**Вычисления**

В курсе программирования можно выделить две важнейшие составляющие – алгоритмы и способы организации данных5. В этом параграфе мы познакомимся с простейшими типами данных. В следующих разделах рассматриваются основные алгоритмические конструкции: ветвления, циклы, подпрограммы. В конце главы подробно изучаются сложные (составные) типы данных: списки, символьные строки, а также работа с файлами.

**Типы данных**

Перечислим основные типы данных в языке Python:

* **int** – целые значения;
* **float** – вещественные значения (могут быть с дробной частью);
* **bool** – логические значения, **True** (истина, «да») или **False** (ложь, «нет»);
* **str** – символ или символьная строка, то есть цепочка символов.

Кроме них есть ещё и другие, с ними мы познакомимся позже.

Тип переменной определяется в тот момент, когда её присваивается новое значение. Мы уже видели, что для определения типа переменной можно использовать стандартную функцию **type**. Например, программа

**a = 5**

**print ( type(a) )**

**a = 4.5**

**print ( type(a) )**

**a = True**

**print ( type(a) )**

**a = "Вася"**

**print ( type(a) )**

выдаст на экран такой результат:

**<class 'int'>**

**<class 'float'>**

**<class 'bool'>**

**<class 'str'>**

Сначала в переменной **a** хранится целое значение 5, и её тип – целый (**int**). Затем мы записываем в неё вещественное значение 4,5, переменная будет вещественного типа (**float**, от англ. *floating point* – с плавающей точкой). Третье присваивание – логическое значение (**bool**, от англ. *boolean* – булевская величина, в честь Дж. Буля). Последнее значение – символьная строка (**str**, от англ. *string* – строка), которая записывается в апострофах или в кавычках.

Целые переменные в Python могут быть сколь угодно большими (или, наоборот, маленькими, если речь идет об отрицательных числах): транслятор автоматически выделяет область памяти такого размера, который необходим для сохранения результата вычислений. Поэтому в Python легко (в отличие от других языков программирования) точно выполнять вычисления с большим количеством значащих цифр.

Вещественные переменные, как правило, занимают 8 байтов, что обеспечивает точность 15 значащих десятичных цифр. Как мы обсуждали в главе 4, большинство вещественных чисел хранится в памяти неточно, и в результате операций с ними накапливается вычислительная ошибка.

Поэтому для работы с целочисленными данными не стоит использовать вещественные переменные.

Логические переменные относятся к типу **bool** и принимают значения **True** (истина) или **False** (ложь).

Как вы увидели, одна и та же переменная в различных частях программы может хранить значения различных типов. Дело в том, что в Python имя переменной связывается с некоторым объектом, этот объект имеет определённый тип и содержит данные. При выполнении оператора

**a = 5**

в памяти создаётся объект – ячейка для хранения целого числа, которая связывается с именем **a**

(рис. a). Затем команда

**a = 4.5**

создаёт в памяти другой объект (для хранения вещественного числа) и переставляет ссылку для имени **a** (рис. б). Объект, на который не ссылается ни одно имя, удаляется из памяти «сборщиком мусора».



**b= a**

то два имени будут связаны с одним и тем же объектом (рис. в). Можно было бы подумать, что изменение одной из переменных приведет к такому же изменению другой. Однако *для чисел* это не так. Дело в том, что при присваивании

**a = 10**

в памяти будет создан новый объект – целое число 10, который связывается с именем **a** (рис. г). Это отличается от других языков программирования, где в таких случаях просто изменяется значение той же самой ячейки памяти.

**Арифметические выражения и операции**

Арифметические выражения в любом языке программирования записываются в строчку, без многоэтажных дробей. Они могут содержать числа, имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например,

**a =(c + 5 - 1) / 2\*d**

Если запись длинного выражения не поместилась в одной строке на экране, её можно перенести на следующую с помощью знака «**\**» (он называется «обратный слэш»):

**a =(c + 5 - 1) \**

**/ 2\*d**

При переносе внутри скобок знак «**\**» вставлять не обязательно:

**a =(c + 5**

**- 1) / 2\*d**

Эти правила переноса справедливы и для других операторов языка Python.

