编程题

1. 生成器（5分）

（1）创建生成器

```
def test1(number):
    a=0
    b=1
    n=0
    while n < number:
        yield b
        a,b=b,(a+b)
        n+=1
    return 'done'

g=test1(6)
print(g)
```

输出结果：<generator object test1 at 0x0000021018DF90E0>

(2)创建生成器，利用faker随机生成资料并输出

```
import faker #随机生成资料

fk = faker.Faker(locale='zh_CN')
result = fk.phone_number()
print(result)
# 随机生成公司名称
print(fk.company())
# 随机生成地址
print(fk.address())
# 随机生成一个城市
print(fk.city())
```

输出结果：

13764103591
方正科技传媒有限公司
山西省杨市白云南宁路U座 224148
台北县

(3)创建一个生成器，查看其类型

```
g = (x * x for x in range(5))
print(g)

for n in g:
    print(n)
```

输出结果：<generator object at 0x00000211535EC040>
0
1
4
9
16

(4)生成消费者系统

```
def consumer():
    r = ''
    while True:
        n = yield r
        if not n:
            return
        print('[CONSUMER] Consuming %s...' % n)
        r = 'done'

# 生产者
def produce(c):
    c.send(None)
    n = 0
    while n < 5:
        n = n + 1
        print('[PRODUCER] Producing %s...' % n)
        r = c.send(n)
        print('[PRODUCER] Consumer return: %s' % r)
    c.close()

c = consumer()
produce(c)
```

输出结果：

[PRODUCER] Producing 1...
[CONSUMER] Consuming 1...
[PRODUCER] Consumer return: done
[PRODUCER] Producing 2...
[CONSUMER] Consuming 2...
[PRODUCER] Consumer return: done
[PRODUCER] Producing 3...
[CONSUMER] Consuming 3...
[PRODUCER] Consumer return: done
[PRODUCER] Producing 4...
[CONSUMER] Consuming 4...
[PRODUCER] Consumer return: done
[PRODUCER] Producing 5...
[CONSUMER] Consuming 5...
[PRODUCER] Consumer return: done

2. 异常管理（10分）

（1）异常捕获

```
while True:
    try:
        # 提示用户输入一个整数
        num = int(input("输入一个整数: "))
        # 输入不是整数就报错，输入0也是报错，这里我们需要捕获异常 赋值错误的第一个单词作为关键字进行处理
        result = 8 / num
        print(result)
    except ZeroDivisionError:
        print("数学错误")
    except ValueError:
        print("请输入数字")
    except Exception as result:
        print("未知错误 %s" % result)
```

输出结果：

```
输入一个整数: 3
2.6666666666666665
输入一个整数: 6
1.3333333333333333
输入一个整数: gsrg
请输入数字
输入一个整数:
```

（2）密码案例

```
需求案例1:
# 定义 input_password函数，提示用户输入密码
# 如果用户输入长度<8，抛出异常
# 如果用户输入长度>=8 返回输入的密码

def input_password():
    #1.提示用户输入密码
    result =input("请输入密码")
    #2.判断密码长度 >=8 ,返回用户输入的密码
    if len(result) >=8:
        return result
    #3.如果<8 主动抛出异常
    print("主动抛出异常！")
    #1>创建异常对象 -可以使用错误信息字符串作为参数
    ex =Exception("密码长度不够！")
    #2> 主动抛出异常
    raise ex
#提示用户输入密码
try:
    print(input_password())
except Exception as result:
    print(result)
```

输出结果：

请输入密码123456789
123456789

请输入密码ased
主动抛出异常!
密码长度不够!

