2

ВЫЧИСЛЕНИЯ И ПЕРЕМЕННЫЕ

Итак, вы установили Python и знаете, как запускать его командную оболочку, а значит, пора использовать его по назначению. Мы начнем с простых математических расчетов, а затем перейдем к важной части языка — переменным. *Переменные* — это удобный способ хранения данных в программе, и они пригодятся нам для решения самых разных задач.

Вычисления в Python

Если нужно перемножить два числа, к примеру узнать, сколько будет 8 × 3,57, мы обычно пользуемся калькулятором либо берем ручку и умножаем в столбик на листе бумаги. А что если использовать для подсчетов оболочку Python? Давайте попробуем.

Запустите оболочку, дважды кликнув по значку IDLE на рабочем столе, либо, если у вас система Ubuntu, кликнув по значку IDLE в меню **Applications**. Затем после значка >>> введите выражение и нажмите Enter:

```
>>> 8 * 3.57
28.56
```

Обратите внимание, что при записи числа 3,57 используется не запятая, а точка. Кроме того, в Python числа перемножаются с помощью звездочки (*), а не знака умножения (×).

Теперь рассмотрим более полезную задачу.

Представьте, что вы рыли яму и случайно нашли кошелек с 20 золотыми монетами. На следующий день вы тихонько залезли в подвал, где

стоит изобретение вашего дедушки — работающий на паровом ходу механизм для копирования предметов, и, на ваше счастье, в него удалось запихнуть все 20 монет. Раздался свист, потом щелчок, и устройство выдало еще 10 новеньких монеток.

Сколько монет вы накопите, если будете проделывать эту операцию каждый день в течение года? На бумаге эти расчеты выглядят примерно так:

$10 \times 365 = 3650$
$20 + 3650 = 3670$

Что ж, ничего сложного, осталось лишь выяснить, как посчитать то же в оболочке Python. Первым делом умножаем 10 монет на 365 дней, получится 3650. Затем добавим 20 монет, которые были изначально, и выйдет 3670.

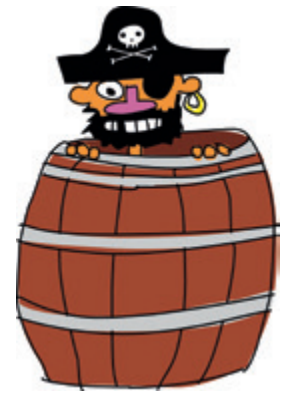
```
>>> 10 * 365
3650
>>> 20 + 3650
3670
```

Но что если о вашем богатстве узнает пронырливая ворона? Предположим, она будет каждую неделю залетать в окно и красть по три монетки.

Сколько у вас будет монет через год? В оболочке Python эти расчеты будут выглядеть так:

```
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Сперва умножаем 3 монеты на 52 недели в году, получаем 156. Затем вычитаем это значение из общего количества монет. Выходит, через год у вас останется 3514 монет.



Получилась очень простая программа. Изучая эту книгу дальше, вы узнаете, как писать более сложные и полезные программы.

Операторы в Python

В оболочке Python можно умножать, складывать, вычитать и делить числа, а также совершать некоторые другие операции, о которых мы узнаем позже. Символы, с помощью

которых выполняются математические действия в языке Python, называются *операторами*. Основные математические операторы перечислены в таблице 2.1.

Символ	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Таблица 2.1. Основные операторы в Python

Прямой слеш (/) обозначает деление, этот символ похож на линию между числителем и знаменателем дроби. Например, у вас 100 пиратов и 20 больших бочек, и вы хотите рассчитать, сколько пиратов можно спрятать в каждой бочке. Для этого следует разделить 100 пиратов на 20 бочек, введя в оболочке 100 / 20. И запомните — прямым слешем называют черту, верх которой наклонен вправо.

Порядок выполнения операций

Операции — это любые действия, которые совершаются с помощью операторов. Математические операции выполняются по очереди в зависимости от их приоритета (если не задать другую очередность с помощью скобок). Умножение и деление имеют более высокий приоритет, чем сложение и вычитание, и это значит, что они будут выполняться первыми. Иначе говоря, при вычислении математического выражения Python сначала умножит и разделит числа, а затем перейдет к сложению и вычитанию.

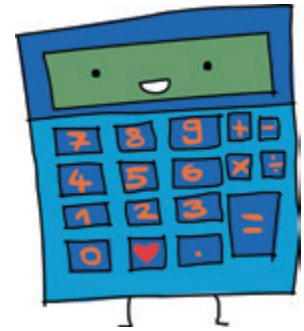
Например, в этом выражении сперва будут перемножены числа 30 и 20, а затем к их произведению будет прибавлено число 5.

