HW4-答案与评注

王瑞环

一、设计模式

1.2 工厂模式

- 参考答案：右边两种皆可
- 主要问题
  - 关于 `or` 在 `if` 判断语句中的使用方式
  - 关于抛出异常的类型

```python
def create_product(product_type):
    if product_type in ["A", "Product A"]:
        return ProductA()
    elif product_type in ["B", "Product B"]:
        return ProductB()
    else:
        raise ValueError("Invalid product type")
```

```python
def create_product(product_type):
    if product_type == "A" or product_type == "Product A":
        return ProductA()
    elif product_type == "B" or product_type == "Product B":
        return ProductB()
    else:
        raise ValueError("Invalid product type")
```

在if中使用or

- 在编程时，常常需要判断某个变量是否为多个值中的一个，在Python中常见的有以下两种写法

```
if (x == a) or (x == b) or (x == c) or ... :
    # do something

if x in [a, b, c, ...]:
    # do something
```

- 但是，以下的两种写法是***错误***的

```
if x == a or b or c or ... :
    # do something

if x == (a or b or c or ...) :
    # do something
```

在if中使用or：第一种错误

- 在Python的运算优先级中，"=="的优先级是高于"or"的，因此以下两种代码是等价的

```
if x == a or b or c or ... :
    # do something

if (x == a) or b or c or ...:
    # do something
```

- 因此，当b或c不为零/空容器/None等值时，if下面的语句必定会被执行，与x的取值无关，这并不符合我们的期望

在if中使用or：第二种错误

- 在详细解释第二种错误之前，先回顾Python中or的用法
- Python中的or是一个“短路”运算符，也即
  - x1 or x2 or ... or xn 的结果为第一个使得bool(xi)为True的**xi**
  - 如果对于所有的xi，bool(xi)都为False，则结果为**xn**
  - **总之，用 or/and 连接的表达式的值并不一定为True/False**
  - **但是使用 not 得到的值一定是True/False**
- 以下是使用or的几个例子

```
'1' or 'abc'                # 值为 '1'
0 or [] or {1: 'a'}         # 值为 {1: 'a'}
[] or None or 10 or 3 or 0  # 值为 10
0 or [] or '' or {}         # 值为 {}
```

在if中使用or：第二种错误

- 了解了or的机制后，再来看第二种错误的写法，在这里我们如果假设bool(a) = True，则以下两种代码等价

```
if x == (a or b or c or ...) :
    # do something

if x == a :
    # do something
```

- 因此，在x取值为b或c时，if下的语句仍然不会被执行，这也不符合我们期望的功能

关于Python中的异常类型

- 本次作业中我们要求在无法完成产品时抛出异常，参考答案中使用的是ValueError，提交的作业中许多同学用的是TypeError，也有直接用Exception的
- 关于Exception：这是Python中异常类的基类，并不建议在代码中直接抛出Exception，而是抛出一个具体的子类（Python中已有或自定义的异常类），这样做的一个好处是在try-except结构中针对多种异常实现不同的处理，一个简单例子如下

```
def divide_numbers(x, y):
    try:
        return = x / y
    except ZeroDivisionError:
        print("y cannot be zero")
    except TypeError:
        print("input should be numbers")
```

关于Python中的异常类型

- 关于TypeError/ValueError的区别：
  - TypeError一般用于表示某个变量的**类型**不符合要求的情况，如我们需要一个字符串，但调用函数时传递了一个数字类型的参数等
  - ValueError一般用于表示某个变量的**值**不符合要求，如本题中的情况
- 以下是一个例子，可以借此体会TypeError/ValueError的区别

```
def create_product(product_type):
    if not isinstance(product_type, str):
        raise TypeError("product_type should be a str")
    if product_type in ["A", "Product A"]:
        return ProductA()
    elif product_type in ["B", "Product B"]:
        return ProductB()
    else:
        raise ValueError("Invalid product type")
```

