牛艾科技

贪吃其实不是错，贪吃蛇的游戏制作

目录 Contents

目录 Contents

第一部分

游戏基本原理

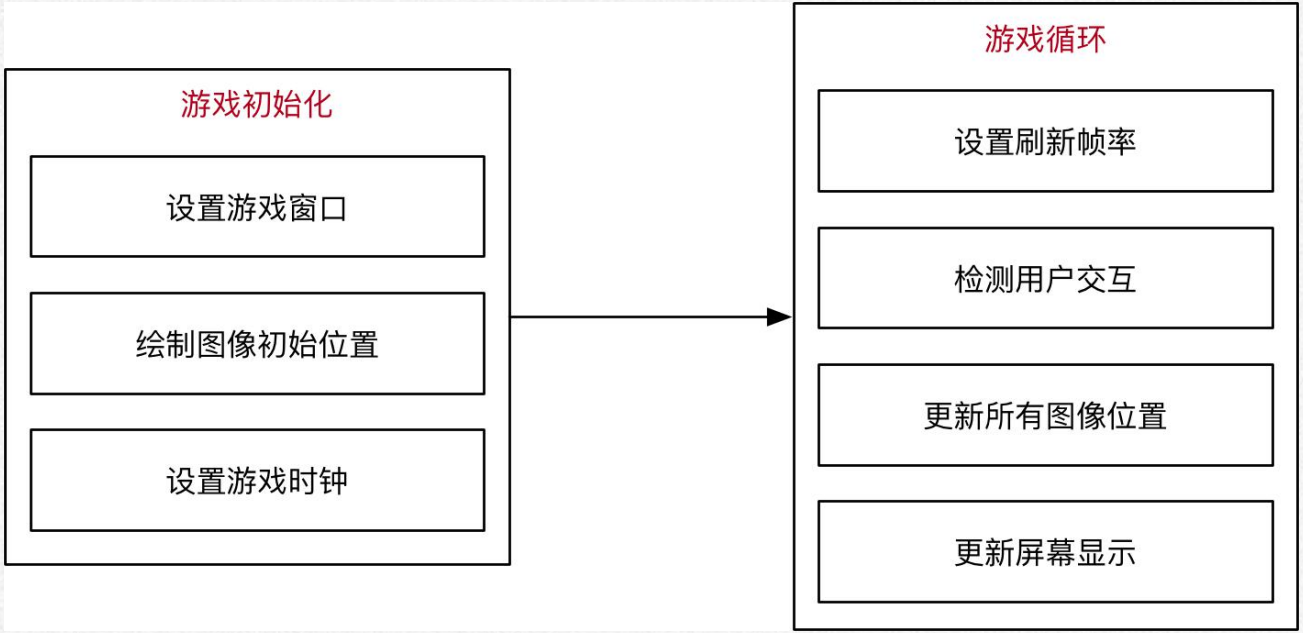
游戏基本原理

- 可交互的动画片

跟电影的原理类似，游戏中的动画效果，
本质上是快速的在屏幕上绘制图像
一般在电脑上 每秒绘制 60 次，就能够达到非常连续高品质的动画效果

游戏基本原理

- 程序基本流程



- 游戏循环的作用

1. 保证游戏不会直接退出
2. 变化图像位置 — 动画效果
3. 每隔一段时间移动或更新一下所有图像的位置
4. 检测用户交互 — 按键、鼠标等

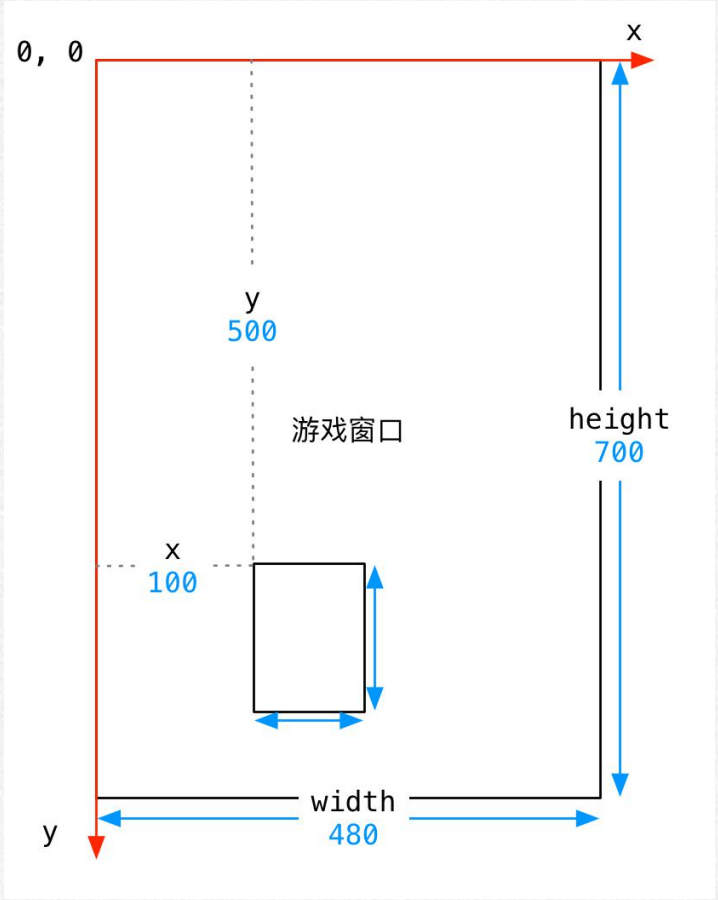
游戏基本原理

- 游戏中的坐标系描述

1. 原点在左上角 (0, 0)
2. x 轴水平方向向右，逐渐增加
3. y 轴垂直方向向下，逐渐增加

- 在游戏中，所有可见的元素都是以矩形区域来描述位置的

- 要描述一个矩形区域有四个要素：(x, y) (width, height)