（3）判断是否为数值，不是的话输出异常

```
while True:
    try:
        price=float(input("请输入价格: "))
        print('价格为:%5.2f' % price)
        break
    except ValueError:
        print('您输入的不是数字。')
```

(4)创建异常并测试异常是否出现并输出，模板:

```
try:
    <可能出现异常的语句块>
except <异常类名字name1>:
    <异常处理语句块1>  #如果在try部份引发了'name1'异常，执行这部分语句块
except <异常类名字name2> as e1:
    <异常处理语句块2>  #如果在try部份引发了'name2'异常，执行这部分语句块
except <（异常类名字name3，异常类名字name4，…）> as e2:
    <异常处理语句块3>  #如果引发了'name3'、'name4'、…中任何一个异常，执行该语句块
…
except:
    <异常处理语句块n>   #如果引发了异常，但与上述异常都不匹配，执行此语句块
else:
    <else语句块>          #如果没有上述所列的异常发生，执行else语句块
finally:
    <任何情况下都要执行的语句块>
```

（5）案例2

```
try:
   x=float(input('请输入设备成本: '))
   y=int(input('请输入分摊年数: '))
   z=x/y
   print('每年分摊金额为%.2f'% z)
except ZeroDivisionError:
   print("发生异常，分摊年数不能为0.")
except:
   print('输入有误')
else:
   print("没有错误或异常")
finally:
   print('不管是否有异常发生，始终执行finally部分的语句')
```

输出结果:

请输入设备成本：5
请输入分摊年数：6
每年分摊金额为0.83
没有错误或异常
不管是否有异常发生，始终执行finally部分的语句

（6）

```
# 代码:长度不低于3为
class ShortInputException(Exception):
    def __init__(self, length, atleast):
        self.length = length
        self.atleast = atleast
def main():
    try:
        s = input('请输入 --> ')
        if len(s) < 3:
            # raise引发一个你定义的异常
            raise ShortInputException(len(s), 3)
    except ShortInputException as result:#x这个变量被绑定到了错误的实例
        print('ShortInputException: 输入的长度是 %d,长度至少应是 %d'%
    (result.length, result.atleast))
    else:
        print('没有异常发生')
main()
```

3. 四种模式的一种（10分）

(1.1)单例模式（闭包+装饰器）

```
#_new_方法：制造CLass
#_call_方法：制造obj

def singleton(cls): #装饰器
    _instance = {} #闭包

    def inner(*args,**kwargs):#定义内部方法
        if cls in _instance:
            return _instance[cls]

        obj = cls(*args,**kwargs)
        _instance[cls] = obj #存在字典

        return obj
    # return cls
    return inner


@singleton#加入装饰器，类就自动变成单例只能创造一个对象
class Person: #创建类
    pass

p_1 = Person()
```

```
p_2 = Person()#没有创造对象

print(p_1 is p_2)
```

输出结果：True

（1.2）单例模式:metaclass 利用类自身来产生类成员，来记录变量

```
class SingletonMeta(type):
    _instance = {}

    def __call__(cls, *args, **kwargs):
        if hasattr(cls,'_instance'):#类里面有没有一个叫instance的类属性
            return getattr(cls,'_instance')

        obj = super().__call__(cls,*args,**kwargs)
        setattr(cls,'_instance',obj)

        return obj

# @singleton#加入装饰器，类就自动变成单例只能创造一个对象
class Person(metaclass=SingletonMeta): #创建类
    pass

p_1 = Person()
p_2 = Person()#没有创造对象

print(p_1 is p_2)
```

输出结果：True

（2.1）代理模式：

```
class Actor(object):
    def __init__(self):
        self.is_empty = True

    def show_film(self):
        self.is_empty = False
        print(type(self).__name__, "show_film")

    def listen_music(self):
        self.is_empty = True
        print(type(self).__name__, "listen_music")


class Agent(object):
    def __init__(self):
        self.actor = Actor()

    def work(self):
        if self.actor.is_empty:
            self.actor.show_film()
        else:
```

```
            self.actor.listen_music()


if __name__ == '__main__':
    agent = Agent()
    agent.work()
    agent.work()
```

输出结果：

Actor show_film
Actor listen_music

（2.2）银行

```
from abc import ABC, abstractmethod

# 支付接口
class Payment(ABC):
    @abstractmethod
    def do_pay(self):
        pass


# 银行类：真实主题
class Bank(Payment):
    def check_account(self):
        print("账户检查中...")
        return True

    def do_pay(self):
        self.check_account()
        print("银行结算完成")


# 银行类的代理
class DebitCard(Payment):
    def __init__(self):
        self.bank = Bank()

    def do_pay(self):
        print("借记卡即将去银行支付")
        self.bank.do_pay()
        print("借记卡完成银行支付")


# 客户端
class You(object):
    def __init__(self):
        self.debit_card = DebitCard()

    def make_payment(self):
        print("借记卡支付开始")
        self.debit_card.do_pay()
        print("借记卡支付结束")
```

```
if __name__ == '__main__':
    you = You()
    you.make_payment()
    """
    借记卡支付开始
    借记卡即将去银行支付
    账户检查中...
    银行结算完成
    借记卡完成银行支付
    借记卡支付结束
    """
```

