1.2 解压可迭代对象赋值给多个变量¶

问题¶

如果一个可迭代对象的元素个数超过变量个数时，会抛出一个 `ValueError` 。 那么怎样才能从这个可迭代对象中解压出 N 个元素出来？

解决方案¶

Python 的星号表达式可以用来解决这个问题。比如，你在学习一门课程，在学期末的时候， 你想统计下家庭作业的平均成绩，但是排除掉第一个和最后一个分数。如果只有四个分数，你可能就直接去简单的手动赋值， 但如果有 24 个呢？这时候星号表达式就派上用场了：

```
def drop_first_last(grades):
    first, *middle, last = grades
    return avg(middle)
```

另外一种情况，假设你现在有一些用户的记录列表，每条记录包含一个名字、邮件，接着就是不确定数量的电话号码。 你可以像下面这样分解这些记录：

```
>>> record = ('Dave', 'dave@example.com', '773-555-1212', '847-555-1212')
>>> name, email, *phone_numbers = record
>>> name
'Dave'
>>> email
'dave@example.com'
>>> phone_numbers
['773-555-1212', '847-555-1212']
>>>
```

值得注意的是上面解压出的 `phone_numbers` 变量永远都是列表类型，不管解压的电话号码数量是多少（包括 0 个）。所以，任何使用到 `phone_numbers` 变量的代码就不需要做多余的类型检查去确认它是否是列表类型了。

星号表达式也能用在列表的开始部分。比如，你有一个公司前 8 个月销售数据的序列， 但是你想看下最近一个月数据和前面 7 个月的平均值的对比。你可以这样做：

```
*trailing_qtrs, current_qtr = sales_record
trailing_avg = sum(trailing_qtrs) / len(trailing_qtrs)
return avg_comparison(trailing_avg, current_qtr)
```

下面是在 Python 解释器中执行的结果：

```
>>> *trailing, current = [10, 8, 7, 1, 9, 5, 10, 3]
>>> trailing
[10, 8, 7, 1, 9, 5, 10]
>>> current
3
```

讨论¶

扩展的迭代解压语法是专门为解压不确定个数或任意个数元素的可迭代对象而设计的。 通常，这些可迭代对象的元素结构有确定的规则（比如第 1 个元素后面都是电话号码）， 星号表达式让开发人员可以很容易的利用这些规则来解压出元素来。 而不是通过一些比较复杂的手段去获取这些关联的元素值。

值得注意的是，星号表达式在迭代元素为可变长元组的序列时是很有用的。 比如，下面是一个带有标签的元组序列：

```
records = [
    ('foo', 1, 2),
    ('bar', 'hello'),
    ('foo', 3, 4),
]
```

```
def do_foo(x, y):
    print('foo', x, y)

def do_bar(s):
    print('bar', s)

for tag, *args in records:
    if tag == 'foo':
        do_foo(*args)
    elif tag == 'bar':
        do_bar(*args)
```

星号解压语法在字符串操作的时候也会很有用，比如字符串的分割。

代码示例：

```
>>> line = 'nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false'
>>> uname, *fields, homedir, sh = line.split(':')
>>> uname
'nobody'
>>> homedir
'/var/empty'
>>> sh
'/usr/bin/false'
>>>
```

有时候，你想解压一些元素后丢弃它们，你不能简单就使用 `*`， 但是你可以使用一个普通的废弃名称，比如 `_` 或者 `ign` （ignore）。

代码示例：

```
>>> record = ('ACME', 50, 123.45, (12, 18, 2012))
>>> name, *_, (*_, year) = record
>>> name
'ACME'
>>> year
2012
>>>
```

在很多函数式语言中，星号解压语法跟列表处理有许多相似之处。比如，如果你有一个列表， 你可以很容易的将它分割成前后两部分：

```
>>> items = [1, 10, 7, 4, 5, 9]
>>> head, *tail = items
>>> head
1
>>> tail
[10, 7, 4, 5, 9]
>>>
```

如果你够聪明的话，还能用这种分割语法去巧妙的实现递归算法。比如：

```
>>> def sum(items):
...     head, *tail = items
...     return head + sum(tail) if tail else head
...
>>> sum(items)
36
>>>
```

然后，由于语言层面的限制，递归并不是 Python 擅长的。 因此，最后那个递归演示仅仅是个好奇的探索罢了，对这个不要太认真了。