

# 目标

- 注释的作用
- 注释的分类及语法
- 注释的特点

## 一. 注释的作用

- 没有注释的代码

```
2048.py
83 def is_win(self):
84     return any(any(i >= self.win_value for i in row) for row in self.field)
85
86 def is_gameover(self):
87     return not any(self.move_is_possible(move) for move in actions)
88
89 def draw(self, screen):
90     help_string1 = '(W)Up (S)Down (A)Left (D)Right'
91     help_string2 = '      (R)Restart (Q)Exit'
92     gameover_string = '                GAME OVER'
93     win_string = '                YOU WIN!'
94     def cast(string):
95         screen.addstr(string + '\n')
96
97     def draw_hor_separator():
98         line = '+' + ('+-----' * self.width + '+')[1:]
99         separator = defaultdict(lambda: line)
100         if not hasattr(draw_hor_separator, "counter"):
101             draw_hor_separator.counter = 0
102         cast(separator[draw_hor_separator.counter])
103         draw_hor_separator.counter += 1
104
105     def draw_row(row):
106         cast(''.join('|{: ^5} '.format(num) if num > 0 else '|' for num in row)
107
108     screen.clear()
109     cast('SCORE: ' + str(self.score))
110     if 0 != self.highscore:
111         cast('HGHSCORE: ' + str(self.highscore))
112     for row in self.field:
113         draw_row(row)
114         draw_hor_separator()
```

- 添加注释的代码

```

2048.py
158 def main(stdscr):
159     def init():
160         #重置游戏棋盘
161         game_field.reset()
162         return 'Game'
163
164     def not_game(state):
165         #画出 GameOver 或者 Win 的界面
166         game_field.draw(stdscr)
167         #读取用户输入得到action, 判断是重启游戏还是结束游戏
168         action = get_user_action(stdscr)
169         responses = defaultdict(lambda: state) #默认是当前状态, 没有行为就会一直在当前界面循环
170         responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态
171         return responses[action]
172
173     def game():
174         #画出当前棋盘状态
175         game_field.draw(stdscr)
176         #读取用户输入得到action
177         action = get_user_action(stdscr)
178
179         if action == 'Restart':
180             return 'Init'
181         if action == 'Exit':
182             return 'Exit'
183         if game_field.move(action): # move successful
184             if game_field.is_win():
185                 return 'Win'
186             if game_field.is_gameover():
187                 return 'Gameover'
188             return 'Game'

```

以#开头的就是注释

- 通过用自己熟悉的语言，在程序中对某些代码进行标注说明，这就是注释的作用，能够大大增强程序的可读性。

## 二. 注释的分类及语法

注释分为两类：单行注释 和 多行注释。

- 单行注释

只能注释一行内容，语法如下：

```
1 | # 注释内容
```

- 多行注释

可以注释多行内容，一般用在注释一段代码的情况，语法如下：

```

1 | """
2 |     第一行注释
3 |     第二行注释
4 |     第三行注释
5 | """
6 |
7 | ...
8 |     注释1
9 |     注释2
10 |    注释3
11 | ...

```

快捷键： `ctrl + /`

## 2.1 快速体验

- 单行注释

```
1 # 输出hello world
2 print('hello world')
3
4 print('hello Python') # 输出(简单的说明可以放到一行代码的后面，一般习惯代码后面添加
   两个空格再书写注释文字)
```

- 多行注释

```
1 """
2     下面三行都是输出的作用，输出内容分别是：
3     hello Python
4     hello itcast
5     hello itheima
6 """
7 print('hello Python')
8 print('hello itcast')
9 print('hello itheima')
10
11
12 '''
13     下面三行都是输出的作用，输出内容分别是：
14     hello Python
15     hello itcast
16     hello itheima
17 '''
18 print('hello Python')
19 print('hello itcast')
20 print('hello itheima')
```

注意：解释器不执行任何的注释内容。

## 总结

- 注释的作用

用人类熟悉的语言对代码进行解释说明，方便后期维护。

- 注释的分类

- 单行： `# 注释内容`，快捷键`ctrl+/`
- 多行： `""" 注释内容 """` 或 `''' 注释内容 '''`

- 解释器不执行注释内容