```
>>> 5 + 30 * 20
605
```

По сути это выражение означает «умножить 30 на 20 и прибавить к результату 5». Получается 605. Однако мы можем изменить порядок операций, заключив первые два числа в скобки. Вот так:

```
>>> (5 + 30) * 20
700
```

В результате получилось 700, а не 605, поскольку Python выполняет операции в скобках прежде, чем операции вне скобок. Другими словами, это выражение означает «прибавить 5 к 30 и умножить результат на 20».



Скобки могут быть *вложенными*, то есть внутри скобок могут стоять еще одни скобки:

```
>>> ((5 + 30) * 20) / 10
70.0
```

В этом примере Python сперва вычислит выражение во внутренних скобках, затем во внешних и в самом конце выполнит стоящую за скобками операцию деления.

Иначе говоря, это выражение означает «прибавить 5 к 30, затем умножить результат на 20, потом разделить результат на 10». Вот что при этом происходит:

- сложение 5 и 30 дает 35;
- умножение 35 на 20 дает 700;
- деление 700 на 10 дает окончательный результат — 70.

Если бы мы не использовали скобки, результат вышел бы другим:

```
>>> 5 + 30 * 20 / 10
65.0
```

В этом случае сперва 30 умножается на 20 (получается 600), затем 600 делится на 10 (выходит 60) и, наконец, к 60 прибавляется 5, что дает в итоге 65.

Запомните, что умножение и деление всегда выполняются прежде, чем сложение и вычитание, если не менять порядок вычислений с помощью скобок.

Переменные как ярлыки для данных

В программировании слово *переменная* обозначает именованное место для хранения данных, например чисел, текста, списков с числами или символами и так далее. Также переменную можно рассматривать как ярлык, которым помечены некие данные.

Например, чтобы создать переменную с именем `fred`, нужно указать имя, поставить знак «равно» (=) и ввести соответствующие данные. Давайте создадим переменную `fred` (Фред), указав, что ей соответствует значение 100 (однако из этого не следует, что другая переменная не может иметь такое же значение):

```
>>> fred = 100
```

Чтобы напечатать значение нашей переменной, введите в оболочке Python команду `print` и следом за ней — имя переменной в скобках. Вот так:

```
>>> print(fred)
100
```

Можно изменить значение переменной `fred` — сделать так, чтобы ей соответствовали другие данные. Например, вот как заменить значение `fred` числом 200:

```
>>> fred = 200
>>> print(fred)
200
```

В первой строке говорится, что переменной `fred` теперь соответствует число 200. Во второй строке мы запрашиваем значение `fred`, чтобы убедиться, что оно поменялось. Последней строкой Python печатает ответ.

Можно использовать несколько переменных для одного и того же значения:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

В этом примере знак «равно» между именами `john` (Джон) и `fred` говорит о том, что переменной `john` соответствует значение переменной `fred`.

Конечно, `fred` — не самое удачное имя переменной, поскольку оно не поясняет, для чего эта переменная используется. Лучше назовем переменную не `fred`, а, допустим, `number_of_coins` (количество монет):

```
>>> number_of_coins = 200
>>> print(number_of_coins)
200
```

Number of coins — количество монет

Теперь понятно, что речь идет о двухстах монетах.

Имена переменных могут состоять из латинских букв, цифр и знака подчеркивания (_), однако начинаться с цифры они не могут. В остальном допустимо использовать любые имена, которые могут состоять как из отдельных букв (например, `a`), так и из целых предложений (пробелы в именах недопустимы, но слова можно разделять знаками подчеркивания). Для небольших программ часто удобны короткие имена, но в целом желательно, чтобы имя переменной отражало смысл, который вы вкладываете в ее использование.

Теперь вы знаете, как создавать переменные. Давайте посмотрим, что с ними можно делать.

Использование переменных

Помните, как мы вычисляли, сколько монет накопится за год, если каждый день создавать новые монеты с помощью изобретения вашего дедушки? Итак, вот на чем мы остановились:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Все это можно записать одной строкой кода:

```
>>> 20 + 10 * 365 - 3 * 52
3514
```

А что если заменить в этом выражении числа переменными? Введите:

```
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
```

Found coins — найденные монеты
Magic coins — волшебные монеты
Stolen coins — украденные монеты

Мы создали три переменные: `found_coins`, `magic_coins` и `stolen_coins`.