

Python编程基础

实验指导手册

前言

简介

掌握本实验手册内容，您将能够进行基础 Python 编程。

内容描述

本实验指导书共包含三个实验，分别是 Python 基础编程、Python 高级和 Python 实现的资产管理系统，希望学员或者读者能够熟悉 Python 语言，具备 Python 基础编程能力。

读者知识背景

为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 了解计算机编程。

实验环境说明

Python 开发工具：

本实验环境是基于 Python3.6 环境开发编译的。

目录

简介	2
内容描述	2
读者知识背景	2
实验环境说明	2
1 Python 编程基础.....	5
1.1 实验介绍	5
1.1.1 关于本实验.....	5
1.1.2 实验目的.....	5
1.2 实验代码	5
1.2.1 Hello world	5
1.2.2 数据类型：数值	5
1.2.3 数据类型：字符串	6
1.2.4 数据类型：列表	6
1.2.5 数据类型：元组	7
1.2.6 数据类型：字典	8
1.2.7 数据类型：集合	8
1.2.8 深拷贝和浅拷贝	9
1.2.9 if 语句	9
1.2.10 循环语句	9
1.2.11 自定义函数	10
1.2.12 面向对象	11
1.2.13 标准库的使用	14
1.2.14 IO 操作	17
2 Python 高级.....	19
2.1 实验介绍	19
2.1.1 关于本实验.....	19
2.1.2 实验目的.....	19

2.2 实验代码	19
2.2.1 数据库编程.....	19
2.2.2 多任务	26
2.2.3 迭代器、生成器和装饰器.....	28
2.2.4 正则表达式.....	31
3 资金管理系统.....	34
3.1 实验介绍	34
3.1.1 关于本实验.....	34
3.1.2 实验目的.....	34
3.2 实验代码	34
3.2.1 实验思路.....	34
3.2.2 实验实现.....	34

1 Python 编程基础

1.1 实验介绍

1.1.1 关于本实验

本实验介绍了 Python (Python3) 语言的基础，帮助读者快速掌握 Python 语言中的基本数据类型、Python 编程语言的基本语法、Python 面向对象编程和 Python 的文件操作。

1.1.2 实验目的

通过以下小的实验可以帮助我们掌握 Python 这门编程语言，为以后的 AI 实验打下基础。

1.2 实验代码

1.2.1 Hello world

第一个 Python 程序，打印 hello world。

```
print('hello world') #打印出: hello world  
print("hello world") #打印出: hello world。单双引号输出相同。
```

1.2.2 数据类型：数值

熟悉 Python 中数值的基本运算。注意，Python 中的“与或非”布尔操作不是使用操作符，而是使用关键词 and/or/not。

```
print(True+False)      # 输出 1, True 默认为 1, False 为 0  
print(True or False)  # 输出 True, 关键字 or 执行“或”操作  
print(5//2)           # 输出 2, //为取整运算符  
print(5%2)            # 输出 1, %为取余运算符  
print(3**2)           # 输出 9, **表示乘方操作  
print(5+1.6)          # 输出 6.6, 不同精度的类型的数字相加默认取高精度类型作为结果
```

1.2.3 数据类型：字符串

步骤 1 字符串的基本操作。

```
S = 'python'                                # 给变量 S 赋值 python
# len(obj)：返回对象的长度
print(len(S))                               # 输出 6
print(S[0],S[1],S[-1])                      # 输出 pyn，按照索引获取元素
print(S+'1',S*2)                            # 输出 python1 pythonpython：合并和重复
```

步骤 2 字符串的不可变性。

```
S = 'python'                                # 变量赋值
S[0] = 'z'                                    # 程序异常
S1 ='z'+S[1:]                                # 生成了新的字符串 zython，并赋值给 S1
print("S:%s,S1:%s"%(S,S1))                  # 输出 S:python, S1:Zython
```

步骤 3 字符串的常用操作。

```
S = "python"                                # 变量赋值
# str.split(str="", num=-1)：通过指定分隔符对字符串进行切片，如果参数 num 有指定值，则分隔 num+1 个子字符串，-1 表示分割所有。
print(S.split('h'))                          # 输出 ['pyt', 'on']，根据 h 对字符串切割
# str.replace(old, new[, max])：返回字符串中的 old (旧字符串) 替换成 new(新字符串) 后生成的新字符串，如果指定第三个参数 max，则替换不超过 max 次。
print(S.replace('py', 'PY'))                 # Python，将字符串中的 py 替换为 PY
# str.upper()：返回小写字符转化为大写后的值。
print(S.upper())                            # PYTHON
# str.lower()：返回大写字符转化为小写后的值。
print('PYTHON'.lower())                     # python，字符串转小写
line='aa,bb,ccc,dd\n'                        # \n 为换行
# str.join(sequence)：sequence：要连接的序列，返回指定字符连接序列中元素后生成的新字符串。
print('.'.join(['life', 'is', 'short']))    # 输出 life is short，join 拼接字符串
hw12='%s %s %d' % ('hello','world',12)     # 格式化字符串
print(hw12)                                 # 输出 hello world 12
```

1.2.4 数据类型：列表

列表的常用操作：

```
animals = ['cat', 'dog', 'monkey']
# list.append(obj)：在列表末尾添加新的对象。
animals.append('fish')                      # 追加元素
```

```

print(animals)           # 输出 ['cat', 'dog', 'monkey', 'fish']

# list.remove(obj): 移除列表中某个值的第一个匹配项。
animals.remove('fish')   # 删除元素 fish

print(animals)           # 输出 ['cat', 'dog', 'monkey']

# list.insert(index, obj): 用于将指定对象插入列表的指定位置。index: 插入位置
animals.insert(1, 'fish') # 在下标 1 的地方插入元素 fish

print(animals)           # 输出 ['cat', 'fish', 'dog', 'monkey']

# list.pop([index=-1]): 要移除列表中对下标对应的元素（默认是最后一个）。Index: 下标
animals.pop(1)           # 删除下标为 1 的元素

print(animals)           # 输出 ['cat', 'dog', 'monkey']

# 遍历并获取元素和对应索引
# enumerate(sequence) : 将一个可遍历的数据对象组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。
for i in enumerate(animals):
    print(i)             # 元素下标和元素所组成的索引

输出: (0, cat)
      (1, dog)
      (2, monkey)

# 列表推导式
squares = [x*x for x in animals] # 批量生成符合规则的元素组成的列表
print(squares)                  # ['catcat ', 'dogdog ', 'monkeymonkey ']

list1 = [12, 45, 32, 55]
# list.sort(cmp=None, key=None, reverse=False): cmp 为可选参数，如果指定了该参数，会使用该参数的方法进行排序。key 是用来进行比较的元素。reverse 为排序规则，False 为升序。
list1.sort()                   # 对列表进行排序

print(list1)                   # 输出 [12, 32, 45, 55]

# list.reverse(): 反向列表中元素。
list1.reverse()                # 对列表进行逆置

print(list1)                   # 输出 [55, 45, 32, 12]

```

1.2.5 数据类型：元组

元祖的常用操作。

```

T=(1,2,3)                   # 创建元组

print(T+(4,5))              # 元组合并，输出: (1, 2, 3, 4, 5)

t=(42,)                      # 只有一个元素的元组，区别于数字

tuple1 = (12,45,32,55,[1,0,3]) # 创建元组

tuple1[0] = "good"            # 程序异常，元组的不可变性

tuple1[4][0] = 2               # 元组中可变的元素是可以变得

print(tuple1)                 # (12, 45, 32, 55, [2, 0, 3])

