типы данных, хранящиеся в переменных, могут быть легко преобразованы (приведены) к другим типам с помощью использования функций языка Python. Значение, которое требуется преобразовать, указывается в функции внутри скобок, идущих сразу после имени функции. Приведение строковых величин к целым позволяет затем использовать их уже для арифметических выражений, например, `8 + 4 = 12`.

Встроенные функции преобразования типов данных в Python возвращают новый объект, представляющий преобразованную величину. Наиболее часто используемые из этих функций представлены в таблице ниже.

## Внимание



Преобразование типа данных с плавающей точкой (`float`) в целочисленный тип данных (`int`) отбрасывает десятичную часть числа.

Функция	Описание
<code>int( x )</code>	Преобразует <code>x</code> в целое число
<code>float( x )</code>	Преобразует <code>x</code> в число с плавающей точкой
<code>str( x )</code>	Преобразует <code>x</code> в строковое представление
<code>chr( x )</code>	Преобразует целое <code>x</code> в символ
<code>unichr( x )</code>	Преобразует целое <code>x</code> в символ Юникода (Unicode)
<code>ord( x )</code>	Преобразует символ <code>x</code> в соответствующее ему целое число
<code>hex( x )</code>	Преобразует целое <code>x</code> в шестнадцатеричную строку
<code>oct( x )</code>	Преобразует целое <code>x</code> в восьмеричную строку