游戏基本原理

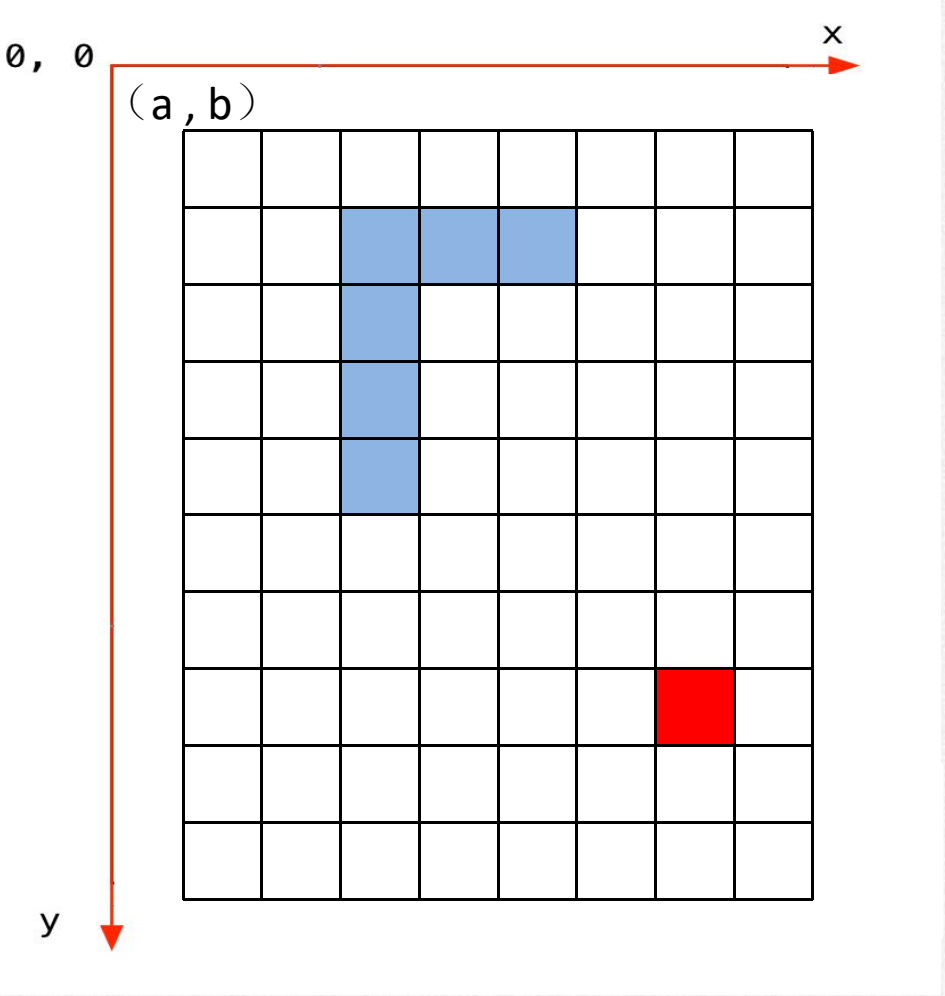
- 贪吃蛇的网格坐标

1. 将屏幕分成若干10*10的网格
2. 对指定网格填充颜色形成蛇的身体
3. 对指定网格填充颜色形成食物

- 网格左上角坐标和屏幕坐标的变换

x = 网格横坐标 * 8 + a

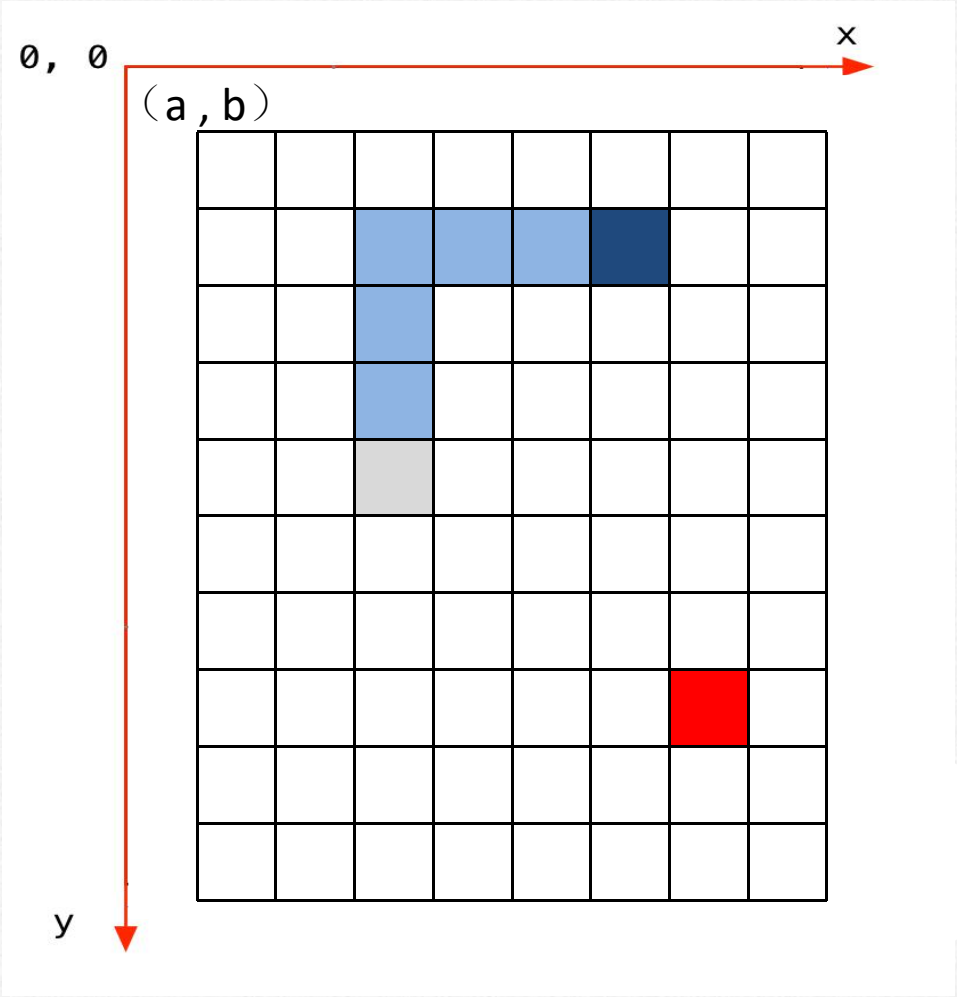
y = 网格纵坐标 * 8 + b



游戏基本原理

- 贪吃蛇的移动

1. 移动方向填充身体颜色
2. 蛇尾部填充背景颜色
3. 不能向自己的反方向前进



目录 Contents

第二部分

贪吃蛇的代码分析

贪吃蛇的代码分析

- Grid网格类

```
class Grid(object):
    def __init__(self, master=None,x=8, y=8, w=12, h=12):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.width=w
        self.height=h
        self.bg=disp.WHITE
        disp.clr(disp.WHITE)

    def draw(self, pos, color):
        x = pos[0] * 8 + self.x+1
        y = pos[1] * 8 + self.y+1
        disp.putrect(x,y,8,8,color)
```

网格基类

初始化左上角坐标
宽和高

渲染背景颜色和清屏

网格转换为屏幕坐标

对指定坐标长宽为8*8
区域填充color参数的颜色

贪吃蛇的代码分析

- 食物类

```
class Food(object):
    def __init__(self, grid, color = disp.GREEN):
        self.grid = grid
        self.color = color
        self.set_pos()
        self.type = 1
    def set_pos(self):
