HW5-作业讲评

王瑞环

1.1 正则表达式

- 电话号码的判断

```
def phone_num_valid(phone_num):

    pattern = re.compile(r'^1\d{10}$')

    return bool(pattern.match(phone_num))
```

- 邮箱的判断

```
def email_valid(email):

    pattern = re.compile(r'^[a-zA-Z0-9]+([\.\-]?[a-zA-Z0-9]+)*@test\.com$')

    return bool(pattern.match(email))
```

- 严格来说不能用\w，因为\w除了匹配字母和数字外，还会匹配下划线，但题目要求的格式中并不允许下划线的存在（用\w没扣分）

1.1 正则表达式

- 邮箱匹配的几种错误答案（但是能过assert，没扣分）

```
pattern = re.compile(r'[A-Za-z0-9]([A-Za-z0-9]|.|-)*@test.com$')
        # 错误例子：a-@test.com
pattern = re.compile(r"^\w[\w.-]*\w@test.com$")
        # 错误例子：a@test.com
pattern = re.compile(r'^[a-zA-Z0-9][a-zA-Z0-9\.\-]*[a-zA-Z0-9]@test\.com$')
        # 错误例子：a@test.com
pattern = re.compile(r'^[^\.-][a-zA-Z0-9\.-]+[^\.-]@test\.com$')
        # 错误例子：a@test.com
pattern = re.compile(r'^(?<!\.\-)[a-zA-Z0-9\.\-]+(?<!\.\-)@test\.com$')
        # 错误例子：a-@test.com
pattern = re.compile(r'(\w|(\w[\w.-]*\w))@test.com$') # 把.com换成\.com就正确
        # 错误例子：a@testacom
```

- 零宽断言的正确用法

```
pattern = re.compile(r'^(?![-.])[\w.-]+(?<![-.])@test\.com$')     #刘和金
pattern = re.compile(r'^(?=[\w])[\w.-]+(?<=[\w])@test\.com$')     #黄学郅
```

1.1 正则表达式

- 关于正则表达式里的"."和"-"在[]和()中的几种用法

	"."	"B"	"-"
[.]	√		
[-]			√
[A-]			√
[-A]			√
[A-Z]		√	
[.-Z]	√	√	
[\.-Z]	√	√	
[--Z]	√	√	√
[.-A-Z]	√		√
(A|.)	√	√	√
(A|-)			√

	"."	"B"	"-"
[\.]	√		
[\-]			√
[A\-]			√
[\-A]			√
[A\-Z]			√
[.\-Z]	√		√
[\.\-Z]	√		√
[-\-Z]			√
[.\-A-Z]	√	√	√
(A|\.)	√		
(A|\-)			√

1.1 正则表达式

- （附加题）合理密码的判断

```
def passwd_valid(passwd, use_regex=False):
    if use_regex:

        pattern = re.compile(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)[A-Za-z\d]{9,}$')

        return bool(pattern.match(passwd))
```

- 另一种写法

```
# 崔鹤龄
pattern = re.compile(r'^(?![0-9a-z]+$)(?![a-zA-Z]+$)(?![0-9A-Z]+$)[0-9A-Za-z]{9,}$')
```

- 关于零宽断言的一个简单例子

```
pattern = re.compile(r'(?=.*a)\w+')  # 判断对\w+的匹配结果中是否有.*a结构
pattern.match('bac')  # 能够匹配
pattern.match('bc')   # 不能匹配
```

1.2 服务器

```
for i in range(30):
    client_socket, addr = server.accept()

    client_handler = threading.Thread(
        target=self.handle_client, args=(i, client_socket))
    all_threads.append(client_handler)
    client_handler.start()

for thread in all_threads:
    thread.join()
```

2.1 yield用法

```
def running_mean():
    total = 0.0
    count = 0
    value = yield

    while True:
        if value is None:
            total = 0.0
            count = 0
            value = yield (0.0, 0)
        elif isinstance(value, (int, float)):
            total += value
            count += 1
            value = yield (total / count, count)
        else:
            value = yield "Wrong input"
```

2.1 yield用法

- 可拓展为带历史记录的计算器

```
#黄婧扬
def running_mean():
    count = 0
    numlist = []
    res = None
    while True:
        num = yield res
        if isinstance(num, int) or isinstance(num, float):
            count += 1
            numlist.append(num)
            res = (sum(numlist)/count, count)
        elif num is None:
            count = 0
            numlist = []
            res = (0.0,0)
        else:
            res = "Wrong input"
```

