实验四 Python字典和while循环

班级： 21计科1

学号： 202302200000

姓名： 张三

Github地址：https://github.com/yourusername/python_course

CodeWars地址：https://www.codewars.com/users/yourusername

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

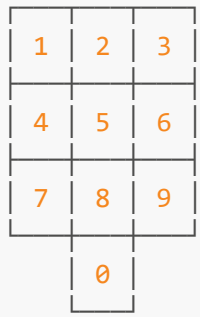
---

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：



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

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

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

---

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：

```
1100110011001100000011000000111111001100111111001111110000000000000011001111110011
1111001111110000001100110011111100000011111100110011000000011
```

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

Coderwars 挑战

第一题： 淘气还是乖孩子（Naughty or Nice）

```
def naughty_or_nice(data):
    nice = 0
    for month in data:
        for day in data[month]:
            if data[month][day] == "Nice":
                nice = nice +1
            else :
                nice = nice -1
    return "Nice!" if nice >=0 else "Naughty!"
```

算法思路：

遍历data字典计数统计“Naughty”和“Nice”出现的次数，较大的则返回,若相等返回“Nice“

第二题： 观察到的PIN（The observed PIN）

```
def get_pins(observed):
   PIN_DiCT = {   #1234
   '1':['1','2','4'],
   '2':['2','1','5','3'],
   '3':['3','2','6'] ,
   '4':['4','1','5','7'],
   '5':['5','2','4','8','6'],
   '6':['6','3','5','9'],
   '7':['7','4','8'],
   '8':['8','7','9','5','0'],
   '9':['9','6','8'],
   '0':['0','8']
   }
   pins = [""]
   cnt = 0
   for obs in observed:
      new_pins = []
      for adj in PIN_DiCT[obs]:
         for pin in pins:
            new_pins.append(pin + adj)
      pins = new_pins
   return pins
```

算法思路：

创建一个PIN_DICT字典，把每个数字相邻的数字以及本身存入字典的‘值’（value）中,遍历observed（由数字组成的字符串）中的每一位，每一位数字都是键值，都可以在字典中找到都对应一个”值“（数组），然后将'值'追加进数组new_pins[],每一次循环结束刷新pins[]保存new.pins[]中的值,同时将new_pins[]置为空，循环结束返回pins

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

```
def protein(rna):
    # your code here
    #每次取大小为3的字符串
    cnt = 0
    length =  int(len(rna)/3)
    print(length)
    rna_str = []
    j = 0
    for i in range(0,length):  #从0开始
        array = []
        for y in range(0,3):
            array.append(rna[j])
            j= j + 1
        print(array)
        rna_portion = "".join(array)
        print(rna_portion)
        if PROTEIN_DICT[rna_portion] == 'Stop' :
            rna_str.append("")
            break
        else:
            rna_str.append(PROTEIN_DICT[rna_portion])
        str_rna = ''.join(rna_str)
    return str_rna
```

算法思路：

每次只取rna字符串中的三个字符，将三个字符组成新的字符串作为‘键值’，然后在蛋白质编译字典中找对应的‘值’，将这个‘值’追加进字符数组 rna_str[],循环以上步骤，最终得到编译后的蛋白质序列，利用''.join() 返回字符串 rna_str

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

```
def fillable(stock, merch, n):
    # Your code goes here.
    if merch not in stock:
        return False
    if stock[merch] >= n and n >= 0:
        return True
    else:
        return False
```

算法思路：

通过键值找对应的值，通过商品找对应的库存量，如果需求量N大于库存则返回False,如果需求量小于等于库存量则返回True,如果找不到该商品则返回False

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

```
def min_union(lists,bits,word):
    cnt = 0
    for bit in bits:
        if bit == word :
            cnt += 1
        else:
            if cnt !=0 :
                lists.append(cnt)
                cnt = 0
    if cnt!=0:
        lists.append(cnt)
    print(lists)
    return lists
def ReturnMinNum(A,B):
    if A >= B:
        return B
    if A <= B :
        return A
def decode_bits(bits):
    zero_list = []
    one_list  = []
    bits = bits.strip('0')
    #获取列表
    if '0' not in bits:
        minunit = len(bits)
        print(minunit)
    else:
        zero_list = min_union(zero_list,bits,'0')
        one_list =  min_union(one_list,bits,'1')
    #获取两个列表的最小单位
        min_zero = min(zero_list)
        min_one  = min(one_list)
    #求最小单位
        minunit = ReturnMinNum(min_one,min_zero)

    print(minunit)
    # ToDo: Accept 0's and 1's, return dots, dashes and spaces
    return bits.replace('111'*minunit, '-
').replace('0000000'*minunit,'/').replace('000'*minunit, ' ').replace('1'*minunit,
'.').replace('0'*minunit, '')

def decode_morse(morseCode):
    # ToDo: Accept dots, dashes and spaces, return human-readable message
    print(morseCode)
    letter_str = []
    str = []
    if morseCode == '':
        return ''
    morseCode = morseCode.strip()
    print(morseCode)
```

```
    for mcode in morseCode:
        if mcode != ' ':
            letter_str.append(mcode)
        if mcode == ' ':
            str.append(MORSE_CODE[''.join(letter_str)]) #添加到数组里
            #print(letter_str)
            letter_str = []     #然后重置
        if mcode == '/':
            letter_str[-1] =''
            str.append(MORSE_CODE[''.join(letter_str)])
            str.append(' ')
            #print(letter_str)
            letter_str =[]
    str.append(MORSE_CODE[''.join(letter_str)])
    return ''.join(str)
```

