纳什均衡简介

纳什均衡，又称为非合作博弈均衡，是博弈论的一个重要术语，以约翰·纳什命名。在一个博弈过程中，无论对方的策略选择如何，当事人一方都会选择某个确定的策略，则该策略被称作支配性策略。如果两个博弈的当事人的策略组合分别构成各自的支配性策略，那么这个组合就被定义为纳什均衡。

一个策略组合被称为纳什均衡，当每个博弈者的均衡策略都是为了达到自己期望收益的最大值，与此同时，其他博弈者也遵循这样的策略。

纳什均衡的得来

关于纳什均衡的普遍意义和存在性定理的证明等奠定非合作博弈理论发展基础的重要成果，是约翰·纳什在普林斯顿大学攻读博士学位时完成的。实际上，博弈论的研究起始于1944年冯·诺依曼（Von Neumann）和奥斯卡·摩根斯坦（Oskar Morgenstern）的《博弈论和经济行为》。然而却是纳什首先用严密的数学语言和简明的文字准确地定义了纳什均衡这个概念，并在包含“混合策略（mixed strategies）”的情况下，证明了纳什均衡在n人有限博弈中的普遍存在性，从而开创了与诺依曼和摩根斯坦框架路线完全不同的“非合作博弈（Non-cooperative Game）”理论，进而对“合作博弈（Cooperative Game）”和“非合作博弈”做了明分和定义。阿尔伯特·塔克（Albert tucker）教授评价其论文，“这是对博弈理论发展的极重要的贡献，[illegible]意义的n人有限非合作博弈的概念和性质。并且它很可能开拓出许多在两人零和问题以外的，至今尚未涉及的问题。在概念和方法两方面，该论文都是作者的独立创造。

Nash equilibrium 纳什均衡 完全信息静态博弈

Eric Martin, CSE, UNSW

COMP9021 Principles of Programming, trimester 1, 2019

```
In [2]: from itertools import product
        import matplotlib.pyplot as plt
        from matplotlib.patches import Rectangle, Circle
        import re
```

纯策略和帕累托最优性

1 2×2 games, pure strategies, and Pareto optimality

Consider two players, Ruth and Charlie, who both can take one of two possible decisions. Each of the four pairs (d_R, d_C) of possible decisions taken by both players is associated with a pair (p_R, p_C) of expected *payoffs* (the indexes R and C refer to Ruth and Charlie, respectively).

For a first example, consider the *prisoner's dilemma*. Ruth and Charlie have been arrested and charged with robbery. It is believed that they were actually carrying guns, making them liable for the more severe charge of armed robbery. Ruth and Charlie, who are held in separate cells and cannot communicate, are each offered the following deal, knowing that the other is offered the same deal:

第一个例子，考虑囚犯的困境。露丝和查理被捕并被指控犯有抢劫罪。 据信它们实际上是携带枪支，使它们对更严重的人负责武装抢劫指控。露丝和查理，他们被关押在不同的单元格中，无法沟通，每个人都提供以下协议，知道另一个是同样的交易：

- if they testify that their partner was armed and their partner does not testify against them, then their sentence will be suspended while their partner will spend 15 years in jail;
- if they testify against each other, then they will both spend 10 years in jail;
- if neither testifies against the other, then they will both spend 5 years in jail.

如果他们证明他们的伴侣是武装的并且他们的伴侣没有对他们作证，那么他们的判决将被暂停，而他们的伴侣将在监狱中度过15年；

	Not testify	Testify
Testify	(-15, 0)	(-10, -10)
Not testify	(-5, -5)	(0, -15)

Charlie

Ruth

Both Ruth and Charlie deciding to testify is a *Nash equilibrium*:

- Charlie testifying, Ruth finds herself better off by testifying (-10) than by not testifying (-15);
- Ruth testifying, Charlie finds himself better off by testifying (-10) than by not testifying (-15).

Note that Ruth and Charlie would in fact be better off if they did not testify (they would spend 5 years rather than 10 years in jail). But that is not what Game theory recommends, and that is not what is observed in practice either... A way to express that Ruth and Charlie would be better off if they did not testify is to say that this pair of decisions is not *Pareto optimal*: there exists another pair of decisions (namely, both of them not testifying) associated with an outcome which is at least as good for both players, and better for at least one of them (in this case, better for both of them).

For a second example, consider the *game of chicken*. Ruth and Charlie drive towards each other at high speed following a white line drawn on the middle of the road. If both swerve (chicken out), then it is a

露丝和查理在道路中间画出一条白线后高速驶向对方。

再举一个例子，考虑鸡的游戏。 露丝和查理在道路中间画出一条白线后高速驶向对方。 如果两个都转向（鸡出），那么它是平局（0到两个）。 如果一个转弯而另一个不转弯，那么没有出鸡的人会赢（1）而另一个输掉（-1）。 如果两只鸡都没出来，那么它们将没有机会再次玩，并且它们的共同收益可以合理地设置为-∞，但是我们将它任意设置为-10

转弯

draw (0 to both). If one swerves and the other does not, then the one who did not chicken out wins (1) while the other one loses (-1). If neither chickens out, then they won't have a chance to play again, and their common payoff could be reasonably set to $-\infty$, but we set it arbitrarily to -10.

	Not swerve	Swerve
Swerve	(1, -1)	(0, 0)
Not swerve	(-10, -10)	(-1, 1)

Charlie

Ruth

露丝没有出去玩，查理出去玩的是纳什均衡
查理突然转过身来，露丝发现自己因没有转向（1）而不是转向（0）而变得更好；
露丝没有突然转向，查理通过转向（-1）而不是不转弯（-10）发现自己变得更好
通过对称性，露丝转弯，而查理没有转弯，也是纳什均衡

Ruth not chickening out and Charlie chickening out is a Nash equilibrium:

- Charlie swerving, Ruth finds herself better off by not swerving (1) than by swerving (0);
- Ruth not swerving, Charlie finds himself better off by swerving (-1) than by not swerving (-10).

By symmetry, Ruth chickening out and Charlie not chickening out is also a Nash equilibrium.

2 Mixed strategies 混合策略

Testifying or not testifying, swerving or not swerving, are *pure strategies*. More generally, Ruth can testify or swerve with probability p, and Charlie can testify or swerve with probability q. Ruth opts for a pure strategy iff p is 0 or 1, and similarly Charlie opts for a pure strategy iff q is 0 or 1; otherwise, they opt for a *mixed strategy*.

