Python.

Деление нацело и остаток от деления.

В языке Python для обозначения операции целочисленного деления используется следующая комбинация символов **//** (два слэша без пробела).

```
print(20 // 3)
```

Результат **6**.

Для обозначения операции остатка от деления используется следующая символ **%** (знак процента).

```
print(20 % 3)
```

Результат **2**.

Рассмотрим особенности работы этих операторов в языке Python.

Логика работы этих операторов зависит от ***знака делителя***.

Если делитель ***положительный***, то результатом целочисленного деления ***a // b*** будет такое максимально возможное число ***c***, что будет выполняться неравенство ***c * b <= a***.

```
20 // 3 = 6   (6 * 3 = 18,  18 < 20)
-20 // 3 = -7 (-7 * 3 = -21,  -21 < -20)
```

При этом остатком от деления будет такое число ***d >= 0***, при котором будет выполняться равенство ***c * b + d = a***.

```
20 % 3 = 2   (6 * 3 + 2 = 20)
-20 % 3 = 1  (-7 * 3 + 1 = -20)
```

Если делитель ***отрицательный***, то результатом целочисленного деления ***a // b*** будет такое минимально возможное число ***c***, что будет выполняться неравенство ***c * b >= a***.

```
20 // -3 = -7   (-7 * -3 = 21,  21 > 20)
-20 // -3 = 6 (6 * -3 = -18,  -18 > -20)
```

При этом остатком от деления будет такое число ***d <= 0***, что будет выполняться равенство ***c * b + d = a***.

```
20 % -3 = -1   (-7 * -3 -1 = 20)
-20 % -3 = -2  (6 * -3 -2 = -20)
```

Заметим, что в языке Python *знак остатка от деления всегда совпадает со знаком делителя*.

Операции целочисленного деления и остатка от деления имеют указанный смысл только если оба операнда являются целыми числами. Язык Python допускает использование и вещественных чисел, однако смысл использования данных операций в этом случае теряется.

Арифметические операции с присваиванием

В языке Python могут быть использованы операции присваивания следующего вида:

```
a <арифметический оператор> <знак равно> b
```

Такая запись эквивалентна следующему выражению

```
a <знак равно> a <арифметический оператор> b
```

Например

a += b равносильно **a = a + b**

a %= b равносильно **a = a % b**

Допустимы следующие операции

+= присвоение результата сложения

-= присвоение результата вычитания

***=** присвоение результата умножения

/= присвоение результата от деления

****=** присвоение степени числа

//= присвоение результата целочисленного деления

%= присвоение остатка от деления

Ввод данных

Для ввода данных используется функция ***input()***.

```
n = input()
```

При вызове этой функции выполнение программы приостанавливается. Происходит ожидание ввода данных. После набора на клавиатуре некоторого текста и нажатия клавиши ввода, набранный текст присваивается переменной n.

Допускается в круглых скобках указать некоторый текст, заключённый в апострофы или кавычки. В этом случае при приостановке программы на экране высвечивается данный текст в виде подсказки.

```
n = input('введите значение:')
print(n)
```

При выполнении этого фрагмента программы на экране можно будет увидеть следующее

```
введите значение:23
23
```

Немного исправим программу

```
n = input('введите значение:')
print(3 * n)
```

Получим следующий результат

```
введите значение:23
232323
```

Дело в том, что функция ***input()*** возвращает всегда строку. Если нам необходимо ввести численные значения, то надо воспользоваться функциями преобразования ***int()*** или ***float()***, соответственно для целых и вещественных данных.

```
n = int(input('введите значение:'))
print(3 * n)
```

Получим следующий результат

```
введите значение:23
69
```

Работа с другими системами счисления

В языке Python есть возможность переводить целые числа из произвольной системы счисления (не белее 36-ричной) в десятичную. Для этого служит функция ***int()***, с двумя параметрами. Первый параметр – это строка, содержащее число, записанное в некоторой системе счисления, а второй параметр – основание системы счисления, в которой записано число.

Например,

```
n = int('1001', 2)
print(n)
n = int('FF', 16)
print(n)
n = int('20', 36)
print(n)
```

В результате получим

```
9
255
72
```

Функции, которая бы переводила числа из десятичной системы в произвольную, в языке Python нет.

