Lecture 7 第 7 讲

Regular Expressions 正则表达式

- Regular expressions or "regexes" will enable us to examine patterns within our code. For example, we might want to validate that an email address is formatted correctly. Regular expressions will enable us to examine expressions in this fashion.

  正则表达式或"regexe"将使我们能够检查代码中的模式。例如，我们可能想要验证电子邮件地址的格式是否正确。正则表达式将使我们能够以这种方式检查表达式。

- To begin, type `code validate.py` in the terminal window. Then, code as follows in the text editor:

  首先，在终端窗口中键入 `code validate.py`。然后，在文本编辑器中编写如下代码：

```
email = input("What's your email? ").strip()

if "@" in email:
    print("Valid")
else:
    print("Invalid")
```

  Notice that `strip` will remove whitespace at the beginning or end of the input. Running this program, you will see that as long as an `@` symbol is inputted, the program will regard the input as valid.

  请注意，`strip` 将删除输入开头或结尾的空格。运行此程序，您将看到只要输入了 `@` 符号，程序就会认为输入有效。

- You can imagine, however, that one could input `@@` alone and the input could be regarded as valid. We could regard an email address as having at least one `@` and a `.` somewhere within it. Modify your code as follows:

  但是，您可以想象，可以单独输入 `@@`，并且该输入可以被视为有效。我们可以将电子邮件地址视为其中至少有一个 `@` 和一个 `.`。按如下方式修改代码：

```
email = input("What's your email? ").strip()

if "@" in email and "." in email:
    print("Valid")
else:
    print("Invalid")
```

  Notice that while this works as expected, our user could be adversarial, typing simply `@.` would result in the program returning `valid`.

  请注意，虽然这按预期工作，但我们的用户可能是对抗性的，只需键入 `@.` 即可导致程序返回 有效。

- We can improve the logic of our program as follows:

  我们可以按如下方式改进程序的逻辑：

```
email = input("What's your email? ").strip()

username, domain = email.split("@")

if username and "." in domain:
    print("Valid")
else:
    print("Invalid")
```

  Notice how the `strip` method is used to determine if `username` exists and if `.` is inside the `domain` variable. Running this program, a standard email address typed in by you could be considered `valid`. Typing in `malan@harvard` alone, you'll find that the program regards this input as `invalid`.

  请注意如何使用 `strip` 方法来确定 `username` 是否存在以及 `.` 是否在 域 变量内。运行此程序时，您输入的标准电子邮件地址可被视为 有效。单独键入 `malan@harvard`，您会发现程序将此输入视为 无效。

- We can be even more precise, modifying our code as follows:

  我们可以更精确地修改我们的代码，如下所示：

```
email = input("What's your email? ").strip()

username, domain = email.split("@")

if username and domain.endswith(".edu"):
    print("Valid")
else:
    print("Invalid")
```

  Notice how the `endswith` method will check to see if `domain` contains `.edu`. Still, however, a nefarious user could still break our code. For example, a user could type in `malan@.edu` and it would be considered valid.

  请注意 `endswith` 方法将如何检查 `domain` 是否包含 `.edu`。但是，恶意用户仍然可能破坏我们的代码。例如，用户可以键入 `malan@.edu`，它将被视为有效。

- Indeed, we could keep iterating upon this code ourselves. However, it turns out that Python has an existing library called `re` that has a number of built-in functions that can validate user inputs against patterns.

  事实上，我们自己可以不断迭代这段代码。但是，事实证明 Python 有一个名为 `re` 的现有库，该库具有许多内置函数，可以根据模式验证用户输入。

- One of the most versatile functions within the library `re` is `search`.

