目录

12 练习—密码生成器

更新时间：2019-09-09 10:32:53

> 天才就是长期劳动的结果。
>
> ——牛顿

学习编程一定要多加练习，只靠单纯地阅读是无法真正掌握编程方法的，只有反复练习才能真正领悟编程思想。我们已经学习了一些 Python 知识了，说多不多说少也不少，是时候来运用一下了。

开始之前我们先来聊聊账号的话题。当今互联网十分普及，大家一定注册了很多 APP 和网站吧，大大小小的账号少则十几个多则可能数十个。大家的密码是都怎么设置的呢，所有账号用的是同一个密码吗？

所有账号用同一个密码是件很危险的事，一个平台上的账号泄漏了，有可能殃及其它平台。安全的做法是每个平台使用单独的密码，并且密码间的关联性尽可能的小，这样就算一个密码泄漏了也不会将影响扩大。

每个平台都使用一个单独的密码，并且密码间的关联性尽要可能的小，那十几个甚至几十个平台的密码要怎么来取呢？我们可以用密码自动生成器呀，现在就来动手做一个！

密码生成器要求

我们对密码生成器的要求是：

1. 密码需要随机生成
2. 至少包含一个大写字母（A ~ Z）
3. 至少包含一个小写字母（a ~ z）
4. 至少包含一个数字（0 ~ 9）
5. 至少包含一个特殊字符（~!@#$%^&*）
6. 长度可以自由设置，范围是 8 ~ 20

实现思路

要求有了，怎么来实现呢？

实现方法非常多，不同的人有不同的思路。在这里我们一起来分析吧。

- 首先，随机生成 N 位密码——换一种角度这其实相当于，准备好大写字母集合、小写字母集合、数字集合和特殊字符集合，从中随机挑出 N 个字符，然后将它们排成一排。你看，这样我们不就将一个笼统的需求转化成了可以用编程来解决的实际问题了吗？
- 其次，密码至少要包含一个大写字母、一个小写字母、一个数字、一个特殊字符，且可指定密码长度——要满足这个要求，有个简单的办法，我们从头开始，密码第一位放大写字母，第二位放小写字母，第三位放数字，第四位放特殊字符，剩余的 `N - 4` 个字符就依次放任意的字符。
- 再次，要解决从字符集合中随机取字符的问题——我们之前学习过 `random.randint()` 函数，它可以随机生成一个数字，我们就将这个随机数字当作索引去字符集合中取值（字符集合可以是 `str` 或 `list` 形式），这样就达到了随机从字符集合中取字符的目的。
- 最后，通过命令行交互接收密码长度，这个比较简单，使用 `input()` 即可。

实现

经过刚才的分析，我们可以将这个程序划分为三个主要部分：

1. 命令行交互部分：进行命令行交互、接收输入参数
2. 随机字符生成部分：可随机生成一个大写字母，或一个小写字母，或一个数字，或一个特殊字符，或一个任意字符
3. 密码逻辑部分：按照密码长度，操作「随机字符生成部分」来生成全部密码

这三个部分各司其职，共同构成我们的密码生成器。

我们完全可以只用我们之前学习过的那些知识，来实现这三个部分，并完整地构建整个程序。

命令行交互部分的实现

程序被执行后，首先给出提示信息要求用户指定密码长度，然后接收用户所输入的值，并判断值是否符合要求。

实现如下：

```
password_length = input('请输入密码长度（8~20）：')
password_length = int(password_length)

if password_length < 8 or password_length > 20:
    raise ValueError('密码长度不符')
```

获取到密码长度以后，就该使用「密码逻辑部分」来进一步完成工作，在这里我们把「密码逻辑部分」封装成一个函数，只需调用它就可以获取到想要的密码。也就是下面代码中的 `generate_password()` 函数。

```
password = generate_password(password_length)
print(password)
```

