**数据应用开发与服务(Python)**

**职业技能等级证书(初级) 实操考试试题**

**考试时间：90分钟**

**重要说明：**

1. 本考试包括5道编程大题，每道大题设有若干个填空或选择小题，共计40小题。各个大题之间相互独立，没有前后联系。
2. 每道编程题均已提供了详细的代码框架，考生应首先仔细阅读理解代码框架，尤其是注释行中的解释和要求，然后回答问题。
3. 填空题应特别注意区分大小写，注意是否需要添加单引号、双引号、括号等字符。
4. 不得在代码框架中添加任何其它代码，不得更改或删除已有的任何代码。

# 题目一、实现程序的基本流程

**1.题目简介**

本试题要求分别使用递归和循环方式来实现计算Fibonacci序列中第num个元素的值，并粗略统计两种方法分别消耗的时间。

Fibonacci序列是指满足下列条件的序列：

1. 序列中的第1个和第2个元素值都为1
2. 从第3个元素起，其值为前两个元素值之和

按照上述规则，前10个元素值分别为：1, 1, 2, 3, 5, 8, 13, 21, 34, 55

**2.代码框架**

本任务程序位于源代码目录01中，仅包含main.py文件。

代码框架包括了2个任务，共6个小题。

**3.任务步骤**

**任务1：使用递归调用实现Fibonacci函数**

在main.py文件中，根据上下文，填写空缺的4处内容（小题编号【1】、【2】、【3】、【4】）：

# 以递归方式实现的函数

def fibonacci\_recursive(num):

    if num <= 2:

        return \_\_\_\_【1】\_\_\_\_

    return fibonacci\_recursive(num - 1) + fibonacci\_recursive(num - \_\_\_\_【2】\_\_\_\_)

# 以循环方式实现的函数

def fibonacci\_traversive(num):

    # 初始化，第1、2个Fibonacci数的值都为1

    temp1 = temp2 = 1

    # 从第3个元素开始，依次根据前2个元素值，计算后1个元素的值

    for i in range(3, num + 1):

        temp = temp1 + temp2      # 计算前两个数之和

        temp1 = \_\_\_\_【3】\_\_\_\_   # temp1和temp2分别更新赋值

        temp2 = temp

    # 返回第num个元素的计算结果

    return temp2

# 用两种方式分别计算Fibonacci序列前10个元素的值

for i in range(1, \_\_\_\_【4】\_\_\_\_):

    print("第%d个元素值为：%d, %d" %

        (i, fibonacci\_traversive(i), fibonacci\_recursive(i)))

**任务2：粗略计算函数运行的时间**

在main.py文件中，填写空缺的2处内容（小题编号【5】、【6】）

import time

# 计算Fibonacci指定元素值并统计运行耗时

# func：传入的函数，本例中可传入递归调用fibonacci\_recursive或循环调用fibonacci\_traversive的函数

# num：传入待计算的第num个元素

# 返回值：Fibonacci计算结果，函数运行耗时

def test\_fibonacci(func, num):

    start = time.time()     # 记录函数调用初始时间戳

    result = \_\_\_\_【5】\_\_\_\_(num)      # 调用函数

    end = time.time()       # 记录函数调用结束时间

    # 返回计算结果和耗时

    return (result, end - \_\_\_\_【6】\_\_\_\_)

# 通过test\_fibonacci分别调用递归和循环函数，输出计算结果和耗时

num = 38        # 计算Fibonacci中第num个元素的值

print("计算Fibonacci序列第%d个元素的值并衡量耗时......" % num)

result, t = test\_fibonacci(fibonacci\_recursive, num)

print("递归调用，结果为：%d， 耗时：%.4f秒" % (result, t))

result, t = test\_fibonacci(fibonacci\_traversive, num)

print("循环调用，结果为：%d， 耗时：%.4f秒" % (result, t))

# 题目二、编写面向对象的程序

**1.题目简介**

本试题要求对一批员工和部门对象进行处理，包括对象的创建、对象集合的管理、查找集合中的对象，对象排序等。

**2.代码框架**

本任务程序位于源代码目录02中，包含main.py和entity.py两个文件。entity.py中定义了员工(Employee)和部门(Department)两个实体类；main.py文件中给定了5个任务，针对员工和部门对象数据进行处理。

代码框架包括了5个任务，共10个小题。其中，第1、3、4、5任务的小题号出现在main.py中，第2个任务的小题号出现在entity.py中。

**3.任务步骤**

**任务1：导入必要的类**

在main.py文件中，根据上下文，填写空缺的1处内容（小题编号【1】）：

# 从指定模块中导入实体类

from \_\_\_\_【1】\_\_\_\_ import Employee, Department

**任务2：创建初始的部门和员工对象**

在entity.py文件中，填写空缺的2处内容（小题编号【2】、【3】）

    # 以指定的文本格式输出员工的信息

    def \_\_str\_\_(self):

