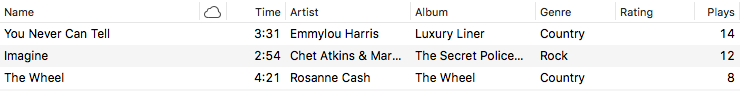
计算中许多有趣的数据都是表格式的。 ，就像表格一样。 首先，我们将看到它们的一些例子，然后再尝试确定它们的共同点。 以下是其中一些:

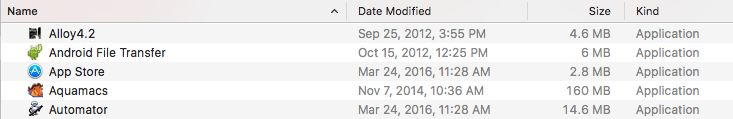
•  电子邮件收件箱是一个消息列表。 对于每条消息，您的收件箱都会存储一组信息:发件人、主题行、对话内容、正文，以及更多信息。

http://tranpic.nos.netease.com/3F20CE43FBB346EE8633F8BA71B7C95A-rId5

•  一个音乐播放列表。 对于每首歌，您的音乐播放器都会维护一组信息:它的名称、歌手、长度、类型等等。



•  文件系统文件夹或目录。 对于每个文件，您的文件系统记录一个名称、修改日期、大小和其他信息。



***现在做!***

你能举出更多的例子吗?

如何:

•  对派对邀请的回应。

•  成绩单发布。

•  一个日历日程。

你可以在你的生活中想出更多!

所有这些有什么共同之处? 表列数据的特点是:

•  它们由行和列组成。 例如，每首歌或电子邮件消息或文件都是一行。 它们的每一个特征——歌曲标题、信息主题、电影胶片——都是一个列。

•  每一行都具有与其他行相同的列，顺序相同。

•  给定的列具有相同的类型，但是不同的列可以具有不同的类型。 例如，电子邮件消息有一个发送者的名称，它是一个字符串; 主题行，它是一个字符串; 发送日期，即日期; 是否被读取，这是一个布尔值; 等等。

•  这些行通常有特定的顺序。 例如，电子邮件是按照最近发送的邮件排序的。

**锻炼**

在上面描述的其他示例和您描述的示例中查找表格数据的特征。

现在，我们将学习如何对表进行编程，并考虑分解涉及表的任务。 您还可以为表操作查找完整的Pyret文档。 (本段包含链接：[table operations](https://www.pyret.org/docs/latest/tables.html" \t "_blank)  )

**4.1创建表格数据**

Pyret提供了多种创建表格数据的简单方法。 最简单的方法是在程序中定义数据如下:

**表:姓名、年龄**

**行:“爱丽丝”,30岁**

**行:“Bob”, 40岁**

**行:“颂歌”,25岁**

**结束**

也就是说，一个表后面是按所需顺序排列的列的名称，然后是一系列行。 每一行必须包含与列声明相同数量的数据，且顺序相同。

**锻炼**

改变以上例子的不同部分。 ，从一行中删除一个必要的值，添加一个无关的值，删除一个逗号，添加一个额外的逗号，在一行的末尾留下一个额外的逗号——然后看看会得到什么错误。

注意，在一个表中，列的顺序很重要:如果两个表在其他方面是相同的，但是列的顺序不同，则认为它们是不相等的。

**检查:**

**表:姓名、年龄**

**行:“爱丽丝”,30岁**

**行:“Bob”, 40岁**

**行:“颂歌”,25岁**

**结束**

**不是**

**表:年龄、姓名**

**行:30,“爱丽丝”**

**行:40岁的“Bob”**

**行:25岁的“颂歌”**

**结束**

**结束**

注意，上面的例子使用is-not，即。 ，测试通过，表示表不相等。

当然，我们可以使用table编写文字表。 然而，Pyret还提供了其他获取表格数据的方法! 特别是，您可以从电子表格[FILL]导入表格数据，因此任何允许您创建这样一个表格的机制都可以使用。 你可能会:

