1. Python 编程语言基本成分分析

- python 语言基本成分分析
  - 数据成分
    - 标准数据类型
      - numbers
      - string
      - list
      - tuple
      - dictionary
    - python 的变量不需要提前申明变量类型和分配空间
  - 运算成分
    - 算术运算符
      - 常规运算符
      - ** 幂运算
      - //取整除
    - 比较（关系）运算符
      - 与c++基本相同
    - 赋值运算符
      - =
      - 支持上述算数运算符的算数赋值操作
    - 位运算符
      - 与c++相同
    - 逻辑运算符
      - and
      - or
      - not
    - 成员运算符
      - in 在序列中返回True
      - not in 不在序列中返回False
    - 身份运算符
      - is 判断两个标识符是不是引自同一个对象
      - is not 判断连个标识符是不是引自不同对象
    - 
      运算符优先级
  - 控制成分
    - 条件语句
      - if
      - else
      - elif
    - 循环语句
      - while
      - for
    - 控制语句
      - pass
      - continue
      - break
  - 传输成分
    - input
    - print
    - open
      - f.tell()
      - f.read()
      - f.readline()
      - f.write()
      - f.seek()
      - f.close()
    - write

2. C 语言特性
   a) 心理特性
      i. 紧致性好：python 只有 31 个关键字，算术及逻辑操作符都与正常的数学学习相似，控制语句只有五种，甚至没有 switch。还有为了异常控制而特别设置的 try 语句，便于使用。
      ii. 局部性高：程序由模块组成，程序的各个部分的独立性高，高内聚低耦合，模块之间彼此独立，有良好的使用性能，python 是脚本式语言，也被称为胶水语言，在项目中有良好的可移植性。
      iii. 线性：在逻辑上符合线性的程序流，不会出现像 C 语言的 goto 语句那样对程序的易读性产生伤害的语句。
      iv. 易用性：python 的语法简单清晰，减少了如变量的类型声明等限制，使得语言的可移植性增强。
      v. 易上手：python 有强大的各种外部库支持，新手很容易用很低的学习成本完成一个实用性很强的项目，对于学习者来说是很好的鼓励。
   b) 工程特性
      i. 设计翻译成代码的便利程度： 这在很大程度上取决于设计与计算机实现的贴合程度，python 有实用性强的数据类型，以及强大的函数库支持，都利于将设计翻译成代码。
      ii. 编译器的效率：python 是较为高级的语言，相对于 C 和 C++来说，python 的编译和运行速度都较慢，所以一些对于程序运行速度要求较高的项目很少使用 python。
      iii. 代码的可移植性：python 被称为胶水语言，局部性好，不用声明变量类型等特点使得 python 有良好的可移植性。
      iv. 配套的开发工具：在 python 发展的过程中，已经发展出了强大的配套开发工具，作为脚本语言，python 可以在控制台时实交互检查自己的代码，查找文档，十分方便。Python 已经发展出了强大的各种库，可以应用在不同领域的开发。
      v. 可维护性：Python 遵循程序的顺序流结构，没有 goto 等伤害代码可读性的语句，语法结构清晰，对于异常处理等情况有特别的控制语句，清晰易懂。但是由于 Python 的灵活性，比如说没有类型声明，很容易在并不严格的规范中产生失误。
      vi. 开发快：Python 没有 JAVA 那样繁琐的规定，十分灵活，适用开发期限短的项目。
   c) 应用特性：
      i. 科学研究：比如说在生物统计，化学计算方面都有完善的专业库可以使用，为没有太多计算机基础的其他学科的研究人员提供了实用性强的工具。
      ii. 数据挖掘和分析：Python 以其脚本语言的特性，有强大的爬虫和数据处理功能。
      iii. 统计与计算：Python 现在在统计方面占据着统治地位，pandas ipython 等库更是极大地丰富了功能。Python 作为一个成熟的编程语言，既可以做逻辑结构比较复杂的项目，也可以被初级的入门人员作为数据处理工具使用，适用于各种规模的项目，简单易用。
3. 经典 Python 语言开源项目分析“Records: SQL for Humans™”
   a) 项目介绍：record 是一个简单但是强大的用于查询数据库原始数据的库，用 Python 编写，由 kennethreitz 完成并在 github 上维护。
   b) 项目源码来源：https://github.com/kennethreitz/records
   c) 项目分析：
      i. 序言性注释：

```
A
Kenneth
Reitz
project.
            Usage:
              records <query> [<format>] [<params>...] [--url=<url>]
              records (-h | --help)
            Options:
              -h --help     Show this screen.
              --url=<url>   The database URL to use. Defaults to $DATABASE_URL.
            Supported Formats:
               %(formats_lst)s
               Note: xls, xlsx, dbf, and ods formats are binary, and should only be
                     used with redirected output e.g. '$ records sql xls > sql.xls'.
            Query Parameters:
                Query parameters can be specified in key=value format, and injected
```

```
    into your query in :key format e.g.:
    $ records 'select * from repos where language ~= :lang' lang=python
Notes:
  - While you may specify a database connection string with --url, records
    will automatically default to the value of $DATABASE_URL, if available.
  - Query is intended to be the path of a SQL file, however a query string
    can be provided instead. Use this feature discernfully; it's dangerous.
  - Records is intended for report-style exports of database queries, and
    has not yet been optimized for extremely large data dumps.
```

