有道翻译 支持免费文档翻译240万字符/年 立即下载→ https://fanyi.youdao.com/download

作业 1

2025 年第一学期 COMP9021 课程

1 一般事项

1.1 目标

该作业的目的是:

- 培养你的解决问题的能力;
- 请您仔细阅读规格说明并严格遵循;
- 让您以小型 Python 程序的形式设计并实现解决问题的方案;
- 让您练习算术运算、测试、循环、基本的 Python 数据类型以及 Unicode 字符的使用;
- 能够控制打印语句。

1.2 提交

您的程序将分别存储在名为 solitaire_1.py 和 solitaire_2.py 的文件中。在开发并测试好程序后，请使用 Ed 进行上传（除非您是在 Ed 中直接编写程序）。作业可以多次提交，以最后一次提交的版本为准。作业截止日期为 3 月 31 日上午 10 点。

1.3 评估

该作业满分为 13 分。它将通过多个输入进行测试。对于每次测试，自动评分脚本会让您的程序运行 30 秒。

作业可在截止日期后最多 5 天内提交。每逾期一天（即超过 24 小时但不超过 48 小时），可获得的最高分减少 5%，最多减少 5 天。因此，如果学生 $_{A}$ 和 $_{B}$ 提交的作业满分分别为 12 分和 11 分，且都逾期两天提交，那么可获得的最高分是 11.7 分，所以 A 获得 min(11.7, 12) = 11.7 分，B 获得 min(11.7, 11) = 11 分。

您的程序输出应与所指示的**完全一致**。

1.4 关于抄袭政策的提醒

您被允许，甚至被鼓励与其他同学讨论解决作业的方法。但讨论必须围绕算法展开，而非代码。不过，您必须独立完成代码实现。提交的作业会定期接受检查，以发现抄袭、修改他人代码或多人密切合作完成同一份作业的情况。一旦发现，将予以严厉处罚。

2 纸牌，洗牌

2.1 演示文稿

第一个练习模拟一种用 32 张牌玩的单人纸牌游戏，即每种花色的 A、7、8、9、10、J、Q 和 K，遵循以下规则。

数字 0 到 7 分别代表红桃牌，从红桃 A 到红桃 K。

数字 8 至 15 代表红桃牌，从红桃 A 到红桃 K。

16 至 23 这几个数字代表黑桃牌，从黑桃 A 到黑桃 K。

数字 24 至 31 代表黑桃牌，从黑桃 A 到黑桃 K。

例如，6 代表红桃皇后，26 代表黑桃 8 。

第二个练习模拟一种用 52 张牌玩的单人纸牌游戏，规则如下。

数字 0 到 12 分别代表红桃牌，从红桃 A 到红桃 K。

13 至 25 这些数字代表红桃牌，从红桃 A 到红桃 K。

· 26 至 38 这些数字代表黑桃牌，从黑桃 A 到黑桃 K。

数字 39 至 51 代表黑桃牌，从黑桃 A 到黑桃 K。

例如，16 代表红桃 4，36 代表梅花 J。

2.2 洗牌

这两种练习都需要洗牌，要么洗整副牌（32 张或 52 张），要么洗整副牌中的部分牌。为此，遵循以下约定。

通过洗牌一副牌，我们指的是通过将相应数字集合随机化来实现，具体做法是将这些数字**按升序**排列后作为参数传递给 random 模块的 shuffle（） 函数。例如，

· 要将一副 52 张的牌全部洗乱，我们可以这样做

```
>>> cards = list(range(52))
>>> shuffle(cards)
```

· 要将除红桃 4 和梅花 J 之外的 52 张牌洗牌，我们可以这样做

```
>>> cards = sorted(set(range(52)) - {16, 36})
>>> shuffle(cards)
```

为了确保结果可预测，在调用 `shuffle()` 函数之前，应使用给定的参数调用 `random` 模块的 `seed()` 函数。通过使用 *678* 作为*种子*值对包含*全部（52 张）牌的牌组*进行*洗牌*，我们指的是执行与以下操作等效的操作：

```
>>> cards = list(range(52))
>>> seed(678)
>>> shuffle(cards)
```