•  自己创建表格，

•  与朋友合作制作一张表格，

•  在Web上查找可以导入到工作表中的数据，

•  创建一个让其他人填写的谷歌表单，并从他们的回答中获得一个表单

等等。 让你的想象力驰骋吧! 一旦数据位于Pyret中，语言就不关心它们来自何处。

**4.2处理行**

现在我们来学习如何处理表格。 Pyret提供了各种内置操作，可以很容易地在表上执行有趣的计算。 此外，正如我们稍后将看到的[REF]，如果我们发现这些还不够，我们可以自己编写。 现在，我们将主要关注Pyret提供的操作。

让我们来思考一些关于我们的数据的问题:

•  哪些邮件是由特定用户发送的?

•  哪首歌是某位艺术家唱的?

•  播放列表中播放频率最高的歌曲是什么?

•  在播放列表中，哪些歌曲播放频率最低?

我们看到，其中一些对应于保留一些行，另一些对应于排序。 Pyret提供了与这些相对应的表格操作。

**4.2.1保持准备**

我们保留表中的行如下:

**使用发件人筛选电子邮件:**

   发件人== 'Matthias Felleisen'

**结束**

表示要使用电子邮件表，并特别要使用发件人列。 这个操作处理表的每一行。 在每一行中，sender引用该行的sender列的值。 body (between: and end)中的表达式必须计算为布尔值; 如果为真，则Pyret将该行保存在结果表中，否则将丢弃该行。 运行这个查询的结果是一个具有相同列的新表，但是只有一些行(可能少到没有行); 剩下的行与原始表中的行顺序相同。

用同样的方法，我们可以根据艺术家保留行:

**滤网播放列表使用艺术家:**

   (美图== '深紫色')或(美图== 'Van Halen')

**结束**

这表明我们可以编写复杂的表达式来选择行。

**4.2.2订购**

我们可以对表中的行进行类似的排序，这将生成一个新表，其中的行按描述的顺序排列:

**顺序播放列表:**

   play-count提升

**结束**

用播放计数值按升序排列行，以便表中最早的行告诉我们哪些歌曲我们听得最少。

注意:和结束之间的内容不是表达式。 因此，我们不能在这里编写任意代码。 我们只能命名列并指出它们应该以何种方式排序。

**4.2.3保管与订货相结合**

当然，我们不限于执行这些操作中的一个。 由于它们都使用一个表并生成一个表，所以我们可以轻松地组合它们。 让我们先想想我们可能想用英语做什么:

•  在某个人的电子邮件中，哪封是最古老的?

•  在某一位艺术家的歌曲中，我们最不常演奏的是哪首?

***现在做!***

花点时间想想你会如何用你目前所看到的来写这些。

这是第一个例子:

mf-email =使用发件人筛选邮件:

   发件人== 'Matthias Felleisen'

**结束**

**mf-emails顺序:**

   发送日期升序

**结束**

注意，在order表达式中，我们订购的不是电子邮件，它是所有电子邮件的表，而是mf-email，只包含来自该发件人的电子邮件的表。 现在，看看结果中最早的几行，我们就能得到那个人最早的电子邮件。

**锻炼**

将第二个示例编写为对播放列表表进行keep和order操作的组合。

**4.2.4扩展**

有时，我们希望创建一个新列，其值基于现有列的值。 例如，我们的表可能反映员工记录，并具有名为小时工资和工时的列，表示相应的数量。 我们现在想要扩展这个表格，增加一个新的列来反映每个员工的工资总额:

**延长员工使用小时工资，工作时间:**

   总工资:小时工资\*工作时间

**结束**

这将创建一个新列total-wage，它在每行中的值是该行中两个指定列的乘积。 塔将把新列放在右边的末端; 我们很快就会看到，我们可以很容易地更改列的顺序(选择)。 (本段包含链接：[Selecting](https://papl.cs.brown.edu/2018/intro-tabular-data.html" \t "_blank)  )