本项目将序言性注释写在了最后，说明了主要的库使用方法，作者，和功能以及库的一些特性，有助于拿到这个库的使用者开始使用。

ii. 功能性注释（在以下内容中为了排版方便，会截取注释相关内容和相关的代码行来做分析）

行号	注释
17-18	Defisexception(obj): """Given an object, return a boolean indicating whether it is an instance or subclass of :py:class:`Exception`. """
次函数用于检测对象是一个项目的子类，还是 Python 的 exception 错误信息	
28	class Record(object): """A row, from a query, from a database."""
这个类是用来封装一条查询讯息或者是一条从数据库返回的信息的。	
35	#Ensurethatlengthsmatchproperly. assert len(self._keys) == len(self._values)
确保数据的长度相同，能够被函数解析。	
39	Defkeys(self): """Returns the list of column names from the query."""
返回查询的关键字	
43	def values(self):

	"""Returns the list of values from the query."""
返回查询的值	
50,54	def __getitem__(self, key): # Support for index-based lookup. # Support for string-based lookup.
两个部分分别用来支持下标查询和字符串作为关键字的查询	
71	def __dir__(self): # Merge standard attrs with generated ones (from column names).
在一列信息中，将一些生成的属性名合并到标准的内置属性名上。	
81	"""Returns the row as a dictionary, as ordered."""
以字典的形式将查询行进行解析	
89	def dataset(self): """A Tablib Dataset containing the row."""
用于解析成一个含有查询行的 Tablib 数据集	
99	def export(self, format, **kwargs): """Exports the row to the given format."""
将查询的数据用给定的格式输出	
104	class RecordCollection(object): """A set of excellent Records from a query."""
封装来自查询的数据	
114-115	def __iter__(self): """Iterate over all rows, consuming the underlying generator only when necessary."""
循环遍历所有的查询行，只有在必要的时候消耗底层的生成器	

146

```
# Convert
RecordCollection[1]
into slice.
                                if is_int:
                                    key = slice(key, key + 1)
```

将一个 recordcollection 转换成切片器

171-

172

```
def
dataset(self):
                            """A Tablib Dataset representation of the RecordCollection."""
                            # Create a new Tablib Dataset.
```

一个代表着 record collection 的数据集

229-

233

```
def one(self,
default=None,
as_dict=False,
as_ordereddict=False):
                                    """Returns a single record for the RecordCollection,
                            ensuring that it
                                    is the only record, or returns `default`. If `default`
                            is an instance
                                    or subclass of Exception, then raise it instead of
                            returning it."""


                                    # Ensure that we don't have more than one row.
```

值返回一条数据库中的查询信息，如果出现错误的话，raise 错误信息，而不是返回类。

295-

298

```
def query(self,
query,
fetchall=False,
**params):
                                    """Executes the given SQL query against the Database.
                            Parameters can,
                                    optionally, be provided. Returns a RecordCollection, which
                            can be
                                    iterated over to get result rows as dictionaries.
                                    """
```

向数据库执行 SQL 操作，这些参数可以被选择性地赋予，会赶回一个数据记录，这个数据记录可以像字典一样被遍历或获取。

381-

382

```
def
query_file(self,
path,
fetchall=False,
**params):
                                     """Like Connection.query, but takes a filename to load a
                             query from."""
```

这是一个 filename 用来加载查询指令。

```
421     def
        bulk_query(self,
        query,
        *multiparams):
                                """Bulk insert or update."""
```

次函数用来确认是否需要拦截一个不合法的数据库更新

```
430
        def
        transaction(self):
                                """Returns a transaction object. Call ``commit`` or
                            ``rollback``
                                on the returned object as appropriate."""
```