```

1.2.6 数据类型：字典

字典的常用操作：

```
# 字典的三种赋值操作
x = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
x = dict(food='Spam', quantity=4, color='pink')
x = dict([("food", "Spam"), ("quantity", "4"), ("color", "pink")])
# dict.copy(): 拷贝数据
d = x.copy()
d['color'] = 'red'
print(x)                                # {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
print(d)                                # {'food': 'Spam', 'quantity': 4, 'color': 'red'}
```

#元素访问

```
print(d['name'])                         # 得到错误信息
print(d.get('name'))                      # 输出 None
print(d.get('name', '键值不存在! '))       # 输出 键值不存在
print(d.keys())                           # 输出 dict_keys(['food', 'quantity', 'color'])
print(d.values())                         # 输出 dict_values(['Spam', 4, 'red'])
print(d.items())                          # 输出 dict_items([('food', 'Spam'), ('quantity', 4), ('color', 'red')))
d.clear()                                # 清空字典中的所有数据
print(d)                                  # 输出 {}
del(d)                                    # 删除字典
print(d)                                # 程序异常, 提示 "d" 未定义
```

1.2.7 数据类型：集合

集合的常用操作：

```
sample_set = {'Prince', 'Techs'}
print('Data' in sample_set)                # 输出 False, in 的作用是检查集合中是否存在某一元素
# set.add(obj): 给集合添加元素, 如果添加的元素在集合中已存在, 则不执行任何操作。
sample_set.add('Data')                    # 向集合中增加元素 Data
print(sample_set)                        # 输出 {'Prince', 'Techs', 'Data'}
print(len(sample_set))                  # 输出 3
# set.remove(obj): 移除集合中的指定元素。
sample_set.remove('Data')                # 删除元素 Data
print(sample_set)                        # {'Prince', 'Techs'}
list2 = [1, 3, 1, 5, 3]
print(list(set(list2)))                 # 输出 [1, 3, 5], 利用集合元素的唯一性进行列表去重
sample_set = frozenset(sample_set) # 不可变集合
```

1.2.8 深拷贝和浅拷贝

使用 Python 中的 copy 模块来实现深拷贝的功能。

```
import copy
Dict1 = { 'name': 'lee', 'age':89, 'num':[1,2,8]}      # 新建字典
Dict_copy = Dict1.copy()          # 浅拷贝
Dict_dcopy = copy.deepcopy(Dict1) # 深拷贝
Dict1['num'][1] = 6            # 修改原数据中嵌套列表的值
print('Dict1:' + str(Dict1) + "\n",' Dict_copy:' + str(Dict_copy) + "\n",' Dict_dcopy:' +
str(Dict_dcopy))

输出结果：
Dict1:{'name':'lee', 'age':89, 'num':[1,6,8]}
Dict_copy :{'name':'lee', 'age':89, 'num':[1,6,8]}          # 浅拷贝数据一起被修改
Dict_dcopy :{'name':'lee', 'age':89, 'num':[1,2,8]}        # 深拷贝没有修改
```

1.2.9 if 语句

接收一个用户输入的分数，然后判断用户所输入的分数属于什么级别。使用 Python 中的 if 语句可以完成此功能。

```
#根据输入的分数判断
# input(): 用于接收输入。
score = input("请输入你的分数")                      # input 函数接收输入，为字符串类型
score = float(score)                                  # 将分数转化为数字类型
# try:... except Exception:... 是 Python 中用于捕获异常的语句，如果 try 中的语句出现错误，则会
# 执行 except 中的语句。
try:
    if 100>=score>=90:                                # 判断输入的值是否大于等级分数
        print("优")                                     # 满足条件后输出等级
    elif 90 > score >= 80:
        print("良")
    elif 80>score>0:
        print("中")
    else:
        print("差")
except Exception:
    print("请输入正确的分数")
```

1.2.10 循环语句

步骤 1 for 循环

使用 for 循环打印九九乘法表。

```
for i in range(1,10): # 定义外层循环
    for j in range(1,i+1):# 定义内层循环
        # 字符串的格式化输出，让打印结果进行对齐，end 属性设置打印结尾符号默认为/n
        print("%d*%d=%2d"%(i,j,i*j), end=" ")
    print()
```

输出结果：

```
1*1= 1
2*1= 2 2*2= 4
3*1= 3 3*2= 6 3*3= 9
4*1= 4 4*2= 8 4*3=12 4*4=16
5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25
6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

步骤 2 while 循环

当满足条件时循环执行语句块，想要结束循环时，使用 break 或 continue 结束循环。

```
#while 循环
i = 0                                     # 新建 i 变量
while i<9:                                # 设置循环条件
    i+=1                                    # 每次循环 i 增加 1
    if i == 3:                             # 判断条件是否满足
        print("跳出此次循环")
        continue                            # continue 跳出当前的这一次循环
    if i == 5:
        print("跳出当前大的循环")
        break                               # 跳出当前的大的循环
    print(i)
```

输出结果：

```
1
2
跳出此次循环
4
跳出当前大的循环
```

1.2.11 自定义函数

步骤 1 定义函数

自定义一个函数，返回一个序列。序列中每个数字都是前两个数字之和（斐波那契数列）。

```
def fibs(num):                                # 位置参数
    result = [0,1]                                # 新建列表存储数列的值
    for i in range(2,num):                         # 循环 num-2 次
        a = result[i-1] + result[i-2]
        result.append(a)                            # 将值追加至列表
    return result                                  # 返回列表

fibs(5)
```

输出结果：

```
[0, 1, 1, 2, 3]
```

步骤 2 参数

自定义函数，打印不同方式传入的参数。

```
def hello(greeting='hello',name='world'):      # 默认参数
    print('%s, %s!' % (greeting, name))        # 格式化输出

hello()                                         # hello, world  默认参数
hello('Greetings')                            # Greetings, world  位置参数
hello('Greetings','universe')                 # Greetings, universe  位置参数
hello(name='Gumby')                           # hello, Gumby  关键字参数
```

输出结果：

```
hello, world!
Greetings, world!
Greetings, universe!
hello, Gumby!
```

1.2.12 面向对象

步骤 1 创建和使用类

创建 Dog 类。

根据 Dog 类创建的每个实例都将存储名字和年龄。我们将赋予了每条小狗蹲下 (sit ()) 和打滚 (roll_over ()) 的能力：

```
class Dog():
    """一次模拟小狗的简单尝试"""
    def __init__(self, name, age):
        """初始化属性 name 和 age"""
        self.name = name
        self.age = age
    def sit(self):
        """模拟小狗被命令时蹲下"""

    def roll_over(self):
        """模拟小狗打滚"""

    def __str__(self):
        return "%s is %d years old." % (self.name, self.age)
```

```

        print(self.name.title()+"is now sitting")
    def roll_over(self):
        """模拟小狗被命令时打滚"""
        print(self.name.title()+"rolled over!")

```

实例化类

```

dog = Dog("哈士奇", 2)
dog.sit()
dog.roll_over()

```

输出结果::

```

哈士奇 is now sitting
哈士奇 rolled over!

```

步骤 2 访问属性

```

class Employee:
    '所有员工的基类'
    empCount = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
    def displayCount(self):
        print("Total Employee %d" % Employee.empCount )
    def displayEmployee(self):
        print("Name : ", self.name, ", Salary: ", self.salary)
# 创建 Employee 类的第一个对象"
emp1 = Employee("Zara", 2000)
# 创建 Employee 类的第二个对象"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)

```

输出结果:

```

Name : Zara ,Salary: 2000
Name : Manni ,Salary: 5000
Total Employee 2

```

步骤 3 类的继承

面向对象的编程带来的主要好处之一是代码的重用, 实现这种重用的方法之一是通过继承机制。继承完全可以理解成类之间的类型和子类型关系。

```

class Parent:      # 定义父类
    parentAttr = 100
    def __init__(self):

```

```

print("调用父类构造函数")
def parentMethod(self):
    print('调用父类方法')
def setAttr(self, attr):
    Parent.parentAttr = attr
def getAttr(self):
    print("父类属性 :", Parent.parentAttr)

class Child(Parent): # 定义子类
    def __init__(self):
        print("调用子类构造方法")
    def childMethod(self):
        print('调用子类方法')

c = Child()          # 实例化子类
c.childMethod()      # 调用子类的方法
c.parentMethod()    # 调用父类方法
c.setAttr(200)       # 再次调用父类的方法 - 设置属性值
c.getAttr()          # 再次调用父类的方法 - 获取属性值

```

输出结果：

```

调用子类构造方法
调用子类方法
调用父类方法
父类属性 : 200

```

步骤 4 类属性和方法

```

class JustCounter:
    __secretCount = 0 # 私有变量
    publicCount = 0   # 公开变量
    def count(self):
        self.__secretCount += 1
        self.publicCount += 1
        print(self.__secretCount)
counter = JustCounter()
counter.count()
counter.count()
print(counter.publicCount)
print(counter.__secretCount) # 报错, 实例不能访问私有变量

```

输出结果：

```

1
2
2

```

```
-----  
AttributeError Traceback (most recent call last)  
<ipython-input-5-5dff96b3b703> in <module>  
    10 counter.count()  
    11 print(counter.publicCount)  
--> 12 print(counter.__secretCount) # 报错，实例不能访问私有变量  
    13  
    14  
  
AttributeError: 'JustCounter' object has no attribute '__secretCount'
```

图 1-1 报错信息

1.2.13 标准库的使用

步骤 1 sys

sys.exit([n]): 此方法可以是当前程序退出, n 为 0 时表示正常退出, 其他值表示异常退出。

```
import sys  
for i in range(100):  
    print(i)  
    if i ==5:  
        sys.exit(0)
```

输出结果:

```
0  
1  
2  
3  
4  
5  
An exception has occurred, use %tb to see the full traceback.
```

sys.path: 获取模块搜索路径。

```
sys.path
```

输出结果:

```
['D:\\\\python3.6\\\\python36.zip',  
 'D:\\\\python3.6\\\\DLLs',  
 'D:\\\\python3.6\\\\lib',  
 'D:\\\\python3.6',  
 '',  
 'D:\\\\python3.6\\\\lib\\\\site-packages',  
 'D:\\\\python3.6\\\\lib\\\\site-packages\\\\IPython\\\\extensions',  
 'C:\\\\Users\\\\xxx\\\\.ipython']
```

sys.platform: 获取当前系统平台。

```
sys.platform
```

输出结果：

```
'win32'
```

sys.argv：从程序外部向程序传递参数，参数以列表的形式传递，第一个为当前文件名。

新建 py 文件 test.py（在当前文件夹下或者桌面），写入以下代码：

```
print(sys.argv[1])
```

在命令行中切换到文件路径，运行程序：

```
python test.py hello
```

输出结果：



```
C:\Users\...esktop>python test.py hello  
这是在程序外部接收的参数 hello
```

图 1-2 外部接收参数

步骤 2 os

```
import os  
  
# os.getpid()：获取当前进程 id  
print("当前进程的 ID: ", os.getpid())  
  
# os.getppid()：获取当前父进程 id  
print("当前父进程的 ID: ", os.getppid())  
  
# os.getcwd()：获取当前所在路径  
cwd = os.getcwd()  
print("当前所在路径为: ", cwd)  
  
# os.chdir(path)：改变当前工作目录  
os.chdir("C:\\\\")  
print("修改后当前所在路径为: ", os.getcwd())  
  
# os.listdir()：返回目录下所有文件  
print("当前目录下的文件有: ", os.listdir(cwd))  
  
# os.walk()：输出当前路径下的所有文件  
for root, dirs, files in os.walk(cwd, topdown=False):  
    for name in files:  
        print(os.path.join(root, name))  
    for name in dirs:  
        print(os.path.join(root, name))
```

输出结果：

```
当前进程的ID: 11004
当前父进程的ID: 9012
当前所在路径为: D:\python project\HCIA - AI实验
修改后当前所在路径为: C:\\
当前目录下的文件有: ['.ipynb_checkpoints', 'ai.py', 'peiqi.py', 'python实验.ipynb', 'text.txt', '语音识别']
D:\python project\HCIA - AI实验\.ipynb_checkpoints\python实验-checkpoint.ipynb
D:\python project\HCIA - AI实验\语音识别\voice_dataset.zip
D:\python project\HCIA - AI实验\语音识别\yuyinchall.py
D:\python project\HCIA - AI实验\语音识别\yuyinutils.py
D:\python project\HCIA - AI实验\ai.py
D:\python project\HCIA - AI实验\peiqi.py
D:\python project\HCIA - AI实验\python实验.ipynb
D:\python project\HCIA - AI实验{text.txt}
D:\python project\HCIA - AI实验\.ipynb_checkpoints
D:\python project\HCIA - AI实验\语音识别
```

图 1-3 os 模块执行结果

os.path 模块：主要用于获取文件的属性。

```
import os
# os.path.abspath(path): 返回绝对路径
print("text.txt 的绝对路径为: ",os.path.abspath("text.txt")) # text.txt 为当前文件夹下的一个文件 (上一个实验中将当前路径切换为 C:\, 需要将路径切换回来)

# os.path.exists(path): 文件存在则返回 True, 不存在返回 False
print("text.txt 是否存在: ",os.path.exists("text.txt"))

# os.path.getsize(path): 返回文件大小, 如果文件不存在就返回错误
print("text.txt 的文件大小: ",os.path.getsize("text.txt"))

# os.path.isfile(path): 判断路径是否为文件
print("text.txt 是否为文件: ",os.path.isfile("text.txt"))

# os.path.isdir(path): 判断路径是否为文件夹
print("text.txt 是否为文件夹: ",os.path.isdir("text.txt"))

输出结果:
text.txt 的绝对路径为: D:\python project\text.txt
text.txt 是否存在: True
text.txt 的文件大小: 0
text.txt 是否为文件: True
text.txt 是否为文件夹: False
```

步骤 3 time

```
import time
# time.time(): 用于获取当前时间戳
time_now = time.time()
print("时间戳: ",time_now)
```

```

# time.localtime(): 获取时间元组
localtime = time.localtime(time_now)
print("本地时间为 :", localtime)

# time.asctime(): 获取格式化的时间
localtime = time.asctime(localtime)
print("本地时间为 :", localtime)

# time.strftime(format[, t]): 接收时间元组，并返回以可读字符串表示的当地时间，格式由参数
# format 决定。
print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
输出结果：

时间戳: 1555950340.4777014
本地时间为 : time.struct_time(tm_year=2019, tm_mon=4, tm_mday=23, tm_hour=0,
tm_min=25, tm_sec=40, tm_wday=1, tm_yday=113, tm_isdst=0)
本地时间为 : Tue Apr 23 00:25:40 2019
2019-04-23 00:25:40

```

1.2.14 IO 操作

步骤 1 文件写入：

