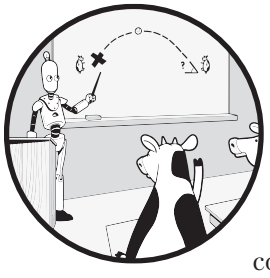
2

Принятие решений



Большинство программ, которыми мы пользуемся в обычной жизни, ведут себя по-разному в зависимости от тех или иных обстоятельств, возникающих во время их выполнения. Например, когда текстовый редактор спрашивает нас, хотим ли мы сохранить выполненную работу, он принимает решение в зависимости от нашего ответа: сохраняет работу, если мы отвечаем «да», и не сохраняет, если отвечаем «нет». В этой главе поговорим об операторе `if`, который позволяет программам принимать решения.

Мы попробуем решить две задачи: определить результат баскетбольного матча и выяснить, принадлежит ли номер телефона телемаркетологу.

Задача 3. Команда-победитель

Нам потребуется вывести сообщение, содержание которого будет зависеть от результата баскетбольного матча. Для этого нужен оператор `if`. Вы также узнаете, как можно хранить в программе истинные и ложные значения и манипулировать ими.

Задача с сайта DMOJ, код `ccc19j1`.

Постановка задачи

В баскетболе очки даются за трехочковый, двухочковый и штрафной бросок с одним очком.

Мы посмотрели баскетбольный матч между командами «Яблоки» и «Бананы» и записали количество успешных трех-, двух- и одноочковых бросков каждой команды. Нужно определить, какая команда выиграла или же игра закончилась ничьей.

Входные данные

Вводятся шесть строк. Первые три — очки «Яблок», последние три — очки «Бананов».

- В первой строке указано количество успешных трехочковых бросков команды «Яблок».
- Во второй строке указано количество успешных двухочковых бросков команды «Яблок».
- В третьей строке указано количество успешных штрафных бросков команды «Яблок».
- В четвертой строке указано количество успешных трехочковых бросков команды «Бананы».
- В пятой строке указано количество успешных двухочковых бросков команды «Бананы».
- В шестой строке указано количество успешных штрафных бросков команды «Бананы».

Все эти строки — целые числа от 0 до 100.

Выходные данные

На выходе будет один символ.

- Если «Яблоки» набрали больше очков, чем «Бананы», выведите `A` (A — Apples, то есть «Яблоки»).
- Если «Бананы» набрали больше очков, чем «Яблоки», выведите `B` (B — Bananas, то есть «Бананы»).
- Если команды набрали одинаковое количество очков, выведите букву `T` (tie — «ничья»).

Условное выполнение

Мы значительно упростим себе жизнь, если воспользуемся знаниями, которые получили в главе 1. С помощью функций `input` и `int` считаем каждое из шести целых чисел. Создадим переменные, чтобы сохранить эти значения. Мы можем умножить количество успешных трехочковых бросков на 3, а количество успешных двухочковых — на 2. С помощью функции `print` выведем нужную букву.

Вы пока не знаете, как научить программу принимать решение об исходе игры. Покажу, зачем это вообще нужно, с помощью двух примеров.

Сначала рассмотрим тестовый пример:

```
5
1
3
1
1
1
```

«Яблоки» набрали 5 · 3 + 1 · 2 + 3 = 20 очков, а «Бананы» — 1 · 3 +1 · 2 + 1 = 6 очков. «Яблоки» выиграли, поэтому выведем букву

```
A
```

Во втором примере баллы команд поменяем местами:

```
1
1
1
5
1
3
```

На этот раз выиграли «Бананы», так что нужно вывести другую букву:

```
B
```

Наша программа должна уметь сравнивать общее количество баллов, набранных «Яблоками» и «Бананами», и на основании полученного результата выводить `A`, `B` или `T`. Для принятия подобных решений можно использовать оператор `if`. *Условие* — это истинное или ложное выражение, а оператор `if` задействует условия, чтобы определить, что делать дальше. Операторы `if` так и называются — *операторы условного выполнения*, так как на выполнение нашей программы влияют различные условия.

Сначала давайте введем новый тип данных, который позволяет хранить истинные или ложные значения, и посмотрим, как создавать выражения этого типа. Именно их мы позднее будем передавать оператору `if`.