2.2 Tkinter

- 可以用简单的try-except判断是否能转为浮点数

```
def calculate_running_mean(self):
    input_str = self.input_entry.get()
    if len(input_str) == 0:
        cur_avg, cur_n = self.running_mean_fn.send(None)
        self.output_entry_show(self.output_entry1, str(cur_avg))
        self.output_entry_show(self.output_entry2, str(cur_n))
    else:
        try:
            cur_avg, cur_n = self.running_mean_fn.send(float(input_str))
            self.output_entry_show(self.output_entry1, str(cur_avg))
            self.output_entry_show(self.output_entry2, str(cur_n))
        except ValueError:
            msg = self.running_mean_fn.send(input_str)
            self.output_entry_show(self.output_entry1, msg)
            self.output_entry_show(self.output_entry2, '')
    self.input_entry.delete(0, tk.END)
```

2.2 Tkinter

- 用字符串操作判断能否转为浮点数
- 基于字符串的isdigit()方法

```
#曾为帅
def Is_Float(self, a):
    lst = a.split(".")
    if len(lst) == 2 and lst[0].isdigit() and lst[1].isdigit():
        return True
    return False
```

- 基于正则表达式

```
# 周宇亮
re.match(r"[+-]?(\d*\.\d+|[1-9]\d*|0)$", s)
```

3.1 任务调度-task

```
while total_time > 0:
    actual_duration = 0

    if customized_duration is None:
        actual_duration = min(default_duration, total_time)
    else:
        actual_duration = min(customized_duration, total_time)

    total_time = max(total_time - actual_duration, 0)

    print(f'Working on task1 for time {actual_duration}, ' \
            f'time left {total_time}')
    customized_duration = yield
```

3.2 任务调度-RoundRobin

- 参考答案

```
while len(tasks_list) > 0:
    print(f'At timestamp {i}:')

    for task, custom_times in zip(tasks_list[:], customized_times_list[:]):
        try:
            custom_time = custom_times.get(i)
            task.send(custom_time)
        except StopIteration:
            tasks_list.remove(task)
            customized_times_list.remove(custom_times)

    i += 1
```

- 易错点1：在for x in lst结构中操作lst
- 易错点2：忘记同时remove customized_time_list的元素

3.2 任务调度-RoundRobin

- 另一种写法，并不直接在调度任务的循环中remove，而是记录已完成的任务，在一次调度结束后统一删除

```
# 刘浩伦
itm_to_rmv=[]
for j,(tsk,dct) in enumerate(zip(tasks_list,customized_times_list)):
    try:
        if i in dct.keys():
            tsk.send(dct[i])
        else:
            tsk.send(None)
    except StopIteration:
        itm_to_rmv.append(j)
for id in itm_to_rmv:
    del tasks_list[id]
    del customized_times_list[id]
```

3.2 任务调度-RoundRobin

- **错误写法一**：忘记移除customized_times_list对应元素

```
for p in tasks_list[:]:
    try:
        custimedic = customized_times_list[tasks_list.index(p)]
        if i in custimedic:
            custime = custimedic[i]
        else:
            custime = None
        p.send(custime)
    except StopIteration:
        tasks_list.remove(p)
```

错误输出

```
At timestamp 5:
Task 1 Finished
Working on task2 for time 2, time left 2
At timestamp 6:
Working on task2 for time 2, time left 0
At timestamp 7:
Task 2 Finished
All tasks finished
```

正确输出

```
At timestamp 5:
Task 1 Finished
Working on task2 for time 2, time left 2
At timestamp 6:
Working on task2 for time 1, time left 1
At timestamp 7:
Working on task2 for time 1, time left 0
At timestamp 8:
Task 2 Finished
All tasks finished
```

3.2 任务调度-RoundRobin

- **错误写法二**：在for x in lst中直接操作lst

```
for p in tasks_list:
    index = tasks_list.index(p)
    try:
        if i in customized_times_list[index].keys():
            tasks_list[index].send(customized_times_list[index][i])
        else:
            tasks_list[index].send(None)
    except StopIteration:
        tasks_list.remove(tasks_list[index])
        customized_times_list.remove(customized_times_list[index])
```

错误输出

```
At timestamp 5:
Task 1 Finished
At timestamp 6:
Working on task2 for time 1, time left 3
At timestamp 7:
Working on task2 for time 2, time left 1
At timestamp 8:
Working on task2 for time 1, time left 0
At timestamp 9:
Task 2 Finished
All tasks finished
```