转向

Let us consider the game of chicken with both Ruth and Charlie swerving with probability 0.9. Then Ruth's expectation is $0.1(0.1 \times -10 + 0.9 \times 1) + 0.9(0.1 \times -1 + 0.9 \times 0) = -0.1$; by symmetry, Charlie's expectation is also -0.1. It turns out that this strategy is also a Nash equilibrium, as we now show.

期望

If Ruth swerves with probability p and Charlie swerves with probability q, then Ruth's expectation is equal to

$$(1-p)[(1-q) \times -10 + q \times 1] + p[(1-q) \times -1 + q \times 0]$$

which simplifies to

$$(-10q + 9)p + 11q - 10$$

Ruth's aim is to *maximise* her expectation, that is, maximise the value of the above expression, which is achieved by:

- setting p to 0 if $q > 0.9$;
- setting p to 1 if $q < 0.9$;
- taking for p an arbitrary value if $q = 0.9$.

对称的　　任意值

By symmetry, Charlie maximises his expectation by:

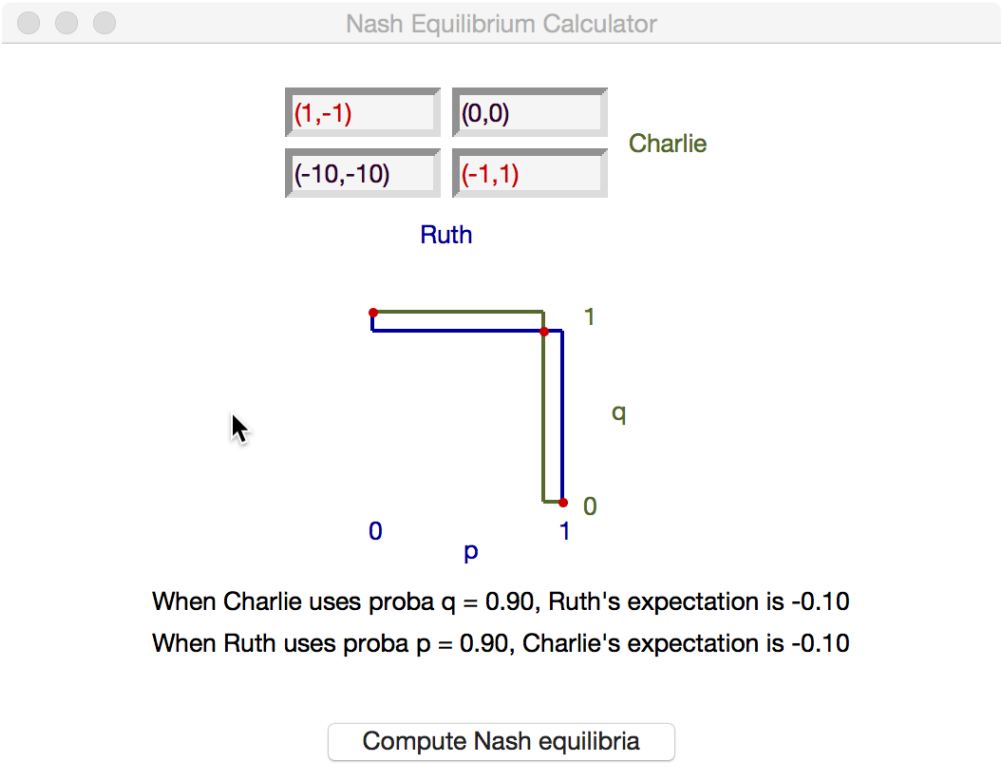
- setting q to 0 if $p > 0.9$;
- setting q to 1 if $p < 0.9$;
- taking for q an arbitrary value if $p = 0.9$.

这意味着两位球员都接受了对手的战略（他们只能自己决定），露丝和查理都不会后悔他们在三种情况下的策略，三种纳什均衡：

This means that both players accepting the opponent's strategy as it is (they can only decide for themselves), Ruth and Charlie will both not regret their strategy in three cases, the three Nash equilibria:

- $p = 0$ and $q = 1$
- $p = 1$ and $q = 0$

- $p = 0.9$ and $q = 0.9$



3 No regret graphs

Use the following notation for Ruth's and Charlie's payoffs:

符号

(a_2, b_2)	(a_4, b_4)
(a_1, b_1)	(a_3, b_3)

Charlie

Ruth

Then Ruth's expectation is

$$(1-p)[(1-q)a\_1 + qa\_2] + p[(1-q)a\_3 + qa\_4]$$

which can be written as

$$(a\_1 - a\_2 - a\_3 + a\_4)pq + (a\_3 - a\_1)p + (a\_2 - a\_1)q + a\_1$$

whereas Charlie's expectation is

$$(1-p)[(1-q)b\_1 + qb\_2] + p[(1-q)b\_3 + qb\_4]$$

which can be written as

$$(b\_1 - b\_2 - b\_3 + b\_4)pq + (b\_3 - b\_1)p + (b\_2 - b\_1)q + b\_1$$

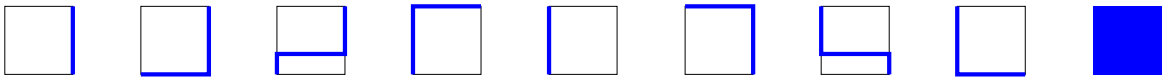
Set:

- $D_R = a_1 - a_2 - a_3 + a_4$ and $D_C = b_1 - b_2 - b_3 + b_4$;
- $E_R = a_3 - a_1$ and $E_C = b_2 - b_1$;
- $F_R = a_2 - a_1$ and $F_C = b_3 - b_1$.

So Ruth's expectation is $(D_R q + E_R)p + F_R q + a_1$ and Charlie's expectation is $(D_C p + E_C)q + F_C p + b_1$.
Both players aim at maximising their expectation, which determines: 两位球员都旨在最大化他们的期望，这决定了：

- Ruth's No regret graph, consisting of all pairs of numbers of the form: 露丝没有遗憾的图表，包括所有形式的数字对
  - $(0, q)$ with $D_R q + E_R < 0$,
  - (p, q) with $0 \leq p \leq 1$ and $D_R q + E_C = 0$,
  - $(1, q)$ with $D_R q + E_R > 0$;
- Charlie's No regret graph, consisting of all pairs of numbers of the form:
  - $(p, 0)$ with $D_C p + E_C < 0$,
  - (p, q) with $0 \leq q \leq 1$ and $D_C p + E_C = 0$,
  - $(q, 1)$ with $D_C q + E_C > 0$.

The possible No regret graphs for Ruth are:



The possible No regret graphs for Charlie are:

Any possible No regret graph for Ruth intersects any No regret graph for Charlie, which shows the existence of a Nash equilibrium; the Nash equilibria are all intersection points.