  `re` 库中最通用的功能之一是 搜索。

- The `search` library follows the signature `re.search(pattern, string, flags=0)`. Following this signature, we can modify our code as follows:

  搜索 库遵循签名 `re.search(pattern, string, flags=0)`。按照这个签名，我们可以按如下方式修改我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search("@", email):
    print("Valid")
else:
    print("Invalid")
```

Notice this does not increase the functionality of our program at all. In fact, it is somewhat a step back.

请注意，这根本不会增加我们程序的功能。事实上，这在某种程度上是一种倒退。

- We can further our program's functionality. However, we need to advance our vocabulary around `validation`. It turns out that in the world of regular expressions there are certain symbols that allow us to identify patterns. At this point, we have only been checking for specific pieces of text like `@`. It so happens that many special symbols can be passed to the compiler for the purpose of engaging in validation. A non-exhaustive list of those patterns is as follows:

  我们可以进一步扩展我们程序的功能。但是，我们需要围绕 验证 来改进我们的词汇。事实证明，在正则表达式的世界中，有一些符号可以让我们识别模式。此时，我们只检查了 `@` 等特定文本片段。碰巧可以将许多特殊符号传递给编译器以进行验证。这些模式的非详尽列表如下：

```
.   any character except a new line
*   0 or more repetitions
+   1 or more repetitions
?   0 or 1 repetition
{m} m repetitions
{m,n} m-n repetitions
```

- Implementing this inside of our code, modify yours as follows:

  在我们的代码中实现它，修改你的代码，如下所示：

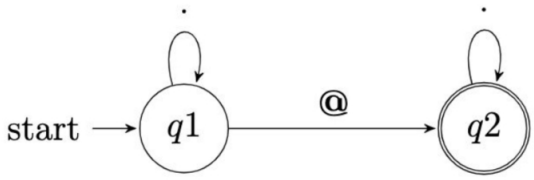
```
import re

email = input("What's your email? ").strip()

if re.search(".+@.+", email):
    print("Valid")
else:
    print("Invalid")
```

Notice that we don't care what the username or domain is. What we care about is the pattern. `.+` is used to determine if anything is to the left of the email address and if anything is to the right of the email address. Running your code, typing in `malan@`, you'll notice that the input is regarded as `invalid` as we would hope.

请注意，我们并不关心用户名或域是什么。我们关心的是模式。 `.+` 用于确定电子邮件地址左侧是否有任何内容，以及电子邮件地址右侧是否有任何内容。运行您的代码，键入 `malan@`，您会注意到输入被视为 无效，正如我们所希望的那样。

- Had we used a regular expression `.*@.*` in our code above, you can visualize this as follows:

  如果我们在上面的代码中使用了正则表达式 `.*@.*`，您可以按如下方式可视化：

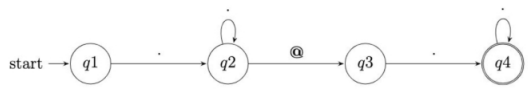


Notice the depiction of the `state machine` of our regular expression. On the left, the compiler begins evaluating the statement from left to right. Once we reach `q1` or question 1, the compiler reads time and time again based on the expression handed to it. Then, the state is changed looking now at `q2` or the second question being validated. Again, the arrow indicates how the expression will be evaluated time and time again based upon our programming. Then, as depicted by the double circle, the final state of state machine is reached.

请注意正则表达式的状态 机 的描述。在左侧，编译器开始从左到右计算语句。一旦我们到达 `q1` 或问题 1，编译器就会根据交给它的表达式一次又一次地读取。然后，状态发生变化，现在查看 `q2` 或正在验证的第二个问题。同样，箭头指示如何根据我们的编程一次又一次地计算表达式。然后，如双圆所示，达到 state machine 的最终状态。

- Considering the regular expression we used in our code, `.+@.+`, you can visualize it as follows:

  考虑到我们在代码中使用的正则表达式 `.+@.+`，您可以按如下方式对其进行可视化：



Notice how `q1` is any character provided by the user, including 'q2' as 1 or more repetitions of characters. This is followed by the '@' symbol. Then, `q3` looks for any character provided by the user, including `q4` as 1 or more repetitions of characters.

请注意 `q1` 是用户提供的任何字符，包括 'q2' 作为 1 个或多个重复的字符。后跟 '@' 符号。然后， `q3` 查找用户提供的任何字符，包括 `q4` 作为 1 个或多个重复的字符。

- The `re` and `re.search` functions and ones like them look for patterns.

  `re` 和 `re.search` 函数以及类似的函数寻找模式。

- Continuing our improvement of this code, we could improve our code as follows:

  继续改进此代码，我们可以按如下方式改进我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search(".+@.+.edu", email):
```

```
    print("Valid")
else:
    print("Invalid")
```

Notice, however, that one could type in `malan@harvard?edu` and it could be considered valid. Why is this the case? You might recognize that in the language of validation, a `.` means any character!