顺便说一下，这使得卡片代表

```
[11, 12, 22, 38, 15, 16, 14, 28, 4, 34, 46, 48, 33,
   18, 5, 17, 27, 37, 50, 51, 31, 41, 9, 1, 39, 3,
       29, 40, 43, 23, 25, 13, 19, 35, 26, 42, 24, 32, 44,
           45, 6, 36, 8, 47, 2, 30, 10, 49, 21, 0, 20, 7]
```

3 首款单人纸牌游戏

3.1 游戏描述

该游戏使用 32 张牌进行。目标是弃掉足够多的牌，最终手中剩下按顺序排列的 4 张 A，可能还有 A 之前的牌或之后的牌。牌的弃除分 3 个阶段进行，第一阶段所有牌在 4 堆中分布，第二阶段在 3 堆中分布，第三阶段也是最后一个阶段在 2 堆中分布。牌只在游戏开始前洗一次。

在第一阶段开始时，牌堆中的牌都是面朝下的，牌堆最底下的牌在最下面，最上面的牌在最上面，然后从牌堆顶部到底部分成 4 堆，所以牌堆顶部的第一张、第二张、第三张和第四张牌分别成为第一堆（最左边）、第二堆、第三堆和第四堆（最右边）的最下面的牌，第五张、第六张、第七张和第八张牌分别成为第一堆、第二堆、第三堆和第四堆最下面牌的上面一层，以此类推。将第一堆牌翻转过来，使其牌面朝上。将堆顶的所有牌丢弃，直到露出一张 A，除非第一堆中没有 A，这种情况下整堆牌都被丢弃。如果露出了一张 A，那么将剩余的牌翻转过来，然后放在一边，此时 A 面朝下放在桌上。对第二堆、第三堆和第四堆也进行同样的操作，每次将剩余的牌（如果有）翻转过来，然后放在之前保留的牌的上面（如果有）。

在第二阶段，遵循同样的步骤，只是将之前放一边的牌分到 3 堆而不是 4 堆。如果剩余牌数不是 3 的倍数，则第一堆比第三堆多一张牌；如果剩余牌数除以 3 余 2，则第二堆比第三堆多一张牌。请注意，最后分发的牌（牌面朝下）是第一阶段中首次露出的 A。

对于第三阶段也是这样操作，只是之前被搁置的牌要分到两个牌堆里。

在第三阶段结束时，将之前最后放一边的牌从上到下依次取出，从左到右正面朝上展示。四张 A 必然都在其中，但只有当它们依次出现，且其间没有其他牌时，才算获胜。当然，如果只剩下 4 张牌，那么在翻开之前就已知游戏获胜。

3.2 玩一局游戏（3.5 分）

您的程序将被存储在一个名为 solitaire_1.py 的文件中。执行

$ python3 solitaire_1.py

在 Unix 提示符下运行该命令应产生以下输出（以一个空格结尾）：

请输入一个整数作为 seed（） 函数的参数:

此时程序正在等待您的输入，该输入应为一个整数，您可以假定输入的一定是整数。您的程序会在调用 shuffle（） 函数之前，按照第 2 节所述将该整数传递给 seed（） 函数，以对 32 张牌的牌组进行洗牌。

以下是一个可能的关于游戏失败的互动场景。

以下是一个可能的游戏获胜互动场景。

输出以一个空行开始，接着是一行文字，内容为：

```
Deck shuffled, ready to start!
```

下一行输出代表 32 张牌的牌组，所有牌都朝下（32 个“]”）。其后是一个空行。

第一轮开始的标志是一行文字，上面写着：

```
Distributing the cards in the deck into 4 stacks.
```

接下来两轮的比赛安排由以下这条信息公布：

```
Distributing the cards that have been kept into _ stacks.
```

其中 _ 在第二轮为 3，在第三轮为 2。该行之后是 a 开头的 5 行：

- 剩余栈的表示，相邻两个栈的起始位置相隔 12 个字符；
- 一个空行（其用途将在下文进一步说明）；
- 所有已弃牌的牌面朝上排列的表示形式，除最后一张（最上面那张）外，其余每张牌都用 [表示，而第一阶段该行为空；
- 一个空行（其用途将在下文进一步说明）；
- 一个空行。