Note from the graphs that if Ruth achieves a Nash equilibrium using a pure strategy, then Charles can also also use a pure strategy; similarly, if Charlie achieves a Nash equilibrium using a pure strategy then Ruth can also also use a pure strategy.

从方程中注意到当Ruth使用混合策略达到纳什均衡时，则DR q + ER = 0并且她的期望不依赖于她自己选择一个决策的概率
替代；类似地，当查理使用混合策略达到纳什均衡时，则DC p + EC = 0并且他的期望不依赖于他自己选择一个决策而不是替代方案的概率

Note from the equations that when Ruth achieves a Nash equilibrium using a mixed strategy, then $D_R q + E_R = 0$ and her expectation does not depend on her own probability of choosing one decision over the alternative; similarly, when Charlie achieves a Nash equilibrium using a mixed strategy, then $D_C p + E_C = 0$ and his expectation does not depend on his own probability of choosing one decision over the alternative.

Write D, E and F to refer to either D_R, E_R and F_R, or to D_C, E_C and F_C. What determines the actual No regret graph depends on: 什么决定了实际的无遗憾图取决于

- whether D is equal to 0 or not.
  - If D is equal to 0, whether E is strictly negative, equal to 0, or strictly positive.
  - If D is not equal to 0, whether $-\frac{E}{D}$ is strictly negative, or equal to 0, or strictly betweeen 0 and 1, or equal to 1, or strictly greater than 1.
    - If $-\frac{E}{D}$ is strictly negative or strictly greater than 1, whether E is positive or negative.
    - If $-\frac{E}{D}$ is between 0 and 1, whether D is positive or negative.

The cases where D is equal to 0 and E is either strictly negative or strictly positive determine the same No regret graphs as the cases where D is not equal to 0 and $-\frac{E}{D}$ is not between 0 and 1 and E is either negative or positive, respectively.

4 Implementation 执行

为了系统地检查所有案例，让我们以4元组的形式定义适当的支付例子，意在表示：

To systematically examine all cases, let us define appropriate payoffs examples in the form of 4-tuples, meant to denote:

他们代表露丝的收益

- (a_1, a_2, a_3, a_4) if they represent Ruth's payoffs;
- (b_1, b_2, b_3, b_4) if they represent Charlie's payoffs.

```
In [3]: payoffs_examples = (1, 2, 2, 4), (0, -1, 0, 0), (3, 1, 1, 4),\
                           (1, 1, 0, 1), (4, 2, 2, 1), (-1, -1, 0, -1),\
                           (2, 3, 4, -3), (0, 1, 0, 0), (0, 0, 0, 0),\
                           (1, 2, 2, 3), (2, 1, 1, 0)
```

We will design and implement functions where the payoffs for one player are not meant to be represented as one 4-tuple, but as two 2-tuples:

- $((a_1, a_2), (a_3, a_4))$ for Ruth
- $((b_1, b_2), (b_3, b_4))$ for Charlie

由于a1对应于b1，a2到b3，a3到b2和a4到b4，我们从payoffs_examples创建以下成对对列表，便让payoffs_examples的每个成员扮演对称角色 露丝和查理，并简化测试

Since a_1 corresponds to b_1, a_2 to b_3, a_3 to b_2, and a_4 to b_4, we create from `payoffs_examples` the following lists of pairs of pairs, so as to let each member of `payoffs_examples` play a symmetric role for Ruth and Charlie, and simplify testing:

```
In [4]: ruth_payoffs_examples = [((a1, a2), (a3, a4))
                                 for (a1, a2, a3, a4) in payoffs_examples
                                ]
        charlie_payoffs_examples = [((a1, a3), (a2, a4))
                                    for (a1, a2, a3, a4) in payoffs_examples
```

使用这两个列表，让我们证明payoffs_examples的成员说明了之前组]织的所有可能的无遗憾图案例

Using those two lists, let us demonstrate that the members of `payoffs_examples` illustrate all possible cases of No regret graphs as previously organised:

```
In [5]: def case_analysis(payoffs_per_player):
            print()
            D = dict.fromkeys('Ruth', 'Charlie')
            E = dict.fromkeys('Ruth', 'Charlie')
            for player in payoffs_per_player:
                payoffs = payoffs_per_player[player]
                D[player] = payoffs[0][0] - payoffs[0][1] -\
                            payoffs[1][0] + payoffs[1][1]
                 E[player] = payoffs[int(player == 'Ruth')]\
                                    [int(player == 'Charlie')] - payoffs[0][0]
                if D[player]:
                    cut = -E[player] / D[player]
                    if cut < 0 or cut > 1:
                        if E[player] > 0:
                            graph = '1(a)'
                        else:
```

```
                        graph = '5(a)'
                    if cut == 0:
                        if D[player] > 0:
                            graph = 2
                        else:
                            graph = 8
                    elif cut == 1:
                        if D[player] > 0:
                            graph = 4
                        else:
                            graph = 6
                    elif 0 < cut < 1:
                        if D[player] > 0:
                            graph = 3
                        else:
                            graph = 7
                else:
                    cut = 'undef'
                    if E[player] > 0:
                        graph = '1(b)'
                    elif E[player] < 0:
                        graph = '5(b)'
                    else:
                        graph = 9
                print(f'{player:9}{D[player]:3}{E[player]:5}{cut:>8}   {graph}')
```

```
In [6]: print('Player     D    E    -E/D   No regret graph')
        for payoffs_per_player in ({'Ruth': ruth_payoffs,
                                    'Charlie': charlie_payoffs
                                   } for (ruth_payoffs, charlie_payoffs) in
                                           zip(ruth_payoffs_examples,
                                               charlie_payoffs_examples
                                              )
                                   ):
            case_analysis(payoffs_per_player)
```

```
Player     D    E    -E/D   No regret graph

Ruth       1    1    -1.0   1(a)
Charlie    1    1    -1.0   1(a)

Ruth       1    0     0.0   2
Charlie    1    0     0.0   2

Ruth       5   -2     0.4   3
Charlie    5   -2     0.4   3

Ruth       1   -1     1.0   4
```

Charlie	1	−1	1.0	4
Ruth	1	−2	2.0	5(a)
Charlie	1	−2	2.0	5(a)
Ruth	−1	1	1.0	6
Charlie	−1	1	1.0	6
Ruth	−8	2	0.25	7
Charlie	−8	2	0.25	7
Ruth	−1	0	−0.0	8
Charlie	−1	0	−0.0	8
Ruth	0	0	undef	9
Charlie	0	0	undef	9
Ruth	0	1	undef	1(b)
Charlie	0	1	undef	1(b)
Ruth	0	−1	undef	5(b)
Charlie	0	−1	undef	5(b)

