上海大学

SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号　第 8 组

实 验 序 号　1

学　　号　20121034

姓　　名　胡才郁

日　　期　2022 年 3 月 13 日

实验一 PYTHON 基础与数据结构

1 实验目的与要求

1. 熟悉 Python 的开发调试环境
2. 熟悉 Python 外部库的调用
3. 掌握 Python 语言基本语法
4. 熟悉 Python 的数据结构

2 实验环境

Python 3.9.7 [MSC v.1916 64 bit (AMD64)]
PyCharm 2021.3.2

3 实验内容

3.1 Python代码理解 polygon.py

（1）运行和阅读代码
（2）理解代码功能
（3）修改代码，练习调用文件中其他几个图形函数。

3.2 输入输出

编写脚本文件，设计友好的用户输入输出提示，用户输入一个时间（24 小时制，包含时、分、秒），输出 1 秒后的时间。

3.3 反序对

如果一个单词是另一个单词的反向序列，则称这两个单词为“反向对”。编写代码输出 word.txt 中词汇表包含的反向对。

3.4 文本分析算法设计

（1）设计 Python 程序读入一个英文单词组成的文本文件，统计该文本文件中各个单词出现的次数。设计测试用例验证代码的正确性。

（2）设计 Python 程序读入一个英文单词组成的文本文件，统计其中包含的某给定关键词列表中各个单词出现的频率。设计测试用例验证代码的正确性。

4 设计与实现

4.1 Python代码理解 polygon.py

swampy.TurtleWorld 是一个画图工具库，海龟为画笔，通过控制海龟在画板上移动，来画出相应的图案。通过阅读代码，海龟的主要控制函数如下。

海龟前移	turtlename.fd(distance=1)
海龟后移	turtlename.bk(distance=1)

海龟右转	turtlename.rt(angle=90)
海龟左转	turtlename.lt(angle=90)
画笔落下	turtlename.pd()
画笔抬起	turtlename.pu()

下面对于海龟控制的方法对于 Python 面向对象设计思想的进一步思考。

4.1.1 Python 面向对象设计思想分析

在 TurtleWorld 模块之中，可以通过两类写法控制乌龟：

1. bob.fd(100)----> Turtlename.functionname() 对象.方法(形参)

2. fd(bob, 100) ----> functionname(Turtlename) 方法(形参)

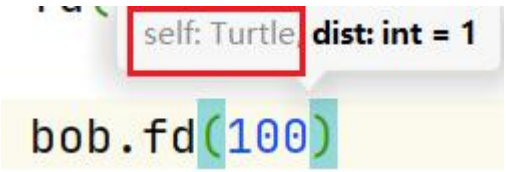
4.1.2 对象.方法(形参)

第 1 种方式易于理解，原因是 Turtle 类之中实现了 fd 方法，则由 Turtle 类创建出的对象可以使用 fd 方法。

和普通函数相比，在类中定义的成员方法有一点不同(类方法、静态方法此处不讨论)，就是第一参数永远是类的本身实例变量 self，并且调用时，不用传递该参数。例如下图(TurtleWorld.py, line 176)中的类内成员函数

```
176     def fd(self, dist=1):
177         """Moves the turtle foward by the given distance."""
178         x, y = self.x, self.y
179         p1 = [x, y]
180         p2 = self.polar(x, y, dist, self.heading)
181         self.x, self.y = p2
182 
183         # if the pen is down, draw a line
184         if self.pen and self.world.exists:
185             self.world.canvas.line([p1, p2], fill=self.pen_color)
186         self.redraw()
```

下图使用 IDE 工具验证：

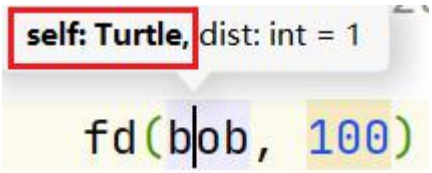


4.1.2 方法(形参)

第 2 种方式，通过阅读 TurtleWorld 模块的源码，可以找到此模块的作者在下图处(**TurtleWorld.py, line 225**)定义了全局变量，这些全局变量(fd/bk/lt...)为 Turtle 类中成员方法的引用。由于均为全局变量，因此可以在__main__函数之中直接使用，并且这些全局变量(fd/bk/lt...)的值也没有被改变，也无需再使用 global 声明

```
225     """Add the turtle methods to the module namespace
226     so they can be invoked as simple functions (not methods).
227     """
228     fd = Turtle.fd
229     bk = Turtle.bk
230     lt = Turtle.lt
231     rt = Turtle.rt
232     pu = Turtle.pu
233     pd = Turtle.pd
234     die = Turtle.die
235     set_color = Turtle.set_color
236     set_pen_color = Turtle.set_pen_color
```