然后，对于每个阶段，输出由每组 12 或 13 行组成，每组的结构如下。

该组中的第一行内容如下所示：

-翻开后，牌堆中没有 A 。

用 _ 填写 first（第一）、second（第二）、third（第三）或 fourth（第四）中的一个，或者

在 _ **堆**牌中翻转的最后一张牌是一张 A 。

和……在一起

“First”中的第一个“一”

“第二”（在“第一、第二、第三和第四”中的“第二”）

*（并且只有当该卡片确实是牌堆中的最后一张时才添加）

该组中的第二行描绘了正在处理的那一叠牌剩余的部分，即这叠牌被倒置后，所有上面没有 A 的牌都被依次丢弃后剩下的部分；所以要么这叠牌已经空了，要么剩下的牌顶上有一张 A。

该组中的第三行描绘了从正在处理的牌堆中依次丢弃的所有牌，如果没有找到 A，则顶部为牌堆中的最后一张牌；如果找到了 A，则顶部为 A 上方的那张牌。

· 该组中的第四行展示的是截至当时已弃掉的牌，牌面朝上。

该组中的第五行描绘的是截至当时被搁置一旁的牌，牌面朝下。

该组中的第六行是空的。

· 该组中的第七行内容如下所示：

- 将堆叠中的所有牌都丢弃到已丢弃的牌中。或者
- 将 A 之前的一张牌也丢弃 | 把 A 之前的那张牌也加入到已丢弃的牌中。或者
- 将在 A 之前丢弃的牌中再丢弃 _ 张牌，其中 _ 为至少等于 2 的整数。

如果尚未弃牌，则使用弃牌操作；否则，使用将牌添加到已弃牌中这一操作。

如果发现了一张 A，那么下一行会是以下内容之一：

-保留|同时保留这张 A，将其翻开。
或者
-保留|同时保留这张 A 和后面的那张牌，然后把它们翻过来。或者
-保留|同时保留 A 和 _ 张牌，随后将其翻开。其中 _ 为至少等于 2 的整数。

如果尚未留出任何东西，就用“Keeping”；否则，就用“Also keeping”。

· 接下来的那行显示了有待处理的堆栈（如果有）。

· 下一行是空行。

· 接下来的一行显示了已被弃掉的牌（可能未变）。

· 接下来的一行显示的是已被搁置（可能未作改动）的卡片。

该组的最后一行是空的。

输出以一组 6 行结束：

该组中的第一行内容为：

显示已保留的 _ 张牌，其中 _ 为整数（必然至少等
于 4）。

该组中的第二行显示：你输了！或者你赢了！

· 该组接下来的两行是空白的。

该组中的倒数第二行描绘了游戏过程中已弃掉的牌。

该组的最后一行展示了游戏结束时被搁置的所有牌，这些牌正面朝上并排摆放。

请注意，输出中**任何地方都没有制表符，且任何一行都没有尾随空格**。

3.3 玩多种游戏并估算概率（3 分）

执行

$ python3

在 Unix 提示符下，然后

>>> 从 solitaire_1 导入 simulate

在 Python 提示符下，您应该能够调用 simulate（） 函数，该函数接受两个参数。

第一个参数，比如说 $_n$，应当是一个严格意义上的正整数，并且您可以假定它是一个严格意义上的正整数，代表要进行的游戏场数。

第二个参数，比如说 $_i$，应当是一个整数，并且您可以假定它是一个整数。

该函数模拟游戏进行 $_n$ 次。

· 首次使用给定的 i 值作为种子对所有牌的牌组进行洗牌时，

· 如果 n≥2，则第二次洗牌时将所有牌的序列号 i + 1 作为 seed（） 函数的参数。

· …

第 n 次也是最后一次，将所有牌洗牌，用 i + n - 1 作为 seed（） 的种子值。

以下是一种可能的互动。

概率以浮点数形式计算，并保留小数点后两位。仅输出严格正的概率以及获胜时剩余的牌数（包括概率小于 0.005% 的情况，此时输出为 0.00%）。输出结果按照获胜时剩余牌数的递增顺序排列。