（3.1）观察者模式:Python设计模式：观察者模式 - 知乎 (zhihu.com)

```
# 看股票的职员
class StockClerk:
    def __init__(self, name):
        self.name = name

    def close_stock_software(self):
        print(f"{self.name} 关闭了股票软件，并开始办公")


# 睡着的职员
class SleepingClerk:
    def __init__(self, name):
        self.name = name

    def open_word(self):
        print(f"{self.name} 打开了word，并开始办公")

class Receptionist:
    actions = []

    @classmethod
    def attach(cls, action):
        cls.actions.append(action)

    @classmethod
    def notify(cls):
        print("老板回来了，各同事行动...")
        for actioin in cls.actions:
            actioin()
# 实例化职员
c1 = StockClerk('Chris')
c2 = SleepingClerk('Ryan')

# 告诉前台小姐姐如何通知
Receptionist.attach(c1.close_stock_software)
Receptionist.attach(c2.open_word)

# 前台小姐姐发布通知
```

```
Receptionist.notify()
```

输出结果：

老板回来了，各同事行动...
Chris 关闭了股票软件，并开始办公
Ryan 打开了word，并开始办公

(4.1)适配器模式

```
# （二）类适配器
# 步骤:
# 1.创建一个新的类
# 2.初始化传入需要适配的对象
# 3.接口调用初始化中的对象的老方法
class ExecuteCmd:
    def __init__(self, cmd):
        self.cmd = cmd

    def cmd_exe(self):
        print(f"使用cmd_exe执行{self.cmd}命令")


class NewExecuteCmd:
    def __init__(self, cmd):
        self.cmd = cmd

    def run_cmd(self):
        print(f"使用run_cmd执行{self.cmd}命令")


class ExecuteAdapter():
    def __init__(self, new_exec_obj):
        self.exec_obj = new_exec_obj

    def cmd_exe(self):
        """直接在适配器中实现旧系统的接口"""
        return self.exec_obj.run_cmd()


# 旧接口
old_obj = ExecuteCmd("ls")
old_obj.cmd_exe()

# 新接口需要进行适配
new_obj = ExecuteAdapter(NewExecuteCmd("ls"))
new_obj.cmd_exe()
```

输出结果：

使用cmd_exe执行ls命令
使用run_cmd执行ls命令

4. 并行（10分）

(1)计算并行时间

```
import time
import os
from multiprocessing import Pool
def countdown(n):
    while n > 0:
        n -= 1

if __name__ == "__main__":
    count = 2e7
    start = time.time()
    # n_processes = os.cpu_count()
    n_processes = 8 # 进程数
    pool = Pool(processes=n_processes) # 进程池
    for i in range(n_processes):
        pool.apply_async(countdown, (count//n_processes,)) # 启动多进程
    pool.close() # 使进程池不能添加新任务
    pool.join() # 等待进程结束
    print(time.time() - start)
```

输出结果：0.8108816146850586

（2）利用treading 创建并行

```
import threading

def my_function(arg):
    # Do something with 'arg'
    print(arg)

threads = []
for i in range(5):
    thread = threading.Thread(target=my_function, args=(i,))
    thread.start()
    threads.append(thread)

# Wait for all threads to complete
for thread in threads:
    thread.join()
```

输出结果：

0
1
2
3
4

（3）使用线程池或进程池来分配任务并行执行

```python
import concurrent.futures

def my_function(arg):
    # Do something with 'arg'
    return arg

# Create a thread pool
with concurrent.futures.ThreadPoolExecutor() as executor:
    # Submit work to the pool
    results = [executor.submit(my_function, i) for i in range(5)]

    # Wait for the results to complete
    for result in concurrent.futures.as_completed(results):
        print(result.result())
```

输出结果:

0
1
2
3
4