现在让我们对所有九个非遗憾图案例进行全面分析（使用payoffs_examples而没有最后
两个成员，所以只选择一个单独的代表来处理No的第一和第五案例
遗憾图。 继续写入D，E和F以指代DR，ER和FR，或指向DC，EC和FC，并且还写入v以分
别参考q或p（注意：不是p或q）。 下面的函数analyze（）定义了一个集合和三个字
典：

Let us now conduct the full analysis for all nine cases of Non regret graphs (using `payoffs_examples` without its last two members, so just selecting a single representative for each of Cases 1 and 5 of the No regret graphs). Keep writing D, E and F to refer to either D_R, E_R and F_R, or to D_C, E_C and F_C, and also write v to refer to q or p (note: not p or q), respectively. The function below, `analyse()`, defines one set and three dictionaries:

set `all_good`：它被初始化为空集，并且如果玩家的No Regret图属于案例9，则将'Ruth'或'Charlie'作为成员
字典段：将播放器设置为“ Ruth ”或“ Charlie ”，将[player]段初始化为[None，None]

- the set `all_good`: it is initialised to the empty set, and will have `'Ruth'` or `'Charlie'` as member if the player's No regret graph falls under Case 9.
- the dictionary `segments`: with `player` set to either `'Ruth'` or `'Charlie'`, `segments[player]` is initialised to `[None, None]`.
  - `segments[player][0]` will be changed to a tuple of the form (a, b), $0 \leq a < b \leq 1$, in case $Dv + E \leq 0$ holds and either $D \neq 0$ or $E \neq 0$, so if the No regret graph for `player` does not fall under Case 9 and has a "low" boundary line segment (vertically on the left for Ruth, horizontally at the bottom for Charlie).
  - `segments[player][1]` will be changed to a tuple of the form (a, b), $0 \leq a < b \leq 1$, in case $Dv + E \geq 0$ holds and either $D \neq 0$ or $E \neq 0$, so if the No regret graph for `player` does not fall under Case 9 and has a "high" boundary line segment (vertically on the right for Ruth, horizontally at the top for Charlie).
- the dictionary `probas`: with `player` set to either `'Ruth'` or `'Charlie'`, `probas[player]` is initialised to `None`. If there is a unique $v \in [0, 1]$ such that $Dv + E = 0$, then it will be changed to v's value.
- the dictionary `expectations`: with `player` set to either `'Ruth'` or `'Charlie'`, `probas[player]` is initialised to `None`. If there is a unique $v \in (0, 1)$ such that $Dv + E = 0$, then it will be changed

segment [player] [0]将被改为形式为（a，b）的元组，0≤a<b≤1，如果Dv +E≤0成立并且D 6 = 0或E 6 = 0，因此，如果玩家的无后悔图不属于案例9并且具有“低”边界线段（左侧垂直为露丝，水平位于底部为查理）

segment [player] [1]将被改为形式为（a，b）的元组，0≤a<b≤1，如果Dv +E≥0成立并且D 6 = 0或E 6 = 0，因此，如果玩家的无遗憾图不属于案例9并且具有“高”边界线段（右侧垂直为露丝，水平位于顶部为查理）

字典问题：当玩家设置为“ 露丝 ”或“ 查理 ”时，普罗卡斯[玩家]被归为无。 如果存在唯一的v [0,1]，使得Dv + E = 0，则它将被更改为v值。
• 词典期望：当玩家设置为'Ruth'或'Charlie'时，probas [player]初始化为None。 如果有一个唯一的v (0,1) 使得Dv + E = 0，那么它将被改变家的期望，以防其他玩家选择不以概率v作证/转向/ tes。
这是唯一一种期望可能不是收益之一的情况，它对应于纳什均衡，即当后悔线时两条内线线段的交叉点。
两名球员属于案例3和7中的一个

to `player`'s expectation in case the other player choses not to testify/swerve/... with probability v. This is the only case where the expectation might not be one of the payoffs, and it corresponds to the Nash equilibrium that is the intersection the two inner line segments when the Regret lines of both players fall under one of Cases 3 and 7.

Cases 4 and 8 of the Non regret graphs have a “low” boundary line segment, while Cases 2 and 6 have a “high” boundary line segment. As `case_analysis()` shows, under the assumptions that $D \neq 0$ and $-\frac{E}{D} \in \{0, 1\}$:

- Case 4 or Case 8 holds iff either $-\frac{E}{D} = 0$ and $D < 0$, or $-\frac{E}{D} = 1$ and $D > 0$;
- Case 2 or Case 6 holds iff either $-\frac{E}{D} = 0$ and $D > 0$, or $-\frac{E}{D} = 1$ and $D < 0$.

Boolean xor, `^`, that returns `True` iff one operand evaluates to `True` and the other to `False`, offers a good way to capture the previous distinction:

```
In [7]: print('cut    D    low (0) or')
        print('            right (1)')
        for cut, D in product((0, 1), (-1, 1)):
            print(f'{cut:3}{D:5}    {int((cut == 1) ^ (D > 0))}')

cut    D    low (0) or
            right (1)
  0   -1    0
  0    1    1
  1   -1    1
  1    1    0
```