        x = randint(0, 12)
        y = randint(0, 12)
        self.pos = (x, y)
    def display(self):
        self.grid.draw(self.pos, self.color)
```

食物类

初始化

随机产生
网格坐标

调用grid类的draw
方法进行渲染

贪吃蛇的代码分析

- 蛇类

```
class Snake(object):
    def __init__(self, grid, color = disp.BLUE):
        self.grid = grid
        self.color = color
        self.body = [(5, 5), (5, 6), (5, 7)]
        self.direction = "Up"
        for i in self.body:
            self.grid.draw(i, self.color)
```

设置蛇身的网格坐标

调用grid类的draw方法进行渲染

贪吃蛇的代码分析

- 蛇类

```
    def move(self, new):
        self.body.insert(0, new)
        pop = self.body.pop()
        self.grid.draw(pop, self.grid.bg)
        self.grid.draw(new, self.color)
    def add(self ,new):
        self.body.insert(0, new)
        self.grid.draw(new, self.color)

#蛇吃到了特殊食物1，剪短自身的长度
    def cut_down(self,new):
        self.body.insert(0, new)
        self.grid.draw(new, self.color)
        for i in range(0,3):
            pop = self.body.pop()
            self.grid.draw(pop, self.grid.bg)
```

蛇身的列表插入新的节点

弹出蛇尾的节点

蛇尾渲染背景颜色

新的节点渲染蛇身的颜色

增加长度的方法，只新增节点，渲染蛇身颜色

吃到特殊食物，增加新节点并渲染蛇身的颜色

循环从尾部弹出三个节点，渲染背景颜色，也就是把蛇身减少三个长度

贪吃蛇的代码分析

- 蛇类

```
#蛇吃到了特殊食物2，回到最初长度
def init(self, new):
    self.body.insert(0, new)
    self.grid.draw(new, self.color)
    while len(self.body) > 3:
        pop = self.body.pop()
        self.grid.draw(pop, self.grid.bg)

 #蛇吃到了特殊食物3，改变了自身的颜色,纯属好玩
def change(self, new, color):
    self.color = color
    self.body.insert(0, new)
    for item in self.body:
        self.grid.draw(item, self.color)
```

蛇身的列表插入新的节点

循环从蛇尾弹出节点，渲染背景色，直到只剩三个节点

蛇身的列表插入新的节点

循环改变所有蛇身的节点的颜色

贪吃蛇的代码分析

- 游戏类

```
class SnakeGame():
    def __init__(self):
        self.grid = Grid()
        self.snake = Snake(self.grid)
        self.food = Food(self.grid)
        self.gameover = False
        self.score = 0
        self.status = ['run', 'stop']
        self.speed = 300
        self.display_food()
```