当然，我们可以将扩展与其他表操作结合起来。 例如，我们可能已经注意到标题行较短的消息通常不包含高优先级任务。 因此，我们可以首先扩展电子邮件表的主题行长度:

ext-email =使用主题扩展电子邮件:

   subject-length: - length(主题)

**结束**

**ext-email顺序:**

   subject-length下行

**结束**

这将创建一个表，其中最长的主题行位于顶部，最短的主题行位于底部。

**4.2.5改造、清洗、正火**

有时候，一个表“几乎是正确的”，但是需要进行一些调整。 例如，我们可能有一个客户请求免费样本的表，并且希望将每个客户限制在最多一个特定的数量。 我们可能会从不同的国家得到不同格式的温度读数，并希望将它们全部转换成一种格式。 因为单位错误可能是危险的! 我们可能会有一个评分本不同的年级使用不同的精度水平，想要标准化他们所有人都有相同的精度水平。 (本段包含链接：[unit errors can be dangerous](http://mentalfloss.com/article/25845/quick-6-six-unit-conversion-disasters" \t "_blank)  )

在所有这些情况下，我们都希望得到的表具有与原始表相同的“形状”—相同的列、相同的行、相同的顺序—但是有些列值稍微进行了转换。 Pyret提供了转换来实现这一点。 例如，我们是这样限制客户订单的:

**使用count转换订单:**

   数:num-min(统计,3)

**结束**

下面是我们对总分的四舍五入:

**转换级电子书使用总等级:**

   总成绩:num-round(总成绩)

**结束**

当然，转换可以包含被转换列以外的列:

**使用temp转换天气，单位:**

   温度:

**如果单位== "F":**

       fahrenheit-to-celsius(临时)

**其他:**

       临时

**结束**

   单位:

**如果单位== "F":**

       “C”

**其他:**

       单位

**结束**

**结束**

这改变了桌子，所以所有的温度都转换成摄氏度。

***现在做!***

在这个例子中，为什么还要转换单位呢?

因为我们应该保持温度和单位同步。 如果我们转换温度而不是单位，这个表的后面的用户可能会假设单位列是准确的，并且意外地将转换后的温度当作仍然是华氏温度。

**4.2.6选择**

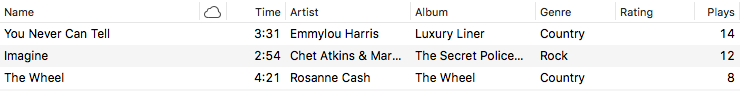
最后，出于表示的目的，有时只看到一些列是有用的，特别是在包含许多列的表中; 更改列的顺序也会很有帮助，这样可以使原本要一起查看的项相邻。 假设我们的成绩本有很多列代表所有的中间成绩，最后是总分; 当我们完成分数分配时，我们希望看到每个学生的名字和他们的最终分数:

Many interesting data in computing are *tabular*—i.e., like a table—in form. First we’ll see a few examples of them, before we try to identify what they have in common. Here are some of them:

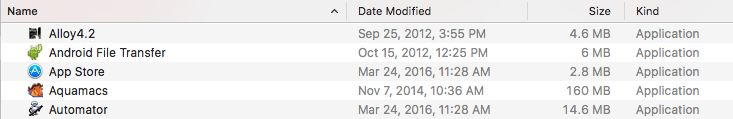
•  An email inbox is a list of messages. For each message, your inbox stores a bunch of information: its sender, the subject line, the conversation it’s part of, the body, and quite a bit more.

http://tranpic.nos.netease.com/3F20CE43FBB346EE8633F8BA71B7C95A-rId5

•  A music playlist. For each song, your music player maintains a bunch of information: its name, the singer, its length, its genre, and so on.



•  A filesystem folder or directory. For each file, your filesystem records a name, a modification date, size, and other information.



***Do Now!***

Can you come up with more examples?

How about:

•  Responses to a party invitation.

•  A gradebook.

•  A calendar agenda.