下图粗体说明光标位置处对应第一个形参 `self,`形参类型为 `Turtle`



4.2 输入输出

本例之中对于输出的时间进行了异常处理，即对于明显不合理的时间，例如 29:20:10 进行处理，继续读入直至用户输入了合理的时间。采用 datetime 库之中的 timedelta、strftime、strptime 方法对于用户输入的时间字符串进行格式化处理，并将下一秒的时间输出到控制台上。

4.3 反序对

首先利用 Python 文件读写方法，将文件之中的每个单词去除换行符之后，构建集合推导式，再对集合之中的每一个单词进行遍历，将每一个单词的倒置之后，再次在集合之中查找，如果查找到则输出，并统计一共的反序对数量。

在此时实验之中，set 与 list 存储单词的效率有极大差异，小组的实验测试数据如下表

数据结构	list	set
用时	143.0031	0.026018
时间复杂度	O(n)	O(1)

查阅资料可知，在 list 之中查找一个元素时，需要从第一个元素遍历至最后一个元素，以数据结构的角度来看，时间复杂度为$O(n)$，而对于集合来说，内部数据结构为哈希表，即散列表。通过把关键码值映射到表中一个位置(index)，来加快查找的速度，时间复杂度为$O(1)$。

4.4 文本分析算法设计

调用 collections 模块之中的 counter 类对于文本文件中的单词进行读取，统计各个单词出现的次数，并且可以根据出现次数进行排序操作，并转换为键值对的字典形式进行输出。

在读如文本时，要对于特殊标点符号、换行符、分割符等特殊字符进行过滤筛除，采用列表推导式的构造方法，将此操作一并执行，如下图：

```
words_list = [word.strip(string.punctuation).lower() for word in text.split()]
words_counter = Counter(words_list)
```

5 测试用例

5.1 Python代码理解 polygon.py

海龟的移动不仅仅局限于直线，通过 Python 流程控制，在海龟前进与转向两个操作进行配合，即可画出曲线等复杂图形。

设计编写代码，画出了以下图案。

5.2 输入输出

```
请输入一个时间,格式为 (__:__:__)：29:20:10
请输入合法的时间!
请输入一个时间,格式为 (__:__:__)：23:59:59
1秒后的时间为：  00:00:00
输入合法,程序退出

Process finished with exit code 0
```

由程序输出可知，此程序对于异常输入进行了判断与处理。有较好的鲁棒性。

5.3 反序对

采用集合的存储方式，部分输出如下，未去除回文字符串时，结果如下，有较好的时间结果：

```
depots stoped
rep per
had dah
oud duo
ante etna
lotos sotol
deled deled
snoot toons
map pam
bots stob
时间开销为: 0.03398442268371582
```

5.4 文本分析算法设计

对于自选的文本，**左图**展示了任务 1 中频率最多的几个单词以及对应的次数，**右图**展示了任务 2 中给定部分关键词列表时的输出结果。

```
the 2253
i 1241
and 1230
of 1151
a 803
to 678
was 548
in 541
my 438
that 436
it 416
```

```
['time', 'machine', 'well', 'said']
time 196
machine 85
well 33
said 89
```

6 测试用例

本次实验之中，我体会到了不同数据结构之间处理不同问题之间时间复杂度的差距，感受到针对问题一定要分析之后再进行处理，否则就如同第 2 小题之中对于列表与集合的处理问题，时间开销大。