正确输出

```
At timestamp 5:
Task 1 Finished
Working on task2 for time 2, time left 2
At timestamp 6:
Working on task2 for time 1, time left 1
At timestamp 7:
Working on task2 for time 1, time left 0
At timestamp 8:
Task 2 Finished
All tasks finished
```

3.2 任务调度-RoundRobin

- **错误写法二**：在for x in lst中直接操作lst，更为简单但体现本质的例子

```
lst = [1, 2, 3]
for x in lst:
    print(x)
    lst.remove(x)
```

- 这段代码的输出为1和3，并没有2
- 所以，当确实需要在循环中对被循环的列表lst进行增删操作，则在for语句中应该使用其复制lst[:]

```
lst = [1, 2, 3]
for x in lst[:]:
    print(x)
    lst.remove(x)
```

4.1 生产者部分

```
async def factory(transfer_center, products):

    for product in products:
        prod_time = random.uniform(1, 3)
        await asyncio.sleep(prod_time)
        print(f"工厂生产: {product}")
        await transfer_center.put(product)

    await transfer_center.put(None)

    print('工厂生产完毕')
    await transfer_center.join()
```

4.1 消费者部分

```
async def supermarket(transfer_center):

    while True:

        product = await transfer_center.get()
        if product is None:
            transfer_center.task_done()
            break
        else:
            recv_time = random.uniform(1, 3)
            await asyncio.sleep(recv_time)
            print(f"超市接收：{product}")
            transfer_center.task_done()

    print('超市接收完毕')
```

4.1 消费者部分

- 不使用None在Queue中传输的写法，使用wait_for和timeout

```
# 尹奕涵
try:
    receiving_time = random.randint(1, 3)
    await asyncio.sleep(receiving_time)
    product = await asyncio.wait_for(transfer_center.get(), timeout=5)
    transfer_center.task_done()
    print('超市接收: %s' % product)
except asyncio.TimeoutError:
    break
```

4.1 关于各版本Python的协程写法

- Python 3.6 ~ Python 3.10

```
async def f(delay):
    await asyncio.sleep(delay)
    print(delay)

loop = asyncio.get_event_loop()
tasks = [f(3), f(2)]
loop.run_until_complete(asyncio.wait(tasks))
```

- ≤ Python 3.12 写法一

```
async def f(delay):
    await asyncio.sleep(delay)
    print(delay)

loop = asyncio.get_event_loop()
tasks = [asyncio.create_task(f(3)),
         asyncio.create_task(f(2))]
loop.run_until_complete(asyncio.wait(tasks))
```

4.1 关于各版本Python的协程写法

- ≤ Python 3.12 写法二

```
async def f(delay):
    await asyncio.sleep(delay)
    print(delay)

async def main():
    tasks = [asyncio.create_task(f(3)),
             asyncio.create_task(f(2))]
    await asyncio.gather(*tasks)

asyncio.run(main())
```

- 尽量避免在Jupyter notebook环境里直接运行协程程序

4.2 （附加题）异步生成器

```
async def take_products(spmarket, num, name):
    count = 0
    while count < num:

        if len(spmarket.all_products[name]) == 0:
            await restock_products(spmarket, name)
        product = spmarket.all_products[name].pop()

        count += 1
        print(f'获得第 {count:2d} 个 {name}')
        yield product
```

4.2 （附加题）异步生成器

- 错误：加了else

```
async def take_products(spmarket, num, name):
    count = 0
    while count < num:

        if len(spmarket.all_products[name])==0:
            await restock_products(spmarket, name)
        else:
            product=spmarket.all_products[name].pop(0)

        count += 1
        print(f'获得第 {count:2d} 个 {name}')
        yield product
```

```
获得第  1 个 商品2
获得第  2 个 商品2
获得第  3 个 商品2
获得第  4 个 商品2
获得第  5 个 商品2
获得第  6 个 商品2
获得第  7 个 商品2
获得第  1 个 商品1
获得第  2 个 商品1
获得第  3 个 商品1
工作人员为 商品1 补货  2 件
获得第  4 个 商品1
获得第  5 个 商品1
获得第  6 个 商品1
工作人员为 商品2 补货  1 件
获得第  8 个 商品2
获得第  9 个 商品2
工作人员为 商品1 补货  2 件
获得第  7 个 商品1
获得第  8 个 商品1
获得第  9 个 商品1
工作人员为 商品2 补货  2 件
获得第 10 个 商品2
获得第 11 个 商品2
获得第 12 个 商品2
工作人员为 商品1 补货  2 件
获得第 10 个 商品1
获得第 11 个 商品1
获得第 12 个 商品1
```