在分隔的竖线左右各有一个空格，所有行均由恰好 45 个字符组成。

4 第二款单人纸牌游戏

4.1 游戏描述

游戏使用 52 张牌。将 7 点牌从牌堆中取出，正面朝上放在桌上，从左至右依次为红桃 7、梅花 7、黑桃 7 和红心 7，确保桌上有足够的空间在 7 点牌上方放置从 8 点到 K 点的牌，在 7 点牌下方放置从 6 点到 A 点的牌，且同一花色的牌最终都在同一列。最多允许 3 个阶段来完成所有牌的摆放。每个阶段，剩余的所有牌都正面朝下叠放，然后从牌堆顶部一张一张地取牌，直到牌堆取空。从牌堆顶部取出的牌，如果其上方或下方的同花色牌已经放置好，则将其放置在相应位置；如果无法放置，则正面朝上放在所有已放置牌的上方。如果可以放置该牌，我们接着检查已放一边的牌堆（如果有）顶部的牌是否能为同花色的牌列进行扩展，如果可以，就将其放置在应放置的位置，然后再次检查已放一边的牌堆（如果有）顶部的牌，直到已放一边的牌堆中没有牌，或者虽有牌但顶部的牌无法为同花色的牌列进行扩展为止，此时，从剩余待处理的牌堆顶部取下一张牌（如果有）。当牌堆为空时，要么所有牌都已正确放置在桌面上，游戏获胜，要么至少还有一个阶段未完成，此时将已放一边的牌堆翻转过来，作为待处理的牌堆，继续按照前一阶段的方式进行操作。如果游戏进行了 3 个阶段，且在第 3 个阶段结束时仍有牌未正确放置在桌面上，则游戏失败。48 张牌（整副牌去掉所有 7 点牌）仅在游戏开始前洗牌一次。

4.2 玩一局游戏（3.5 分）

您的程序将被存储在一个名为 solitaire_2.py 的文件中。执行

$ python3 solitaire_2.py

在 Unix 提示符下运行该命令应产生以下输出（以一个空格结尾）

```
Please enter an integer to feed the seed() function:
```

程序现在正在等待您的输入，该输入应为一个整数，您可以假定输入的一定是整数。您的程序会在调用 shuffle（） 函数之前，按照第 2 节所述将该整数传递给 seed（） 函数，以对 52 张牌（减去四张 7）进行洗牌。

输出以一个空行开始，然后是

```
There are _ lines of output; what do you want me to do?

Enter: q to quit
       a last line number (between 1 and _)
       a first line number (between -1 and -_)
       a range of line numbers (of the form m--n with 1 <= m <= n <= _)
```

其中所有下划线“_”均表示相同的数字。程序应等待在下一行输入，该行应与“q”以及上述最左边的三个字符对齐。在输入“q”之前，程序应输出一个

空行，如果输入正确，则执行所要求的操作，输出空行并再次提示用户。当输入“q”时程序退出。输入正确是指完全符合要求，包括整数在规定的范围内（注意正数前面不应有“+”号），但输入的开头、结尾、输入范围时第一个“-”之前以及输入范围时第二个“-”之后可以有任意数量的空格（负数的减号和数字之间不能有空格……）。

第一种输入会让程序输出收集到的输出结果中的前 n 行，其中 n 为作为输入提供的数字。

第二种输入会让程序输出收集到的输出结果中的最后 n 行，其中 -n 为作为输入提供的数字，且

· 第三种输入将使程序输出所收集输出中位于 m^{th} 和 n^{th} 行之间的部分，包括 m^{th} 和 n^{th} 行，其中 m 和 n 是作为输入提供的数字。

以下是一种可能的互动。

以下是一个可能的交互示例，展示了游戏失败时收集到的完整输出。

以下是一个可能的交互示例，展示了赢得游戏时收集到的完整输出。

当输入正确时，收集到的输出的第一行显示

```
All 7s removed and placed, rest of deck shuffled, ready to start!
```

第二行收集的输出代表 48 张牌，所有牌都朝下（48 个“]”）。其后是一个空行（稍后会解释其用途），接着是 6 个空行，分别用于放置 K、Q、J、10、9 和 8。然后是一行表示 7 的牌，其后是 6 个空行，分别用于放置 6、5、4、3、2 和 A，最后再跟一个空行。

收集到的其余输出内容由以下格式的行组成

开始第 _ 轮...