但是请注意，您可以输入 `malan@harvard?edu` 并且可以认为它是有效的。为什么会这样呢？您可能会认识到，在验证语言中，`a .` 表示任何字符！

- We can modify our code as follows:

  我们可以按如下方式修改我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search(r".+@.+\.edu", email):
    print("Valid")
else:
    print("Invalid")
```

Notice how we utilize the "escape character" or `\` as a way of regarding the `.` as part of our string instead of our validation expression. Testing your code, you will notice that `malan@harvard.edu` is regarded as valid, where `malan@harvard?edu` is invalid.

请注意我们如何使用"转义字符"或 `\` 作为将 `.` 视为字符串的一部分，而不是我们的验证表达式。测试你的代码，你会注意到 `malan@harvard.edu` 被认为是有效的，而 `malan@harvard?edu` 是无效的。

- Now that we're using escape characters, it's a good time to introduce "raw strings". In Python, raw strings are strings that *don't* format special characters—instead, each character is taken at face-value. Imagine `\n`, for example. We've seen in an earlier lecture how, in a regular string, these two characters become one: a special newline character. In a raw string, however, `\n` is treated not as `\n`, the special character, but as a single `\` and a single `n`. Placing an `r` in front of a string tells the Python interpreter to treat the string as a raw string, similar to how placing an `f` in front of a string tells the Python interpreter to treat the string as a format string:

  现在我们使用的是转义字符，现在是引入"raw strings"的好时机。在 Python 中，原始字符串是不格式化特殊字符的字符串，而是每个字符都按面值取。例如，想象一下 `\n`。在前面的讲座中，我们已经看到了在常规字符串中，这两个字符如何合二为一：一个特殊的换行符。但是，在原始字符串中，`\n` 不被视为特殊字符 `\n`，而是被视为单个 `\` 和单个 `n`。将 `r` 放在字符串前面会告诉 Python 解释器将字符串视为原始字符串，类似于将 `f` 放在字符串前面会告诉 Python 解释器将字符串视为格式字符串：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^.+@.+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Now we've ensured the Python interpreter won't treat `\.` as a special character. Instead, simply as a `\` followed by a `.`—which, in regular expression terms, means matching a literal ".":

现在我们已经确保 Python 解释器不会将 `\.` 视为特殊字符。相反，简单地作为 `\` 后跟 `.` – 在正则表达式术语中，表示匹配文本 "."。

- You can imagine still how our users could create problems for us! For example, you could type in a sentence such as `My email address is malan@harvard.edu.` and this whole sentence would be considered valid. We can be even more precise in our coding.

  您仍然可以想象我们的用户如何给我们制造问题！例如，您可以键入一个句子，`My email address is malan@harvard.edu.` 例如，整个句子将被视为有效。我们可以在编码中更加精确。

- It just so happens we have more special symbols at our disposal in validation:

  碰巧我们在验证中可以使用更多特殊符号：

```
^   matches the start of the string
$   matches the end of the string or just before the newline at the end of the string
```

- We can modify our code using our added vocabulary as follows:

  我们可以使用添加的词汇来修改我们的代码，如下所示：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^.+@.+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice this has the effect of looking for this exact pattern matching to the start and end of the expression being validated. Typing in a sentence such as `My email address is malan@harvard.edu.` now is regarded as invalid.

请注意，这具有查找与正在验证的表达式的开头和结尾匹配的确切模式的效果。键入 now 之类 `My email address is malan@harvard.edu.` 的句子被视为无效。

- We propose we can do even better! Even though we are now looking for the username at the start of the string, the `@` symbol, and the domain name at the end, we could type in as many `@` symbols as we wish! `malan@@@harvard.edu` is considered valid!

  我们建议我们可以做得更好！即使我们现在正在寻找字符串开头的用户名、`@` 符号和末尾的域名，我们也可以输入任意数量的 `@` 符号！`malan@@@harvard.edu` 被视为有效！

- We can add to our vocabulary as follows:

  我们可以按如下方式添加到我们的词汇表中：