密码逻辑部分的实现

「密码逻辑部分」是一个函数，它以密码长度作为参数，返回我们所要的随机密码。

它生成密码的策略是，先随机生成一个大写字母，以此作为起始密码；再生成一小写字母，追加到密码末尾；再生成一个数字，追加到密码末尾；再生成一个特殊字符，追加到密码末尾。这样就拥有 4 位密码了，且满足包含大写字母、小写字母、数字、特殊字符的要求。密码剩余的几位，依次随机取任意字符并追加到密码末尾。

上述生成随机字符的功能将由「随机字符生成部分」提供，我们将「随机字符生成部分」封装成 `RandomChar` 类，并单独放置在 `randomchar` 模块中。使用 `RandomChar` 类对象的方法即可获取随机字符：

- 获取大写字母： `random_char.uppercase()`
- 获取小写字母： `random_char.lowercase()`
- 获取数字： `random_char.digit()`
- 获取特殊字符： `random_char.special()`
- 获取上述任意一种字符： `random_char.anyone()`

无需在这个层面上关心 `RandomChar` 类对象是怎么做到获取随机字符的，对当前这个部分来讲这并不重要，重要的是如何运用其它部分的能力来达到当前部分的目的。

我们来实现整个「密码逻辑部分」：

```
def generate_password(length):
    if length < 4:
        raise ValueError('密码至少为 4 位')

    random_char = randomchar.RandomChar()

    password  = random_char.uppercase()  # 用一个随机的大写字符作为起始密码
    password += random_char.lowercase()  # 将一个随机的小写字符拼接在密码末尾
    password += random_char.digit()      # 将一个随机的数字拼接在密码末尾
    password += random_char.special()    # 将一个随机的特殊字符拼接在密码末尾

    count = 5  # 此时的密码长度为 4，再向后拼接要从第 5 位开始，所以 count 为 5。
    while count <= length:  # 如果 count 大于密码长度则退出循环
        password += random_char.anyone()  # 随机取出一个字符拼接在密码末尾
        count += 1

    return password
```

上面代码中以 `#` 号开头的代码，称为**注释**，如 `# 用一个随机的大写字符作为起始密码` 。注释用于对代码作注解，只是写给代码阅读者看的，并不会被解释器执行。注释的范围是 `#` 及其之后的该行的所有字符。

随机字符生成部分的实现

「随机字符生成部分」被封装成 `RandomChar` 类，并单独放置在 `randomchar` 模块中，使用它的对象方法即可获取随机字符。

我们需要先准备好各类字符的完整集合，这里采用字符串的形式存放：

- 数字： '0123456789'
- 特殊字符： '~!@#$%^&*'

可以把这些字符串分别保存在对象属性中：

```
class RandomChar:
    def __init__(self):
        self.all_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        self.all_lowercase = 'abcdefghijklmnopqrstuvwxyz'
        self.all_digits = '0123456789'
        self.all_specials = '~!@#$%^&*'
```

再来准备一个方法 pick_random_item() ，这个方法接受一个字符串作为参数，随机返回这个字符串中的一个字符。其内部可以使用 random.randint() 随机生成一个数字，然后把这个随机数字当作索引去字符串中取值，以此生成随机字符。

pick_random_item() 方法实现如下：

```
def pick_random_item(self, sequence):
    random_int = random.randint(0, len(sequence) - 1)  # 调用 random.randint() 生成一个随机数
    return sequence[random_int]
```

有了上面这个从任意字符串中随机取值的功能，我们就可以把它应用到大写字母、小写字母、数字、特殊字符的集合（字符串形式）中去，这样就可以随机获取这四种字符了。

分别对应四个方法：

```
def uppercase(self):
    return self.pick_random_item(self.all_uppercase)  # 调用 pick_random_item 随机从 all_upp

def lowercase(self):
    return self.pick_random_item(self.all_lowercase)  # 调用 pick_random_item 随机从 all_lowe

def digit(self):
    return self.pick_random_item(self.all_digits)     # 调用 pick_random_item 随机从 all_digits 与

def special(self):
    return self.pick_random_item(self.all_specials)   # 调用 pick_random_item 随机从 all_specia
```

