ОГЛАВЛЕНИЕ

1

ИНТЕРАКТИВНАЯ СРЕДА РАЗРАБОТКИ

Прежде чем приступить к разработке игр, вам нужно изучить несколько базовых концепций программирования. Эту главу вы начнете с того, что научитесь использовать интерактивную среду разработки Python и выполнять базовые арифметические задачи.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Операторы
- Целые числа и числа с плавающей запятой
- Значения
- Выражения
- Синтаксические ошибки в коде
- Сохранение значений в переменных

Немного арифметики

Запустите IDLE, выполнив указания из раздела «Запуск интерактивной среды разработки» во введении. На первых порах вы будете использовать Python для решения простых арифметических задач. IDLE может быть использована как калькулятор. Введите `2 + 2` в интерактивной оболочке после символов приглашения, **>>>**, и нажмите клавишу **Enter**. (На некоторых клавиатурах эта клавиша обозначена как **Return**.) На рис. 1.1 показано, как

эта арифметическая задача выглядит в интерактивной оболочке — обратите внимание, что получен ответ в виде числа 4.

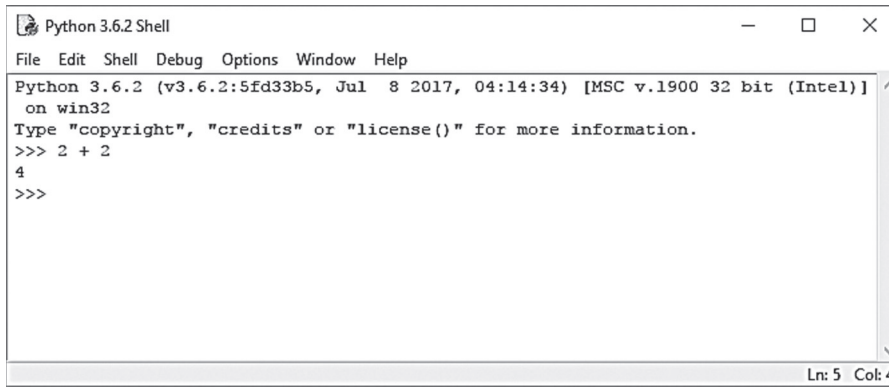


Рис. 1.1. Ввод `2 + 2` в интерактивной оболочке

Эта арифметическая задача — простая инструкция программирования. Символ «плюс» (+) указывает компьютеру сложить 2 и 2. Компьютер выполняет операцию и отвечает числом 4 в следующей строке. Таблица 1.1 содержит доступные в Python арифметические операторы.

Таблица 1.1. Арифметические операторы

Оператор	Операция
`+`	Прибавить
`-`	Вычесть
`*`	Умножить
`/`	Разделить

Символ минус (-) вычитает числа, звездочка (*) умножает, а косая черта (/) делит. При использовании таким образом, знаки +, -, * и / называются *операторами*. Операторы сообщают Python, какую операцию необходимо произвести над числами.

Целые числа и числа с плавающей запятой

Целые числа (на английском `int`) представляют собой числа, такие как 4, 99 или 0. *Числа с плавающей запятой* (на английском `float`) представляют собой дроби или числа с десятичными точками, такие как 3.5, 42.1 и 5.0. Для Python число 5 — целое, а 5.0 — число с плавающей запятой.

Эти числа называют *значениями*. Позже вы узнаете о других видах значений помимо чисел. В арифметической задаче, введенной в интерактивной оболочке, 2 и 2 — это целые (или *целочисленные*) значения.

Выражения

Арифметическая задача `2 + 2` — пример *выражения*. Как демонстрирует рис. 1.2, выражения состоят из значений (чисел), связанных между собой с помощью операторов (арифметических символов), что создает новое значение, которое может быть использовано в коде. Компьютер способен вычислить значения миллиардов выражений в секунду.

Введите следующие выражения в интерактивной оболочке, нажимая клавишу **Enter** после каждой строки.

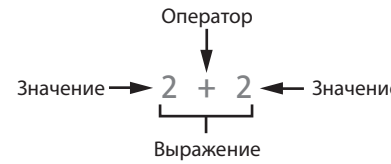


Рис. 1.2. Выражение, состоящее из значений и операторов

```
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2 +  2
4
```

Все эти выражения выглядят как обычные арифметические уравнения, но обратите внимание на пробелы в примере `2 +  2`. В Python вы можете поставить любое количество пробелов между значениями и операторами. Тем не менее вы всегда должны начинать инструкцию в начале строки без пробелов при вводе их в интерактивной оболочке.