```
[]    set of characters
[^]   complementing the set
```

- Using these newfound abilities, we can modify our expression as follows:

  使用这些新发现的功能，我们可以按如下方式修改表达式：

```
import re

email = input("What's your email? ").strip()
```

```
if re.search(r"^[^@]+@[^@]+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice that `^` means to match at the start of the string. All the way at the end of our expression, `$` means to match at the end of the string. `[^@]+` means any character except an `@`. Then, we have a literal `@`. `[^@]+\.edu` means any character except an `@` followed by an expression ending in `.edu`. Typing in `malan@@@harvard.edu` is now regarded as invalid.

请注意，`^` 表示在字符串的开头匹配。在表达式的末尾，`$` 表示在字符串的末尾匹配。`[^@]+` 表示除 `@` 之外的任何字符。然后，我们有一个文本 `@`。`[^@]+\.edu` 表示除 `@` 后跟以 `.edu` 结尾的表达式之外的任何字符。键入 `malan@@@harvard.edu` 现在被视为无效。

- We can still improve this regular expression further. It turns out there are certain requirements for what an email address can be! Currently, our validation expression is far too accomodating. We might only want to allow for characters normally used in a sentence. We can modify our code as follows:

  我们仍然可以进一步改进这个正则表达式。事实证明，电子邮件地址是有一定要求的！目前，我们的验证表达式太过包容。我们可能只想允许句子中通常使用的字符。我们可以按如下方式修改我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^[a-zA-Z0-9_]+@[a-zA-Z0-9_]+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice that `[a-zA-Z0-9_]` tells the validation that characters must be between `a` and `z`, between `A` and `Z`, between `0` and `9` and potentially include an `_` symbol. Testing the input, you'll find that many potential user mistakes can be indicated.

请注意，`[a-zA-Z0-9_]` 告诉验证字符必须介于 `a` 和 `z` 之间、`A` 和 `Z` 之间、`0` 和 `9` 之间，并且可能包含 `_` 符号。测试输入，您会发现可以指示许多潜在的用户错误。

- Thankfully, common patterns have been built into regular expressions by hard-working programmers. In this case, you can modify your code as follows:

  值得庆幸的是，勤奋的程序员已经将通用模式构建到正则表达式中。在这种情况下，您可以按如下方式修改代码：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice that `\w` is the same as `[a-zA-Z0-9_]`. Thanks, hard-working programmers!

请注意，`\w` 与 `[a-zA-Z0-9_]` 相同。感谢辛勤工作的程序员！

- Here are some additional patterns we can add to our vocabulary:

  以下是我们可以添加到词汇表中的其他模式：

```
\d    decimal digit
\D    not a decimal digit
\s    whitespace characters
\S    not a whitespace character
\w    word character, as well as numbers and the underscore
\W    not a word character
```

- Now, we know that there are not simply `.edu` email addresses. We could modify our code as follows:

  现在，我们知道不仅仅是 `.edu` 电子邮件地址。我们可以按如下方式修改我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w.+\.(com|edu|gov|net|org)$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice that the `|` has the impact of an `or` in our expression.

请注意，在我们的表达式中，`|` 具有 `or` 的影响。

- Adding even more symbols to our vocabulary, here are some more to consider:

  在我们的词汇表中添加更多符号，以下是一些需要考虑的符号：

```
A|B     either A or B
(...)   a group
(?:...) non-capturing version
```

Case Sensitivity 区分大小写

- To illustrate how you might address issues around case sensitivity, where there is a difference between `EDU` and `edu` and the like, let's rewind our code to the following:

  为了说明如何解决区分大小写的问题，其中 `EDU` 和 `edu` 等之间存在差异，让我们将代码倒回去如下：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Notice how we have removed the `|` statements provided previously.