To conduct the full analysis and generate all relevant information for all cases of No regret graphs, `analyse()` is essentially a “fleshed out” version of `case_analysis()`:

```
In [8]: def analyse(payoffs_per_player):
            all_good = set()
            segments = {player: [None, None] for player in payoffs_per_player}
            probas = dict.fromkeys(payoffs_per_player)
            expectations = dict.fromkeys(payoffs_per_player)
            D = dict.fromkeys(payoffs_per_player)
            E = dict.fromkeys(payoffs_per_player)
            F = dict.fromkeys(payoffs_per_player)
            for player in payoffs_per_player:
                payoffs = payoffs_per_player[player]
                D[player] = payoffs[0][0] - payoffs[0][1] -\
                            payoffs[1][0] + payoffs[1][1]
                E[player] = payoffs[int(player == 'Ruth')]\
                                   [int(player == 'Charlie')] - payoffs[0][0]
                if D[player]:
                    cut = -E[player] / D[player]
                    if cut < 0 or cut > 1:
                        segments[player][1 - int(E[player] < 0)] = 0, 1
```

```
                    else:
                        probas[player] = cut
                             segments[player][int((cut == 1) ^
                                                  (D[player] > 0)
                                                 )
                                             ] = 0, 1
                        else:
                            F[player] = payoffs[int(player == 'Charlie')]\
                                               [int(player == 'Ruth')] -\
                                                   payoffs[0][0]
                            expectations[player] = F[player] * probas[player] +\
                                                    payoffs[0][0]
                            segments[player][int(D[player] < 0)] = 0, probas[player]
                            segments[player][int(D[player] > 0)] = probas[player], 1
                    elif E[player]:
                        segments[player][1 - int(E[player] < 0)] = 0, 1
                    else:
                        all_good.add(player)
                return all_good, segments, probas, expectations
```

```
In [9]: print('Player   all_good   low segment   high segment   proba   expectation')
        for payoffs_per_player in ({'Ruth': ruth_payoffs, 'Charlie': charlie_payoffs}
                                        for (ruth_payoffs, charlie_payoffs) in
                                             zip(ruth_payoffs_examples[: -2],
                                                 charlie_payoffs_examples[: -2]
                                                )
                                   ):
            print()
            outcome = analyse(payoffs_per_player)
            for player in 'Ruth', 'Charlie':
                s_1, s_2, s_3, s_4, s_5 = (str(player in outcome[0]),
                                           str(outcome[1][player][0]),
                                           str(outcome[1][player][1]),
                                           str(outcome[2][player]),
                                           str(outcome[3][player])
                                          )
                print(f'{player:9}{s_1:11}{s_2:14}{s_3:15}{s_4:8}{s_5}')
```

```
Player   all_good   low segment   high segment   proba   expectation

Ruth     False      None          (0, 1)         None    None
Charlie  False      None          (0, 1)         None    None

Ruth     False      None          (0, 1)         0.0     None
Charlie  False      None          (0, 1)         0.0     None

Ruth     False      (0, 0.4)      (0.4, 1)       0.4     2.2
Charlie  False      (0, 0.4)      (0.4, 1)       0.4     2.2
```

```
Ruth     False      (0, 1)        None          1.0     None
Charlie  False      (0, 1)        None          1.0     None

Ruth     False      (0, 1)        None          None    None
Charlie  False      (0, 1)        None          None    None

Ruth     False      None          (0, 1)        1.0     None
Charlie  False      None          (0, 1)        1.0     None

Ruth     False      (0.25, 1)     (0, 0.25)     0.25    2.25
Charlie  False      (0.25, 1)     (0, 0.25)     0.25    2.25

Ruth     False      (0, 1)        None          -0.0    None
Charlie  False      (0, 1)        None          -0.0    None

Ruth     True       None          None          None    None
Charlie  True       None          None          None    None
```

For the the game of chicken, the analysis yields the following:

```
In [10]: analyse({'Ruth': ((-10, 1), (-1, 0)),
                  'Charlie': ((-10, -1), (1, 0))
                 }
                )

Out[10]: (set(),
          {'Ruth': [(0.9, 1), (0, 0.9)], 'Charlie': [(0.9, 1), (0, 0.9)]},
          {'Ruth': 0.9, 'Charlie': 0.9},
          {'Ruth': -0.09999999999999964, 'Charlie': -0.09999999999999964})
```

The following function determines which of the lower left corner ($i = 0$ and $j = 0$), top left corner ($i = 0$ and $j = 1$), lower right corner ($i = 1$ and $j = 0$), and upper right corner ($i = 1$ and $j = 1$) of the No regret graphs are (pure) Nash equilibria:

```
In [11]: def is_pure_equilibrium(i, j, all_good, segments, probas):
             return all(X[0] in all_good or\
                        segments[X[0]][X[1]] and\
                            segments[X[0]][X[1]][X[2]] == X[2] or\
                        probas[X[0]] == X[2]
                            for X in {('Ruth', i, j), ('Charlie', j, i)}
                       )
```

We test `is_pure_equilibrium()` on all 9 cases of corresponding pairs of No regret graphs:

```
In [12]: print('Lower left  Lower right Upper left  Upper right')
         for payoffs_per_player in ({'Ruth': ruth_payoffs,
                                     'Charlie': charlie_payoffs
```

```
                                    } for (ruth_payoffs, charlie_payoffs) in
                                            zip(ruth_payoffs_examples[: -2],
                                                charlie_payoffs_examples[: -2]
                                               )
                                   ):
            print()
            for (i, j) in product(range(2), repeat = 2):
                v = is_pure_equilibrium(i, j, *analyse(payoffs_per_player)[: 3])
                print(f'{str(v):12}', end = '')
```

```
Lower left  Lower right Upper left  Upper right

False       False       False       True
True        False       False       True
True        False       False       True
True        False       False       True
True        False       False       False
False       True        True        True
False       True        True        False
True        True        True        False
True        True        True        True
```

最后，我们为露丝和查理绘制了9种无遗憾图案的81种可能组合。 我们用红色圆圈表示纯纳什均衡，在无遗憾的交叉点画出红色圆圈内部线段，用红色绘制无后悔边界线段的部分，与其他玩家的无后悔图的线段重叠，而线段的所有其他部分则使用玩家的专用颜色绘制，并且对于第9个没有遗憾图的情况，我们用适当的颜色填充整个区域。 对于构成图形的9 x 9个单元中的任何一个的每个轴ax

Finally, we draw the 81 possible combinations of 9 cases of No regret graphs for Ruth and for Charlie. We indicate the pure Nash equilibria with red circles, also draw red circles at the intersection of No regret inner line segments, draw in red the parts of the No regret boundary line segments that overlap with a line segment of the No regret graph for the other player, while all other parts of the line segments are drawn using the players' dedicated colours, and for the 9th case of No regret graphs, we fill the whole area with the appropriate colour. For each axes `ax` for any of the 9 x 9 cells that make up the figure, we make use of:

删除框架，标签和刻度；

- `ax.axis(`off')` to remove frame, labels and ticks;
- `ax.add_patch()` to draw rectangles and circles, with

从matplotlib.patches导入，它将左下角的坐标对作为第一个参数，第二个参数作为矩形的宽度，第三个参数作为矩形的高度

绘制矩形和圆形

  - `Rectangle()`, imported from `matplotlib.patches`, that takes as first argument the pair of coordinates of the lower left corner, as as second argument the rectangle's width, and as third argument the rectangle's height.
  - `Circle()`, imported from `matplotlib.patches`, that takes as first argument the pair of coordinates of the circle's centre, and as second argument the circle's radius.