对于一个已经返回了的合适的对象，commit 或者 rollback 操作

```
491     # Be
        ready to
        fail on
493     missing
        packages
496             try:
                    # Create the Database.
500                 db = Database(arguments['--url'])
504
                    # Execute the query, if it is a found file.
509                 if os.path.isfile(query):
                        rows = db.query_file(query, **params)

                    # Execute the query, if it appears to be a query string.
                    elif len(query.split()) > 2:
                        rows = db.query(query, **params)

                    # Otherwise, say the file wasn't found.
                    else:
                        print('The given query could not be found.')
                        exit(66)

                    # Print results in desired format.
```

一系列最外层的操作说明，包括创建数据库，在发现文件时执行查询，在找到查询语句时执行查询，没有找到文件的状况，以及对将查询结果用想要的格式输出。

```
532     # Run the CLI when executed directly.
```

直接运行方法

iii.

部分没有注释的主要代码行

```
import os
from sys import stdout
from collections import OrderedDict
from contextlib import contextmanager
from inspect import isclass


import tablib
from docopt import docopt
from sqlalchemy import create_engine, exc, inspect, text


DATABASE_URL = os.environ.get('DATABASE_URL')




def isexception(obj):
    if isinstance(obj, Exception):
        return True
    if isclass(obj) and issubclass(obj, Exception):
        return True
    return False




class Record(object):
    __slots__ = ('_keys', '_values')


    def __init__(self, keys, values):
        self._keys = keys
        self._values = values


        assert len(self._keys) == len(self._values)


    def keys(self):


        return self._keys


    def values(self):


        return self._values


    def __repr__(self):
```

```
    return '<Record {}>'.format(self.export('json')[1:-1])


def __getitem__(self, key):

    if isinstance(key, int):
        return self.values()[key]



    if key in self.keys():
        i = self.keys().index(key)
        if self.keys().count(key) > 1:
            raise KeyError("Record contains multiple '{}' fields.".format(key))
        return self.values()[i]


    raise KeyError("Record contains no '{}' field.".format(key))


def __getattr__(self, key):
    try:
        return self[key]
    except KeyError as e:
        raise AttributeError(e)


def __dir__(self):
    standard = dir(super(Record, self))

    return sorted(standard + [str(k) for k in self.keys()])


def get(self, key, default=None):

    try:
        return self[key]
    except KeyError:
        return default


def as_dict(self, ordered=False):

    items = zip(self.keys(), self.values())


    return OrderedDict(items) if ordered else dict(items)


@property
def dataset(self):
    data = tablib.Dataset()
    data.headers = self.keys()
```

```
        row = _reduce_datetimes(self.values())
        data.append(row)


        return data


    def export(self, format, **kwargs):

        return self.dataset.export(format, **kwargs)




class RecordCollection(object):
    def __init__(self, rows):
        self._rows = rows
        self._all_rows = []
        self.pending = True


    def __repr__(self):
        return '<RecordCollection size={} pending={}>'.format(len(self),
self.pending)


    def __iter__(self):
        i = 0
        while True:
            if i < len(self):
                yield self[i]
            else:


                try:
                    yield next(self)
                except StopIteration:
                    return
            i += 1


    def next(self):
        return self.__next__()


    def __next__(self):
        try:
            nextrow = next(self._rows)
            self._all_rows.append(nextrow)
            return nextrow
        except StopIteration:
```

```
        self.pending = False
        raise StopIteration('RecordCollection contains no more rows.')


def __getitem__(self, key):
    is_int = isinstance(key, int)



    if is_int:
        key = slice(key, key + 1)


    while len(self) < key.stop or key.stop is None:
        try:
            next(self)
        except StopIteration:
            break


    rows = self._all_rows[key]
    if is_int:
        return rows[0]
    else:
        return RecordCollection(iter(rows))


def __len__(self):
    return len(self._all_rows)


def export(self, format, **kwargs):