1.3 观察者模式/发布订阅模式

- 本次作业中写成下面这样已经足够完成功能，但额外判断增加时student是否已在列表中、删除时student是否已不在列表中并抛出可能的异常是更为严谨、也更为推荐的一种做法

```
def add_student(self, student):
    self.students.append(student)

def remove_student(self, student):
    self.students.remove(student)
```

二、正则表达式

2.1 寻找数字字符

- **解法一**：常规想法，以及大部分同学的做法，使用字符串切片

```
def find_number(s, n):
    m = re.search(r'\d',s[n:])
    if m:
        return m.group()
    return None
```

- 除了search+group，也可以用findall等其他操作，合理即可

```
def find_number(s, n):
    m = re.findall(r'\d', s[n:])
    return a[0] if a else None
```

2.1 寻找数字字符

- **解法二**：使用".{n}"匹配前n个字符，用"\D*"匹配非数字字符，最后得到用"\d"匹配的结果

```
def find_number(s, n):
    pattern = ".{" + str(n) + "}\D*\d"
    a = re.search(pattern, s)
    if a is None:
        return None
    else:
        ret = a.group()
        return ret[-1]
```

（By 曾为帅）

2.1 寻找数字字符

- **解法三**：使用非贪婪模式"*?"进行匹配

```
def find_number(s, n):
    m = re.search(f'.{{{n}}}.*?(\d)', s)
    if m == None:
        return None
    return m.group(1)
```

（By 燕骏寒）

- **注**：用非贪婪模式也可以有另一种写法"{n,}?"

```
def find_number(s, n):
    pattern = r'.{%d,}?(\d)' % n
    m = re.search(pattern, s)
    if m:
        return m.group(1)
    return None
```

2.1 寻找数字字符

- **解法四**：最初设计的参考解答，其实既不需要"\D*"也不需要非贪婪匹配还不需要字符串切片，依然可以做出这道题

```
def find_number(s, n):
    pattern = r'.{%d}(\d)' % n
    m = re.search(pattern, s)
    return m.group(1) if m else None
```

- **解法五：**使用re.compile建立re.Pattern类对象，并使用这个类的search方法，可以指定起始搜索位置

```
def find_number(s, n):
    pattern = re.compile('[0-9]', flags = 0)
    match = pattern.search(s, pos = n)
    return match.group() if match else None
```

（By 程一哲）

2.1 寻找数字字符

- **其他解法**：来自一部分（可能不太熟悉正则表达式的）同学，用循环遍历字符串、用len判断字符串长度等操作，虽然也可以解出题目，但是并未充分利用正则表达式**方便简洁***的优点

- ***方便简洁**：指调用一次match、search、findall等方法即可完成所需要求的简单**流程**；不包括设计正则模式内容时的痛苦

```python
def find_number(s, n):
    if len(s) < n:
        return None
    index = n
    while index < len(s) - 1:
        if re.match(r'\d', s[index]):
            break
        else:
            index += 1
    if re.match(r'\d', s[index]):
        return s[index]
    else:
        return None
```

2.2 学号判断

- 参考解答如下

```
def is_valid_student_id(student_id):
    if is_20_to_23:
        pattern = r'^2[0-3]000\d{5}$'
    else:
        pattern = r'^(1[89]|2[0-4])000\d{5}$'
    return bool(re.match(pattern, student_id))
```

- 其中涉及的几个知识点
  - 用^$直接匹配开头结尾，从而无需再判断len(student_id) == 10
  - 用[...]匹配被框起的字符集中的任意字符，如1[89]匹配18或19、2[0-3]匹配20、21、22或23
  - **用(A|B)C匹配模式AC或BC（不少同学用错!）**，如(1[89]|2[0-4])000可以匹配1[89]000或者2[0-4]000
  - 用A{m}匹配模式A连续的m次，如用\d{5}匹配连续五个数字，0{3}匹配三个0