```

f = open("text.txt", 'w') # 打开文件 text.txt, 当文件不存在时会新建一个。
Str = input("请输入要写入的内容: ")
f.write(Str)
f.close()

```

输出结果：

请输入要写入的内容: **python文件操作**

图 1-4 写入的内容



图 1-5 文件内容

步骤 2 文件读取

```
f = open("text.txt", 'r')
```

```
print(f.read(6))    # 读取六个字符，当前光标后移六个字符  
print(f.read())     # 读取光标所在位置至最后  
f.close()
```

输出结果：

```
python  
文件操作
```

步骤 3 使用上下文管理器操作文件

```
# 使用 with 语句进行文件写入  
with open("text1.txt", 'w') as f:  
    f.write("python 文件操作")  
  
# 使用 with 语句读取文件内容  
with open("text1.txt", 'r') as f:  
    print(f.read())
```

输出结果：

```
python 文件操作
```

步骤 4 使用 os 模块操作文件

```
import os  
os.rename("text.txt", "text0.txt") # 文件重命名  
os.remove("text1.txt")      # 删除文件
```

2 Python 高级

2.1 实验介绍

2.1.1 关于本实验

本实验介绍了 Python 高级部分的实验，包括了数据库编程、多任务、迭代器、生成器、装饰器和正则表达式。

2.1.2 实验目的

通过以下的小实验进一步理解和掌握 Python 这一门编程语言。

2.2 实验代码

2.2.1 数据库编程

2.2.1.1 安装数据库

步骤 1 在官网下载安装包 <https://dev.mysql.com/downloads/mysql/>：

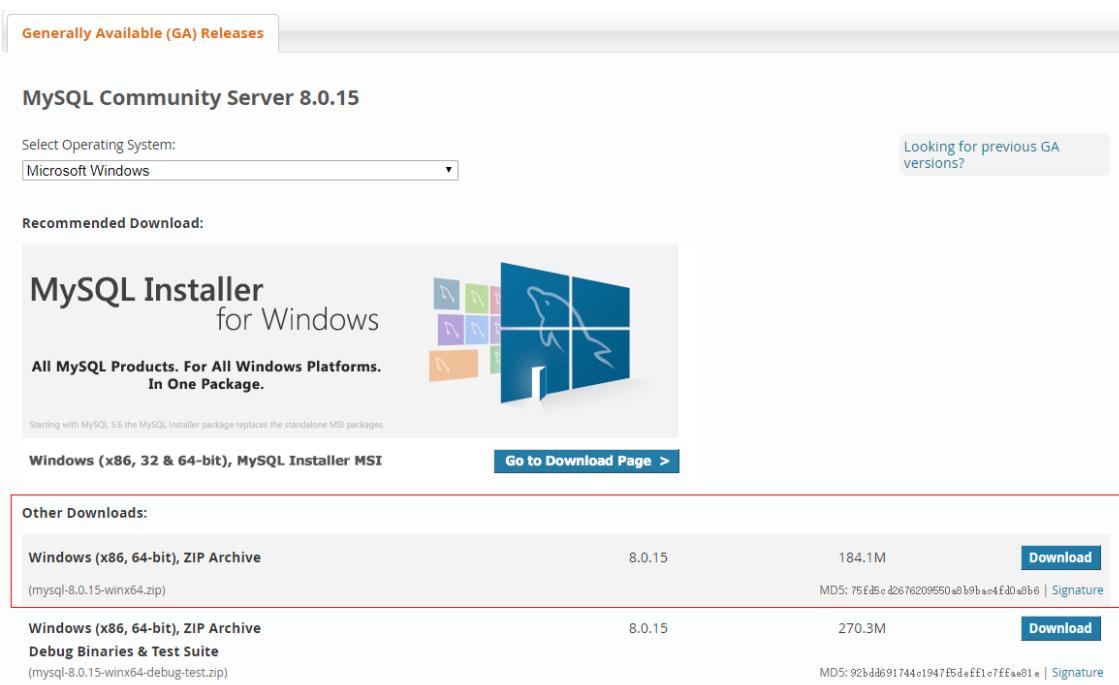


图 2-1 下载 MySQL

下载完后，我们将 zip 包解压到相应的目录（自己选择目录）。

打开刚刚解压的文件，在该文件夹下创建 my.ini 配置文件，编辑 my.ini 配置以下基本信息：

```
[mysql]
# 设置 mysql 客户端默认字符集
default-character-set=utf8

[mysqld]
# 设置 3306 端口
port = 3306
# 设置 mysql 的安装目录，将下面的##换成安装路径
basedir=##
# 设置 mysql 数据库的数据的存放目录
# datadir=##
# 允许最大连接数
max_connections=20
# 服务端使用的字符集默认为 8 比特编码的 latin1 字符集
character-set-server=utf8
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
```

步骤 2 启动 MySQL

以管理员身份打开命令行，切换到解压的文件夹下，进入 bin 目录。

```
mysqld --initialize -console
```

初始化后或获得一个随机密码，形如：root@localhost:随机密码。

使用命令进行安装：