游戏初始化

随机产生不同类型的食物

```
def display_food(self):

    if randint(0, 40) == 5:
        self.food.color = disp.ORANGE
        self.food.type = 3
        while (self.food.pos in self.snake.body):
            self.food.set_pos()
        self.food.display()
    elif randint(0, 4) == 2:
        self.food.color = disp.PINK
        self.food.type = 4
        while (self.food.pos in self.snake.body):
            self.food.set_pos()
        self.food.display()
    elif len(self.snake.body) > 10 and randint(0, 16) == 5:
        self.food.color = disp.YELLOW
        self.food.type = 2
        while (self.food.pos in self.snake.body):
            self.food.set_pos()
        self.food.display()
    else:
        self.food.color = disp.GREEN
        self.food.type = 1
        while (self.food.pos in self.snake.body):
            self.food.set_pos()
        self.food.display()
    print(self.food.type)
```

贪吃蛇的代码分析

- 游戏类

对应不同的方向，产生不同的新的节点坐标，同时判断边界，如果到达边界在另一端产生

如果下一个节点在蛇的身体上，则游戏结束

如果下一个节点是食物，则根据食物的种类调用不同的方法

```
def move(self, color=disp.BLUE):
    # 计算蛇下一次移动的点
    head = self.snake.body[0]
    #print(self.snake.direction)
    if self.snake.direction == 'Up':
        if head[1] - 1 < 0:
            new = (head[0], 17)
        else:
            new = (head[0], head[1] - 1)
    elif self.snake.direction == 'Down':
        new = (head[0], (head[1] + 1) % 17)
    elif self.snake.direction == 'Left':
        if head[0] - 1 < 0:
            new = (13, head[1])
        else:
            new = (head[0] - 1, head[1])
    else:
        new = ((head[0] + 1) % 13, head[1])
        #撞到自己，设置游戏结束的标志位，等待下一循环
    if new in self.snake.body:
        self.gameover=True
    #吃到食物
    elif new == self.food.pos:
        print(self.food.type)
        if self.food.type == 1:
            self.snake.add(new)
            self.snake.change(new, disp.GREEN)

        elif self.food.type == 2:
            self.snake.cut_down(new)
            self.snake.change(new, disp.YELLOW)

        elif self.food.type == 4:
            self.snake.change(new, disp.PINK)
        else:
            self.snake.init(new)
            self.snake.change(new, disp.ORANGE)
        self.display_food()
    elif new == self.food.pos:
        self.snake.add(new)
        self.display_food()
    #什么都没撞到，继续前进
    else:
        self.snake.move(new)
```

贪吃蛇的代码分析

- 游戏类

```
def key_release(self, key):
        keymatch=["Down","Left","Up","Right"]
        key_dict = {"Up": "Down", "Down": "Up", "Left": "Right", "Right": "Left"}
        print(keymatch[key])
    #蛇不可以像自己的反方向走
    if keymatch[key] in key_dict and not keymatch[key] == key_dict[self.snake.direction]:
        self.snake.direction = keymatch[key]
        self.move()
```

