

程序设计大作业：Todo List（计划清单）

设计初衷

- 对于很多程序员而言，gui界面的操作远远不如命令行来得方便，而现在大多的计划清单app都是针对gui界面开发，并没有一个命令行式的项目。因此我决定自己开发一个命令行式的计划清单
- 同时，我还设计了一个类似springboot启动页面的艺术字（只是个人兴趣）即：



使用方法

- 进入mytodo.py下的文件夹，在终端/bash执行 `python mytodo.py -h`，可以看到支持的所有命令：

```
PS E:\AGrade3\程序设计\todo_list> python .\mytodo.py -h
usage: mytodo.py [-h] {add,delete,dump,complete,showdate,showall,showunfinished,merge} ...

Todo List Manager

positional arguments:
  {add,delete,dump,complete,showdate,showall,showunfinished,merge}
                        Available commands
  add                   Add a new task
  delete               Delete a task
  dump                 Dump a certain task
  complete             Mark a task as complete
  showdate             Show tasks for a specific date
  showall              Show all tasks
  showunfinished       Show unfinished tasks
  merge                Merge tasks from previous dates into the specified date

optional arguments:
  -h, --help            show this help message and exit
```

- 我设计了一个todo_list需要的基本操作，并且相比于传统的计划清单，我设计了一个merge操作，把过去未完成任务归并到今天
- add操作---增加任务，执行 `python mytodo.py add -h` 可以看到关于add操作的格式，比如可以执行 `python mytodo.py add --task 复习算法设计 --duetime 2024-1-1 --priority 0`
 - 这时候查看2024-1-1的任务就可以看到

```
PS E:\AGrade3\程序设计\todo_list> python mytodo.py showdate --date 2024-1-1

Date: 2024-01-01, Day: Monday
Tasks:
Priority: 1 Task ID: 1, Task: 复习算法设计, Status: 未完成
```

- delete操作----删除任务。由于我的所有任务其实是存储在一个数组中，然后分配给每一天的，这样就实现了解耦，所以我们删除的时候需要给的是Task ID，并且删除一个任务之后可以实现在每一天如果有这个任务，都会被删除。比如执行 `python mytodo.py delete --id 1` 就会删除所有id为1的任务。
- dump/complete----放弃任务/完成任务。这个会改变任务的status，同时也是传入一个task id。
- showdate----展示某一天的所有任务，如执行 `python mytodo.py showdate --date 2024-1-1` 就可以展示某一天的所有任务
- showall----展示所有天的任务，直接执行 `python mytodo.py showall`。这样可以看到所有任务。

```
PS E:\AGrade3\程序设计\todo_list> python mytodo.py showall

Date: 2023-12-05, Day: Tuesday
Tasks:

Date: 2023-12-06, Day: Wednesday
Tasks:
Priority: 1 Task ID: 2, Task: 复习编译原理1, Status: 未完成
Date: 2023-12-14, Day: Thursday
Tasks:
Priority: 1 Task ID: 1, Task: 做大作业, Status: 未完成
Date: 2024-01-01, Day: Monday
Tasks:
Priority: 1 Task ID: 1, Task: 复习算法设计, Status: 未完成
```

- showunfinished----展示所有未完成的任务，和showall的执行方法差不多，只要给出showunfinished参数就可以了


```
PS E:\AGrade3\程序设计\todo_list> python mytodo.py showunfinished
```



当前未完成的task:
Task ID: 2, Task: 复习编译原理1, Status: 未完成

- 最后是merge，即将未完成的任务归并到某一天。比如 `python mytodo.py merge --date 2024-1-1` 执行之后，会把所有未完成的任务归并到2024-1-1

```
PS E:\AGrade3\程序设计\todo_list> python mytodo.py showall
```



Date: 2023-12-05, Day: Tuesday
Tasks:

Date: 2023-12-06, Day: Wednesday
Tasks:

Priority: 1 Task ID: 2, Task: 复习编译原理1, Status: 未完成

Date: 2023-12-14, Day: Thursday
Tasks:

Date: 2024-01-01, Day: Monday
Tasks:

Priority: 1 Task ID: 2, Task: 复习编译原理1, Status: 未完成

- 如果有不清楚的操作，可以直接执行 `-h` 选项，我为每个参数后面都写了详细的解释

实现逻辑

为了实现系统的解耦，我将任务清单分成了五个模块。一个是个性输出，这个相对独立，即：

```
def todo_output():
    todo_art = """
                                ,-----,
                                ,/  .`|  ,-----..
                                ,-----..
                                ,---,.      ,--,      ,--,
                                /  /  \
                                ,'. .' |      ,---,|'. \  /  .'
                                /  .      :      ;      //  .      :      .' .' \
                                ,---.' |      /_ . / | \ \ ' / ;      .'___,/      ,'.  /  ;.  \,---.'
                                \  .  /  ;.  \
                                |  |  .',---, |  ' :'. \  /  /  .'      |  :      |.  ;  /  ` ;|  |  .' \
                                |.  ;  /  ` ;
    """
```

```

: : : /___/ \. : | \ \ / / ./ ; |.'; ;; | ; \ ; | : : | '
| ; | ; \ ; |
: | | -, \ \ , ' ' \ \ .' / \----' | || : | ; | ' | ' ' ;
: | : | ; | '
| : ; / | \ ; ` , ' \ ; ; ' : ; . | ' ' ' : ' | ; .
|. | | ' ' ' :
| | . ' \ \ ' / \ \ \ | | | ' ; \ ; / || | : |
" ; \ ; / |
' : ' ' ' \ | ; ^ \ \ ' : | \ \ ' , / ' : | /
; \ \ ' , /
| | | \ ; ; /___; \ ; \ ; |.' ; : / | | ' ` , /
; : /
| : \ : \ \ : / \ \ ; '----' \ \ \ .' ; : .'
\ \ \ .'
| | , ' \ ' ; | / \ ' | \----' | , .'
\----' \----' \----' \----' \----' \----' \----'
'''
print(todo_art)

```

剩下的四个是一个从高到低封装的模块。

todo_list_item

这个类是最底层的，设计了一个计划中一项需要的所有元素。设计如下：

```

def __init__(self, task_name, due_date):
    """
    Constructor for TodoListItem.

    Args:
        task_name (str): Name or description of the task.
        due_date (datetime): Due date of the task.
    """
    self.task_name = task_name
    self.task_id = TodoListItem._get_next_id()
    self.due_date = due_date
    # Convert both datetime objects to date objects for comparison
    today_date = datetime.today().date()
    due_date_date = due_date.date()
    self.status = '未完成' if today_date <= due_date_date else '已过期'

```

我设计了四个元素，即名字，id，截止日期和状态。并且，id我类似在数据库中一样选择了自增主键，这个设计主要是为了简单和确保唯一性。同时，为了代码更具有可读性和操作性，我重写了这个类的输出函数：

```
def __str__(self):
    """
    String representation of the TodoListItem.

    Returns:
        str: String representation of the task.
    """
    return f"Task ID: {self.task_id}, Task: {self.task_name}, Status: {self.status}"
```

todo_list_day_items

这个类用于封装每一天里面对于清单项的操作。对于单天而言，只有增加和删除两个操作。这个类偏于简单，并且我也重写了这个类的输出函数。

todo_list

这个类就用于支持所有的操作了。这个类包装了每一天的总清单项，支持的操作就如用户可以使用的操作一样。为了简洁起见，我只想说一下关于持久化的操作。由于我的计划清单是无状态的，每一次进行操作后都要进行持久化，我选择写一个log文件到硬盘，然后第二次读取的时候再进行恢复，如下：

```
def save(self, filename):
    """
    Save the TodoList to a file.

    Args:
        filename (str): The name of the file to save to.
    """
    with open(filename, 'wb') as file:
        # 保存 all_items 列表中的任务项
        pickle.dump(self.all_items, file)
        # 保存每个日期对象，包括日期和其中的任务项的索引
        day_items_data = []
        for day_item in self.day_items:
            day_item_indices = [self.all_items.index(task) for task in
day_item.items]

            day_item_data = {
                'date': day_item.date,
                'day_of_week': day_item.day_of_week,
                'items_indices': day_item_indices
            }
            day_items_data.append(day_item_data)
        pickle.dump(day_items_data, file)

def restore(self, filename):
    """
    Restore the TodoList from a file.

    Args:
        filename (str): The name of the file to restore from.
    """
    with open(filename, 'rb') as file:
        # 恢复 all_items 列表中的任务项
        self.all_items = pickle.load(file)
```

```

# 恢复每个日期对象，包括日期和其中的任务项的索引
day_items_data = pickle.load(file)
self.day_items = []
for day_item_data in day_items_data:
    date = day_item_data['date']
    day_of_week = day_item_data['day_of_week']
    items_indices = day_item_data['items_indices']

    # 根据索引从 all_items 中获取任务项
    items = [self.all_items[index] for index in items_indices]

    # 创建并添加日期对象
    day_item = TodoListDayItems(date.year, date.month, date.day)
    day_item.day_of_week = day_of_week
    day_item.items = items
    self.day_items.append(day_item)

```

并且，这里还利用了自增主键的性质，在恢复和存储的时候我并不会存储task id，而是让系统自己生成，这样可以保证id的唯一性。

mytodo

这就是封装了所有与用户的交互，包括命令行交互的arg，之后对于mytodo的调用等等。

代码风格

我的总体代码风格个人感觉还是不错的，我为每个类和每个函数都写了统一格式的注释，增加代码可读性，所有变量的命名都遵从统一规范，并且类的设计模块化，完全解耦并有层次感