请注意我们是如何删除前面提供的 `|` 语句的。

- Recall that within the `re.search` function, there is a parameter for `flags`.

  回想一下，在 `re.search` 函数中，有一个 `flags` 参数。

- Some built-in flag variables are:

  一些内置的标志变量是：

```
re.IGNORECASE
re.MULTILINE
re.DOTALL
```

  Consider how you might use these in your code.

  考虑如何在代码中使用这些 API。

- Therefore, we can change our code as follows.

  因此，我们可以按如下方式更改我们的代码。

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w+\.edu$", email, re.IGNORECASE):
    print("Valid")
else:
    print("Invalid")
```

  Notice how we added a third parameter `re.IGNORECASE`. Running this program with `MALAN@HARVARD.EDU`, the input is now considered valid.

  请注意我们如何添加第三个参数 `re。IGNORECASE` 的使用 `MALAN@HARVARD.EDU` 运行此程序，则输入现在被视为有效。

- Consider the following email address `malan@cs50.harvard.edu`. Using our code above, this would be considered invalid. Why might that be?

  请考虑 以下电子邮件地址 `malan@cs50.harvard.edu`。使用上面的代码，这将被视为无效。为什么会这样呢?

- Since there is an additional `.`, the program considers this invalid.

  由于有一个额外的 `.`，程序认为这是无效的。

- It turns out that we can, looking at our vocabulary from before, we can group together ideas.

  事实证明，我们可以，看看我们以前的词汇，我们可以把想法组合在一起。

```
A|B     either A or B
(...)   a group
(?:...) non-caputuring version
```

- We can modify our code as follows:

  我们可以按如下方式修改我们的代码：

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@(\w+\.)?\w+\.edu$", email, re.IGNORECASE):
    print("Valid")
else:
    print("Invalid")
```

  Notice how the `(\w+\.)?` communicates to the compiler that this new expression can be there once or not at all. Hence, both `malan@cs50.harvard.edu` and `malan@harvard.edu` are considered valid.

  请注意 `（\w+\.？）` 如何向编译器传达这个新表达式可以存在一次或根本不存在。因此，`malan@cs50.harvard.edu` 和 `malan@harvard.edu` 都被视为有效。

- Interestingly enough, the edits we have done so far to our code do not fully encompass all the checking that could be done to ensure a valid email address. Indeed, here is the full expression that one would have to type to ensure that a valid email is inputted:

  有趣的是，到目前为止，我们对代码所做的编辑并未完全包含为确保电子邮件地址有效而可以进行的所有检查。事实上，这是必须键入的完整表达式，以确保输入了有效的电子邮件：

```
^[a-zA-Z0-9.!#$%&'*+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$
```

- There are other functions within the `re` library you might find useful. `re.match` and `re.fullmatch` are ones you might find exceedingly useful.

  `re` 库中还有其他一些函数可能会很有用。`re.match` 和 `re.fullmatch` 是您可能会发现非常有用的。

- You can learn more in Python's documentation of re.

  您可以在 Python 的 re.

Cleaning Up User Input 清理用户输入

- You should never expect your users to always follow your hopes for clean input. Indeed, users will often violate your intentions as a programmer.

  您永远不应该期望您的用户总是遵循您对干净输入的希望。事实上，用户经常会违背您作为程序员的意图。

- There are ways to clean up your data.

  有多种方法可以清理数据。

- In the terminal window, type `code format.py`. Then, in the text-editor, code as follows:

  在终端窗口中，键入 `code format.py`。然后，在文本编辑器中，编写如下代码：

```
name = input("What's your name? ").strip()
print(f"hello, {name}")
```

  Notice that we have created, essentially, a "hello world" program. Running this program and typing in `David`, it works well! However, typing in `Malan, David` notice how the program does not function as intended. How could we modify our program to clean up this input?

  请注意，我们实际上已经创建了一个 "hello world" 程序。运行此程序并键入 `David`，效果很好！然而，在输入 `Malan 时，David` 注意到该程序并没有按预期运行。我们如何修改我们的程序来清理这个输入呢?