При определении порядка действий используется *приоритет* (старшинство) операций. Они выполняются в следующем порядке:

* действия в скобках;
* возведение в степень (**\*\***), справа налево;
* умножение (**\***) и деление (**/**), слева направо;
* сложение и вычитание, слева направо.

Таким образом, умножение и деление имеют одинаковый приоритет, более высокий, чем сложение и вычитание. Поэтому в приведенном примере значение выражения, заключенного в скобки, сначала разделится на 2, а потом – умножится на **d**.

В Python (как и в языке Cи) разрешено множественное присваивание. Запись

**a = b = 0**

равносильна паре операторов

**b = 0**

**a = b**

Так же, как и в Си, часто используют сокращенную запись арифметических операций:

|  |  |
| --- | --- |
| *сокращенная запись* | *полная запись* |
| **a += b**  **a -= b**  **a \*= b**  **a /= b** | **a = a + b**  **a = a - b**  **a = a \* b**  **a = a / b** |

Если в выражение входят переменные разных типов, в некоторых случаях происходит автоматическое приведение типа к более «широкому». Например, результат умножения целого числа на вещественное – это вещественное число. Переход к более «узкому» типу автоматически не выполняется. Нужно помнить, что результат деления (операции «/») – это вещественное число, даже если делимое и делитель – целые и делятся друг на друга нацело6.Часто нужно получить целый результат деления целых чисел и остаток от деления. В этом случае используют соответственно операторы «**//**» и «**%**»(они имеют такой же приоритет, как умножение и деление):

**d = 85**

**a = d // 10 # = 8**

**b = d % 10 # = 5**

Обратим внимание на результат выполнения этих операций для отрицательных чисел. Программа

**print ( -7 // 2 )**

**print ( -7 % 2 )**

выдаст на экран числа «–4» и 1. Дело в том, что с точки зрения теории чисел остаток – это неотрицательное число, поэтому –7 = (–4)∙2+1, то есть частное от деления (–7) на 2 равно –4, а остаток равен 1. Поэтому в Python (в отличие от многих других языков, например, Паскаля и Си) эти операции выполняются математически правильно.

В Python есть операция возведения в степень, которая обозначается двумя звездочками: «**\*\***». Например, выражение y=2x2+z3 запишется так:

**y = 2\*x\*\*2 + z\*\*3**

Возведение в степень имеет более высокий приоритет, чем умножение и деление.

**Вещественные значения**

При записи вещественных чисел в программе целую и дробную часть разделяют не запятой (как принято в отечественной математической литературе), а точкой. Например

**x = 123.456**

Вещественные значения по умолчанию выводятся на экран с большим количеством значащих цифр, например

**x = 1/3**

**print ( x ) # 0.3333333333333333**

При выводе очень больших или очень маленьких чисел используется так называемый научный (или *экспоненциальный*) формат. Например, программа:

**x = 100000000000000000/3**

**print ( x )**

выведет

**3.3333333333333332e+16**

что означает 3,3333333333333332⋅1016 , то есть до буквы «e» указывают значащую часть числа, а после нее – порядок.

Часто используют *форматный вывод*: все данные, которые нужно вывести, сначала преобразуют в символьную строку с помощью функции **format**:

**a = 1/3**

**print ( "{:7.3f}".format(a) )**

В данном случае использован формат «**7.3f**», который определяет вывод числа с фиксированной запятой (**f** от англ. *fixed* – фиксированный) в 7 позициях с тремя знаками в дробной части:

**◦◦0.333**

Поскольку эта запись занимает 5 позиций (а под нее отведено 7), перед числом добавляются два пробела, обозначенные знаком «**◦**».