以“_”开头，若存在第二阶段则为第二行，若存在第三阶段则为第三行，其后空一行。接下来是一系列结构如下所示的行。

第一行是这样写的

无法从剩余牌堆的顶部放置牌 ☒☒ ☹

或者

无法从放在一边的牌堆顶部取牌 ☒☒ ☹

或者

从剩下的牌堆顶部放一张牌 ☺

或者

从放在一边的那叠卡片顶部抽出一张卡片 ☺

在所有情况中，除了“无法将放在一旁的牌堆顶部的牌放置到位 ☒☒”之外，接下来会有一行显示剩余待处理的牌堆（牌面朝下），然后是一行显示已从牌堆中取出但无法放置的牌（牌面朝上）。

如果能从剩余的牌堆或放一边的牌堆中放置一张牌，接下来的 13 行将从国王到 A 显示已放置的牌。

最后是一行空行。

当使用“从剩余牌堆顶部放置一张牌 ”时，待处理的牌堆减少一张。当使用“从搁置牌堆顶部放置一张牌 ”时，无法放置的牌堆减少一张。在这两种情况下，可放置的牌都会显示在预期位置。在第一种情况下，这张牌是作为刚添加到已放置牌堆中的牌而被“发现”的；而在第二种情况下，这张牌是已知的，因为它朝上放置，所以我们知道在已放置牌堆中刚添加的牌就是它。

当从可处理的牌堆顶部取牌时，若该牌无法放置，则使用“无法从牌堆顶部放置牌”表情符号。当刚放置了一张牌，而无法放置的牌堆不为空且其顶部的牌仍无法放置时，则使用“无法从放置一旁的牌堆顶部放置牌”表情符号。在第一种情况下，该牌被“发现”，并成为无法放置牌堆顶部的新牌。

一旦所有卡片都已放置完毕，会在收集到的输出结果中添加最后一行，其内容为：

```
You placed all cards, you won 👍
```

如果存在第三阶段且结束时仍有部分卡片未放置到位，则会在收集到的输出结果中添加最后一行，其内容为：

```
You could not place _ cards, you lost 👎
```

加上未能放置的卡片数量。

每张放置的牌都会显示相应的 Unicode 字符，其前会**依次有 1 个、**2 个、**3 个或** 4 个**制表符**，具体取决于该牌在行**中的位置**是第一张、第二张、第三张还是第四张，这由牌的花色决定。当然，行中每张牌前还会显示该行中之前**所有**牌的 Unicode 字符。**每行**末尾都**没有**多余的**空格**。

4.3 玩多种游戏并估算概率（3 分）

执行

$ python3

在 Unix 提示符下，然后

```
>>> from solitaire_2 import simulate
```

在 Python 提示符下，您应该能够调用 simulate（） 函数，该函数接受两个参数。

第一个参数，比如说 n，应当是一个严格意义上的正整数，并且您可以假定它是一个严格意义上的正整数，代表要进行的游戏场数。

第二个参数，比如说 i，应当是一个整数，并且您可以假定它是一个整数。

该函数模拟游戏进行 n 次。

- 首次使用给定的 i 值作为种子对所有牌的牌组进行洗牌时，
- 如果 $_{n}$ ⩾$_{2}$，则第二次对所有牌进行洗牌，将 $_{i+1}$ 作为 seed（） 的参数，
- …
- 上次和这次，通过将所有牌的牌堆与 $_{i+n-1}$ 传递给 seed（） 函数来重新洗牌。

以下是一种可能的互动。

概率以浮点数形式计算，并保留小数点后两位。仅输出严格正的概率值以及剩余牌数（包括概率小于 0.005% 的情况，此时输出为 0.00%）。输出结果按照剩余牌数从多到少排序。

在分隔的竖线左右各有一个空格，所有行均由恰好 32 个字符组成。