- Modify your code as follows

- Modify your code as follows:

  按如下方式修改您的代码。

```
name = input("What's your name? ").strip()
if "," in name:
    last, first = name.split(", ")
    name = f"{first} {last}"
print(f"hello, {name}")
```

  Notice how `last, first = name.split(", ")` is run if there is a `,` in the name. Then, the name is standardized as first and last. Running our code, typing in `Malan, David`, you can see how this program does clean up at least one scenario where a user types in something unexpected.

  请注意，如果名称中有 `,` ，则如何 `last, first = name.split(", ")` 运行。然后，名称标准化为 first 和 last。运行我们的代码，键入 `Malan, David` ，您可以看到该程序如何清理至少一个用户键入意外内容的情况。

- You might notice that typing in `Malan,David` with no space causes the compiler to throw an error. Since we now know some regular expression syntax, let's apply that to our code:

  您可能会注意到，在无空格的 `Malan，David` 中键入会导致编译器引发错误。由于我们现在知道了一些正则表达式语法，让我们将其应用于我们的代码：

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    last, first = matches.groups()
    name = first + " " + last
print(f"hello, {name}")
```

  Notice that `re.search` can return a set of matches that are extracted from the user's input. If matches are returned by `re.search`. Running this program, typing in `David Malan` notice how the `if` condition is not run and the name is returned. If you run the program by typing `Malan, David`, the name is also returned properly.

  请注意， `re.search` 可以返回从用户输入中提取的一组匹配项。如果 `re.search` 返回匹配项。运行此程序时，键入 `David Malan` 会注意到 `if` 条件未运行并返回名称。如果通过键入 `Malan, David` 来运行程序，则也会正确返回名称。

- It just so happens that we can request specific groups back using `matches.group`. We can modify our code as follows:

  碰巧我们可以使用 `matches.group` 请求返回特定组。我们可以按如下方式修改我们的代码：

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

  Notice how, in this implementation, `group` is not plural (there is no `s`).

  请注意，在此实现中， `group` 不是复数形式（没有 `s` ）。

- Our code can be further tightened as follows:

  我们的代码可以进一步收紧，如下所示：

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

  Notice how `group(2)` and `group(1)` are concatenated together with a space. The first group is that which is left of the comma. The second group is that which is right of the comma.

  请注意 `group（2）` 和 `group（1）` 是如何与空格连接在一起的。第一组是逗号的左边。第二组是逗号右侧的部分。

- Recognize still that typing in `Malan,David` with no space will still break our code. Therefore, we can make the following modification:

  仍然要认识到，在 `Malan，David` 中键入没有空格仍然会破坏我们的代码。因此，我们可以进行以下修改：

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), *(.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

  Notice the addition of the `*` in our validation statement. This code will now accept and properly process `Malan,David`. Further, it will properly handle `David,Malan with many spaces in front of David`.

  请注意 validation 语句中添加的 `*` 。此代码现在将接受并正确处理 `Malan，David` 。此外，它将正确处理 'David， Malan with many spaces in front of David'。

- It is very common to utilize `re.search` as we have in the previous examples, where `matches` is on a line of code after. However, we can combine these statements:

  像前面的例子中一样，使用 `re.search` 是很常见的，其中 `matches` 在后面的一行代码上。但是，我们可以组合这些语句：

```
import re

name = input("What's your name? ").strip()
if matches := re.search(r"^(.+), *(.+)$", name):
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

  Notice how we combine two lines of our code. The walrus `:=` operator assigns a value from right to left and allows us to ask a boolean question at the same time. Turn your head sideways and you'll see why this is called a walrus operator.

  请注意我们如何组合代码的两行。walrus `:=` 运算符从右到左分配一个值，并允许我们同时提出一个布尔问题。把头转向一边，你就会明白为什么这被称为海象算子。

- You can learn more in Python's documentation of re.

您可以在 Python 的 re.

Extracting User Input 提取用户输入

- So far, we have validated the user's input and cleaned up the user's input.