2.2 学号判断

- **错误解答一**：想使用(A|B)C的模式但漏写括号，当is_20_to_23为False时会匹配出"^1[8-9]"或"2[0-4]000\d{5}$"

```
def is_valid_student_id(student_id):
    if is_20_to_23:
        pattern = r'^2[0-3]000\d{5}$'
    else:
        pattern = r'^1[8-9]|2[0-4]000\d{5}$'
    return re.match(pattern, student_id) is not None
```

- **错误解答二**：想使用(A|B)C的模式但括号写错位置

```
def is_valid_student_id(student_id):
    if is_20_to_23:
        pattern = r'^2[0-3]000\d{5}$'
    else:
        pattern = r'^(1[89])|(2[0-4])000\d{5}$'
    return re.match(pattern, student_id) is not None
```

2.2 学号判断

错误解答三：虽然看起来很有道理，但一个无法通过的例子是"2000000000"，原因在于前一个(.*)的贪婪匹配机制

```
def is_valid_student_id(student_id):
    if len(student_id) != 10:
        return False
    try:
        year = re.match("(.*)000(.*)",student_id).group(1)
        year = int(year)
        random_num = re.match("(.*)000(.*)",student_id).group(2)
        random_num = int(random_num)
        if(is_20_to_23):
            if year >= 20 and year <= 23:
                return True
            else:
                return False
        else:
            if year >= 18 and year <= 24:
                return True
            else:
                return False
    except:
        return False
    pass
```

2.2 学号判断

- 以上错误解答都能通过所有的assert，原因之一是要想出很全面的测试用例真的很难（
- 另一个原因是作业中的assert语句也有bug（但是很少有人留意到这个问题）：由于 == 的运算优先级大于 if-else，因此以下两个语句是等价的

```
assert is_valid_student_id("1800012345" ) == False if is_20_to_23 else True

assert (is_valid_student_id("1800012345") == False) if is_20_to_23 else True
```

- 这导致当is_20_to_23为False的时候，这条语句等价于"assert True"，于是起不到任何测试作用

2.3 单词提取

- 参考答案如下

```
def extract_capitalized_words(s):
    pattern = r'\b[A-Z][a-z]*\b'
    return re.findall(pattern, s)
```

- 主要希望回顾的知识点是
  - [A-Z]和[a-z]匹配大小写字母
  - *匹配任意多次
  - \b匹配单词边界

2.3 单词提取

- 当然，即使不知道\b的用法，也可以做出这道题

```
def extract_capitalized_words(s):
    ans = []
    for word in re.split('[^a-zA-Z]', s):
        if re.match('\A[A-Z][a-z]*\Z', word):
            ans.append(word)
    return ans
```

(By 赵凌哲)

```
def extract_capitalized_words(s):
    word_list_new=[]
    for word in re.findall('[A-Za-z]*', s):
        if re.match('\A[A-Z][a-z]*\Z', word):
            word_list_new.append(word)
    return word_list_new
```

(By 尹奕涵)

2.3 单词提取

- re.findall只能寻找不重叠的匹配(non-overlapping matches)，但是借助第三方库regex可以实现比re更丰富的功能，如在findall中指定overlapped参数允许查找的内容有重叠
- 但是应当指出，re作为Python标准库，通常会比第三方库更可靠、更安全，除非确实有难以实现的特殊需求，否则更建议使用re

```
def extract_capitalized_words(s):
    import regex
    return regex.findall("\W([A-Z][a-z]*)\W", " " + s + " ", overlapped=True)
```

(By 陈锐韬)

2.3 单词提取

- 如果了解正则表达式中一些更高级的用法（零宽断言），也可以手动实现类似\b的效果
- 使用?=的写法

```
def extract_capitalized_words(s):
    res = re.findall("(?<=\W)[A-Z][a-z]*(?=\W)", (" " + s + " "))
    return res
```