```
mysqld install
```

启动 mysql：

```
net start mysql
```

登录 mysql：

```
mysql -uroot -p
```

修改密码：

```
set password for 用户名@localhost = password('新密码')
```

步骤 3 配置环境变量

右键我的电脑，打开属性，点击高级系统设置：

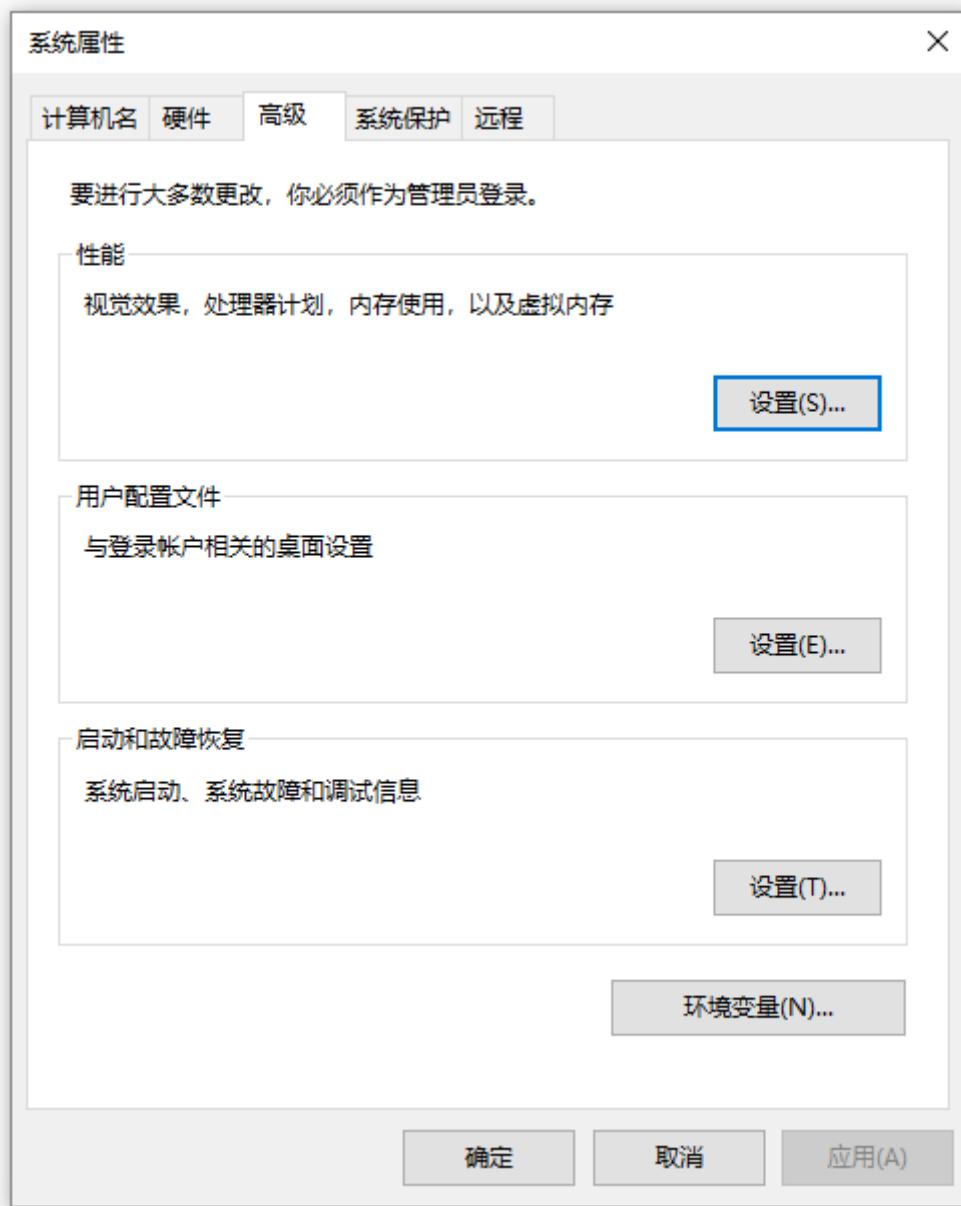


图 2-2 系统属性

点击环境变量。在系统变量中的 path 里面添加数据库的路径。

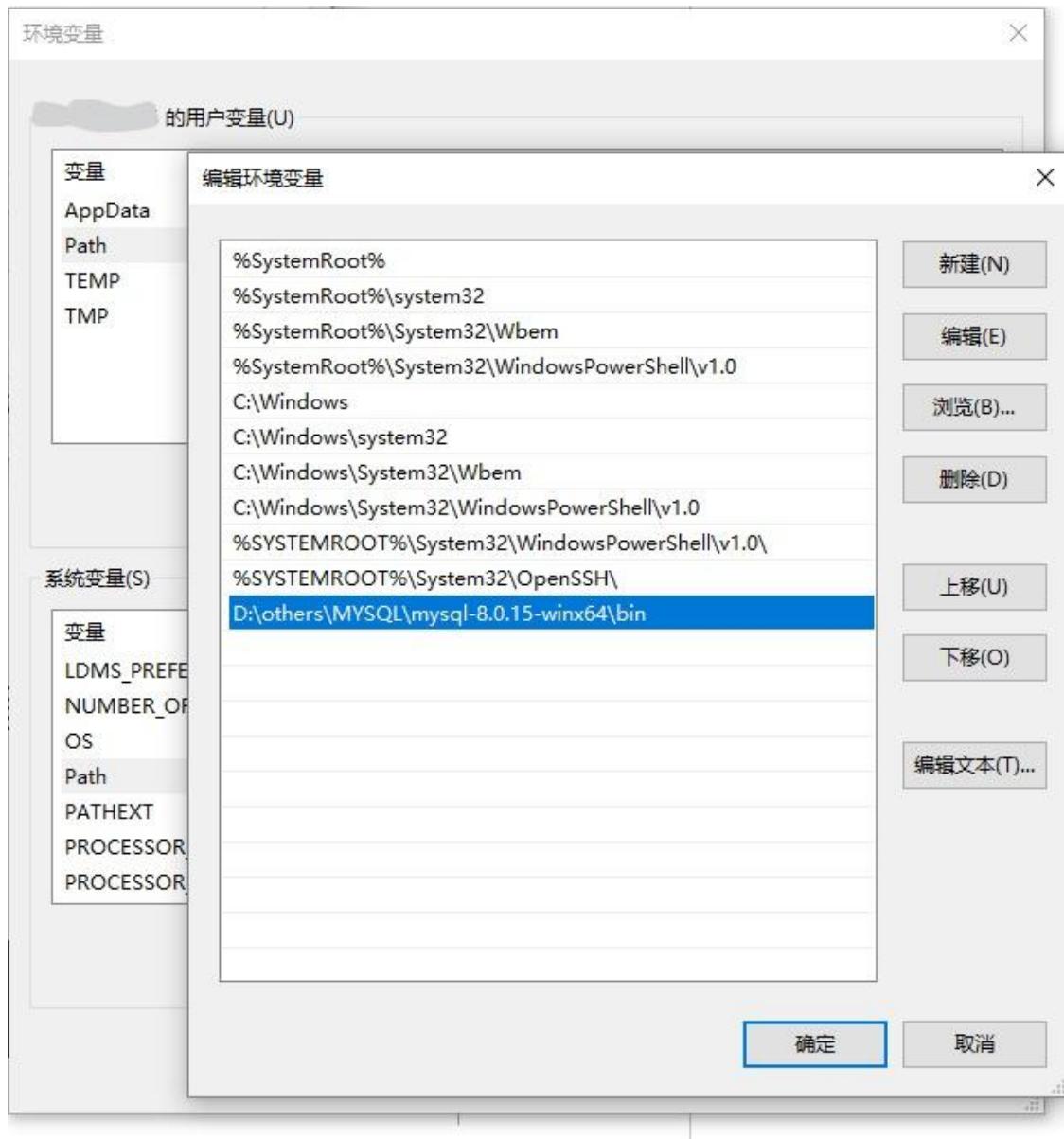


图 2-3 配置环境变量

2.2.1.2 创建数据库

新建数据库 my_database:

```
create database my_database;  
mysql> create database my_database;  
Query OK, 1 row affected (0.09 sec)
```

图 2-4 创建数据库

切换数据库:

```
use my_database;
```

```
mysql> use my_database;
Database changed
```

图 2-5 切换数据库

2.2.1.3 数据库编程

步骤 1 安装数据库所需模块

使用 pip 命令在命令行为 Python 安装第三方库：pip install 库。

```
pip install pymysql
```

步骤 2 连接数据库

```
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost", "root", "mysql", "my_database", charset='utf8' )
# localhost: 本地连接, 也可以换成数据库所在的 ip 地址
# root: mysql 数据库的账户; mysql: 数据库密码
# my_database: 要连接的数据库名称
# 使用 cursor() 方法获取操作游标
cursor = db.cursor()
# 使用 execute 方法执行 SQL 语句
cursor.execute("SELECT VERSION()")
# 使用 fetchone() 方法获取一条数据
data = cursor.fetchone()
print("Database version : %s " % data)
```

输出结果：

```
Database version : 8.0.15
```

步骤 3 使用 Python 操作数据库

```
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost", "root", "mysql", "my_database", charset='utf8' )
# 使用 cursor() 方法获取操作游标
cursor = db.cursor()
# 创建数据表 SQL 语句
sql = """CREATE TABLE my_table (
    id int,
    name varchar(50))"""
# 执行
cursor.execute(sql)
# 关闭数据库连接
db.close()
```

在 mysql 中查看结果：

```
show tables;
mysql> show tables;
+---------------------+
| Tables_in_my_database |
+-----+
| my_table |
+-----+
1 row in set (0.00 sec)
```

图 2-6 查看 mysql 中的数据库

步骤 4 数据操作

插入数据：

