Применение вложенных циклов

Ну и зачем нам нужны все эти вложенные циклы? А затем, что они, к примеру, позволяют найти все доступные *перестановки* и *сочетания* в серии решений.

НОВЫЕ СЛОВА

Перестановка — это математическое понятие, означающее уникальный способ комбинирования некоего множества объектов. Почти аналогичное значение имеет термин «сочетание». Просто в случае *сочетания* порядок следования объектов не имеет значения, в то время как при перестановках за ним строго следят. К примеру, я попросил вас выбрать три числа от 1 до 20, и вы выбрали:

- 5, 8, 14;
- 2, 12, 20;

и т. п. Если мы попытаемся сделать список всех перестановок из трех чисел от 1 до 20, то следующие два варианта должны считаться разными:

- 5, 8, 14;
- 8, 5, 14.

Так происходит как раз потому, что в случае перестановок порядок следования элементов имеет значение. А вот в случае списка сочетаний следующие три варианта будут рассматриваться как одинаковые:

- 5, 8, 14;
- 8, 5, 14;
- 8, 14, 5.

Ведь в случае сочетаний порядок следования элементов не имеет значения.

Проще всего объяснить это на примере. Представьте, что вы открыли палатку с хот-догами и хотите повесить рядом плакат, демонстрирующий, как по номеру можно заказать любое сочетание сосиски, булочки, кетчупа, горчицы и лука. Для этого нам сначала нужно определить все возможные сочетания.

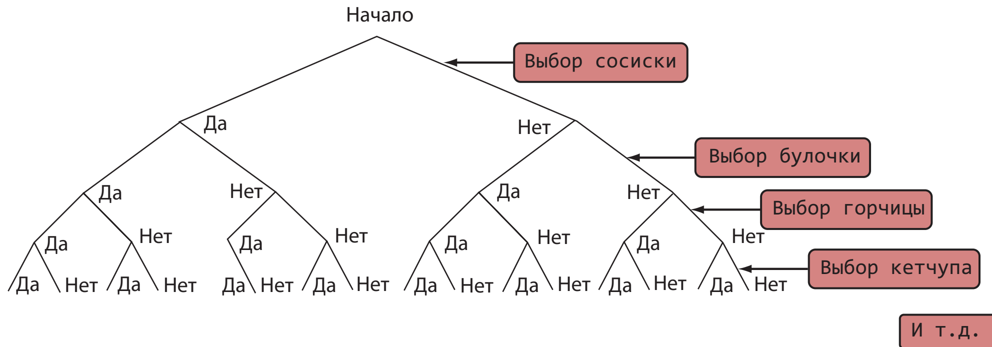
Эту задачу можно представить в виде так называемого *дерева решений*. Вот как оно выглядит для задачи с хот-догами.

В каждой точке принятия решения есть только два варианта: «Да» и «Нет». Все пути вниз по дереву описывают разные комбинации компонентов хот-дога. Выделенный мною путь говорит «Да» сосиске, «Нет» булочке, «Да» горчице и «Да» кетчупу.

А сейчас при помощи вложенных циклов мы сделаем список всех комбинаций — всех путей в дереве решений. Так как у нас пять точек принятия решения, дерево имеет пять

Купить книгу на сайте kniga.biz.ua >>>

уровней, а значит, нам потребуется пять вложенных циклов. (Дерево решений на рисунке имеет всего четыре уровня.)



Введите код из листинга 11.6 в IDLE-редактор и сохраните файл под именем ***hotdog1.py***.

Листинг 11.6. Варианты хот-дога

```
print "\tСосиска \tБулочка \tКетчуп\tГорчица\tЛук"
count = 1
for dog in [0, 1]:
    for bun in [0, 1]:
        for ketchup in [0, 1]:
            for mustard in [0, 1]:
                for onion in [0, 1]:
                    print "#", count, "\t",
                    print dog, "\t", bun, "\t", ketchup, "\t",
                    print mustard, "\t", onion
                    count = count + 1
```

Цикл горчицы
Цикл кетчупа
Цикл сосиски
Цикл булочки
Цикл лука

Видите, как циклы вставляются один в другой? Именно в этом проявляется вся суть вложенных циклов — перебор одних вариантов внутри других вариантов:

- внешний цикл для сосиски (`dog`) запускается дважды;
- цикл для булочки (`bun`) запускается дважды на каждой итерации цикла для сосиски. Значит, он работает $2 \times 2 = 4$ раза;
- цикл для кетчупа (`ketchup`) запускается дважды на каждой итерации цикла для булочки. Значит, всего он будет работать $2 \times 2 \times 2 = 8$ раз.