Можно использовать форматный вывод для нескольких значений сразу (список этих значений заключается в круглые скобки):

**a = 1/3**

**b = 1/9**

**print ( "{:7.3f} {:7.3f}".format(a, b) ) #◦◦0.333◦◦◦0.111**

Запись «**%e**» обозначает экспоненциальный формат:

**print ( "{:10.3e} {:10.3e}".format(a, b) )**

Здесь числа 10 и 3 – это общее количество позиций и число знаков после десятичной точки в записи значащей части:

**◦3.333e-01◦◦1.111e-01**

**Стандартные функции**

Библиотека языка Python содержит большое количество готовых функций, которые можно вызывать из программы. Некоторые функции встроены в ядро языка, например, для вычисления модуля числа используется функция **abs**:

**print ( abs(-1.2) ) # 1.2**

Существуют встроенные функции для перехода от вещественных значений к целым:

**int(x)** – приведение вещественного числа x к целому, отбрасывание дробной части;

**round(x)** – округление вещественного числа x к ближайшему целому.

Большинство стандартных функций языка Python разбиты на группы по назначению, и каждая группа записана в отдельный файл, который называется *модулем*. Математические функции собраны в модуле **math**:

**sqrt(x)** – квадратный корень числа **x**;

**sin(x)** – синус угла **x**, заданного в радианах;

**cos(x)** – косинус угла **x**, заданного в радианах;

**exp(x)** – экспонента числа **x**;

**log(x)** – натуральный логарифм числа **x**.

Для подключения этого модуля используется команда *импорта* (загрузки модуля):

**import math**

После этого для обращение к функциям используется так называемая *точечная запись*: указывают имя модуля и затем через точку название функции:

**print ( math.sqrt(x) )**

Можно поступить по-другому: загрузить в рабочее пространство все функции модуля:

**from math import \***

Теперь к функциям модуля **math** можно обращаться так же, как к встроенным функциям:

**print ( sqrt(x) )**

Этот способ обладает серьёзным недостатком: в рабочем пространстве появляется много дополнительных имён, которые могут совпасть с именами функций, объявленных в других модулях. Поэтому без острой необходимости лучше так не делать.

Третий вариант – загрузить только нужные функции

**from math import sqrt, sin, cos**

В этом случае все функции модуля **math**, кроме перечисленных (**sqrt**, **sin**, **cos**) будут недоступны

**Случайные числа**

В некоторых задачах необходимо моделировать случайные явления, например, результат бросания игрального кубика (на нём может выпасть число от 1 до 6). Как сделать это на компьютере, который по определению «неслучаен», то есть строго выполняет заданную ему программу?

Случайные числа – это последовательность чисел, в которой невозможно предсказать следующее число, даже зная все предыдущие. Чтобы получить истинно случайные числа, можно, например, бросать игральный кубик или измерять какой-то естественный шумовой сигнал (например, радиошум или электромагнитный сигнал, принятый из космоса). На основе этих данных составлялись и публиковались таблицы случайных чисел, которые использовали в разных областях науки.

Вернёмся к компьютерам. Ставить сложную аппаратуру для измерения естественных шумов или космического излучения на каждый компьютер очень дорого, и повторить эксперимент будет невозможно – завтра все значения будут уже другие. Существующие таблицы слишком малы, когда, скажем, нужно получать 100 случайных чисел каждую секунду. Для хранения больших таблиц требуется много памяти. Чтобы выйти из положения, математики придумали алгоритмы получения *псевдослучайных* («как бы случайных») чисел. Для «постороннего» наблюдателя псевдослучайные числа практически неотличимы от случайных, но они вычисляются по некоторой математической формуле: зная первое число («зерно») можно по формуле вычислить второе, затем третье и т.п.

Функции для работы с псевдослучайными числами собраны в модуле **random**. Для получения псевдослучайных чисел в заданном диапазоне используют функции:

**random()** – случайное вещественное число из полуинтервала [0,1);

**randint(a,b)** – случайное целое число из отрезка [a,b].

Для того, чтобы записать в переменную **n** случайное число в диапазоне от 1 до 6 (результат бросания кубика), можно использовать такие операторы:

**from random import randint**

**n = randint(1,6)**

Вещественное случайное число в полуинтервале от 5 до 12 (не включая 12) получается так:

**from random import random**

**x = 7\*random() + 5**