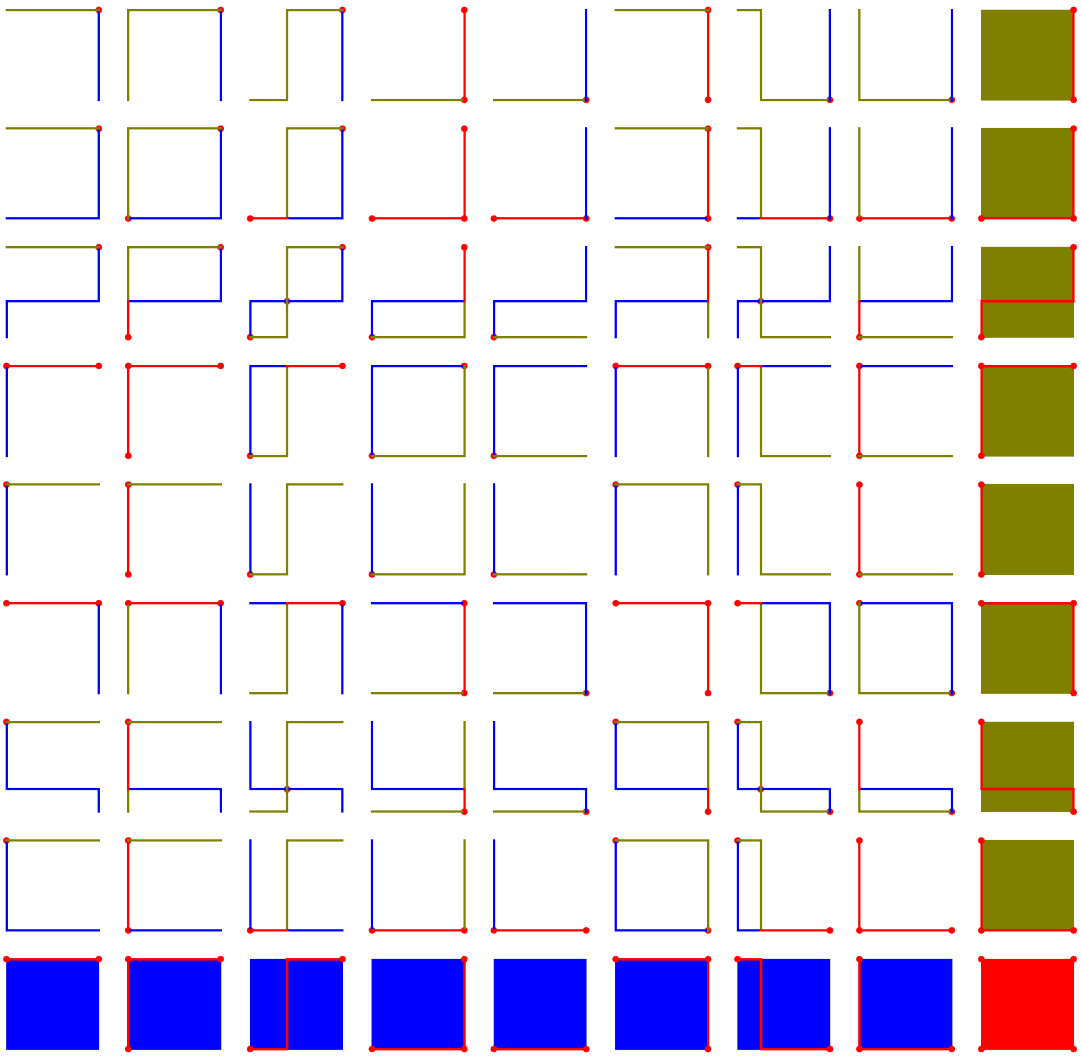
`Circle ()` ，从matplotlib.patches导入，它作为圆的中心坐标对的第一个参数，并作为圆的半径的第二个参数

```
In [13]: def draw_rectangle(colour, ax):
             ax.add_patch(Rectangle((0, 0), 1, 1, color = colour))

         def draw_line(pt_1, pt_2, colour):
             plt.plot(pt_1, pt_2, color = colour, linewidth = 2)

         def draw_rectangles(all_good, colours, ax):
             if 'Ruth' in all_good:
                 if 'Charlie' in all_good:
                     draw_rectangle(colours['Nash'], ax)
                 else:
```

```
            draw_rectangle(colours['Ruth'], ax)
    elif 'Charlie' in all_good:
        draw_rectangle(colours['Charlie'], ax)

def draw_outer_lines(all_good, segments, probas, colours):
    x_y = {'Ruth': lambda x, j: ((x, x), (j[0], j[1])),
           'Charlie': lambda y, i: ((i[0], i[1]), (y, y))
          }
    for i in range(2):
        for player_1, player_2 in ('Ruth', 'Charlie'), ('Charlie', 'Ruth'):
            if segments[player_1][i]:
                # In case the segment S under consideration intersects
                # an inner line segment S' (of length 1) for the other
                # player, then S' has been drawn already.
                # The intersection of S and S' is drawn again using
                # 'Nash' colour.
                colour = colours[player_1]\
                            if player_2 not in all_good and\
                              (probas[player_2] is None or\
                               probas[player_2] != i\
                              )\
                            else colours['Nash']
                draw_line(*x_y[player_1](i, segments[player_1][i]), colour)

def draw_inner_lines(all_good, probas, colours):
    x_y = {'Ruth': lambda p: ((0, 1), (p, p)),
           'Charlie': lambda p: ((p, p), (0, 1))
          }
    for player_1, player_2 in ('Ruth', 'Charlie'), ('Charlie', 'Ruth'):
        if probas[player_1] is not None:
            if player_2 in all_good:
                draw_line(*x_y[player_1](probas[player_1]),
                          colours['Nash']
                         )
            elif probas[player_1] is not None:
                draw_line(*x_y[player_1](probas[player_1]),
                          colours[player_1]
                         )

def draw_intersecting_nash_equilibria(probas, expectations, colours, ax):
    if expectations['Ruth'] is not None and\
       expectations['Charlie'] is not None:
        ax.add_patch(Circle((probas['Charlie'], probas['Ruth']), 0.03,
                            color = colours['Nash']
                           )
                     )

def draw_pure_nash_equilibria(all_good, segments, probas, colours, ax):
```

```
            for i, j in product(range(2), repeat = 2):
                if is_pure_equilibrium(i, j, all_good, segments, probas):
                    ax.add_patch(Circle((i, j), 0.03, color = colours['Nash']))
```

In [14]:
```
def draw_all_no_regret_graphs():
    colours = {'Ruth': 'blue', 'Charlie': 'olive', 'Nash': 'red'}
    plt.figure(figsize = (15, 15))

    i = 0
    for ruth_payoffs in ruth_payoffs_examples[: -2]:
        for charlie_payoffs in charlie_payoffs_examples[: -2]:
            i += 1
            ax = plt.subplot(9, 9, i)
            ax.axis('off');
            all_good, segments, probas, expectations =\
                analyse({'Ruth': ruth_payoffs, 'Charlie': charlie_payoffs})
            draw_rectangles(all_good, colours, ax)
            draw_inner_lines(all_good, probas, colours)
            draw_outer_lines(all_good, segments, probas, colours)
            draw_intersecting_nash_equilibria(probas, expectations,
                                              colours, ax
                                             )
            draw_pure_nash_equilibria(all_good, segments, probas,
                                      colours, ax
                                     )

draw_all_no_regret_graphs();
```