    """Export the RecordCollection to a given format (courtesy of Tablib)."""
    return self.dataset.export(format, **kwargs)


@property
def dataset(self):
    """A Tablib Dataset representation of the RecordCollection."""

    data = tablib.Dataset()



    if len(list(self)) == 0:
        return data


    # Set the column names as headers on Tablib Dataset.
    first = self[0]


    data.headers = first.keys()
```

```
    for row in self.all():
        row = _reduce_datetimes(row.values())
        data.append(row)


    return data


def all(self, as_dict=False, as_ordereddict=False):

    been fetched yet, consume the iterator and cache the results."""


    rows = list(self)


    if as_dict:
        return [r.as_dict() for r in rows]
    elif as_ordereddict:
        return [r.as_dict(ordered=True) for r in rows]


    return rows


def as_dict(self, ordered=False):
    return self.all(as_dict=not(ordered), as_ordereddict=ordered)


def first(self, default=None, as_dict=False, as_ordereddict=False):




    try:
        record = self[0]
    except IndexError:
        if isexception(default):
            raise default
        return default


    if as_dict:
        return record.as_dict()
    elif as_ordereddict:
        return record.as_dict(ordered=True)
    else:
        return record


def one(self, default=None, as_dict=False, as_ordereddict=False):
    try:
```

```
            record = self[0]
        except IndexError:
            if isexception(default):
                raise default
            return default
        try:
            self[1]
        except IndexError:
            pass
        else:
            raise ValueError('RecordCollection contained more than one row. '
                             'Expects only one row when using '
                             'RecordCollection.one')



        if as_dict:
            return record.as_dict()
        elif as_ordereddict:
            return record.as_dict(ordered=True)
        else:
            return record


    def scalar(self, default=None):
        row = self.one()
        return row[0] if row else default
                    ……
                        中间省略部分代码行

    for i in range(len(row)):
        if hasattr(row[i], 'isoformat'):
            row[i] = row[i].isoformat()
    return tuple(row)


def cli():
    supported_formats = 'csv tsv json yaml html xls xlsx dbf latex ods'.split()
    formats_lst=", ".join(supported_formats)
    cli_docs ="""Records: SQL for Humans™
A Kenneth Reitz project.
Usage:
  records <query> [<format>] [<params>...] [--url=<url>]
  records (-h | --help)
Options:
  -h --help     Show this screen.
  --url=<url>   The database URL to use. Defaults to $DATABASE_URL.
Supported Formats:
   %(formats_lst)s
   Note: xls, xlsx, dbf, and ods formats are binary, and should only be
         used with redirected output e.g. '$ records sql xls > sql.xls'.
Query Parameters:
    Query parameters can be specified in key=value format, and injected
```

```
    into your query in :key format e.g.:
    $ records 'select * from repos where language ~= :lang' lang=python
Notes:
  - While you may specify a database connection string with --url, records
    will automatically default to the value of $DATABASE_URL, if available.
  - Query is intended to be the path of a SQL file, however a query string
    can be provided instead. Use this feature discernfully; it's dangerous.
  - Records is intended for report-style exports of database queries, and
    has not yet been optimized for extremely large data dumps.
    """ % dict(formats_lst=formats_lst)


    # Parse the command-line arguments.
    arguments = docopt(cli_docs)


    query = arguments['<query>']
    params = arguments['<params>']
    format = arguments.get('<format>')
    if format and "=" in format:
        del arguments['<format>']
        arguments['<params>'].append(format)
        format = None
    if format and format not in supported_formats:
        print('%s format not supported.' % format)
        print('Supported formats are %s.' % formats_lst)
        exit(62)



    try:
        params = dict([i.split('=') for i in params])
    except ValueError:
        print('Parameters must be given in key=value format.')
        exit(64)


    try:
        db = Database(arguments['--url'])


        if os.path.isfile(query):
            rows = db.query_file(query, **params)


if __name__ == '__main__':
    cli()
```