最后还有需要一个不区分上述字符种类，随机取任意字符的对象方法。

我们可以把大写字母、小写字母、数字、特殊字符的集合拼接在一起，形成一个更大的集合，然后随机从中取值。

可随机取任意字符的 anyone() 方法如下：

```
def anyone(self):
    # 将四种字符拼接在一起，形成一个大字符串 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh
    return self.pick_random_item(self.all_uppercase + self.all_lowercase + self.all_digits + self
```

整个程序的调用链是：「命令行交互部分」-> 「密码逻辑部分」-> 「随机字符生成部分」。每一个部分各司其职，共同完成这个程序。

完整代码

我们的代码位于两个模块中。

「命令行交互部分」和「密码逻辑部分」位于 password_generator.py 模块，完整代码如下：

password_generator.py

```
import randomchar


def generate_password(length):
    if length < 4:
        raise ValueError('密码至少为 4 位')

    random_char = randomchar.RandomChar()

    password  = random_char.uppercase()
    password += random_char.lowercase()
    password += random_char.digit()
    password += random_char.special()

    count = 5
    while count <= length:
        password += random_char.anyone()
        count += 1

    return password


password_length = input('请输入密码长度（8～20）：')
password_length = int(password_length)

if password_length < 8 or password_length > 20:
    raise ValueError('密码长度不符')

password = generate_password(password_length)
print(password)
```

「随机字符生成部分」位于 randomchar.py 模块，完整代码如下：

randomchar.py

```
import random


class RandomChar:
    def __init__(self):
        self.all_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        self.all_lowercase = 'abcdefghijklmnopqrstuvwxyz'
        self.all_digits = '0123456789'
        self.all_specials = '~!@#$%^&*'

    def pick_random_item(self, sequence):
        random_int = random.randint(0, len(sequence) - 1)
```

```
    return self.pick_random_item(self.all_uppercase)

def lowercase(self):
    return self.pick_random_item(self.all_lowercase)

def digit(self):
    return self.pick_random_item(self.all_digits)

def special(self):
    return self.pick_random_item(self.all_specials)

def anyone(self):
    return self.pick_random_item(self.all_uppercase + self.all_lowercase + self.all_digits + s
```

运行示例

来执行一下程序看看：

```
➜ ~ python3 password_generator.py
请输入密码长度（8~20）：16
Aw6~8a3$AeAo4kSN
```

补充说明

为了可以仅利用之前学过的知识来实现这个程序，这里放弃了一些更简洁或更恰当的 Python 用法。比如

- 循环若干次这里用了 `while` 循环，可以使用 `for _ in range(x)` 的方式替代
- 把随机数字当作索引然后从字符串中取值，可以直接使用 `random.choice()` 函数替代
- `RandomChar` 中的对象属性和对象方法，可直接定义成类属性和类方法
- ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’ 这类字符集合不需要手工书写，使用 `string` 模块即可获取，如 `string.ascii_uppercase`

大家若有兴趣可以自己改进这个程序。

高级的用法和概念将会在之后章节中介绍，不过值得一提的是，朴素的方法也是有价值的！

← 11 更大的代码盒子—模块和包

13 这么多的数据结构（一）：列表、元祖、字符串 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

慕用0015072

请问一下，“ 密码逻辑部分 ” ，是因为函数中有while这样的逻辑判断而命名的吗？

黄浮云 回复 **慕用0015072**

我猜你可能对这里的“逻辑”一词有混淆，在程序员的世界里，如果我们说“某某功能的逻辑”、“某某代码的逻辑”，是指这个功能或代码的具体实现细节或思路。

回复 2019-09-16 21:15:17

千学不如一看，千看不如一练