И т. д. Самый внутренний цикл (то есть расположенный в коде ниже всего цикл `onion`) запускается $2 \times 2 \times 2 \times 2 \times 2 = 32$ раза. Он покрывает все возможные сочетания. Значит, у нас есть 32 возможных варианта. Запустив программу из листинга 11.6, вы получите примерно такой результат:

Купить книгу на сайте kniga.biz.ua >>>

```
>>> ========================= RESTART ==========================
>>>
            Сосиска Булочка Кетчуп  Горчица Лук
# 1         0       0       0       0       0
# 2         0       0       0       0       1
# 3         0       0       0       1       0
# 4         0       0       0       1       1
# 5         0       0       1       0       0
# 6         0       0       1       0       1
# 7         0       0       1       1       0
# 8         0       0       1       1       1
# 9         0       1       0       0       0
# 10        0       1       0       0       1
# 11        0       1       0       1       0
# 12        0       1       0       1       1
# 13        0       1       1       0       0
# 14        0       1       1       0       1
# 15        0       1       1       1       0
# 16        0       1       1       1       1
# 17        1       0       0       0       0
# 18        1       0       0       0       1
# 19        1       0       0       1       0
# 20        1       0       0       1       1
# 21        1       0       1       0       0
# 22        1       0       1       0       1
# 23        1       0       1       1       0
# 24        1       0       1       1       1
# 25        1       1       0       0       0
# 26        1       1       0       0       1
# 27        1       1       0       1       0
# 28        1       1       0       1       1
# 29        1       1       1       0       0
# 30        1       1       1       0       1
# 31        1       1       1       1       0
# 32        1       1       1       1       1
```

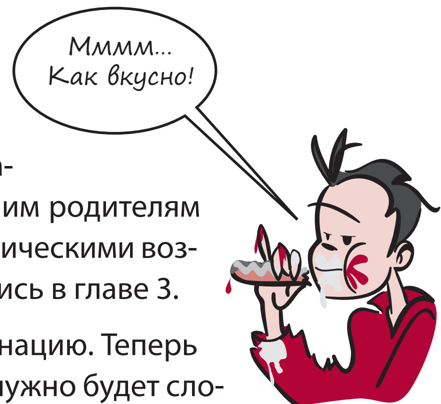
Пять вложенных циклов перебрали все возможные сочетания переменных `dog`, `bun`, `ketchup`, `mustard` и `onion`.

В листинге 11.6 выравнивание осуществлялось при помощи символа табуляции. Он записывается как `\t`. *Форматирование* выводимых результатов мы пока не обсуждали, но если вы хотите узнать об этом больше, загляните в главу 21.

Переменная `count` использовалась для нумерации сочетаний. Например, хот-дог, состоящий из булочки и горчицы, оказался под номером 27. Разумеется, некоторые из сочетаний не имеют смысла. (Хот-дог без булочки, но с горчицей и кетчупом выглядит несколько странно.) В то же время еще никто не отменял правила «Клиент всегда прав!».

Купить книгу на сайте kniga.biz.ua >>>

Подсчет калорий



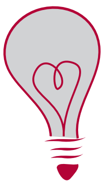
Так как в наши дни многие озабочены здоровым питанием, добавим в наш список комбинаций информацию о калорийности. (Допускаю, что вы совершенно не озабочены тем, сколько калорий потребляете, но вот вашим родителям это наверняка не безразлично!) Воспользуемся математическими возможностями языка Python, с которыми мы познакомились в главе 3.

Мы уже знаем, какие элементы входят в каждую комбинацию. Теперь выясним калорийность каждого элемента. После чего нужно будет сложить их в самом внутреннем цикле. Вот код, задающий калорийность каждого варианта:

```
dog_cal = 140
bun_cal = 120
mus_cal = 20
ket_cal = 80
onion_cal = 40
```

Осталось выполнить сложение. Мы знаем, что каждое сочетание в нашем меню содержит 0 или 1 штуку каждого элемента. Поэтому достаточно умножить это количество на соответствующую калорийность:

```
tot_cal = (dog * dog_cal) + (bun * bun_cal) + \
          (mustard * mus_cal) + (ketchup * ket_cal) + \
          (onion * onion_cal)
```



Так как в соответствии с приоритетом операций сначала выполняется умножение, а только потом сложение, скобки в данном случае не требуются. Я добавил их, чтобы было проще понять происходящее.

Соберем все вместе и напишем новую версию программы с подсчетом калорийности наших хот-догов (листинг 11.7).