  到目前为止，我们已经验证了用户的输入并清理了用户的输入。

- Now, let's extract some specific information from user input. In the terminal window, type `code twitter.py` and code as follows in the text editor window:

  现在，让我们从用户输入中提取一些特定信息。在终端窗口中，在文本编辑器窗口中键入 `code twitter.py` 和 code，如下所示：

```
url = input("URL: ").strip()
print(url)
```

  Notice that if we type in `https://twitter.com/davidjmalan`, it shows exactly what the user typed. However, how would we be able to extract just the username and ignore the rest of the URL?

  请注意，如果我们键入 `https://twitter.com/davidjmalan` ，它会准确显示用户键入的内容。但是，我们如何能够只提取用户名而忽略 URL 的其余部分呢？

- You can imagine how we would simply be able to get rid of the beginning of the standard Twitter URL. We can attempt this as follows:

  您可以想象我们如何能够简单地摆脱标准 Twitter URL 的开头。我们可以按如下方式尝试此作：

```
url = input("URL: ").strip()

username = url.replace("https://twitter.com/", "")
print(f"Username: {username}")
```

  Notice how the `replace` method allows us to find one item and replace it with another. In this case, we are finding part of the URL and replacing it with nothing. Typing in the full URL `https://twitter.com/davidjmalan`, the program effectively outputs the username. However, what are some shortcomings of this current program?

  请注意 `replace` 方法如何允许我们找到一个项目并将其替换为另一个项目。在本例中，我们将查找 URL 的一部分并将其替换为任何内容。键入完整的 URL `https://twitter.com/davidjmalan` ，程序将有效地输出用户名。但是，当前程序有哪些缺点呢？

- What if the user simply typed `twitter.com` instead of including the `https://` and the like? You can imagine many scenarios where the user may input or neglect to input parts of the URL that would create strange output by this program. To improve this program, we can code as follows:

  如果用户只键入 `twitter.com` 而不是包括 `https://` 等，该怎么办？您可以想象许多场景，用户可能会输入或忽略输入 URL 的部分，这些部分会通过该程序产生奇怪的输出。为了改进这个程序，我们可以编写如下代码：

```
url = input("URL: ").strip()

username = url.removeprefix("https://twitter.com/")
print(f"Username: {username}")
```

  Notice how we utilize the `removeprefix` method. This method will remove the beginning of a string.

  请注意我们如何使用 `removeprefix` 方法。此方法将删除字符串的开头。

- Regular expressions simply allow us to succinctly express the patterns and goals.

  正则表达式只是让我们简洁地表达模式和目标。

- Within the `re` library, there is a method called `sub`. This method allows us to substitute a pattern with something else.

  在 `re` 库中，有一个名为 `sub` 的方法。这种方法允许我们用其他东西替换一个模式。

- The signature of the `sub` method is as follows

  `sub` 方法的签名如下

```
re.sub(pattern, repl, string, count=0, flags=0)
```

  Notice how `pattern` refers to the regular expression we are looking for. Then, there is a `repl` string that we can replace the pattern with. Finally, there is the `string` that we want to do the substitution on.

  注意 `pattern` 是如何引用我们正在寻找的正则表达式的。然后，有一个 `repl` 字符串，我们可以用它来替换 pattern。最后，是我们要对其进行替换的 `字符串` 。

- Implementing this method in our code, we can modify our program as follows:

  在我们的代码中实现这个方法，我们可以按如下方式修改我们的程序：

```
import re

url = input("URL: ").strip()

username = re.sub(r"https://twitter.com/", "", url)
print(f"Username: {username}")
```

  Notice how executing this program and inputting `https://twitter.com/davidjmalan` produces the correct outcome. However, there are some problems still present in our code.