程序nash_equilibrium_calculator.py创建一个小部件，期望以这种形式输入4对支付，（a1，b1），（a2，b2），（a3，b3）和（a4，b4），j
数字都是整数，可能 括号和逗号两边的空格。 为了检查输入有效性并提取数据，我们利用正则表达式的语法并使用re模块及其search（）函
数。 此函数尝试并查找模式字符串与数据字符串的一部分之间的匹配，模式和字符串分别作为第一个和第二个参数提供。 当第一个参数可以
匹配第二个参数的许多部分时，search（）会查找最左边的部分。 字符与自身匹配，而点匹配任何字符

The program `nash_equilibrium_calculator.py` creates a widget that expects the 4 pairs of payoffs, (a_1, b_1), (a_2, b_2), (a_3, b_3) and (a_4, b_4) to be entered in this form, all numbers being integers, with possibly spaces on either side of the parentheses and commas. To check input validity and extract the data, we take advantage of the syntax of regular expressions and make use of the `re` module and its `search()` function. This function tries and find a match between a pattern string and part of a data string, the pattern and the string being provided as first and second arguments, respectively. When the first argument can match many parts of the second argument, `search()` looks for the longest leftmost part. A character matches itself, while a dot matches any character:

```
In [15]: re.search('cde', 'abcdef')
         re.search('...', 'abcdef')
         re.search('.c.', 'abcdef')

Out[15]: <re.Match object; span=(2, 5), match='cde'>

Out[15]: <re.Match object; span=(0, 3), match='abc'>
```

```
Out[15]: <re.Match object; span=(1, 4), match='bcd'>
```

\d matches any digit, while \D matches any nondigit:

```
In [16]: re.search('\d\d', '_ab123')
         re.search('\D\D', '_ab123')

Out[16]: <re.Match object; span=(3, 5), match='12'>

Out[16]: <re.Match object; span=(0, 2), match='_a'>
```

\s matches any space, while \S matches any nonspace character:

```
In [17]: re.search('\s\s\s', ' a1? \t\n!')
         re.search('\S\S\S', ' a1? \t\n!')

Out[17]: <re.Match object; span=(4, 7), match=' \t\n'>

Out[17]: <re.Match object; span=(1, 4), match='a1?'>
```

\w matches any character that can be part of a word (Python identifier), that is, a letter, the underscore or a digit, while \W matches any other character:

```
In [18]: re.search('\w\w\w', ' |+a_2| ')
         re.search('\W\W\W', ' |+a_2| ')

Out[18]: <re.Match object; span=(3, 6), match='a_2'>

Out[18]: <re.Match object; span=(0, 3), match=' |+'>
```

A question mark requests an optional occurrence of the pattern it applies to, which can be parenthesised to delimit its scope:

```
In [19]: re.search('\d\s?\d', 'a12b')
         re.search('\d\s?\d', 'a1 2b')
         re.search('\s(\d\d)?\s', 'a 12 b')
         re.search('\s(\d\d)?\s', 'a  b')

Out[19]: <re.Match object; span=(1, 3), match='12'>

Out[19]: <re.Match object; span=(1, 4), match='1 2'>

Out[19]: <re.Match object; span=(1, 5), match=' 12 '>

Out[19]: <re.Match object; span=(1, 3), match='  '>
```

A star requests an arbitrary (possibly empty) repetition of the pattern it applies to, which can be parenthesised to delimit its scope:

一个明星要求任意（可能是空的）重复它所适用的模式，这可以用来划定其范围

```
In [20]: # The empty initial sequence is the longest leftmost
         # substring of 'a123456b' consisting of digits
         re.search('\d*', 'a123456b')
         re.search('\d\d\d*', 'a123456b')
         re.search('\d(\d\d)*', 'a123456b')

Out[20]: <re.Match object; span=(0, 0), match=''>

Out[20]: <re.Match object; span=(1, 7), match='123456'>

Out[20]: <re.Match object; span=(1, 6), match='12345'>
```

A plus requests an arbitrary nonempty repetition of the pattern it applies to, which can be parenthesised to delimit its scope: 加号要求对其应用的模式进行任意非空重复，可以将其括号以划分其范围

```
In [21]: re.search('.\d+', 'a123456b')
         re.search('(.\d)+', 'a123456b')

Out[21]: <re.Match object; span=(0, 7), match='a123456'>

Out[21]: <re.Match object; span=(0, 6), match='a12345'>
```

Square brackets define character classes, surrounding the characters accepted for the match. In this context, any character with a special meaning (dot, question mark, star, plus, parenthesis...) loses its special meaning and is treated as a literal characters; to lose its special meaning outside a character class, such a character can be escaped:

方括号定义字符类，围绕匹配接受的字符。 在这种情况下，任何具有特殊含义的字符（点，问号，星号，加号，括号.....）都会失去其特殊含义，并被视为文字字符；在角色类之外失去其特殊含义，这样的角色可以逃脱

```
In [22]: re.search('[.]', 'abcabc.123456.defdef')
         re.search('[cba]+', 'abcabc.123456.defdef')
         re.search('.[\d]*', 'abcabc.123456.defdef')
         re.search('\.[\d]*', 'abcabc.123456.defdef')
         re.search('[.\d]+', 'abcabc.123456.defdef')

Out[22]: <re.Match object; span=(6, 7), match='.'>

Out[22]: <re.Match object; span=(0, 6), match='abcabc'>

Out[22]: <re.Match object; span=(0, 1), match='a'>

Out[22]: <re.Match object; span=(6, 13), match='.123456'>

Out[22]: <re.Match object; span=(6, 14), match='.123456.'>
```

Ranges of letters or digits can be provided within square brackets, letting a hyphen separate the first and last characters in the range. A hyphen placed after the opening square bracket or before the closing square bracket is interpreted as a literal character:

```
In [23]: re.search('[e-h]+', 'ahgfea')
         re.search('[B-D]+', 'ABCBDA')
         re.search('[4-7]+', '154465571')
         re.search('[-e-gb]+', 'a--bg--fbe--z')
         re.search('[73-5-]+', '14-34-576')
```

```
Out[23]: <re.Match object; span=(1, 5), match='hgfe'>

Out[23]: <re.Match object; span=(1, 5), match='BCBD'>

Out[23]: <re.Match object; span=(1, 8), match='5446557'>

Out[23]: <re.Match object; span=(1, 12), match='--bg--fbe--'>

Out[23]: <re.Match object; span=(1, 8), match='4-34-57'>
```