You can think of many more in your life!

What do all these have in common? The characteristics of tabular data are:

•  They consists of rows and columns. For instance, each song or email message or file is a row. Each of their characteristics—the song title, the message subject, the filename—is a column.

•  Each row has the same columns as the other rows, in the same order.

•  A given column has the same type, but different columns can have different types. For instance, an email message has a sender’s name, which is a string; a subject line, which is a string; a sent date, which is a date; whether it’s been read, which is a Boolean; and so on.

•  The rows are usually in some particular order. For instance, the emails are ordered by which was most recently sent.

**Exercise**

Find the characteristics of tabular data in the other examples described above, as well as in the ones you described.

We will now learn how to program with tables and to think about decomposing tasks involving them.You can also look up the full Pyret documentation for table operations.

**4.1 Creating Tabular Data**

Pyret provides multiple easy ways of creating tabular data. The simplest is to define the datum in a program as follows:

**table**: name, age

**row**: "Alice", 30

**row**: "Bob", 40

**row**: "Carol", 25

**end**

That is, a table is followed by the names of the columns in their desired order, followed by a sequence of rows. Each row must contain as many data as the column declares, and in the same order.

**Exercise**

Change different parts of the above example—e.g., remove a necessary value from a row, add an extraneous one, remove a comma, add an extra comma, leave an extra comma at the end of a row—and see what errors you get.

Note that in a table, the order of columns matters: two tables that are otherwise identical but with different column orders are not considered equal.

**check**:

**table**: name, age

**row**: "Alice", 30

**row**: "Bob", 40

**row**: "Carol", 25

**end**

**is-not**

**table**: age, name

**row**: 30, "Alice"

**row**: 40, "Bob"

**row**: 25, "Carol"

**end**

**end**

Observe that the example above uses **is-not**, i.e., the test passes, meaning that the tables are *not* equal.

Of course, we can write *literal* tables using table. However, Pyret provides other ways to get tabular data, too! In particular, you can import tabular data from a spreadsheet [FILL], so any mechanism that lets you create such a sheet can also be used. You might:

•  create the sheet on your own,

•  create a sheet collaboratively with friends,

•  find data on the Web that you can import into a sheet,

•  create a Google Form that you get others to fill out, and obtain a sheet out of their responses

and so on. Let your imagination run wild! Once the data are in Pyret, the language doesn’t care where they came from.

**4.2 Processing Rows**

Let’s now learn how we can actually process a table. Pyret offers a variety of built-in operations that make it quite easy to perform interesting computations over tables. In addition, as we will see later [REF], if we don’t find these sufficient, we can write our own. For now, we’ll focus on the operations Pyret provides.

Let’s think about some of the questions we might want to ask about our data:

•  Which emails were sent by a particular user?

•  Which songs were sung by a particular artist?

•  Which are the most frequently played songs in a playlist?

•  Which are the least frequently played songs in a playlist?

We see that some of these correspond to *keeping* some rows and some correspond to *ordering* them. Pyret provides tabular operations corresponding to these.

**4.2.1 Keeping**

We keep rows from a table as follows:

**sieve** email **using** sender:

sender == 'Matthias Felleisen'

**end**

says to use the email table, and specifically to employ the sender column. This operation processes every row of the table. In each row, sender refers to the value of the sender column of that row. The expression in the body (between : and **end**) must evaluate to a Boolean; if it is true, then Pyret keeps that row in the resulting table, otherwise it is discarded. The outcome of running this *query* is a fresh table with the same columns but only some (perhaps as few as none) of the rows; those rows that remain will be in the same order as in the original table.

In the same way, we can keep rows based on the artist:

**sieve** playlist **using** artist:

(artist == 'Deep Purple') **or** (artist == 'Van Halen')

**end**

This shows that we can write complex expressions to select rows.

**4.2.2 Ordering**

We can similarly order the rows of a table, which produces a new table that has the rows in the described order:

**order** playlist:

play-count **ascending**

**end**

orders the rows with the play-count values in ascending order, so that the earliest rows in the table tell us which songs we’ve listened to least frequently.