  请注意执行此程序并输入 `https://twitter.com/davidjmalan` 如何产生正确的结果。但是，我们的代码中仍然存在一些问题。

- The protocol, subdomain, and the possibility that the user inputted any part of the URL after the username are all reasons that this code is still not ideal. We can further address these shortcomings as follows:

  协议、子域以及用户在用户名后输入 URL 的任何部分的可能性都是此代码仍然不理想的原因。我们可以进一步解决这些缺点，如下所示：

```
import re

url = input("URL: ").strip()

username = re.sub(r"^(https?://)?(www\.)?twitter\.com/", "", url)
print(f"Username: {username}")
```

  Notice how the `^` caret was added to the url. Notice also how the `.` could be interpreted improperly by the compiler. Therefore, we escape it using a `\` to make it `\.` For the purpose of tolerating both `http` and `https`, we add a `?` to the end of `https?`, making the `s` optional. Further, to accommodate `www` we add `(www\.)?` to our code. Finally, just in case the user decides to leave out the protocol altogether, the `http://` or `https://` is made optional using `(https?://)`.

  请注意 `^` 插入符号是如何添加到 URL 中的。还要注意编译器如何错误地解释 `.` 。因此，我们使用 `\` 对其进行转义，使其成为 `\.` 为了同时容忍 `http` 和

`https`，我们在 `https?` 的末尾添加了一个 `?`，使 `s` 成为可选的。此外，为了容纳 `www`，我们将 `(www\.?)` 添加到我们的代码中。最后，以防万一用户决定完全省略协议，则使用 `(https?://)` 将 `http://` 或 `https://` 设为可选。

- Still, we are blindly expecting that what the user inputted a url that, indeed, has a username.

  尽管如此，我们还是盲目地期望用户输入的 URL 确实具有用户名。

- Using our knowledge of `re.search`, we can further improve our code.

  利用我们对 `re.search` 的了解，我们可以进一步改进我们的代码。

```
import re

url = input("URL: ").strip()

matches = re.search(r"^https?://(www\.)?twitter\.com/(.+)$", url, re.IGNORECASE)
if matches:
    print(f"Username:", matches.group(2))
```

  Notice how we are searching for the regular expression above in the string provided by the user. In particular, we are capturing that which appears at the end of the URL using `(.+)$` regular expression. Therefore, if the user fails to input a URL without a username, no input will be presented.

  请注意我们如何在用户提供的字符串中搜索上面的正则表达式。特别是，我们使用 `(.+)$` 正则表达式捕获出现在 URL 末尾的内容。因此，如果用户输入没有用户名的 URL 失败，则不会显示任何输入。

- Even further tightening up our program, we can utilize our `:=` operator as follows:

  进一步加强我们的程序，我们可以按如下方式使用我们的 `:=` 运算符：

```
import re

url = input("URL: ").strip()

if matches := re.search(r"^https?://(?:www\.)?twitter\.com/(.+)$", url, re.IGNORECASE):
    print(f"Username:", matches.group(1))
```

  Notice that the `?:` tells the compiler it does not have to capture what is in that spot in our regular expression.

  请注意，`?:` 告诉编译器它不必捕获正则表达式中该位置的内容。

- Still, we can be more explicit to ensure that the username inputted is correct. Using Twitter's documentation, we can add the following to our regular expression:

  尽管如此，我们可以更明确地确保输入的用户名是正确的。使用 Twitter 的文档，我们可以将以下内容添加到正则表达式中：

```
import re

url = input("URL: ").strip()

if matches := re.search(r"^https?://(?:www\.)?twitter\.com/([a-z0-9_]+)", url, re.IGNORECASE):
    print(f"Username:", matches.group(1))
```

  Notice that the `[a-z0-9_]+` tells the compiler to only expect `a-z`, `0-9`, and `_` as part of the regular expression. The `+` indicates that we are expecting one or more characters.

  请注意，`[a-z0-9_]+` 告诉编译器只期望 `a-z`、`0-9` 和 `_` 作为正则表达式的一部分。`+` 表示我们期望使用一个或多个字符。

- You can learn more in Python's documentation of re.

  您可以在 Python 的 re.

Summing Up 总结

Now, you've learned a whole new language of regular expressions that can be utilized to validate, clean up, and extract user input.

现在，您已经学习了一种全新的正则表达式语言，可用于验证、清理和提取用户输入。

- Regular Expressions 正则表达式
- Case Sensitivity 区分大小写
- Cleaning Up User Input 清理用户输入
- Extracting User Input 提取用户输入