Однако есть возможность работать с целыми числами в двоичной, восьмеричной и шестнадцатеричной системах счисления.

Для записи числа в одной из этих систем счисления перед его значением в качестве префикса ставится цифра ***0*** и соответствующая латинская буква:

b – для двоичных чисел;

o – для восьмеричных чисел;

x – для шестнадцатеричных чисел;

Примеры:

0b1001 – запись двоичного числа 1001

0o177 – запись восьмеричного числа 177

0xFF – запись шестнадцатеричного числа FF

Такие числа могут участвовать в арифметических выражениях и операциях присваивания. При этом системой автоматически подставляется их десятичное значение.

В результате работы следующей программы

```
n = 0b1001
print(n)
n = 0xF0 + 0x10
print(n)
```

получим

```
9
256
```

Имеется возможность перевести десятичное число в двоичное, восьмеричное и шестнадцатеричное представление. Для этого служат функции bin(x), oct(x) и hex(x) соответственно.

Например, после выполнения следующего фрагмента программы

```
n = 0xF0 + 0x10
print(n)
s = bin(n)
print(s)
s = oct(n)
print(s)
s = hex(n)
print(s)
```

будет получен следующий результат

```
256
0b100000000
0o400
0x100
```

Важно, отметить, что тип результата этих функция – строка. Поэтому следующий пример

```
s = bin(2) + bin(2)
print(s)
```

даст результат

0b100b10

То есть, склеилось две строки, каждая из которых содержала двоичную запись числа 2.

А в результате выполнения следующей программы

```
s = 0b10 + 0b10
print(bin(s))
```

будет выведено

0b100

Складывается десятичное представление двух двоичных чисел со значением два. Результат – число 4. Выводится его двоичное представление.

Встроенные функции

Имеется ряд функций, встроенных в язык Python. С некоторыми из них мы уже знакомы. Например, ***int()***, ***float()***, ***bin()*** и т.д.

Познакомимся еще с несколькими полезными встроенными функциями

Функция	Назначение	Входные параметры	Тип результата	Примеры
abs(x)	Возвращает абсолютное значение (модуль) аргумента	x - значение типа int или float	Такой же, как и тип аргумента	abs (-2.745) = -2.745 abs (34) = 34
pow(x, y[, r])	(x ** y) % r, взятие остатка возможно только в случае целых x, y	x, y - значение типа int или float r - значение типа int	Тип результата аналогичен операции **	pow(4,2) = 16 pow(4.0,2) = 16.0 pow(4,2,3) = 1
max(a[, b[, c[, ...]]])	Возвращает максимальный элемент из a, b, c, ...	a, b, c, … - значение типа int или float	Такой же, как и тип элемента, оказавшегося максимальным	max(4.5, 2, 5) = 5 max(3.6, 2, 7.2, 4) = 7.2
min(a[, b[, c[, ...]]])	Возвращает минимальный элемент из a, b, c, ...	a, b, c, … - значение типа int или float	Такой же, как и тип элемента, оказавшегося минимальным	min(4.5, 2, 5) = 2 min (3.6, 1.7, 7.2, 2) = 1.7

round(X)	Возвращает округлённое по модулю до ближайшего целого значение числа X	X - значение типа int или float	значение типа int	round (3.1) = 3;
				round (-3.1)= -3**;**
				round (3.8) = 4;
				round (3.5) = 4;
				round (2.5) = 2;
				round (-3.8)= -4;

round(X, N)	Округление до N знаков после запятой	X - значение типа int или float N- значение типа int	Такой же, как и тип аргумента X	round(21.1234, 2) = 21.12 round(-21.783, 1) = -21.8 round(-21.783, 10) = -21.783 round(2, 5) = 2

Работа с модулями

Мы познакомились с некоторыми встроенными функциями в язык Python. Их количество ограничено наиболее часто употребимыми. Но язык Python позволяет подключать дополнительные наборы функций. Для этого используются готовые модули.

Рассмотрим для примера модуль ***math***. Этот модуль содержит много функций, полезных при выполнении математических вычислений.

Для того, чтобы начать работать с модулем, его необходимо подключить. Для этого служит оператор ***import***.