算法思路：

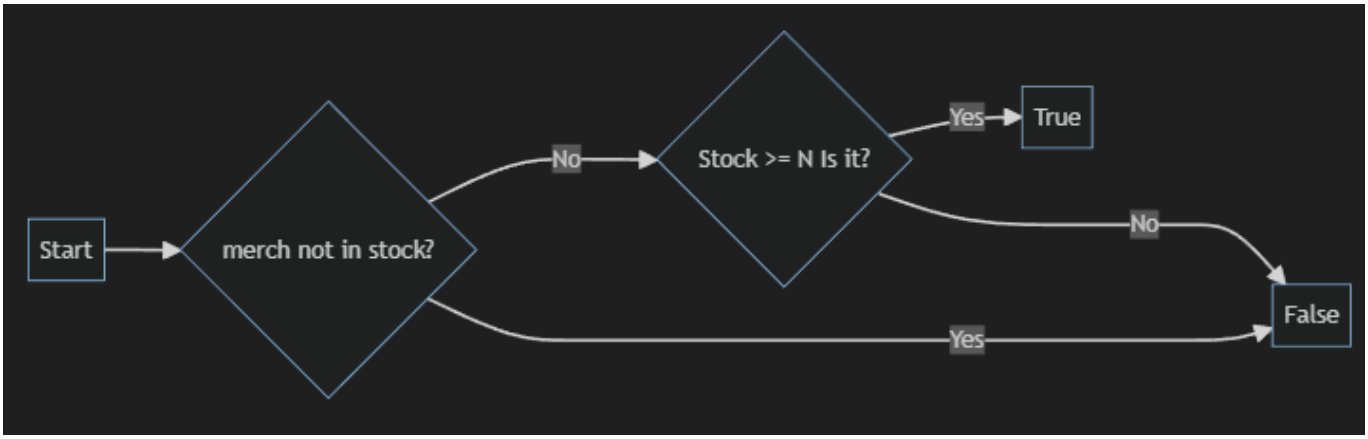
先通过strip()方法去除bits中多余干扰的'0'，将bits分为0和1的长度大小的数组，求两个最小的数组中的 最小单位，通过'最小单位' minunit 按照一定规则将bits解码为莫斯密码，然后返回。最后通过调用decode_morseCode() 函数来对莫斯密码进行解码：循环遍历morseCode，通过在字典中找到对应的值，存入字符数组循环结束将字符数组转化为字符串返回最终解码出的可读信息

Mermaid 流程图

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

Mermaid流程图显示效果如下：

```
flowchart LR
    B{Stock >= N Is it?} -->|Yes| C[True]
    A[Start] --> E{merch not in stock?}
    E ---> |No| B
    E ---> |Yes| D
    B ---->|No| D[False]
```



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?
   字典相当于保存了两组数据,其中一组数据是关键数据,被称为 key;另一组数据可通过 key 来访问,被称为 value。key是十分关键的数据,而且访问value的时候需要通过key进行访问,因此字典的key不允许重复

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?
   可以使用 dict.get(key, default) 方法来读取字典中的值，并且如果键不存在于字典中，它可以返回一个默认值。如果键存在于字典中，则返回对应的值；如果键不存在，则返回默认值。

3. Python中的while循环和for循环有什么区别?
   控制条件：while 循环的执行取决于一个条件表达式的真假，只要条件为真，循环将一直执行。而 for 循环则是基于一个可迭代对象（如列表、元组、字符串或迭代器）的元素进行迭代，每次循环从可迭代对象中取出一个元素，直到取完所有元素为止。

   迭代次数：使用 while 循环时，你需要自己管理循环的迭代次数，通常需要在循环体内更新迭代条件，否则可能导致无限循环。而 for 循环会自动遍历可迭代对象中的所有元素，无需手动管理迭代次数。

   循环变量：在 while 循环中，你需要在循环体外定义并初始化一个循环变量，然后在循环体内对其进行操作和更新。而在 for 循环中，循环变量会自动被定义，并且每次循环都会自动从可迭代对象中获取一个元素赋值给循环变量。

   适用场景：while 循环通常用于需要根据条件动态控制循环的情况，例如根据用户输入的内容判断是否继续循环。而 for 循环适用于已知要迭代的元素集合的情况，可以更方便地遍历和处理集合中的每个元素。

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。

实验总结

在学习 Python 字典的过程中，我掌握了如何使用字典进行键值对的存储和访问。字典是一种无序的数据结构，可以通过键来快速查找和修改对应的值。我了解了字典的基本操作，如创建字典、添加和删除键值对、获取值、遍历字典等。在 Codewars 上的挑战题目帮助我巩固了这些知识，并提升了我在字典操作上的熟练度。字典在 Python 中被广泛应用，对于处理键值关系的场景非常有用。