如果下一个方向，不是自己的相反方向，则记录方向，并调用方法进行下一步移动

贪吃蛇的代码分析

- 游戏类

游戏主循环

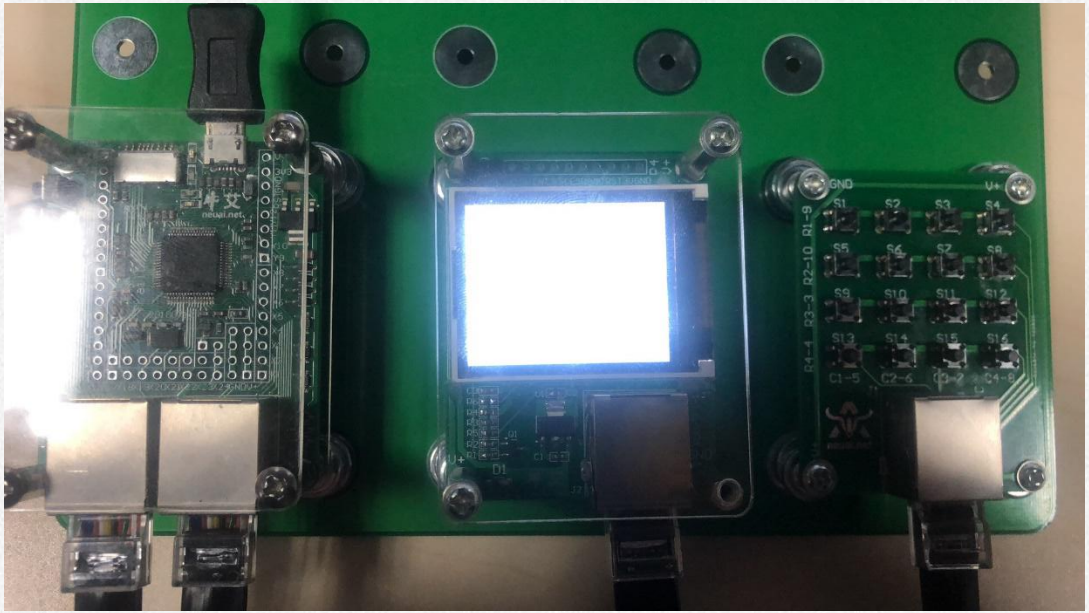
判断当前的按键，调用相应的处理事件

判断游戏是否结束，没结束的话继续延当前方向移动

```
def run(self):
    print('5')
    while True:
        i=0
        j=-1
        for k in keys:
            if k.value()==0:
                if i!=j:
                    print("i=",i)
                    print("j=",j)
                    j=i
                    self.key_release(i)

            i=i+1
            if i>3:
                i=0
        #首先判断游戏是否暂停
        if not self.status[0] == 'stop':
            if self.gameover == True:
                self.initial()
            else:
            #判断游戏是否结束
                self.move()
            time.sleep_ms(125)
```

贪吃蛇运行效果



trailBreaker开发板

目录 Contents

第二部分

认识列表

认识列表

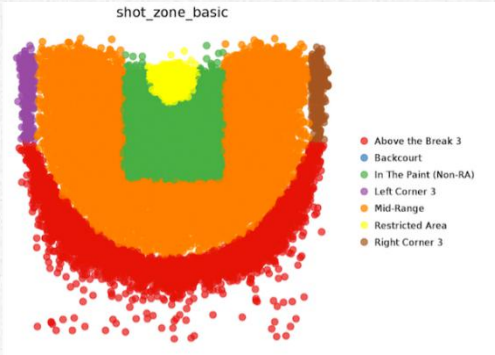
- 列表举例：

```
[2, 'Jump Shot', 'Los Angeles Lakers']
```

```
keymatch=["Down","Left","Up","Right"]
```

- 列表（list）是一个有序的序列结构，序列中的元素可以是不同的数据类型
- 列表可以进行一系列序列操作，如索引、切片、加、乘和检查成员等

科比投篮数据集



认识列表

- 将列表中的各元素用逗号分隔开，并用中括号将所有元素包裹起来

```
kobe_list = [2,'Jump Shot','Los Angeles Lakers']
print kobe_list
print type(kobe_list)
```

```
[2, 'Jump Shot', 'Los Angeles Lakers']
<type 'list'>
```

认识列表

- 使用append()方法添加元素，该方法会在列表末尾位置添加数据元素
- 使用remove()方法删除元素

```
kobe_list.append('POR')
print kobe_list
```

```
[2, 'Jump Shot', 'Los Angeles Lakers', 'POR']
```

```
kobe_list.remove(2)
print kobe_list
```

```
['Jump Shot', 'Los Angeles Lakers', 'POR']
```

认识列表

- remove()方法适用于知道要删除的值的情况，当我们不知道具体元素值，但是知道元素的索引位置时，我们可以使用 del 函数配合列表索引，删除索引位置的元素或者使用 pop()方法

```
kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']
del kobe_list[-2]
print kobe_list
kobe_list = [2, 'Jump Shot', 'Los Angeles Lakers', 'POR']
kobe_list.pop(-2)
print kobe_list
```

```
[2, 'Jump Shot', 'POR']
[2, 'Jump Shot', 'POR']
```

认识列表

- 通过 insert() 方法在指定的索引位置添加数据元素

```
[2, 'Jump Shot', 'POR']
```

```
kobe_list.insert(1,'Los Angeles Lakers')
print kobe_list
```

```
[2, 'Los Angeles Lakers', 'Jump Shot', 'POR']
```