```
import pymysql

# 打开数据库连接
db = pymysql.connect("localhost", "root", "mysql", "my_database", charset='utf8' )
# 使用 cursor() 方法获取操作游标
cursor = db.cursor()

# SQL 插入语句
sql = """INSERT INTO my_table(id,
                               name)
          VALUES (1, '张三')"""\n      # 删除、修改等操作只需要修改执行的 SQL 语句即可
try:
    # 执行 sql 语句
    cursor.execute(sql)
    # 提交到数据库执行
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

在数据库中查看结果：

```
select * from my_table;
```

图 2-7 查看插入的数据

```
mysql> select * from my_table;
+----+-----+
| id | name |
+----+-----+
| 1  | 张三 |
+----+-----+
1 row in set (0.00 sec)
```

2.2.2 多任务

步骤 1 多线程

使用多线程执行任务：

```
import threading
from time import sleep, ctime
def work1():
    for i in range(3):
        print("work1 正在执行...%d"%i)
        sleep(1)

def work2():
    for i in range(3):
        print("work2 正在执行...%d"%i)
        sleep(1)

if __name__ == '__main__':
    print('---开始---:%s'%ctime())

    t1 = threading.Thread(target=work1) # 线程 1
    t2 = threading.Thread(target=work2) # 线程 2
    # 启动线程
    t1.start()
    t2.start()

    sleep(5)
    print('---结束---:%s'%ctime())
```

输出结果：

```
---开始---:Mon Apr 15 10:55:16 2019
work1 正在执行...0
work2 正在执行...0
work1 正在执行...1
work2 正在执行...1
work1 正在执行...2
```

```
work2 正在执行...2
---结束---:Mon Apr 15 10:55:21 2019
```

线程同步:

```
import threading
import time
g_num = 0
def test1(num):
    global g_num    # 使用全局变量
    for i in range(num):
        mutex.acquire()  # 上锁
        g_num += 1
        mutex.release()  # 解锁
    print("----test1---g_num=%d"%g_num)

def test2(num):
    global g_num
    for i in range(num):
        mutex.acquire()  # 上锁
        g_num += 1
        mutex.release()  # 解锁

    print("----test2---g_num=%d"%g_num)

# 创建一个互斥锁
# 默认是未上锁的状态，可以删除锁后查看资源争夺的结果
mutex = threading.Lock()

# 创建 2 个线程，让他们各自对 g_num 加 1000000 次
p1 = threading.Thread(target=test1, args=(1000000,))
p1.start()

p2 = threading.Thread(target=test2, args=(1000000,))
p2.start()

# 等待计算完成
time.sleep(5)

print("2 个线程对同一个全局变量操作之后的最终结果是:%s" % g_num)
```

输出结果:

```
----test2---g_num=1971982----test1---g_num=2000000
```

```
2 个线程对同一个全局变量操作之后的最终结果是:2000000
```

步骤 2 多进程

```
from multiprocessing import Process
import os
import time

nums = [11, 22]

def work1():
    """子进程要执行的代码"""
    print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums)) # 获取进程号
    for i in range(3):
        nums.append(i)
        time.sleep(1)
        print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums))

def work2():
    """子进程要执行的代码"""
    print("in process2 pid=%d ,nums=%s" % (os.getpid(), nums))

if __name__ == '__main__':
    p1 = Process(target=work1)
    p1.start()
    p1.join()

    p2 = Process(target=work2)
    p2.start()
```

输出结果：

```
in process1 pid=9932 ,nums=[11, 22]
in process1 pid=9932 ,nums=[11, 22, 0]
in process1 pid=9932 ,nums=[11, 22, 0, 1]
in process1 pid=9932 ,nums=[11, 22, 0, 1, 2]
in process2 pid=13524 ,nums=[11, 22]
# 注 因为编辑器的缘故，jupyter notebook 不会输出本实验的结果。可以新建一个 py 文件，将代码写入后在命令行执行。
```

2.2.3 迭代器、生成器和装饰器

2.2.3.1 迭代器

步骤 1 可迭代对象

```
# 使用 isinstance() 方法判断对象是否是可迭代对象
from collections import Iterable # 可迭代对象
print(isinstance([], Iterable))
print(isinstance('abc', Iterable))
```

```
print(isinstance(100, Iterable))
```

输出结果：

```
True  
True  
False
```

步骤 2 迭代器

通过 `iter()` 函数获取这些可迭代对象的迭代器。然后我们可以对获取到的迭代器不断使用 `next()` 函数来获取下一条数据。

```
l = [1, 2, 3, 4, 5]  
l_iter = iter(l)  
next(l_iter)  
>>>1  
next(l_iter)  
>>>2
```

步骤 3 判断是否是迭代器

```
from collections import Iterator # 迭代器  
print(isinstance([], Iterator))  
print(isinstance(iter([]), Iterator))
```

输出结果：

```
False  
True
```

2.2.3.2 生成器

步骤 1 列表推导式的形式创建生成器

```
G = (x*2 for x in range(5))  
print(type(G))
```

输出结果：

```
<class 'generator'>
```

步骤 2 使用 `yield` 创建生成器

使用 `yield` 关键字创建一个生成器，用以生成斐波那契数列。

```
def fib(n):  
    current = 0  
    num1, num2 = 0, 1  
    while current < n:  
        num = num1  
        num1, num2 = num2, num1+num2  
        current += 1  
        yield num  
    return 'done'
```

```
g=fib(5)
while True:
    try:
        x = next(g)
        print("value:%d"%x)
    except StopIteration as e:
        print("生成器返回值:%s"%e.value)
        break
```

输出结果：

```
value:0
value:1
value:1
value:2
value:3
生成器返回值:done
```

步骤 3 send

除了使用 `next` 方法以外，还可以使用 `send` 方法唤醒生成器，相比于 `next` 方法，`send` 在唤醒生成器时还可以向断点处传入一个数据。

```
def gen():
    i = 0
    while i<5:
        temp = yield i
        print(temp)
        i+=1
f = gen()
next(f)
>>>0
f.send('haha')
>>>haha
>>>1
next(f)
>>>None
>>>2
```

2.2.3.3 装饰器

步骤 1 构造装饰器

创建一个简单的装饰，其功能是计算一个函数的运行时间。

```
import time
def func(f):
    def inner(*args, **kwargs):
        start_time = time.time()
        f(*args, **kwargs)
```

```
    end_time = time.time()
    print('耗时: %s 秒' % (end_time - start_time))
return inner
```

步骤 2 装饰函数

```
@func
def test():
    time.sleep(2)
```

test()

输出结果:

```
耗时: 2.000276803970337 秒
```

2.2.4 正则表达式

步骤 1 re.match 函数

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match() 就返回 none。

函数语法:

```
re.match(pattern, string, flags=0)
```

示例:

```
import re
print(re.match('www', 'www.huawei.com').span()) # 在起始位置匹配
print(re.match('com', 'www.huawei.com')) # 不在起始位置匹配
```

输出结果:

```
(0, 3)
None
```

步骤 2 re.search 方法

re.search 扫描整个字符串并返回第一个成功的匹配。

函数语法:

```
re.search(pattern, string, flags=0)
```

示例:

```
import re
line = "Cats are smarter than dogs"
searchObj = re.search(r'(.*) are (.*) .*', line, re.M|re.I)
if searchObj:
    print("searchObj.group() : ", searchObj.group())
    print("searchObj.group(1) : ", searchObj.group(1))
    print("searchObj.group(2) : ", searchObj.group(2))
```

```
else:  
    print("Nothing found!!")  
  
输出结果：  
  
searchObj.group() : Cats are smarter than dogs  
searchObj.group(1) : Cats  
searchObj.group(2) : smarter
```