4. 对于自己项目的修改
   a) Views.py 文件

```
5.  # -*- coding: utf-8 -*-

    from django.http import HttpResponse , HttpResponseRedirect
```

```
from django.shortcuts import render_to_response,render
from django import forms
from userpw.models import User
from userpw.models import event
from matplotlib import pyplot as plt
import matplotlib as mpl
import numpy as np
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
import re
# Create your views here.
##定义了 UserForm 表单用于注册和登录页面，ChangeForm 表单用于修改密码页面
class UserForm(forms.Form):
    username = forms.CharField(label='用户名')
    password = forms.CharField(label='密　码',widget=forms.PasswordInput())

def regist(request):
    if request.method == 'POST':
        uf = UserForm(request.POST)
        if uf.is_valid():
            username = uf.cleaned_data['username']
            password = uf.cleaned_data['password']

            ##判断用户原密码是否匹配
            user = User.objects.filter(username = username)
            if user:
                info='已经被注册了'
                return render_to_response('users/error_alert.html', {'error': info})
            elif len(user) == 0:
                user = User()
                user.username = username
                user.password = password
                user.save()
                return HttpResponseRedirect("/log/login/")
    else:
        uf = UserForm()

    return render_to_response('users/regist.html', {'uf': uf})

def login(request):
    if request.method == 'POST':
        ##获取表单信息
        uf = UserForm(request.POST)
        if uf.is_valid():
            print(request.POST)
            username = uf.cleaned_data['username']
            password = uf.cleaned_data['password']
            ##判断用户密码是否匹配
            user = User.objects.filter(username = username)
            if user:
                passwd = User.objects.filter(username = username, password = password)
                if passwd:
                    request.session["username"]=username
                    return HttpResponseRedirect("/log/if_timeline/")
                else:
```

```
                info="密码错误"
            elif len(user) == 0:
                info = '请检查用户名是否正确!'

            return render_to_response('users/error_alert.html', {'error': info})
    else:
        uf = UserForm()

    return render_to_response('users/login.html', {'uf': uf})

#控制时间线函数
def timeline(request):
    username=request.session.get("username")
    eventlist=event.objects.filter(username=username)
    list2=[]
    list3=[]
    for i in eventlist:
        detail=i.detail
        if detail!=None and len(detail)>0:
            list2.append(i)
    for i in list2:
        templist=[]
        #转化为需要的时间格式
        period=i.begin_h+":"+i.begin_m+":"+i.begin_s+"---"+i.end_h+":"+i.end_m+":"+i.end_s
        detail=i.detail
        if detail!=None:
            day = i.weekday
            weekday = ""
            if day == "1":
                weekday += "星期一"
            elif day == "2":
                weekday += "星期二"
            elif day == "3":
                weekday += "星期三"
            elif day == "4":
                weekday += "星期四"
            elif day == "5":
                weekday += "星期五"
            elif day == "6":
                weekday += "星期六"
            elif day == "7":
                weekday += "星期日"
            templist.append(period)
            templist.append(weekday)
            templist.append(detail)
            list3.append(templist)
    #按照最近的时间输出事件
    list4=list3[::-1]
    print(list4)
    return render(request, 'users/timeline.html',{'eventlist':list4})

#是否进入时间线页面
def if_timeline(request):
    return render(request,'users/if_timeline.html')
```

```
#错误信息提示页面
def error(request):
    return render(request, 'users/error_alert.html')

倒计时页面
def countdown(request):
    if request.is_ajax():
        # print 时间
        if request.POST.get('time'):
            b=request.POST.get('time')
            a=b.split("[")[1].split("]")[0].split(",")
            print(a)
            event1=event() #类的实例化是要有括号的
            username=request.session.get("usernam")
             #按照用户名的信息查找以及转化
            if username:
                print(username)
                event1.username=username
                event1.begin_h=a[0]
                event1.begin_m=a[1]
                event1.begin_s=a[2]
                event1.end_h=a[3]
                event1.end_m=a[4]
                event1.end_s=a[5]
                event1.weekday=a[6]
                event1.save()
                request.session["time"]= a
                request.session["id"]=event1.id
                return HttpResponseRedirect("/log/finish/")