Существует несколько способов подключения модуля. Рассмотрим их.

1. import math

В этом случае подключается весь модуль. Для обращения к конкретной функции модуля используется следующий синтаксис:

< имя модуля > < точка > < имя функции >

```
import math
x = math.trunc(123.9)
print(x)
```

В данном случае будет использована функция целой части числа ***trunc()***, которая встроена в модуль ***math***.

2. Возможно использовать синоним имени модуля для более краткой записи

```
import math as m
x = m.trunc(123.9)
print(x)
```

Здесь имя ***m*** заменяет название модуля.

3. Если нужны не все функции модуля, а только несколько конкретных, то можно это сделать следующим образом:

```
from math import sin, trunc
x = trunc(123.9)
print(x)
```

В этом примере из модуля math импортируется только две функции sin() и trunc(). В этом случае уже нет необходимости ссылаться на сам модуль при вызове функции.

4. Для того, чтобы пользоваться всеми функциями модуля, не ссылаясь на него, можно воспользоваться еще одним способом подключения модуля.

```
from math import *
x = trunc(123.9)
print(x)
```

Однако при таком способе подключения надо помнить, что некоторые функции могут иметь одинаковое имя со встроенными функциями и перекрывать их. Например, функция pow(), в модуле math работает несколько иначе по сравнению с со встроенной функцией pow().

Так как модуль math имеет много полезных функций, то ниже приведен список функций, входящих в него.

Список функций, входящих в модуль math.

math.ceil(X) – округление до ближайшего большего числа.

math.copysign(X, Y) - возвращает число, имеющее модуль такой же, как и у числа X, а знак - как у числа Y.

math.fabs(X) - модуль X.

math.factorial(X) - факториал числа X.

math.floor(X) - округление вниз.

math.fmod(X, Y) - остаток от деления X на Y.

math.frexp(X) - возвращает мантиссу и экспоненту числа.

math.ldexp(X, I) - $X * 2^i$. Функция, обратная функции math.frexp().

math.fsum(последовательность) - сумма всех членов последовательности. Эквивалент встроенной функции sum(), но math.fsum() более точна для чисел с плавающей точкой.

math.isfinite(X) - является ли X числом.

math.isinf(X) - является ли X бесконечностью.

math.isnan(X) - является ли X NaN (Not a Number - не число).

math.modf(X) - возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.

math.trunc(X) - усекает значение X до целого.

math.exp(X) - e^X.

math.expm1(X) - e^X - 1. При X → 0 точнее, чем math.exp(X)-1.

math.log(X, [base]) - логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.

math.log1p(X) - натуральный логарифм (1 + X). При X → 0 точнее, чем math.log(1+X).

math.log10(X) - логарифм X по основанию 10.

math.log2(X) - логарифм X по основанию 2. Новое в Python 3.3.

math.pow(X, Y) - X^Y.

math.sqrt(X) - квадратный корень из X.

math.acos(X) - арккосинус X. В радианах.

math.asin(X) - арксинус X. В радианах.

math.atan(X) - арктангенс X. В радианах.

math.atan2(Y, X) - арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).

math.cos(X) - косинус X (X указывается в радианах).
math.sin(X) - синус X (X указывается в радианах).
math.tan(X) - тангенс X (X указывается в радианах).
math.hypot(X, Y) - вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x * x + y * y)).
math.degrees(X) - конвертирует радианы в градусы.
math.radians(X) - конвертирует градусы в радианы.
math.cosh(X) - вычисляет гиперболический косинус.
math.sinh(X) - вычисляет гиперболический синус.
math.tanh(X) - вычисляет гиперболический тангенс.
math.acosh(X) - вычисляет обратный гиперболический косинус.
math.asinh(X) - вычисляет обратный гиперболический синус.
math.atanh(X) - вычисляет обратный гиперболический тангенс.
math.erf(X) - функция ошибок.
math.erfc(X) - дополнительная функция ошибок (1 - math.erf(X)).
math.gamma(X) - гамма-функция X.
math.lgamma(X) - натуральный логарифм гамма-функции X.
math.pi - pi = 3,1415926...
math.e - e = 2,718281...