

# 实验四

## 实验四 Python字典和while循环

班级： 21计科02班

学号： B20230302210

姓名： 陶鑫

Github地址： <https://github.com/Muyu-ikun>

CodeWars地址： <https://www.codewars.com/users/Muyu-ikun>

---

## 实验目的

1. 学习Python字典
2. 学习Python用户输入和while循环

## 实验环境

1. Git
2. Python 3.10
3. VSCode
4. VSCode插件

## 实验内容和步骤

### 第一部分

Python列表操作

完成教材《Python编程从入门到实践》下列章节的练习：

- 第6章 字典
  - 第7章 用户输入和while循环
-

## 第二部分

在[Codewars网站](https://www.codewars.com/)注册账号，完成下列Kata挑战：

---

### 第一题：淘气还是乖孩子（Naughty or Nice）

难度： 7kyu

圣诞老人要来镇上了，他需要你帮助找出谁是淘气的或善良的。你将会得到一整年的JSON数据，按照这个格式：

```
{
  January: {
    '1': 'Naughty', '2': 'Naughty', ..., '31': 'Nice'
  },
  February: {
    '1': 'Nice', '2': 'Naughty', ..., '28': 'Nice'
  },
  ...
  December: {
    '1': 'Nice', '2': 'Nice', ..., '31': 'Naughty'
  }
}
```

你的函数应该返回 "Naughty!"或 "Nice!", 这取决于在某一年发生的总次数（以较大者为准）。如果两者相等，则返回 "Nice! "。

代码提交地址：

<https://www.codewars.com/kata/5662b14e0a1fb8320a00005c>

提交源代码如下：

```
{

def naughty_or_nice(data):

    Naughty_count = 0

    Nice_count = 0

    for month_data in data.values():

        for day,behavior in month_data.items():

            if behavior == 'Naughty':

                Naughty_count +=1

            elif behavior == 'Nice':

                Nice_count +=1

    if Naughty_count > Nice_count:

        return 'Naughty!'

    else:

        return 'Nice!'

# 示例数据

year_data = {

    'January': {'1': 'Naughty', '2': 'Naughty', '31': 'Nice'},

    'February': {'1': 'Nice', '2': 'Naughty', '28': 'Nice'},

    # ... 其他月份的数据
```

```
'December': {'1': 'Nice', '2': 'Nice', '31': 'Naughty'}

}

result = naughty_or_nice(year_data)

print(result)

}
```

---

## 第二题： 观察到的PIN (The observed PIN)

难度：4kyu

好了，侦探，我们的一个同事成功地观察到了我们的目标人物，抢劫犯罗比。我们跟踪他到了一个秘密仓库，我们认为在那里可以找到所有被盗的东西。这个仓库的门被一个电子密码锁所保护。不幸的是，我们的间谍不确定他看到的密码，当罗比进入它时。

键盘的布局如下：

```
┌───┬───┬───┐
│ 1 │ 2 │ 3 │
├───┴───┴───┤
│ 4 │ 5 │ 6 │
├───┴───┴───┤
│ 7 │ 8 │ 9 │
├───┴───┴───┤
│ 0 │
```

他注意到密码1357，但他也说，他看到的每个数字都有可能是另一个相邻的数字（水平或垂直，但不是对角线）。例如，代替1的也可能是2或4。而不是5，也可能是2、4、6或8。

他还提到，他知道这种锁。你可以无限制地输入错误的密码，但它们最终不会锁定系统或发出警报。这就是为什么我们可以尝试所有可能的（\*）变化。

\*可能的意义是：观察到的PIN码本身和考虑到相邻数字的所有变化。

你能帮助我们找到所有这些变化吗？如果有一个函数，能够返回一个列表，其中包含一个长度为1到8位的观察到的PIN的所有变化，那就更好了。我们可以把这个函数命名为getPINs（在python中为get\_pins，在C#中为GetPINs）。

但请注意，所有的PINs，包括观察到的PINs和结果，都必须是字符串，因为有可能会有领先的"0"。我们已经为你准备了一些测试案例。

侦探，我们就靠你了！

代码提交地址：

<https://www.codewars.com/kata/5263c6999e0f40dee200059d>

提交源代码如下：

```
def get_pins(observed):  
  
    # 定义数字之间的替代关系  
  
    neighbors = {  
  
        '0': ['0', '8'],  
  
        '1': ['1', '2', '4'],  
  
        '2': ['1', '2', '3', '5'],  
  
        '3': ['2', '3', '6'],  
  
        '4': ['1', '4', '5', '7'],  
  
        '5': ['2', '4', '5', '6', '8'],  
  
    }
```

```
        '6': ['3', '5', '6', '9'],

        '7': ['4', '7', '8'],

        '8': ['5', '7', '8', '9', '0'],

        '9': ['6', '8', '9']

    }

    result = ['']

    for digit in observed:

        result = [prefix + neighbor for prefix in result for neighbor in
neighbors[digit]]

    return result
```

