Lecture 3 第 3 讲

Exceptions 异常

- Exceptions are things that go wrong within our coding.
  异常是我们的编码中出错的事情。
- In our text editor, type `code hello.py` to create a new file. Type as follows (with the intentional errors included):
  在我们的文本编辑器中，键入 `code hello.py` 以创建新文件。按如下方式键入（包括故意错误）：

```
print("hello, world)
```

Notice that we intentionally left out a quotation mark.
请注意，我们故意省略了引号。

- Running `python hello.py` in our terminal window, an error is outputted. The compiler states that it is a "syntax error."" Syntax errors are those that require you to double-check that you typed in your code correction.
  在终端窗口中运行 `python hello.py`，输出错误。编译器声明这是一个 "语法错误"。语法错误是指要求您仔细检查是否在代码更正中键入的错误。
- You can learn more in Python's documentation of Errors and Exceptions.
  您可以在 Python 的 Errors and Exceptions 文档中了解更多信息。

Runtime Errors 运行时错误

- Runtime errors refer to those created by unexpected behavior within your code. For example, perhaps you intended for a user to input a number, but they input a character instead. Your program may throw an error because of this unexpected input from the user.
  运行时错误是指由代码中的意外行为造成的错误。例如，也许您打算让用户输入一个数字，但他们输入的却是一个字符。由于用户的意外输入，您的程序可能会引发错误。
- In your terminal window, run `code number.py`. Code as follows in your text editor:
  在终端窗口中，运行 `Code number.py`。在文本编辑器中编写如下代码：

```
x = int(input("What's x? "))
print(f"x is {x}")
```

Notice that by including the `f`, we tell Python to interpolate what is in the curly braces as the value of `x`. Further, testing out your code, you can imagine how one could easily type in a string or a character instead of a number. Even still, a user could type nothing at all – simply hitting the enter key.
请注意，通过包含 `f`，我们告诉 Python 将大括号中的内容插值为 `x` 的值。此外，测试您的代码，您可以想象如何轻松地输入字符串或字符而不是数字。即便如此，用户根本无法键入任何内容 - 只需按 Enter 键即可。

- As programmers, we should be defensive to ensure that our users are entering what we expected. We might consider "corner cases" such as `-1`, `0`, or `cat`.
  作为程序员，我们应该采取防御措施，以确保我们的用户输入的是我们的预期内容。我们可能会考虑 `-1`、`0` 或 `cat` 等 "极端情况"。
- If we run this program and type in "cat", we'll suddenly see `ValueError: invalid literal for int() with base 10: 'cat'` Essentially, the Python interpreter does not like that we passed "cat" to the `int` function.
  如果我们运行这个程序并键入 "cat"，我们会突然看到 `ValueError: invalid literal for int() with base 10: 'cat'` Essentially，Python 解释器不喜欢我们将 "cat" 传递给 `int` 函数。
- An effective strategy to fix this potential error would be to create "error handling" to ensure the user behaves as we intend.
  修复此潜在错误的有效策略是创建 "错误处理" 以确保用户的行为符合我们的预期。
- You can learn more in Python's documentation of Errors and Exceptions.
  您可以在 Python 的 Errors and Exceptions 文档中了解更多信息。

`try` 尝试

- In Python `try` and `except` are ways of testing out user input before something goes wrong. Modify your code as follows:
  在 Python 中，`try` 和 `except` 是在出现问题之前测试用户输入的方法。按如下方式修改代码：

```
try:
    x = int(input("What's x?"))
    print(f"x is {x}")
except ValueError:
    print("x is not an integer")
```

Notice how, running this code, inputting `50` will be accepted. However, typing in `cat` will produce an error visible to the user, instructing them why their input was not accepted.
请注意，运行此代码时，输入 `50` 将如何被接受。但是，键入 `cat` 将产生用户可见的错误，告知他们为什么不接受他们的输入。

- This is still not the best way to implement this code. Notice that we are trying to do two lines of code. For best practice, we should only `try` the fewest lines of code possible that we are concerned could fail. Adjust your code as follows:

  这仍然不是实现此代码的最佳方式。请注意，我们正在尝试执行两行代码。作为最佳实践，我们应该只 尝试 我们担心可能会失败的最少代码行。按如下方式调整代码：

```
try:
    x = int(input("What's x?"))
except ValueError:
    print("x is not an integer")

print(f"x is {x}")
```

  Notice that while this accomplishes our goal of trying as few lines as possible, we now face a new error! We face a `NameError` where `x is not defined`. Look at this code and consider: Why is x not defined in some cases?

  请注意，虽然这实现了我们尝试尽可能少行的目标，但我们现在面临一个新错误！我们面临一个 `NameError`，其中 `x 未定义`。看看这段代码并考虑一下：为什么在某些情况下没有定义 x？

- Indeed, if you examine the order of operations in `x = int(input("What's x?"))`, working right to left, it could take an incorrectly inputted character and attempt to assign it as an integer. If this fails, the assignment of the value of `x` never occurs. Therefore, there is no x to print on our final line of code.

  事实上，如果你检查从 右到左的 `x = int（input（“What's x？”））` 中的运算顺序，它可能会采用错误输入的字符并尝试将其分配为整数。如果此操作失败，则永远不会分配 `x` 的值。因此，在我们的最后一行代码上没有 x 要打印。

`else` 还

- It turns out that there is another way to implement `try` that could catch errors of this nature.