Note that what goes between the : and **end** is *not* an expression. Therefore, we cannot write arbitrary code here. We can only name columns and indicate which way they should be ordered.

**4.2.3 Combining Keeping and Ordering**

Naturally, we are not limited to performing only one of these operations. Since each of them consumes a table and produces one, we can easily combine them. Let’s first think of what we might want to do in English:

•  Of the emails from a particular person, which is the oldest?

•  Of the songs by a particular artist, which have we played the least often?

***Do Now!***

Take a moment to think about how you would write these with what you have seen so far.

Here is the first example:

mf-emails = **sieve** email **using** sender:

sender == 'Matthias Felleisen'

**end**

**order** mf-emails:

sent-date **ascending**

**end**

Note that in the **order** expression, we order not email, which is the table of all emails, but only mf-email, the table of just the emails from that one sender. Now, looking at the earliest rows in the result gives us the earliest emails from that one person.

**Exercise**

Write the second example as a composition of keep and **order** operations on a playlist table.

**4.2.4 Extending**

Sometimes, we want to create a new column whose value is based on those of existing columns. For instance, our table might reflect employee records, and have columns named hourly-wage and hours-worked, representing the corresponding quantities. We would now like to extend this table with a new column to reflect each employee’s total wage:

**extend** employees **using** hourly-wage, hours-worked:

total-wage: hourly-wage \* hours-worked

**end**

This creates a new column, total-wage, whose value in each row is the product of the two named columns in that row. Pyret will put the new column at the right end; as we will soon see, we can easily change the order of columns (Selecting).

Naturally, we can combine extension with other table operations. For instance, we might have noticed that messages with short subject lines usually don’t contain high-priority tasks. Therefore, we might first extend the email table with the length of the subject line:

ext-email = **extend** email **using** subject:

subject-length: string-length(subject)

**end**

**order** ext-email:

subject-length **descending**

**end**

This will create a table where the longest subject lines are at the top and the shortest subject lines are at the bottom.

**4.2.5 Transforming, Cleansing, and Normalizing**

There are times when a table is “almost right”, but requires a little adjusting. For instance, we might have a table of customer requests for a free sample, and want to limit each customer to at most a certain number. We might get temperature readings from different countries in different formats, and want to convert them all to one single format.Because unit errors can be dangerous! We might have a gradebook where different graders have used different levels of precision, and want to standardize all of them to have the same level of precision.

In all these cases, we want the resulting table to have the same “shape” as the original—the same columns, the same rows, in the same order—but with some of the column values transformed slightly. Pyret provides **transform** to do this. For instance, here is how we limit customer orders:

**transform** orders **using** count:

count: num-min(count, 3)

**end**

Here’s how we round the total grades:

**transform** gradebook **using** total-grade:

total-grade: num-round(total-grade)

**end**

Of course, a transformation can involve columns other than the one being transformed:

**transform** weather **using** temp, unit:

temp:

**if** unit == "F":

fahrenheit-to-celsius(temp)

**else**:

temp

**end**

unit:

**if** unit == "F":

"C"

**else**:

unit

**end**

**end**

This alters the table so that all temperatures are converted to celsius.

***Do Now!***

In this example, why do we also transform unit?

It’s because we should keep the temperature and unit in sync. If we transform the temperature but not the unit, a later user of this table might assume the unit column is accurate, and accidentally treat the converted temperature as if it were still in Fahrenheit.

**4.2.6 Selecting**

Finally, for presentation purposes, it is sometimes useful to see just a few of the columns, especially in tables with many of them; it can also be helpful to change the order of columns so that items that are meant to be viewed together are made adjacent. Suppose our gradebook has numerous columns representing all the intermediate scores, at the end of which is the total; when we’re done assigning grades, we want to see each student’s name with just their final score:

文档中第 1 至 1000 字的翻译内容正在加载...

确定