Майк МакГрат

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

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. Капьёв*
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

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** E-mail: **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 E-mail: 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
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

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

ISBN 978-5-699-81406-0

ISBN 978-5-699-81406-0

Оглавление

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**

Арифметические действия

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

Оператор	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
//	Целочисленное деление
**	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.

Совет

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

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 )
```

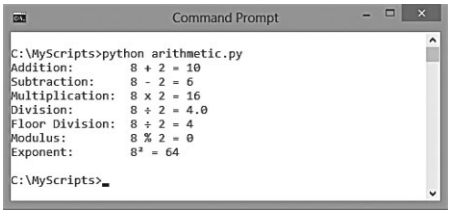
6. Затем выведите остаток от деления одной величины на другую.

```
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.



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
`=`	`a = b`	`a = b`
`+=`	`a += b`	`a = ( a + b )`
`-=`	`a -= b`	`a = ( a – b )`
`*=`	`a *= b`	`a = ( a * b )`
`/=`	`a /= b`	`a = ( a / b )`
`%=`	`a %= b`	`a = ( a % b )`
`//=`	`a //= b`	`a = ( a // b )`
`**=`	`a **= b`	`a = ( a ** b )`

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)' )
```

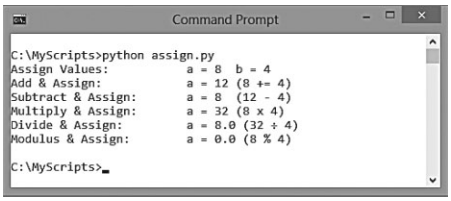
3. Проделайте то же самое, что и в предыдущем шаге, но теперь с использованием вычитания и умножения.

```
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.



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
= =	Равенство
! =	Неравенство
>	Больше
<	Меньше
> =	Больше либо равно
< =	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 символьного сравнения с помощью оператора равенства.

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

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

На заметку

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

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

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

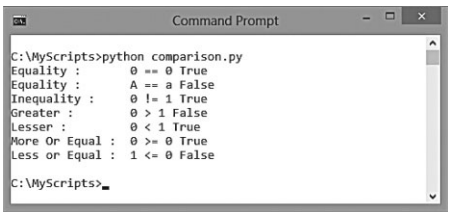
4. Теперь добавим инструкции, чтобы вывести результаты сравнения, выполненного операторами «больше» и «меньше».

```
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.



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
```

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 )
```

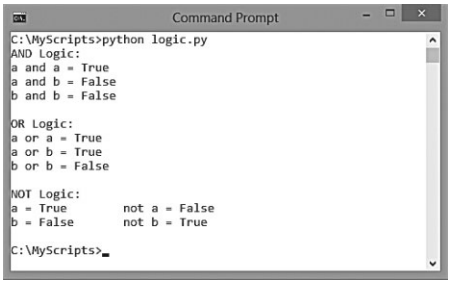
3. Теперь добавьте инструкции, чтобы отобразить вывод результатов работы оператора «логического ИЛИ».

```
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.



logic.py

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.

condition.py

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' )
```

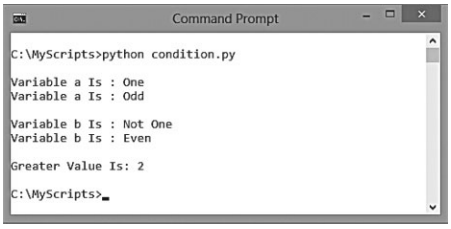
4. Добавьте инструкцию для присваивания его результата новой переменной.

```
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.



Внимание

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

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

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.

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 ) )
```

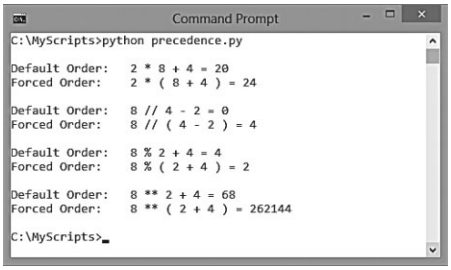
3. Затем добавьте инструкции для вывода результата вычислений с порядком действий по умолчанию и явным указанием приоритета операции вычитания.

```
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.



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
`int( x )`	Преобразует *x* в целое число
`float( x )`	Преобразует *x* в число с плавающей точкой
`str( x )`	Преобразует *x* в строковое представление
`chr( x )`	Преобразует целое *x* в символ
`unichr( x )`	Преобразует целое *x* в символ Юникода (Unicode)
`ord( x )`	Преобразует символ *x* в соответствующее ему целое число
`hex( x )`	Преобразует целое *x* в шестнадцатеричную строку
`oct( x )`	Преобразует целое *x* в восьмеричную строку