            else:
                print("1")
                return HttpResponseRedirect("/log/login/")

    return render(request, 'users/countdown.html')

#用户完成并更新时间页面
def finish(request):
    print(request.session)
    if request.method == 'POST':
        if request.POST:
            #从浏览器中获取需要的信息
            if request.session.get("id"):
                id = request.session.get("id")
                event1=event.objects.get(id=id)
                event1.detail=request.POST['detail']
                event1.type=request.POST['type']
                event1.save()
            info="事件更新成功！"
            return HttpResponseRedirect('/log/timeline/')
        else:
            info="还没有事件哟~"
        return render_to_response('users/error_alert.html', {'error': info})
    else:
        print("error")
```

```
    return render(request, 'users/finish.html')

#时间格式转化成秒
def count_period( begin_h,begin_m,begin_s,end_h,end_m,end_s):
    begin=int(begin_h)*3600+int(begin_m)*60+int(begin_s)
    end=int(end_h)*3600+int(end_m)*60+int(end_s)
    print(begin)
    print(end)
    return abs(begin-end)

#总结页面
def summary(request):
    username=request.session.get("username")
    print(username)
    eventlist=event.objects.filter(username=username)
    daydict=[0]*7
    typedict=[0]*3 #无效拖延;高效工作;休息
    for i in eventlist:
        print(i)
        if i.detail!=None:
            #转化时间格式
            duration=count_period(i.begin_h,i.begin_m,i.begin_s,i.end_h,i.end_m,i.end_s)/3600
            day = i.weekday
            type = i.type
            print(type)
            print(duration)
            #按照工作事件类型来生成表格
            if i.type=="高效工作":
                day_time_now = daydict[int(day) - 1] + duration
                daydict[int(day) - 1] = day_time_now
            if type == "无效拖延":
                duration_temp = typedict[0] + duration
                typedict[0] = duration_temp
            elif type == "高效工作":
                duration_temp = typedict[1] + duration
                typedict[1] = duration_temp
            elif type == "休息":
                duration_temp = typedict[2] + duration
                typedict[2] = duration_temp
    mpl.rcParams['axes.titlesize'] = 20
    mpl.rcParams['xtick.labelsize'] = 16
    mpl.rcParams['ytick.labelsize'] = 16
    mpl.rcParams['axes.labelsize'] = 16
    mpl.rcParams['xtick.major.size'] = 0
    mpl.rcParams['ytick.major.size'] = 0
    weekday_label=["一","二","三","四","五","六","日"]
    print(daydict)
    print(typedict)
    #plt.bar(range(len(daydict)),daydict,tick_label=weekday_label)
    explode=[]
    l=["无效拖延","高效工作","休息"]
    c=[  '#7199cf','#4fc4aa','#e1a7a2'] #lightcoral ;skyblue;palegreen
    colors=[ '#7199cf','#4fc4aa','#e1a7a2', '#7199cf','#4fc4aa','#e1a7a2','#7199cf']
    f=[]
```

```
for i in typedict:
    explode.append(i/300)
    f.append(round(i,2))
#plt.axes(aspect='equal')
#plt.pie(f,explode=explode,labels=l,colors=c,autopct = '%3.1f%%')
#plt.savefig("type.png")
#整体图标签

fig=plt.figure('周统计表')
fig.set_size_inches(13,4)
#http://python.jobbole.com/87471/ 教程来源
#柱状子图
ax=fig.add_subplot(131)
ax.set_title("每日专注时长")
xticks=np.arange(7)
print(xticks)
bar_width=0.5
bars= ax.bar(xticks,daydict,width=bar_width,edgecolor='white')

ax.set_ylabel('时长（h)')
ax.set_xticklabels(weekday_label)
ax.set_xlim([bar_width/2-1,8])
ax.set_ylim([0,int(max(daydict))])
ax.set_xticks(xticks)
for bar,color in zip(bars,colors):
    bar.set_color(color)
#效率分布子图
ax=fig.add_subplot(132)
ax.set_title("效率分布")
labels=['{}\n {}h'.format(type,time) for type,time in zip(l,f) ]
ax.pie(f,explode=explode,labels=l,colors=c,autopct = '%3.1f%%')

#生成叙事分析子图
ax=fig.add_subplot(133)
ax.set_title("趋势分析")
print(daydict)
print(sum(daydict))
weekhour=[7,9,3]
weekhour.append(sum(daydict))
print(weekhour)
xticks=np.arange(4)
bar_width=0.5
bars= ax.bar(xticks,weekhour,width=bar_width,edgecolor='white')
ax.set_ylabel('时长（h)')
ax.set_xlabel("相对周数")
ax.set_xticklabels(['-3', '-2',' -1','0'])
ax.set_xlim([bar_width/2-1,5])
ax.set_ylim([0,int(max(weekhour))])
ax.set_xticks(xticks)
colors2=[ '#7199cf','#4fc4aa','#e1a7a2', '#7199cf']
for bar,color in zip(bars,colors2):
    bar.set_color(color)
plt.savefig(r'C:\Users\Lenovo\Desktop\大二下\金融软件工程\实验\个人项目\focus\static\pic.png')
```

```
    #plt.pie(f, explode=explode, labels=l, colors=c, autopct='%1.2f%%', shadow=True)
    #fig = plt.matplotlib.pyplot.gcf()
    #fig.set_size_inches(9, 8)
    #plt.savefig('type.png')
    return render(request,'users/summary.html')

#休息页面
def rest(request):
    return render(request,'users/rest.html')
```

6. 总结：

a) 较为详细的注释极大地提高了程序的可读性。
b) 功能性的注释有助于代码的后续使用者进行改进
c) 序言性的注释使得代码能够被追根溯源，也会对代码的基本功能和概况做出介绍。