Within a square bracket, a caret after placed after the opening square bracket excludes the characters that follow within the brackets: 在方括号内，放置在开口方括号后面的插入符号排除了括号内的字符

```
In [24]: re.search('[^4-60]+', '0172853')
         re.search('[^-u-w]+', '-stv')

Out[24]: <re.Match object; span=(1, 5), match='1728'>

Out[24]: <re.Match object; span=(1, 3), match='st'>
```

A caret at the beginning of the pattern string matches the beginning of the data string; a dollar at the end of the pattern string matches the end of the data string: 模式字符串开头的插入符号匹配数据字符串的开头；模式字符串末尾的美元与数据字符串的末尾匹配

```
In [25]: re.search('\d*', 'abc')
         re.search('^\d*', 'abc')
         re.search('\d*$', 'abc')
         # re.search('^\d*$', 'abc') returns no match
         re.search('^\s*\d*\s*$', ' 345 ')

Out[25]: <re.Match object; span=(0, 0), match=''>

Out[25]: <re.Match object; span=(0, 0), match=''>

Out[25]: <re.Match object; span=(3, 3), match=''>

Out[25]: <re.Match object; span=(0, 5), match=' 345 '>
```

Escaping a dollar at the end of the pattern string, escaping a caret at the beginning of the pattern string or after the opening square bracket of a character class, makes dollar and caret lose the special meaning they have in those contexts context and let them be treated as literal characters:

在模式字符串的末尾转义一美元，在模式字符串的开头或字符类的开始方括号之后转义插入符号，使得美元和插入符失去特殊含义
他们在这些情境中具有上下文，并将其视为文字字符

```
In [26]: re.search('\$', '$*')
         re.search('\^', '*^')
         re.search('[\^]', '^*')
         re.search('[^^]', '^*')

Out[26]: <re.Match object; span=(0, 1), match='$'>

Out[26]: <re.Match object; span=(1, 2), match='^'>

Out[26]: <re.Match object; span=(0, 1), match='^'>
```

```
Out[26]: <re.Match object; span=(1, 2), match='*'>
```

方括号围绕替代字符，而竖条分隔替代模式：

Whereas square brackets surround alternative characters, a vertical bar separates alternative patterns:

```
In [27]: re.search('two|three|four', 'one three two')
         re.search('|two|three|four', 'one three two')
         re.search('[1-3]+|[4-6]+', '01234567')
         re.search('([1-3]|[4-6])+', '01234567')
         re.search('_\d+|[a-z]+_', '_abc_def_234_')
         re.search('_(\d+|[a-z]+)_', '_abc_def_234_')

Out[27]: <re.Match object; span=(4, 9), match='three'>

Out[27]: <re.Match object; span=(0, 0), match=''>

Out[27]: <re.Match object; span=(1, 4), match='123'>

Out[27]: <re.Match object; span=(1, 7), match='123456'>

Out[27]: <re.Match object; span=(1, 5), match='abc_'>

Out[27]: <re.Match object; span=(0, 5), match='_abc_'>
```

Parentheses allow matched parts to be saved. The object returned by `re.search()` has a `group()` method that without argument, returns the whole match and with arguments, returns partial matches; it also has a `groups()` method that returns all partial matches:

括号允许保存匹配的部分。 `re.search()` 返回的对象有一个`group()` 方法，该方法不带参数，返回整个匹配和带参数，返回部分匹配；它还有一个`groups()` 方法，返回所有部分匹配

```
In [28]: R = re.search('((\d+) ((\d+) \d+)) (\d+ (\d+))',
                       '  1 23 456 78 9 0 '
                      )

         R
         R.group()
         R.groups()
         [R.group(i) for i in range(len(R.groups()) + 1)]

Out[28]: <re.Match object; span=(2, 15), match='1 23 456 78 9'>

Out[28]: '1 23 456 78 9'

Out[28]: ('1 23 456', '1', '23 456', '23', '78 9', '9')

Out[28]: ['1 23 456 78 9', '1 23 456', '1', '23 456', '23', '78 9', '9']
```

成对的括号

Pairs of parentheses can therefore play two roles:

标记，星星或加号

- surround patterns to which question marks, stars or pluses can be applied to;
- delimit the patterns to capture and save.

To let a pair of parentheses play the first role only, let `?:` follow the opening parenthesis:

```
In [29]: # Separated by any sequence of characters, two strings of the form:
         # an optional + or -, followed by
         # - either 0
         # - or a nonzero digit followed with a (possibly empty)
         #   sequence of digits
         R = re.search('([+-]?(?:0|[1-9]\d*)).*([+-]?(?:0|[1-9]\d*))',
                       ' a = -3014, b = 0 '
                      )
         R

         R.groups()
```

```
Out[29]: <re.Match object; span=(5, 17), match='-3014, b = 0'>
```

```
Out[29]: ('-3014', '0')
```

The following function checks that its argument is a string:

0. that from the beginning: `^`
1. consists of possibly some spaces: `␣*` 可能包含一些空格：
2. followed by an opening parenthesis: `\(` 然后是一个左括号
3. possibly followed by spaces: `␣*` 可能后跟空格
4. possibly followed by either + or -: `[+-]?`
5. followed by either 0, or a nonzero digit followed by any sequence of digits: `0|[1-9]\d*`
6. possibly followed by spaces: `␣*`
7. followed by a comma: `,`
8. followed by characters matching the pattern described by 1-7
9. followed by a closing parenthesis: `\)`
10. possibly followed by some spaces: `␣*`
11. all the way to the end: `$`

成对的括号围绕两个数字匹配以捕获它们。 对于第5点，需要一对周围的括号；？：使其无法捕获

Pairs of parentheses surround both numbers to match to capture them. For point 5, a surrounding pair of parentheses is needed; **?:** makes it non-capturing:

```
In [30]: def validate_and_extract_payoffs(provided_input):
             pattern = '^ *\( *([+-]?(?:0|[1-9]\d*)) *,'\
                       ' *([+-]?(?:0|[1-9]\d*)) *\) *$'
             match = re.search(pattern, provided_input)
             if match:
                 return (match.groups())

         validate_and_extract_payoffs('(+0, -7 )')
         validate_and_extract_payoffs('  (-3014,0)  ')
```

```
Out[30]: ('+0', '-7')
```

```
Out[30]: ('-3014', '0')
```