        # 获取所属部门名称(如果没有所属部门，则返回'无部门')

        dept\_name = self.dept.name \_\_\_\_【2】\_\_\_\_ self.dept \_\_\_\_【3】\_\_\_\_ '无部门'

        return "%s, %s, %.2f, %s" % (self.name, self.gender, self.salary, dept\_name)

然后，在main.py文件中，结合任务2中测试代码，检验上述两处内容填空的结果。

**任务3：将部门与员工关联**

在main.py文件中，根据上下文，填写空缺的【2】处内容（小题编号【4】、【5】）：

# 将所有员工加入dept1

for emp in [emp1, emp2, emp3, emp4]:

    dept1.employees.add(\_\_\_【4】\_\_\_)

# 输出dept1部门所有员工信息

print("dep1下所有员工信息：")

dept1.\_\_\_\_【5)】\_\_\_\_()

**任务4：根据条件查找匹配的员工信息**

在main.py文件中，根据上下文，填写空缺的3处内容（小题编号【6】、【7】、【8】）：

# 在dept1部门内查找所有男员工

gender = '男'

results = filter(lambda emp: emp.gender == \_\_\_\_【6】\_\_\_\_, dept1.employees)

# 查询结果转换成list

results\_list = \_\_\_\_【7】\_\_\_\_(results)

for result in results\_list:

    print(result)

# 在dept1部门内查找姓名包含字母'J'的员工

name = 'J'

# 姓名匹配并构造匹配的对象

results = [employee for employee in dept1.employees if name \_\_\_\_【8】\_\_\_\_ employee.name]

print("dept1下姓名包含J的员工：")

for result in results:

    print(result)

**任务5：根据薪资进行排序**

在main.py文件中，根据上下文，填写空缺的2处内容（小题编号【9】、【10】）：

# 定义按salary进行比较的函数

def compare\_by\_salary(emp):

    return emp.salary

# 将dept1的员工集合转换成list

employees = list(dept1.employees)

# 根据salary从高到底排序

employees.sort(key=\_\_\_\_【9】\_\_\_\_, reverse=\_\_\_\_【10】)

for emp in employees:

    print(emp)

# 题目三、使用常用API编程

**1.题目简介**

本试题要求使用文件读写和目录操作API，完成大文件读写操作和获取文件属性的任务。

**2.代码框架**

本任务程序位于源代码目录03中，仅包含main.py文件。

代码框架包括了2个任务，共6个小题。

Origin子目录作为任务1的源目录，里面存放了一个待操作的大文件；target子目录作为任务1的目标目录；test子目录存放了若干个文件，用于任务2处理。

**3.任务步骤**

**任务1：以循环方式实现大文件拷贝**

在main.py文件中，根据上下文，填写空缺的4处内容（小题编号【1】、【2】、【3】、【4】）：

origin\_file\_path = './origin/data.csv'

target\_file\_path = './target/data.csv'

# 如果目标文件已存在，则先删除之

if os.path.\_\_\_\_【1】\_\_\_\_(target\_file\_path):

    os.remove(target\_file\_path)

BATCH\_SIZE = 1024 \* 128    # 每次读取128KB字节

# 以只读形式打开源文件(仅当作文本文件处理)

origin\_file = open(origin\_file\_path, \_\_\_\_【2】\_\_\_\_)

# 以只写方式打开目标文件

target\_file = open(target\_file\_path, \_\_\_\_【3】\_\_\_\_)

# 循环操作：从源文件读取BATCH\_SIZE字节，然后写入目标文件中

batch\_data = origin\_file.read(BATCH\_SIZE)

while batch\_data:

    target\_file.write(batch\_data)

    batch\_data = origin\_file.\_\_\_\_(4)\_\_\_\_(BATCH\_SIZE)

# 关闭文件

origin\_file.close()

target\_file.close()

**任务2：获取指定目录下尺寸最大的文件**

在main.py文件中，填写空缺的2处内容（小题编号【5】、【6】）

import os

max\_size\_file\_name = ''     # 保存尺寸最大的文件路径名

max\_size = 0

dir\_path = '.\\test'

for item in os.listdir(dir\_path):       # 遍历目录中的每个子项

    # 合成完整路径

    full\_path = os.path.\_\_\_\_【5】\_\_\_\_(dir\_path, item)

    # 判断子项是否为文件

    if os.path.isfile(full\_path):

        file\_stat = os.stat(full\_path)      # 获取文件属性信息

        file\_size = file\_stat.st\_size       # 获取文件大小(字节数)

        if file\_size > max\_size:

            max\_size = \_\_\_\_【6】\_\_\_\_

            max\_size\_file\_name = item

print("最大文件：%s, 大小为：%d 字节" % (max\_size\_file\_name, max\_size))