步骤 3 检索和替换

Python 的 re 模块提供了 re.sub 用于替换字符串中的匹配项。

语法：

```
re.sub(pattern, repl, string, count=0, flags=0)
```

示例：

```
import re  
  
phone = "2019-0101-000 # 这是一个电话号码"  
  
# 删除字符串中的 Python 注释  
num = re.sub(r'#.*/$', "", phone)  
print("电话号码是： ", num)  
  
# 删除非数字(-)的字符串  
num = re.sub(r'\D', "", phone)  
print("电话号码是 : ", num)
```

输出结果：

```
电话号码是： 2019-0101-000  
电话号码是 : 20190101000
```

步骤 4 re.compile 函数

compile 函数用于编译正则表达式，生成一个正则表达式（Pattern）对象，供 match() 和 search() 这两个函数使用。

语法格式为：

```
re.compile(pattern[, flags])
```

示例：

```
import re  
  
pattern = re.compile(r'\d+')          # 用于匹配至少一个数字  
n = pattern.match('one12twothree34four')      # 查找头部，没有匹配  
print(n)  
  
m = pattern.search('one12twothree34four') # 从'e'的位置开始匹配，没有匹配  
print(m)  
print(m.group())
```

输出结果：

```
None
<_sre.SRE_Match object; span=(3, 5), match='12'>
12
```

步骤 5 re.split()

split 方法按照能够匹配的子串将字符串分割后返回列表，它的使用形式如下：

```
re.split(pattern, string[, maxsplit=0, flags=0])
```

示例：

```
import re
s = re.split('\W+', 'www.huawei.com')
print(s)
```

输出结果：

```
['www', 'huawei', 'com']
```

3 资金管理系统

3.1 实验介绍

3.1.1 关于本实验

使用 Python 实现一个资金管理系统，功能包括：存款、取款、转账、秘密管理和凭证打印。数据存储在 MySql 数据库中。

3.1.2 实验目的

对于 Python 基础语法和高级语法部分的综合应用，实现一个功能简单的资金管理系统。

3.2 实验代码

3.2.1 实验思路

使用 PyMySql 连接操作数据库，根据数据库内的信息进行登录判断。成功登陆后进入系统欢迎界面，同时为成功登录的用户创建一个用户对象，根据用户作出的操作执行相应的方法，并同步到数据库中。在操作结束后将此次操作打印出来（写入本地文件）。

3.2.2 实验实现

步骤 1 创建数据库和数据表

创建数据库：

```
create database money;
```

创建数据表：

```
CREATE TABLE user(
    username varchar(30) PRIMARY KEY,
    pwd VARCHAR(100) NOT NULL,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    balance FLOAT NOT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

插入数据:

```
INSERT INTO user (username, pwd, start_time, end_time, balance)
VALUES ('admin','123456','2019.04.23', '2019.04.23',100.0);
INSERT INTO user (username, pwd, start_time, end_time, balance)
VALUES ('root','admin','2019.01.01', '2019.02.02',100.0);
```

步骤 2 导入所需的库, 定义操作字典

```
import time
import sys
import pymysql
import getpass
action_dict = {1:"存款", 2:"取款", 3:"转账", 4:"修改密码", 5:'退出'}
```

步骤 3 数据库连接

考虑到系统中多次连接数据库, 而连接数据库的语句相似性较高, 所以将其封装成为一个方法。

```
# 定义连接数据库方法, sql 为每次需要执行的数据库操作语句,
def con_mysql(sql):
    try:
        db = pymysql.connect("localhost", "root", "mysql", "money", charset='utf8' )
        # 使用 cursor() 方法获取操作游标
        cursor = db.cursor()
        # 使用 execute 方法执行 SQL 语句
        cursor.execute(sql)
        results = cursor.fetchone() # 查询一条数据
        db.commit() # 提交至数据库
    except Exception as e:
        db.rollback()
        print("系统异常")
        sys.exit()
    db.close() # 关闭数据库
    return results
```

测试方法:

```
sql = "select * from user"
con_mysql(sql)
```

输出结果:

```
('admin',
'123456',
datetime.datetime(2019, 4, 23, 12, 15, 21),
datetime.datetime(2019, 4, 23, 12, 15, 25),
11200.0)
```

图 3-1 数据库连接测试结果

步骤 4 定义用户类

```
class Account(object):
    def __init__(self, username, money, number=0):
        self.money = money # 账户金额
        self.username = username # 用户名
        # 上次登录时间
        self.start_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        self.number = number
    # 存款
    def save(self):
        self.money += self.number
        print("已存入%f 元" % (self.number))
    # 取钱
    def take(self):
        if self.number > self.money:
            print("余额不足")
        self.money -= self.number
        print("以取出%f 元" % (self.number))
    # 修改密码
    def update(self):
        pwd = getpass.getpass("请输入新密码: ")
        sql = "update user set pwd=%s where username=%s" % (pwd, self.username)
        return sql
    # 转账
    def transfer(self):
        user = input("请输入转账用户: ")
        if self.number > self.money:
            print("余额不足")
            return
        else:
            sql = "select username from user where username='%s'" % (user)
            result = con_mysql(sql)
```

```

        if result == None:
            print("转账用户不存在")
            self.number=0
        else:
            return user
    # 执行用户所选的操作
    def implement(self, action):
        if action == 5:
            sys.exit()
        elif action == 1:
            try:
                self.number = float(input("请输入存入得金额: "))
            except Exception as e:
                print("请输入正确的金额")

                self.save()
        elif action == 2:
            try:
                self.number = float(input("请输入取出的金额: "))
            except Exception as e:
                print("请输入正确的金额")
                self.take()
                sql = "update user set balance=%f where
username=%s"%(self.number,self.username)
                con_mysql(sql)
        elif action == 3:
            try:
                self.number = float(input("请输入转账的金额: "))
            except Exception as e:
                print("请输入正确的金额")
                User = self.transfer()
                if User:
                    sql = "update user set balance=%f where
username=%s"%(self.number,User)
                    con_mysql(sql)
                else:
                    self.update(pwd)
    # 打印操作后的凭证
    def voucher(self,end_time, action):
        str_action = """用户: %s \n 操作: %s\n 操作金额: %s\n 登录时间:
%s\n 结束时间: %s"""%(self.username, action_dict[action],
self.number, self.start_time, end_time)
        with open("%s.txt"%(self.username), 'w') as f:
            try:
                f.write(str_action)

```

```
        except Exception as e:  
            print("凭证打印失败, 请联系管理员")  
            print("打印成功, 请收好您的凭证")
```

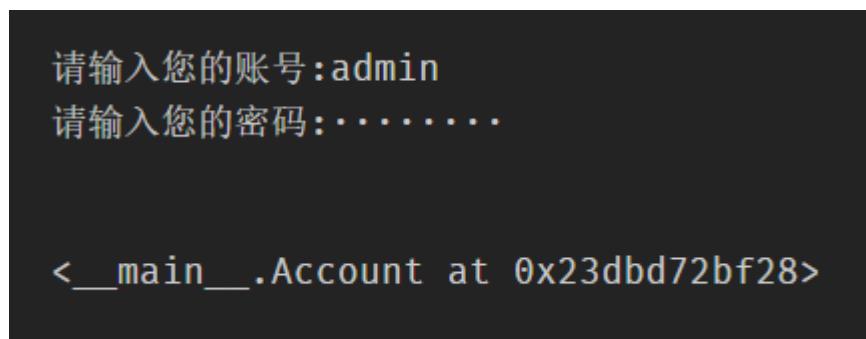
步骤 5 登录功能