---

### 第三题： RNA到蛋白质序列的翻译（RNA to Protein Sequence Translation）

难度：6kyu

蛋白质是由DNA转录成RNA，然后转译成蛋白质的中心法则。RNA和DNA一样，是由糖骨架（在这种情况下是核糖）连接在一起的长链核酸。每个由三个碱基组成的片段被称为密码子。称为核糖体的分子机器将RNA密码子转译成氨基酸链，称为多肽链，然后将其折叠成蛋白质。

蛋白质序列可以像DNA和RNA一样很容易地可视化，作为大字符串。重要的是要注意，“停止”密码子不编码特定的氨基酸。它们的唯一功能是停止蛋白质的转译，因此它们不会被纳入多肽链中。“停止”密码子不应出现在最终的蛋白质序列中。为了节省您许多不必要（和乏味）的键入，已为您的氨基酸字典提供了键和值。

给定一个RNA字符串，创建一个将RNA转译为蛋白质序列的函数。注意：测试用例将始终生成有效的字符串。

```
protein ('UGCGAUGAAUGGGCUCGCUCC')
```

将返回 CDEWARS

作为测试用例的一部分是一个真实世界的例子！最后一个示例测试用例对应着一种叫做绿色荧光蛋白的蛋白质，一旦被剪切到另一个生物体的基因组中，像GFP这样的蛋白质可以让生物学家可视化细胞过程！

## Amino Acid Dictionary

```
# Your dictionary is provided as PROTEIN_DICT

PROTEIN_DICT = {

    # Phenylalanine

    'UUC': 'F', 'UUU': 'F',

    # Leucine

    'UUA': 'L', 'UUG': 'L', 'CUU': 'L', 'CUC': 'L', 'CUA': 'L', 'CUG': 'L',

    # Isoleucine

    'AUU': 'I', 'AUC': 'I', 'AUA': 'I',

    # Methionine

    'AUG': 'M',

    # Valine

    'GUU': 'V', 'GUC': 'V', 'GUA': 'V', 'GUG': 'V',

    # Serine

    'UCU': 'S', 'UCC': 'S', 'UCA': 'S', 'UCG': 'S', 'AGU': 'S', 'AGC': 'S',

    # Proline

    'CCU': 'P', 'CCC': 'P', 'CCA': 'P', 'CCG': 'P',
```

# Threonine

'ACU': 'T', 'ACC': 'T', 'ACA': 'T', 'ACG': 'T',

# Alanine

'GCU': 'A', 'GCC': 'A', 'GCA': 'A', 'GCG': 'A',

# Tyrosine

'UAU': 'Y', 'UAC': 'Y',

# Histidine

'CAU': 'H', 'CAC': 'H',

# Glutamine

'CAA': 'Q', 'CAG': 'Q',

# Asparagine

'AAU': 'N', 'AAC': 'N',

# Lysine

'AAA': 'K', 'AAG': 'K',

# Aspartic Acid

'GAU': 'D', 'GAC': 'D',

# Glutamic Acid

'GAA': 'E', 'GAG': 'E',

# Cystine

'UGU': 'C', 'UGC': 'C',

# Tryptophan

'UGG': 'W',

# Arginine



```

'CGU': 'R', 'CGC': 'R', 'CGA': 'R', 'CGG': 'R', 'AGA': 'R', 'AGG': 'R',

# Glycine

'GGU': 'G', 'GGC': 'G', 'GGA': 'G', 'GGG': 'G',

# Stop codon

'UAA': 'Stop', 'UGA': 'Stop', 'UAG': 'Stop'

}

```

代码提交地址：

<https://www.codewars.com/kata/555a03f259e2d1788c000077>

提交源码如下：

```

def protein(rna):

    # your code here

    protein = ""

    i = 0

    while i < len(rna):

        codon = rna[i:i+3]

        amino_acid = PROTEIN_DICT.get(codon, None)

        if amino_acid == "Stop":

            break

        if amino_acid:

            protein += amino_acid

        i += 3