附录：

实验 1：

```
# 初始化设置
def init():
    world = TurtleWorld()
    shawn = Turtle()
    shawn.set_pen_color('#FF9900')
    shawn.delay = 1e-8
    return world, shawn


# 流汗黄豆
def emoji():
    # 脸
    radius1 = 60
    shawn.pu()
    shawn.fd(radius1)
    shawn.lt()
    shawn.pd()
    circle(shawn, radius1)

    # 嘴
    shawn.set_pen_color('#CC6633')
    radius2 = 50
    arc1 = 15
    shawn.pu()
    shawn.lt()
    shawn.fd(radius1 - radius2)
    shawn.lt()
    shawn.pd()
    arc(shawn, radius2, -180)
    shawn.rt()
    shawn.fd(radius2 * 2)
    shawn.rt()
    shawn.pu()
    arc(shawn, radius2, -arc1)
    shawn.rt(-arc1 + 90)
    shawn.pd()
    shawn.fd(2 * radius2 * math.cos(arc1 / 180 * math.pi))
    shawn.pu()
    shawn.rt(-arc1 + 90)
    arc(shawn, radius2, -arc1)
    shawn.rt()
```

```
shawn.fd(30)

# 眼
k = radius2 / 5 * 2
radius3 = 15
shawn.lt()
shawn.fd(18)
shawn.rt()
shawn.fd(radius3)
shawn.lt()
shawn.pd()
semicircle(shawn, radius3)
shawn.lt(120)
arc(shawn, 30, -60)
shawn.pu()
arc(shawn, 30, -300)
shawn.rt(120)

shawn.lt()
shawn.fd(radius3 * 2 + k * 3 - 20)
shawn.lt()
shawn.pd()
semicircle(shawn, radius3)
shawn.lt(120)
arc(shawn, 30, -60)
shawn.pu()
arc(shawn, 30, -300)
shawn.rt(120)

# 汗
shawn.set_pen_color('#0099FF')
shawn.lt()
shawn.fd(30)
shawn.lt()
shawn.fd(40)

arc2 = 15
radius4 = 40
shawn.lt(180 - arc2)
shawn.pd()
shawn.fd(radius4)
shawn.lt(arc2)
semicircle(shawn, radius4 * math.sin(math.pi * arc2 / 180))
shawn.lt(arc2)
```

```
    shawn.fd(radius4)

    # 爬
    shawn.pu()
    shawn.fd(2000)


if __name__ == '__main__':
    world, shawn = init()
    world.setup_interactive()
    emoji()
    wait_for_user()
```

实验 2：

```
import datetime

if __name__ == '__main__':
    while True:
        try:
            input_time_raw = input("请输入一个时间,格式为 (__:__:__)：")
            input_time_processed = datetime.datetime.strptime(input_time_raw, "%H:%M:%S")
            input_time_processed += datetime.timedelta(seconds=1)
            print("1 秒后的时间为：", input_time_processed.strftime("%H:%M:%S"))
            print("输入合法,程序退出")
            break
        except Exception:
            print("请输入合法的时间！")
# 23:44:20
# 23:59:59
# 29:20:10
```

实验 3：

```
import time


def set_test():
    with open('words.txt', 'r') as f:
        # 文件中每个单词除去换行符之后存入集合
        words_set = {line.strip('\n') for line in f.readlines()}

    # 集合推导式构造逆序集合
    reversed_words_set = {word[::-1] for word in words_set}

    start = time.time()
```

```
    count = 0
    # 集合内元素 Hash 存储,查找时间复杂度 O(1)
    for word in words_set:
        if word in reversed_words_set:
            print(word, word[::-1])
            count+=1

    end = time.time()
    print("时间开销为: {} ".format(end - start))

    print("--------------------")
    print(count)

def list_test():
    with open('words.txt', 'r') as f:
        # 文件中每个单词除去换行符之后存入列表
        words_list = [line.strip('\n') for line in f.readlines()]

    reversed_words_list = [word[::-1] for word in words_list]

    start = time.time()

    for word in words_list:
        if word in reversed_words_list:
            print(word, word[::-1])
    end = time.time()

    start = time.time()

    print("时间开销为: {} ".format(end - start))


if __name__ == '__main__':
    set_test()
    list_test()
```

实验 4：

```
from collections import Counter
import string


def task1():
    with open('timemachine.txt', 'r') as f:
        text = f.read()
```

```
    words_list = [word.strip(string.punctuation).lower() for word in text.split()]
    words_counter = Counter(words_list)

    for k, v in words_counter.most_common():
        print(k, v)


def task2():
    with open('timemachine.txt', 'r') as f:
        text = f.read()

    words_list = [word.strip(string.punctuation).lower() for word in text.split()]
    words_counter = Counter(words_list)

    hot_words = ['time', 'machine', 'well', 'said']
    print(hot_words)
    for word in hot_words:
        print(word, words_counter.get(word))


if __name__ == '__main__':
    task1()
    task2()
```