  事实证明，还有另一种实现 `try` 的方法可以捕获这种性质的错误。

- Adjust your code as follows:

  按如下方式调整代码：

```
try:
    x = int(input("What's x?"))
except ValueError:
    print("x is not an integer")
else:
    print(f"x is {x}")
```

  Notice that if no exception occurs, it will then run the block of code within `else`. Running `python number.py` and supplying `50`, you'll notice that the result will be printed. Trying again, this time supplying `cat`, you'll notice that the program now catches the error.

  请注意，如果没有发生异常，它将在 `else` 中运行代码块。运行 `python number.py` 并提供 `50` 个，您会注意到结果将被打印出来。再次尝试，这次提供 `cat`，您将注意到程序现在捕获了错误。

- Considering improving our code, notice that we are being a bit rude to our user. If our user does not cooperate, we currently simply end our program. Consider how we can use a loop to prompt the user for `x` and if they don't prompt again! Improve your code as follows:

  考虑改进我们的代码，请注意我们对用户有点粗鲁。如果我们的用户不合作，我们目前只需结束我们的程序。考虑一下我们如何使用循环来提示用户输入 `x`，以及他们是否不再提示！按如下方式改进您的代码：

```
while True:
    try:
        x = int(input("What's x?"))
    except ValueError:
        print("x is not an integer")
    else:
        break

print(f"x is {x}")
```

  Notice that `while True` will loop forever. If the user succeeds in supplying the correct input, we can break from the loop and then print the output. Now, a user that inputs something incorrectly will be asked for input again.

  请注意，`虽然 True` 将永远循环。如果用户成功提供了正确的输入，我们可以打破循环，然后打印输出。现在，如果用户输入错误，系统将再次要求其输入。

Creating a Function to Get an Integer

创建函数以获取整数

- Surely, there are many times that we would want to get an integer from our user. Modify your code as follows:

  当然，很多时候我们想从用户那里得到一个整数。按如下方式修改代码：

```
def main():
    x = get_int()
    print(f"x is {x}")


def get_int():
    while True:
        try:
            x = int(input("What's x?"))
        except ValueError:
            print("x is not an integer")
        else:
            break
    return x


main()
```

  Notice that we are manifesting many great properties. First, we have abstracted away the ability to get an integer. Now, this whole program boils down to the first three lines of the program.

  请注意，我们正在展示许多出色的属性。首先，我们抽象了获取整数的能力。现在，整个程序归结为程序的前三行。

- Even still, we can improve this program. Consider what else you could do to improve this program. Modify your code as follows:

  即便如此，我们仍然可以改进这个程序。考虑一下您还可以做些什么来改进这个程序。按如下方式修改代码：

```
def main():
    x = get_int()
    print(f"x is {x}")


def get_int():
    while True:
        try:
            x = int(input("What's x?"))
        except ValueError:
            print("x is not an integer")
        else:
            return x


main()
```

  Notice that `return` will not only break you out of a loop, but it will also return a value.

  请注意，`return` 不仅会让你跳出循环，还会返回一个值。

- Some people may argue you could do the following:

  有些人可能会争辩说您可以执行以下操作：

```
def main():
    x = get_int()
    print(f"x is {x}")


def get_int():
    while True:
        try:
            return int(input("What's x?"))
        except ValueError:
            print("x is not an integer")


main()
```

  Notice this does the same thing as the previous iteration of our code, simply with fewer lines.

  请注意，这与我们代码的上一个迭代做同样的事情，只是行数更少。

`pass` 通过

- We can make it such that our code does not warn our user, but simply re-asks them our prompting question by modifying our code as follows:

  我们可以使我们的代码不会警告我们的用户，而只是通过修改我们的代码来重新询问他们我们的提示问题，如下所示：

```
def main():
    x = get_int()
    print(f"x is {x}")


def get_int():
    while True:
        try:
            return int(input("What's x?"))
        except ValueError:
            pass


main()
```

  Notice that our code will still function but will not repeatedly inform the user of their error. In some cases, you'll want to be very clear to the user what error is being produced. Other times, you might decide that you simply want to ask them for input again.

  请注意，我们的代码仍将运行，但不会重复通知用户他们的错误。在某些情况下，您需要非常清楚地向用户说明正在产生什么错误。其他时候，您可能决定只想再次向他们征求意见。

- One final refinement that could improve the implementation of this `get_int` function. Right now, notice that we are relying currently upon the honor system that the `x` is in both the `main` and `get_int` functions. We probably want to pass in a prompt that the user sees when asked for input. Modify your code as follows.

  可以改进此 `get_int` 函数的实现的最后一项改进。现在，请注意，我们目前依赖于 honor 系统，即 `x` 同时存在于 `main` 和 `get_int` 函数中。我们可能希望传入一个提示，当用户被要求输入时会看到该提示。按如下方式修改您的代码。

```
def main():
    x = get_int("What's x? ")
    print(f"x is {x}")


def get_int(prompt):
    while True:
        try:
            return int(input(prompt))
        except ValueError:
            pass


main()
```

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

  您可以在 Python 的 `pass` 文档中了解更多信息。

Summing Up 总结

Errors are inevitable in your code. However, you have the opportunity to use what was learned today to help prevent these errors. In this lecture, you learned about...

错误代码在代码中是不可避免的。但是，您有机会使用今天学到的知识来帮助防止这些错误。在本次讲座中，您了解了...

- Exceptions 异常
- Value Errors 值错误
- Runtime Errors 运行时错误
- `try` 尝试
- `else` 还
- `pass` 通过