```

```
return protein
```

---

## 第四题： 填写订单 (Thinkful - Dictionary drills: Order filler)

难度：8kyu

您正在经营一家在线业务，您的一天中很大一部分时间都在处理订单。随着您的销量增加，这项工作占用了更多的时间，不幸的是最近您遇到了一个情况，您接受了一个订单，但无法履行。

您决定写一个名为 `fillable()` 的函数，它接受三个参数：一个表示您库存的字典 `stock`，一个表示客户想要购买的商品的字符串 `merch`，以及一个表示他们想购买的商品数量的整数 `n`。如果您有足够的商品库存来完成销售，则函数应返回 `True`，否则应返回 `False`。

有效的数据将始终被传入，并且 `n` 将始终大于等于 1。

代码提交地址：

<https://www.codewars.com/kata/586ee462d0982081bf001f07/python>

提交源代码如下：

```
def fillable(stock, merch, n):  
  
    # Your code goes here.  
  
    if merch in stock:  
  
        stock_quantity = stock[merch]  
  
        return stock_quantity >= n  
  
    else :  
  
        return False
```

---

## 第五题： 莫尔斯码解码器 (Decode the Morse code, advanced)

难度： 4kyu

在这个作业中，你需要为有线电报编写一个莫尔斯码解码器。

有线电报通过一个有按键的双线路运行，当按下按键时，会连接线路，可以在远程站点上检测到。莫尔斯码将每个字符的传输编码为"点"（按下按键的短按）和"划"（按下按键的长按）的序列。

在传输莫尔斯码时，国际标准规定：

- "点" - 1个时间单位长。
- "划" - 3个时间单位长。
- 字符内点和划之间的暂停 - 1个时间单位长。
- 单词内字符之间的暂停 - 3个时间单位长。
- 单词间的暂停 - 7个时间单位长。

但是，该标准没有规定"时间单位"有多长。实际上，不同的操作员会以不同的速度进行传输。一个业余人士可能需要几秒钟才能传输一个字符，一位熟练的专业人士可以每分钟传输60个单词，而机器人发射器可能会快得多。

在这个作业中，我们假设消息的接收是由硬件自动执行的，硬件会定期检查线路，如果线路连接（远程站点的按键按下），则记录为1，如果线路未连接（远程按键弹起），则记录为0。消息完全接收后，它会以一个只包含0和1的字符串的形式传递给你进行解码。

例如，消息 `HEYJUDE`，即 ..... 可以如下接收：

```
11001100110011000000110000001111110011001111110011111100000000000000110011111100111
111001111111000000110011001111110000001111110011001100000011
```

如您所见，根据标准，这个传输完全准确，硬件每个"点"采样了两次。

因此，你的任务是实现两个函数：

函数 `decodeBits(bits)`，应该找出消息的传输速率，正确解码消息为点 (.)、划 (-) 和空格（字符之间有一个空格，单词之间有三个空格），并将它们作为一个字符串返回。请注意，在消息的开头和结尾可能会出现一些额外的0，确保忽略它们。另外，如果你无法分辨特定的1序列是点还是划，请假设它是一个点。

函数 `decodeMorse(morseCode)`，它将接收上一个函数的输出，并返回一个可读的字符串。

注意：出于编码目的，你必须使用ASCII字符.和-，而不是Unicode字符。

莫尔斯码表已经预加载给你了（请查看解决方案设置，以获取在你的语言中使用它的标识符）。

```
morseCodes(".--") #to access the morse translation of ".--"
```

下面是Morse码支持的完整字符列表：

A	..-
B	-... .
C	-... .
D	-... .
E	.
F	.... .
G	--. .
H	.... .
I	.. .
J	.--- .
K	-. .
L	.... .
M	-- .
N	-. .
O	--- .
P	.... .
Q	--- .
R	-. .

S . . .

T -

U . . -

V . . . -

W . - -

X - . . -

Y - . - -

Z - - . .

0 - - - - -

1 . - - - -

2 . . - - -

3 . . . - -

4 . . . . -

5 . . . . .

6 - . . . .

7 - - . . .

8 - - - . .

9 - - - - .

. . - . - . -

, - - . . - -

? . . - . . .

' . - - - - .

! - . - . - -

/     - . . . .  
(     - . - - .  
)     - . - - -  
&     . - . . .  
:     - - - . . .  
;     - . - - .  
=     - . . . -  
+     . - . - .  
-     - . . . . -  
\_     . . - - -  
"     . - . . .  
\$     . . . . . -  
@     . - . . .