Листинг 11.7. Программа создания хот-догов со счетчиком калорий

```
dog_cal = 140
bun_cal = 120
ket_cal = 80
mus_cal = 20
onion_cal = 40
```

Список калорий для каждого компонента хот-дога

Купить книгу на сайте kniga.biz.ua >>>

```
print "\tСосиска \tБулочка \tКетчуп\tГорчица\tЛук \tКалории"
count = 1
for dog in [0, 1]:
    for bun in [0, 1]:
        for ketchup in [0, 1]:
            for mustard in [0, 1]:
                for onion in [0, 1]:
                    total_cal = (bun * bun_cal)+(dog * dog_cal) + \
                        (ketchup * ket_cal)+(mustard * mus_cal) + \
                            (onion * onion_cal)
                    print "#", count, "\t",
                    print dog, "\t", bun, "\t", ketchup, "\t",
                    print mustard, "\t", onion,
                    print "\t", total_cal
                    count = count + 1
```

Вывод заголовков

Цикл dog — это внешний цикл

Подсчет калорий во внутреннем цикле

Вложенные циклы

ДЛИННЫЕ СТРОКИ КОДА

Обратили внимание на символы в виде обратной косой черты (\) на концах строк в предыдущем фрагменте кода? В случае, когда длинное выражение невозможно уместить в одну строку, этот символ сообщает интерпретатору Python: «Строка еще не кончилась. Учти, что следующая строка является продолжением этой». В данном случае, добавив две обратные косые черты, мы разбили одно длинное выражение на три коротких строки. Обратная косая черта, которая называется *символом переноса строки*, присутствует в ряде языков программирования.

Кроме того, выражение можно заключить в дополнительные скобки. Тогда его можно будет расположить на нескольких строках, не прибегая к символам переноса:

```
tot_cal = ((dog * dog_cal) + (bun * bun_cal) +
           (mustard * mus_cal) + (ketchup * ket_cal) +
           (onion * onion_cal))
```

Запустите в IDLE программу из листинга 11.7. Вот что в результате получится:

```
>>> =========================== RESTART ============================
>>>
             Сосиска Булочка Кетчуп  Горчица Лук     Калории
# 1          0       0       0       0       0       0
# 2          0       0       0       0       1       40
# 3          0       0       0       1       0       20
# 4          0       0       0       1       1       60
# 5          0       0       1       0       0       80
# 6          0       0       1       0       1       120
# 7          0       0       1       1       0       100
```

Купить книгу на сайте kniga.biz.ua >>>

```
# 8          0      0      1      1      1       140
# 9          0      1      0      0      0       120
# 10         0      1      0      0      1       160
# 11         0      1      0      1      0       140
# 12         0      1      0      1      1       180
# 13         0      1      1      0      0       200
# 14         0      1      1      0      1       240
# 15         0      1      1      1      0       220
# 16         0      1      1      1      1       260
# 17         1      0      0      0      0       140
# 18         1      0      0      0      1       180
# 19         1      0      0      1      0       160
# 20         1      0      0      1      1       200
# 21         1      0      1      0      0       220
# 22         1      0      1      0      1       260
# 23         1      0      1      1      0       240
# 24         1      0      1      1      1       280
# 25         1      1      0      0      0       260
# 26         1      1      0      0      1       300
# 27         1      1      0      1      0       280
# 28         1      1      0      1      1       320
# 29         1      1      1      0      0       340
# 30         1      1      1      0      1       380
# 31         1      1      1      1      0       360
# 32         1      1      1      1      1       400
```

Только представьте, как утомительно было бы вручную подсчитывать калорийность всех этих сочетаний даже с использованием калькулятора! Гораздо веселее написать программу, которая все сделает за вас. Циклы и немного математики позволяют решить задачу на языке Python быстро и просто!

Что мы узнали

В этой главе мы познакомились с:

- вложенными циклами;
- переменными циклов;
- перестановками и сочетаниями;
- деревьями решений.

Проверь себя

1. Как в Python реализуется цикл с переменной цикла?
2. Как в Python реализуется вложенный цикл?

Купить книгу на сайте kniga.biz.ua >>>

3. Сколько звездочек выведет на экран следующий код?

```
for i in range(5):
    for j in range(3):
        print '*',
    print
```

4. Как будет выглядеть результат работы программы из задания 3?
5. У нас есть четырехуровневое дерево решений, и на каждом уровне предлагается два возможных варианта. Сколько всего существует вариантов (путей вниз по дереву решений)?

Эксперименты

1. Помните таймер обратного отсчета, который мы написали в главе 8? Вот как он выглядел:

```
import time
for i in range (10, 0, -1):
    print i
    time.sleep(1)
print "ПУСК!"
```

Напишите эту программу с использованием переменной цикла. Она должна спрашивать пользователя, с какого момента следует начать обратный отсчет:

```
Таймер обратного отсчета: Сколько секунд? 4
4
3
2
1
ПУСК!
```

2. Доработайте предыдущую программу, чтобы рядом с каждым числом она выводила соответствующее количество звездочек:

```
Таймер обратного отсчета: Сколько секунд? 4
4 * * * *
3 * * *
2 * *
1 *
ПУСК!
```

Подсказка: скорее всего, вам потребуется вложенный цикл.

Купить книгу на сайте kniga.biz.ua >>>