认识列表

- Python语言中所有的索引都是从 0 开始计数的，如果列表中有 n 个元素，那么最后一个元素的索引是 n-1

```
['Jump Shot', 'Los Angeles Lakers', 'POR']
```

表1 kobe_list索引位置及相应的值

index（索引位置）	0	1	2	3
value（值）	2	'Jump Shot'	'Los Angeles Lakers'	'POR'

- 如果我们想要获取 kobe_list 中第 3 个元素及倒数第二个元素

```
kobe_list[2]
```

```
'Los Angeles Lakers'
```

```
kobe_list[-2]
```

```
'Los Angeles Lakers'
```

认识列表

- 切片操作需要提供起始索引位置和最后索引位置，然后用冒号：将两者分开

```
列表名称[起始索引位置：最后索引位置：步长]
```

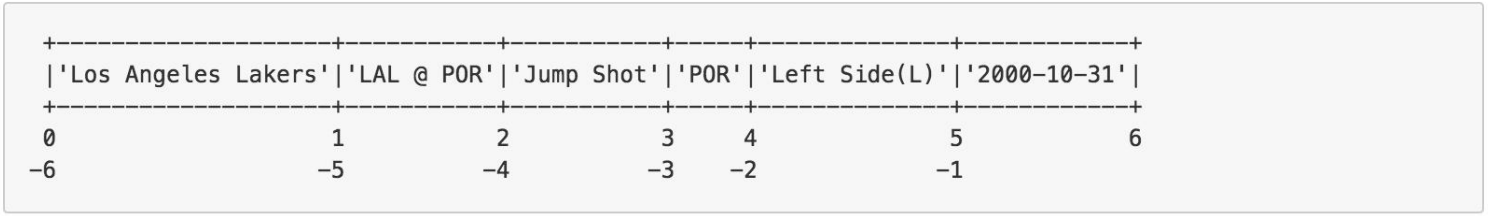
- 如果未输入步长，则默认步长为 1
- 切片操作返回一系列从起始索引位置开始到最后索引位置结束的数据元素
- 需要注意的是，起始索引位置的值包含在返回结果中，而最后索引位置的值不包含在返回结果中

认识列表

- 切片操作需要提供起始索引位置和最后索引位置，然后用冒号：将两者分开

```
kobe_list = ['Los Angeles Lakers', 'LAL @ POR', 'Jump Shot', 'POR', 'Left Side(L)', '2000-10-31']
len(kobe_list)
print kobe_list[1:4]
```

```
['LAL @ POR', 'Jump Shot', 'POR']
```



认识列表

- 逆向切片

```
print kobe_list[-5:-2]
```

```
['LAL @ POR', 'Jump Shot', 'POR']
```

- 我们可以省略起始索引位置，表示从最开始进行切片，当我们将两个索引都省略之后，我们将按原样复制一个列表，如果想要将列表的顺序颠倒，则可以使用::-1

```
print kobe_list[:-2]
print kobe_list[:]
print kobe_list[::-1]
```

```
['Los Angeles Lakers', 'LAL @ POR', 'Jump Shot', 'POR']
['Los Angeles Lakers', 'LAL @ POR', 'Jump Shot', 'POR', 'Left Side(L)', '2000-10-31']
['2000-10-31', 'Left Side(L)', 'POR', 'Jump Shot', 'LAL @ POR', 'Los Angeles Lakers']
```

目录 Contents

第二部分

认识元祖

认识元祖

```
(2, 'Jump Shot', 'Los Angeles Lakers', 'POR')
```

```
self.body = [(8, 11), (8, 12), (8, 13)]
```

- 元组（tuple）数据结构与列表类似，其中元素可以有不同的类型
- 但是元组中的元素是不可变的，即一旦初始化之后，就不能够再做修改（报错：元组对象不支持赋值）
- 由于元组是不可变的，因此元组对象没有append()、insert()和del这样的方法。
- 实际上，tuple的使用可以使得代码更安全，防止错误赋值导致重要对象的改变。

实践练习

- 增加贪吃蛇的初始长度
- 每次吃掉食物之后闪烁两次再前进
- 判断食物的方向，自动选择食物所在方向
- 每次随机产生多个食物


THANK YOU
牛艾科技