代码提交地址：

<https://www.codewars.com/kata/decode-the-morse-code-advanced>

### 第三部分

使用Mermaid绘制程序流程图

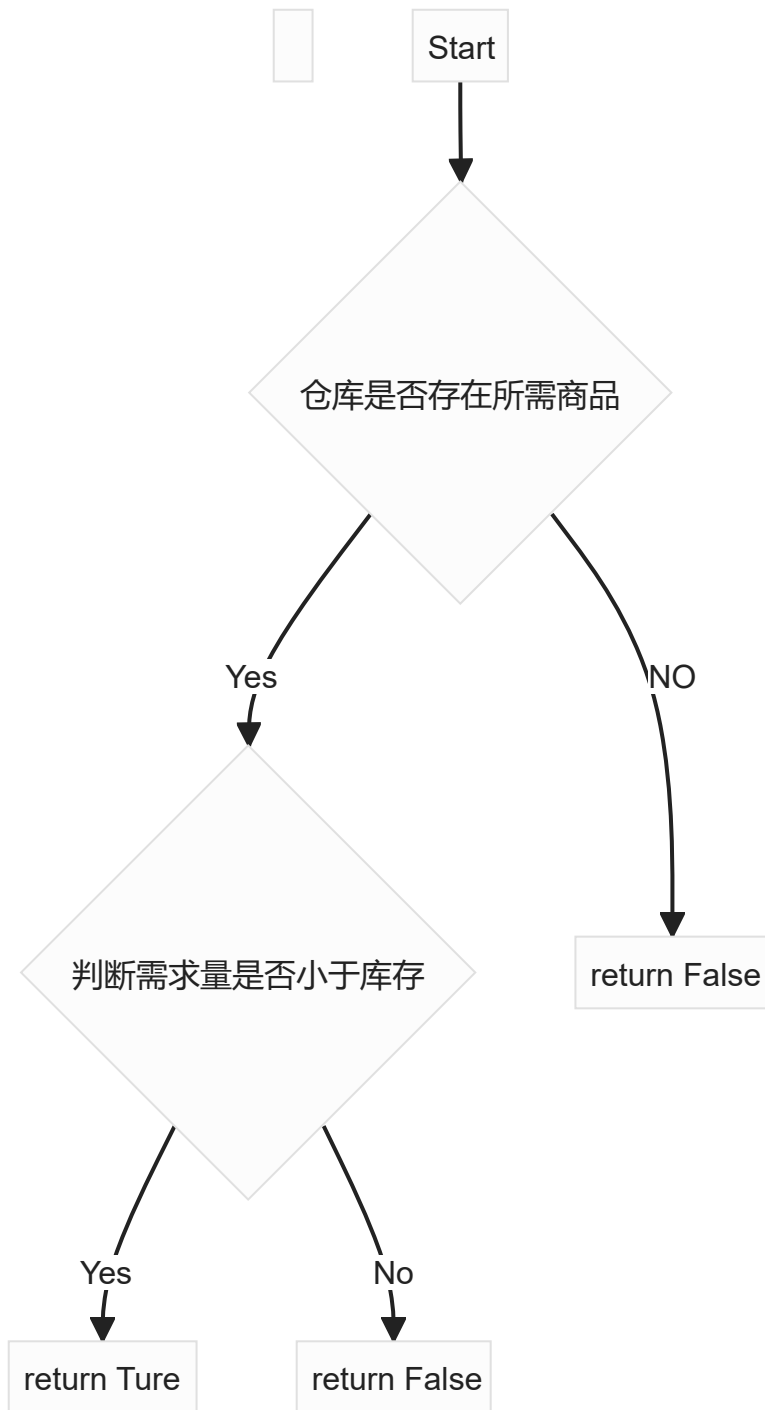
安装VSCode插件：

- Markdown Preview Mermaid Support
- Mermaid Markdown Syntax Highlighting

使用Markdown语法绘制你的程序绘制程序流程图（至少一个）， Markdown代码如下：

"/Experiments/img/2023-08-05-22-00-00.png" is not created yet. Click to create.