```
def login():  
    """  
    用户登录检测  
    :param username: 用户账号  
    :param pwd: 用户密码  
    :return:  
    """  
  
    username = input("请输入您的账号:")  
    pwd = getpass.getpass("请输入您的密码:") # 隐藏输入的密码  
  
    # 编写 SQL 语句从数据库获取账号信息  
    sql = "select * from user where username='%s'"%(username)  
    result = con_mysql(sql)  
    if result:  
        if result[1] == pwd:  
            user_account = Account(result[0], result[4])  
            return user_account  
        else:  
            print("账号或密码错误")  
    else:  
        print("账号不存在")
```

测试登录功能：

```
user_account = login()  
user_account
```

输出结果：



The terminal window shows the following interaction:

```
请输入您的账号:admin
请输入您的密码:.....
<__main__.Account at 0x23dbd72bf28>
```

图 3-2 登录功能测试

步骤 6 欢迎界面

```
def welcome():
```

```

print(' *'*15)
print(" %s%30s"%("*", "*"))
print(" %s      欢迎进入资金管理系统 %s"%("*", "*"))
print(" %s%30s"%("*", "*"))
print(' *'*15)
try:
    action = input("请选择操作: 1.存款 2.取款 3.转账 4.修改密码 5.退出:")
    action = int(action)
except Exception as e:
    print("warn:请输入正确的操作指令! ")
    return -1
if action not in action_dict:
    print("warn:请执行正确的操作! ")
    return -1
return action

```

测试 welcome 方法:

```

action = welcome()
action

```

输出结果:

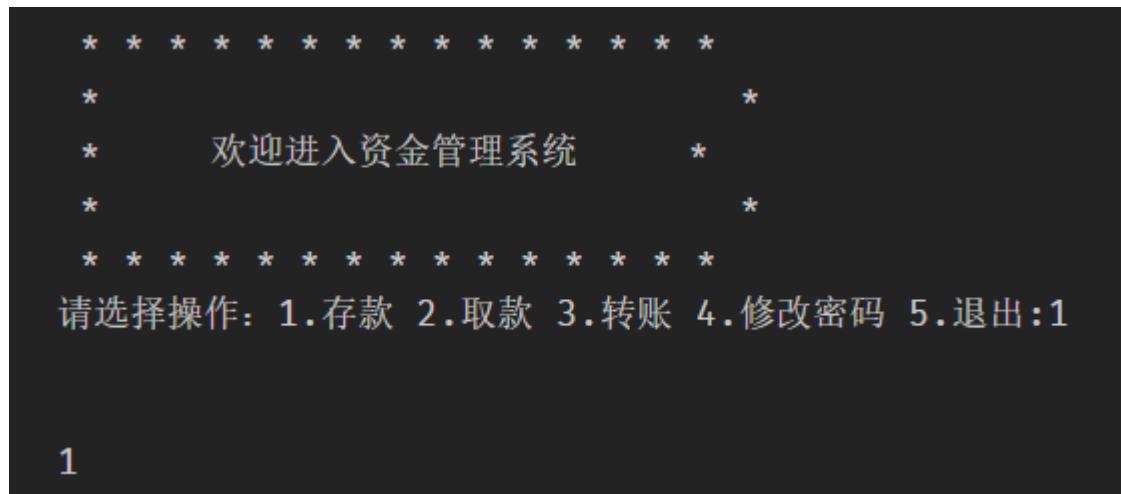


图 3-3 系统欢迎界面

步骤 7 定义系统启动函数

设置启动函数:

```

def run():
    action = welcome()
    user_account.implement(action)
    end_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    sql = """
        update user set balance=%f,start_time='%s',end_time='%s' where username='%s'
    """

```

```
"""%(user_account.money,
user_account.start_time,end_time,user_account.username)
con_mysql(sql)
user_account.voucher(end_time, action)
```

步骤 8 使用装饰器为系统添加计时功能

定义装饰器：

```
def consume_time(func, *args, **kwargs):
    def inner(*args, **kwargs):

        start_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        print("本次登录时间%s"%(start_time))
        func()
        end_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        print("登出时间%s"%(end_time))
        return (start_time, end_time)
    return inner
```

给系统启动函数添加功能：

```
@consume_time
def run():
    action = welcome()
    user_account.implement(action)
    end_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    sql = """
update user set balance=%f,start_time='%s',end_time='%s' where username='%'%
"""%(user_account.money,
user_account.start_time,end_time,user_account.username)
    con_mysql(sql)
    user_account.voucher(end_time, action)
```

步骤 9 启动系统

```
if __name__ == "__main__":
    user_account = login()
    while True:
        if isinstance(user_account, Account):
            break
    while True:
        run()
```

输出结果：

```
请输入您的账号:admin
请输入您的密码:.....
本次登录时间2019-04-22 09:50:48
* * * * *
*          *
*    欢迎进入资金管理系统    *
*          *
* * * * *
请选择操作: 1.存款 2.取款 3.转账 4.修改密码 5.退出:1
请输入存入得金额: 100
已存入100.000000元
打印成功, 请收好您的凭证
登出时间2019-04-22 09:51:00
```

图 3-4 系统执行结果

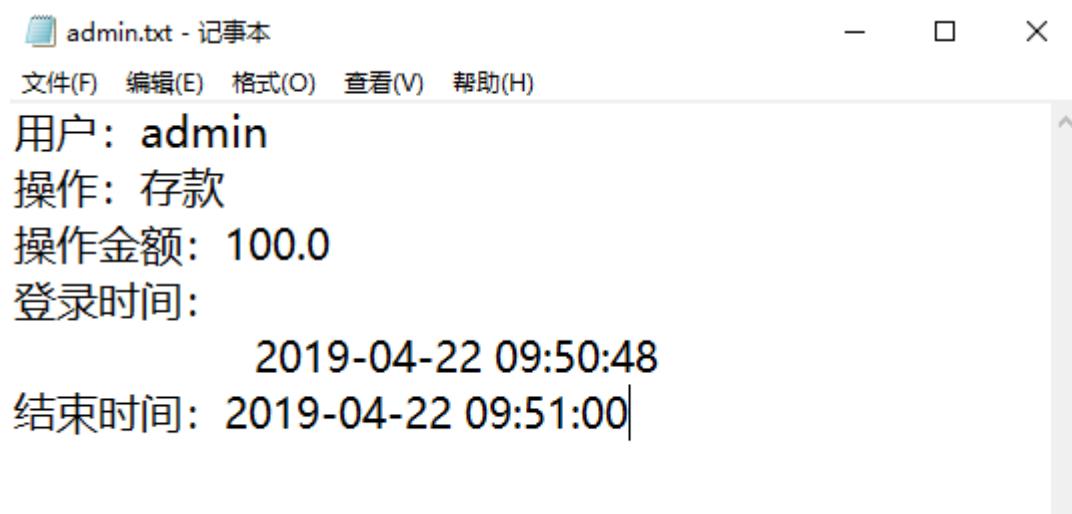


图 3-5 打印的凭证