Вычисление выражений

Когда среда вычисляет выражение `10 + 5` и возвращает значение `15`, это значит, что он его *вычислил*. Вычисление выражения сводит его к единому значению, так же как и решение арифметической задачи сводит ее к единственному числу — ответу. Например, два выражения `10 + 5` и `10 + 3 + 2` вычисляются с результатом `15`.

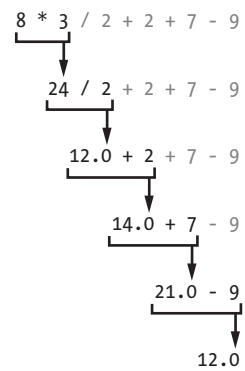
Когда Python вычисляет выражение, он придерживается того же порядка арифметических операций, какому вас научили в школе.

Правила следующие:

- Сначала вычисляются части выражения внутри круглых скобок.
- Умножение и деление выполняются перед сложением и вычитанием.
- Порядок действий слева направо.

Выражение `1 + 2 * 3 + 4` равно `11`, а не `13`, потому что сначала выполняется умножение `2 * 3`. Если бы выражение выглядело как `(1 + 2) * (3 + 4)`, оно бы равнялось `21`, потому что части `(1 + 2)` и `(3 + 4)` находятся внутри скобок и вычисляются перед умножением.

Выражения могут быть любого размера, и Python вычислит значение каждого из них. Даже одиночные цифры — это выражения. Например, выражение `15` является выражением со значением `15`. Выражение `8 * 3 / 2 + 2 + 7 - 9` равняется значению `12.0` после следующих вычислений:



Несмотря на то что компьютер выполняет все эти шаги, вы не видите их в интерактивной оболочке . Интерактивная среда разработки показывает вам только результат.

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Обратите внимание, что выражения с оператором (/) деления Python в итоге приводит к числу с плавающей запятой; Например, `24 / 2` вычисляется как `12.0`. Математические операции хотя бы с одним значением с плавающей запятой так же записываются и в дальнейшем, поэтому `12.0 + 2` вычисляется как `14.0`.

Синтаксические ошибки

Если в интерактивной оболочке вы напишете `5 +` и нажмете клавишу **Enter**, то получите следующее сообщение об ошибке:

```
>>> 5 +
SyntaxError: invalid syntax
```

Эта ошибка произошла потому, что `5 +` не является выражением. Выражения состоят из значений, соединенных операторами. В данном случае оператор сложения `+` подразумевает наличие значений с *двух* сторон. Сообщение об ошибке появляется, когда ожидаемое значение отсутствует.

«Синтаксическая ошибка» означает, что Python не понимает инструкцию, потому что вы задали ее неверно. При написании программ важно не только давать инструкции компьютеру, но и знать, как делать это правильно.

Впрочем, не переживайте из-за ошибок в коде. Ошибки не нанесут вреда вашему компьютеру. Просто правильно перепишите инструкцию в интерактивной оболочке после следующего приглашения `>>>`.

Сохранение значений в переменных

Вы можете использовать полученные значения выражений, сохраняя их в *переменных*. Для простоты можете представить, что переменная — это такая «коробка», в которой вы можете хранить значения. Для сохранения значения выражения в переменной используйте *инструкцию присваивания*. Введите имя переменной, затем знак равенства (`=`), который называется *оператором присваивания*, и затем значение. Например, введите в интерактивной оболочке следующую команду:

```
>>> spam = 15
>>>
```

В «коробке» переменной `spam` теперь хранится значение `15`, как показано на рис. 1.3.

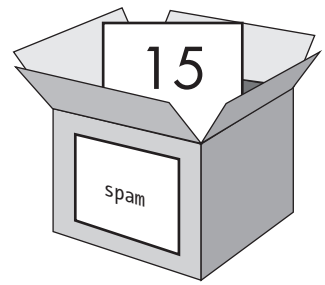


Рис. 1.3. Переменные подобны «коробкам», которые могут содержать значения