（By 崔鹤龄）

- 使用?!的写法（此时s前后无需加额外空格）

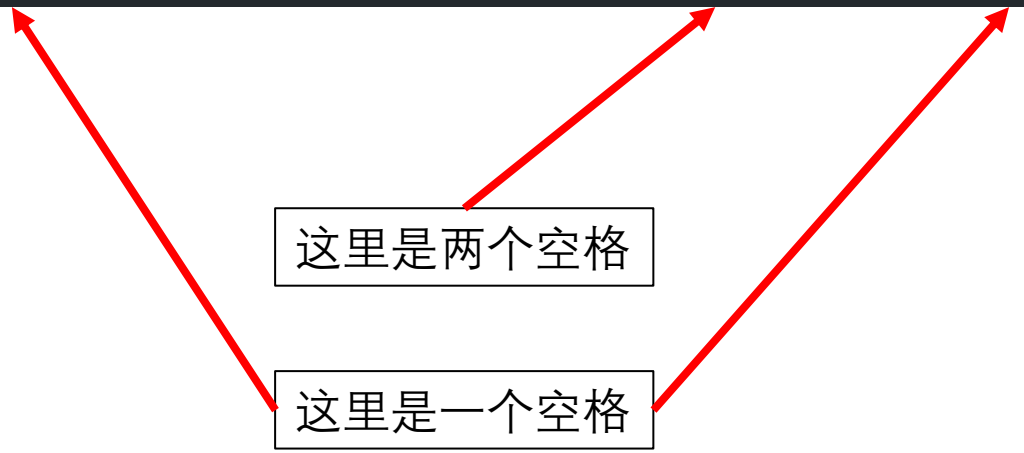
```
def extract_capitalized_words(s):
    pattern = r"((?<!\w)[A-Z][a-z]*(?!\w))"
    return re.findall(pattern, s)
```

（By 陈品璋）

2.3 单词提取

- 还有一种讨巧的方式：既然一个非字母字符无法同时满足左右两边的匹配要求，那就把它变成连续的两个

```
def extract_capitalized_words(s):
    pattern = r'[^A-Za-z]([A-Z][a-z]*)[^A-Za-z]'
    return re.findall(pattern, ' ' + re.sub(r'[^A-Za-z]+', '  ', s) + ' ')
```



2.4 多余空格去除

- 参考答案：以下三种皆可

```python
def remove_multiple_spaces(s):
    return re.sub(r'\s+', ' ', s)

def remove_multiple_spaces(s):
    return re.sub(r'( |\n|\t|\r)+', ' ', s)

def remove_multiple_spaces(s):
    return re.sub(r'[ \n\t\r]+', ' ', s)
```

- 但还是希望大家能尽量掌握\s代表空白字符的知识点

2.4 多余空格去除

- 部分同学精准地捕捉到了一个语文上的bug并给出了更符合题意的解法

给定一个字符串`s`，使用**正则表达式相关操作**将其中的多个连续的空字符(包括空格、换行符`\n`、制表符`\t`、回车符`\r`)替换为单个空格

```
def remove_multiple_spaces(s):
    pattern = r'( |\n|\t|\r){2,}'
    return re.sub(pattern, ' ', s)
```

（By 唐宇）

```
def remove_multiple_spaces(s):
    return re.sub(r"\s\s+", r" ", s)
```

（By 彭子朔）

2.5 字母轮换

- 参考答案如下

```
def shift_characters(s):

    def shift_letter(m):
        letter = m.group()
        if letter != 'Z':
            return chr(ord(letter) + 1)
        else:
            return 'A'

    return re.sub(r'[A-Z]', shift_letter, s)
```

- 主要的知识点re.sub的第二个参数除了可以传入字符串外，还可以传入一个接受re.Match类参数并返回一个str的函数

2.5 字母轮换

- 也可以用lambda表达式来写

```
def shift_characters(s):
    return re.sub(r"[A-Z]", lambda x: chr(ord(x.group()) + 1) if x.group() != 'Z' else 'A', s)
```

- 但通常这不会让阅读者感到愉快(`へ ´*)ノ