显示效果如下：



查看Mermaid流程图语法-->[点击这里](#)

## 实验考查

请使用自己的语言并使用尽量简短代码示例回答下面的问题，这些问题将在实验检查时用于提问和答辩以及实际的操作。

1. 字典的键和值有什么区别？

键是字典中的标识符，用于唯一标识每个值。

键通常是不可变的，通常是字符串、整数、元组等数据类型，但不能是可变的数据类型，如列表。

字典中的键必须是唯一的，不能重复，因为它们用于检索和标识与之关联的值。

键是字典中的索引，用于访问对应的值。

值是与字典中的键关联的数据，可以是任何数据类型，包括整数、字符串、列表、字典、自定义对象等。

字典中的值可以重复，多个键可以关联相同的值。

值存储在字典中，与键一起构成键值对，用于实现数据的关联和检索。

2. 在读取和写入字典时，需要使用默认值可以使用什么方法？

在Python中，可以使用 `setdefault()` 方法或 `defaultdict` 类来设置字典的默认值。

**`setdefault()` 方法：** `setdefault()` 方法是字典对象的方法，用于设置字典中键的默认值。如果键存在，则返回键对应的值；如果键不存在，则设置该键的值为指定的默认值，并返回默认值。它的语法如下：

```
value = my_dict.setdefault(key, default_value)
```

示例：

```
my_dict = {'a': 1, 'b': 2}

value_a = my_dict.setdefault('a', 0) # 返回1，因为键'a'已存在

value_c = my_dict.setdefault('c', 0) # 返回0，因为键'c'不存在，设置默认值为0
```

1. **`defaultdict` 类：** `collections` 模块中的 `defaultdict` 类允许你为字典的每个键设置默认值的数据结构。在创建 `defaultdict` 时，你需要指定一个默认值工厂（一个函数或数据类型），用于指定新键的默认值。示例：

```
from collections import defaultdict
```



```
my_dict = defaultdict(int) # 创建一个默认值为0的字典

my_dict['a'] = 1 # 此时键'a'的值为1

value_b = my_dict['b'] # 此时键'b'不存在，但它的默认值为0
```

### 3. Python中的while循环和for循环有什么区别？

while 循环用于需要根据条件重复执行的情况，条件在循环开始前检查。

for 循环用于已知需要遍历序列中的元素并执行循环体的情况。

### 4. 阅读[PEP 636 – Structural Pattern Matching: Tutorial](#)，总结Python 3.10中新出现的match语句的使用方法。

以下是Python 3.10中新出现的 `match` 语句的主要特点和使用方法的总结：

1. **模式匹配：** `match` 语句允许你根据不同的模式匹配执行不同的代码块，类似于多个 `if-elif-else` 块的结构。这可以帮助编写更清晰、更简洁的代码。
2. **模式：** 模式是一种用于匹配数据的结构，可以包括常量、变量、序列、映射等。模式可以包括通配符 (`_`) 来匹配任何值。例如：

```
```python
```

```
match value:
```

```
    case 1:
```

```
        # 匹配值为1的情况
```

```
    case "hello":
```

```
        # 匹配值为"hello"的情况
```

```
    case (x, y):
```

```
        # 匹配一个包含两个元素的元组
```

```
    case {"name": name}:
```

```
        # 匹配一个包含键"name"的字典
```

```
case _:  
    # 匹配任何其他情况
```

```
...
```

3. **case 子句**: `match` 语句包括一个或多个 `case` 子句, 每个子句对应一个模式和要执行的代码块。 `case` 子句按顺序依次匹配, 一旦找到匹配的模式, 就执行相应的代码块。

4. **as 模式**: `as` 模式用于将匹配的值绑定到变量, 以便在代码块中使用。例如:

```
```python  
  
match value:  
  
    case (x, y) as point:  
  
        # 在代码块中可以使用point来引用元组(x, y)
```

```
...
```

5. **| 操作符**: `|` 操作符用于匹配多个模式中的任何一个。例如:

```
```python  
  
match value:  
  
    case 1 | 2:  
  
        # 匹配值为1或2的情况
```

```
...
```

6. **类型模式**: `match` 语句支持类型模式, 用于匹配对象的类型。例如:

```
```python  
  
from typing import List  
  
match value:  
  
    case int:  
  
        # 匹配整数类型  
  
    case List[int]:  
  
        # 匹配整数列表
```

...

7. **递归模式**：`match` 语句允许你使用递归模式来匹配嵌套数据结构，例如树形结构。
8. **绑定变量**：`match` 语句允许在匹配模式中绑定变量，并将其用于后续的模式匹配和代码块执行。

总之，`match` 语句是Python 3.10引入的一项强大的功能，它可以帮助简化复杂的条件分支逻辑，使代码更易读、更可维护。通过合理使用模式匹配，可以处理多种不同的情况，使代码更具表现力和可读性。

## 实验总结

总结一下这次实验你学习和使用到的知识，例如：编程工具的使用、数据结构、程序语言的语法、算法、编程技巧、编程思想。

这次试验我学会了python中字典和键的具体用法，感觉python用来处理数据很方便，代码量很小。