Логический тип

Если мы передадим любое выражение функции `type`, в ответ она сообщит вам тип переданного выражения:

```
>>> type(14)
<class 'int'>
```

```
>>> type(9.5)
<class 'float'>
>>> type('hello')
<class 'str'>
>>> type(12 + 15)
<class 'int'>
```

Один из типов Python, с которым мы еще не встречались, — это тип Boolean (`bool`). В отличие от целых чисел, строк и чисел с плавающей точкой, у которых миллиарды возможных значений, у этого типа есть лишь два логических значения: `True` и `False`. Именно с их помощью записываются условия:

```
>>> True
True
>>> False
False
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Что с этими значениями можно делать? Например, для работы с числами у нас были математические операторы, такие как `+` и `-`, которые позволяли объединять значения в более сложные выражения. А для работы с логическими значениями понадобятся новые операторы.

Операторы сравнения

Больше ли число 5 числа 2? Меньше ли число 4, чем 1? Ответить на эти вопросы позволяют *операторы сравнения* Python. Они дают `True` или `False`, поэтому часто используются для написания *логических выражений*.

Оператор `>` принимает два операнда и возвращает `True`, если первый больше, чем второй, и `False` в противном случае:

```
>>> 5 > 2
True
>>> 9 > 10
False
```

Аналогично работает оператор `<`:

```
>>> 4 < 1
False
>>> -2 < 0
True
```

Операторы `>=` (больше или равно) и `<=` (меньше или равно):

```
>>> 4 >= 2
True
>>> 4 >= 4
True
>>> 4 >= 5
False
>>> 8 <= 6
False
```

Для определения равенства используется оператор `==`. Это именно два знака равенства, а не один (помните, что в качестве оператора присваивания применяется один знак равенства (`=`), но равенство он не проверяет):

```
>>> 5 == 5
True
>>> 15 == 10
False
```

Для определения неравенства используется оператор `!=`. Он возвращает `True`, если операнды не равны, и `False`, если равны:

```
>>> 5 != 5
False
>>> 15 != 10
True
```

Реальные программы не будут оценивать выражения, значения которых мы уже знаем. Нам и без Python известно, что 15 не равно 10. Как правило, в выражениях такого типа используются переменные. Например, результат выражения `number != 10` зависит от того, что находится в переменной `number`.

Операторы сравнения работают и со строками. При проверке равенства учитывается регистр:

```
>>> 'hello' == 'hello'
True
>>> 'Hello' == 'hello'
False
```

Одна строка меньше другой, если она идет первой по алфавиту:

```
>>> 'brave' < 'cave'
True
>>> 'cave' < 'cavern'
True
>>> 'orange' < 'apple'
False
```

А вот если есть и строчные, и прописные символы, то все интереснее:

```
>>> 'apple' < 'Banana'
False
```

Странно, да? Это связано с тем, как символы хранятся внутри компьютера. Как правило, символы верхнего регистра идут по алфавиту раньше символов нижнего регистра. А как насчет этого:

```
>>> '10' < '4'
True
```

Если бы это были числа, то результат был бы `False`. Но строки сравниваются посимвольно слева направо. Python сравнивает 1 и 4, и поскольку 1 меньше, оператор `<` возвращает `True`. Всегда следите за тем, чтобы значения были именно тех типов, которых вы ожидаете!

Есть оператор сравнения, который работает со строками, но не с числами, — оператор `in`. Он возвращает `True`, если первая строка встречается хотя бы один раз во второй, и `False` в противном случае:

```
>>> 'ppl' in 'apple'
True
>>> 'ale' in 'apple'
False
```

ПРОВЕРИМ ЗНАНИЯ

Какой результат даст следующий код?

```
a = 3
b = (a != 3)
print(b)
```

А. `True`

Б. `False`

В. `3`

Г. Этот код вызывает синтаксическую ошибку.

Ответ: **Б**. Выражение `a! = 3` дает `False`, и оно же выводится.

Оператор if

Теперь рассмотрим несколько вариантов оператора `if` в Python.

Одиночный оператор if

Предположим, у нас есть окончательные результаты в двух переменных, `apple_total` и `banana_total`, и мы хотим вывести `A`, если значение переменной `apple_total` больше, чем `banana_total`. Это делается вот так:

```
>>> apple_total = 20
>>> banana_total = 6
>>> if apple_total > banana_total:
...     print('A')
...
A
```

Python выводит `A`, как и следовало ожидать.

Оператор `if` начинается с ключевого слова `if`. *Ключевое слово* — это слово, которое имеет особое значение для Python и не может использоваться в качестве имени переменной. После слова `if` идет логическое выражение, за которым следует двоеточие, а затем один или несколько операторов с отступом. Операторы с отступом часто называют *блоком* оператора `if`. Блок выполняется, если логическое значение возвращает `True`, и пропускается, если логическое выражение дает `False`.

Обратите внимание на то, что приглашение изменится с `>>>` на `...`. Это напоминание о том, что мы находимся внутри блока оператора `if` и должны делать в коде отступы. Я выбрал отступ в четыре пробела. Некоторые программисты на Python делают отступы клавишей Tab, но мы в этой книге будем использовать исключительно пробелы.

Как только вы введете `print('A')` и нажмете клавишу Enter, появится еще одно приглашение `...`. Поскольку нам нечего добавить в данный оператор `if`, нажмите клавишу Enter еще раз, чтобы закрыть это приглашение и вернуться к приглашению `>>>`. Дополнительное нажатие клавиши Enter — особенность оболочки Python. При написании программы на Python в файл пустые строки не требуются.

Рассмотрим пример помещения двух операторов в блок `if`:

```
>>> apple_total = 20
>>> banana_total = 6
```

```
>>> if apple_total > banana_total:
...     print('A')
...     print('Apples win!')
...
A
Apples win!
```

Оба вызова `print` выполняются, что дает две строки вывода.

Давайте попробуем другой оператор `if`, на этот раз с логическим выражением, которое возвращает `False`:

```
>>> apple_total = 6
>>> banana_total = 20
>>> if apple_total > banana_total:
...     print('A')
...
```

На этот раз функция `print` *не* вызывается: выражение `apple_total > banana_total` дает `False`, поэтому блок оператора `if` пропускается.

Составной оператор с elif

Давайте с помощью последовательных операторов `if` выведем букву `A`, если выиграли «Яблоки», `B`, если выиграли «Бананы», и `T`, если ничья:

```
>>> apple_total = 6
>>> banana_total = 6
>>> if apple_total > banana_total:
...     print('A')
...
>>> if banana_total > apple_total:
...     print('B')
...
>>> if apple_total == banana_total:
...     print('T')
...
T
```

Блоки первых двух операторов `if` пропускаются, потому что их логические выражения ложны. А блок третьего оператора `if` выполняется, выводя `T`.

Когда вы ставите одно выражение `if` за другим, они независимы и каждое логическое выражение оценивается независимо от того, были ли предыдущие выражения равны `True` или `False`.

При любых заданных значениях `apple_total` и `banana_total` может выполняться только один из операторов `if`. Например, если выражение `apple_total > banana_total` имеет значение `True`, то первый оператор `if` сработает, а два других — нет. Можно написать код, чтобы подчеркнуть, что разрешен запуск только одного блока кода. Вот как это сделать:

```
❶ >>> if apple_total > banana_total:
  ...     print('A')
❷ ... elif banana_total > apple_total:
  ...     print('B')
❸ ... elif apple_total == banana_total:
  ...     print('T')
  ...
  T
```

Теперь у нас один оператор `if`, а не три отдельных. Чтобы сделать это, не нажимайте клавишу Enter в приглашении, а вместо этого введите строку `elif`.

Чтобы выполнить этот оператор `if`, Python вычисляет первое логическое выражение ❶. Если оно дает `True`, то выводится `A`, а остальные `elif` пропускаются. Если оно дает `False`, Python вычисляет второе логическое выражение ❷. Если оно дает `True`, то выводится `B` и последний `elif` пропускается. Если оно дает `False`, Python вычисляет третье логическое выражение ❸. Если это `True`, то выводится `T`.

Ключевое слово `elif` означает else-if. С помощью этой расшифровки напоминайте себе о том, что выражение `elif` проверяется только в том случае, если перед ним ни одна из частей не сработала.

Эта версия кода эквивалентна предыдущему коду, где применялись три отдельных оператора `if`. Если бы мы хотели разрешить существование возможности выполнения более чем одного блока, нам пришлось бы использовать три отдельных `if`.

Оператор if с оператором else

Мы можем использовать ключевое слово `else` для запуска альтернативного кода, если все логические выражения в операторе `if` вернули `False`. Вот пример:

```
>>> if apple_total > banana_total:
...     print('A')
... elif banana_total > apple_total:
...     print('B')
... else:
...     print('T')
...
T
```

Python вычисляет логические выражения сверху вниз. Если любое из них имеет значение `True`, он запускает связанный блок и пропускает остальную часть оператора `if`. Если все логические выражения имеют значение `False`, Python выполняет блок `else`.

Обратите внимание, что мы убрали сравнение `apple_total == banana_total`. Мы попадаем в блок `else`, только если `apple_total > banana_total` дает `False` и `banana_total > apple_total` дает `False`, то есть если значения равны.

Что лучше использовать — отдельные операторы `if`? Оператор `if` с несколькими `elif`? Оператор `if` с `else`? Часто это вопрос вкуса. Задействуйте цепочку `elif`, если хотите выполнить не более одного блока кода. `Else` может помочь сделать код более понятным и избавить от необходимости писать обобщающее логическое выражение. Соблюдение правильной логики гораздо важнее точного оформления оператора `if`!

ПРОВЕРИМ ЗНАНИЯ

Какое значение переменной `x` даст следующий код?

```
x = 5
if x > 2:
    x = -3
if x > 1:
    x = 1
else:
    x = 3
```

А. `-3`

Б. `1`

В. `2`

Г. `3`

Д. `5`

Ответ: **Г**. Поскольку `x > 2` дает `True`, выполняется блок первого оператора `if`. Выполняется присвоение `x = -3`. Теперь о втором операторе `if`. Здесь `x > 1` дает `False`, поэтому работает блок `else` и значение переменной `x` становится равным `3`. Попробуйте поменять `if x > 1` на `elif x > 1` и посмотреть, что изменится в поведении программы!

ПРОВЕРИМ ЗНАНИЯ

Одинаково ли работают приведенные фрагменты кода? Предположим, что в переменной `temperature` уже есть число.

Фрагмент 1:

```
if temperature > 0:
    print('warm')
elif temperature == 0:
    print('zero')
else:
    print('cold')
```

Фрагмент 2:

```
if temperature > 0:
    print('warm')
elif temperature == 0:
    print('zero')
print('cold')
```

А. Да.

Б. Нет.

Ответ: **Б**. Фрагмент 2 всегда выводит слово `cold`, поскольку команда `Print('cold')` написана без отступа и не находится внутри операторов условия!

Решение задачи

Вернемся к задаче «Команда-победитель». В книге я обычно показываю полный код, а затем привожу пояснения к нему. Но, поскольку здесь решение длиннее, чем в главе 1, я решил показать код тремя частями и только потом привести его целиком.

Во-первых, нужно прочитать ввод от пользователя. Для этого требуется шесть запросов ввода, потому что у нас есть две команды и три нужных числа для каждой из них. Потребуется преобразовать все введенные строки в целое число. А вот и код:

```
apple_three = int(input())
apple_two = int(input())
apple_one = int(input())
```

```
banana_three = int(input())
banana_two = int(input())
banana_one = int(input())
```

Во-вторых, нужно определить количество очков, набранных командами. Для каждой команды мы складываем очки за трех-, двух- и одноочковые броски. Это можно сделать следующим образом:

```
apple_total = apple_three * 3 + apple_two * 2 + apple_one
banana_total = banana_three * 3 + banana_two * 2 + banana_one
```

В-третьих, выводим результат. Если выигрывают «Яблоки», мы выводим `A`, если выигрывают «Бананы», выводим `B`, в противном случае у нас ничья, поэтому выводим `T`. Мы используем для этого оператор `if`, как показано далее:

```
if apple_total > banana_total:
    print('A')
elif banana_total > apple_total:
    print('B')
else:
    print('T')
```

Это весь код, который нам нужен. Полный код приведен в листинге 2.1.

Листинг 2.1. Решение задачи «Команда-победитель»

```
apple_three = int(input())
apple_two = int(input())
apple_one = int(input())

banana_three = int(input())
banana_two = int(input())
banana_one = int(input())

apple_total = apple_three * 3 + apple_two * 2 + apple_one
banana_total = banana_three * 3 + banana_two * 2 + banana_one

if apple_total > banana_total:
    print('A')
elif banana_total > apple_total:
    print('B')
else:
    print('T')
```

Если вы отправите код на сайт, он, скорее всего, пройдет все тесты.