CS50x Iran

Exceptions

Week note

پاییز و زمستان ۱۴۰۴

1 فهرست

- استثناها یا Exceptions مواردی هستند که در کد ما ایجاد خطا می‌کنند.
- در ویرایشگر متن، `code hello.py` را برای ایجاد یک فایل جدید به صورت زیر (با خطاهای عمدی) تایپ کنید:

```
hello.py

print("hello, world)
```

توجه داشته باشید که ما عمداً یک علامت نقل قول (") را نگذاشته ایم.

- با اجرای `python hello.py` در پنجره ترمینال، یک خطا ظاهر می‌شود. مفسر پایتون نشان می‌دهد که این یک `SyntaxError` یا خطای نگارشی است. خطای نگارشی یعنی شما در قسمتی از کد های خود، قواعد نوشتاری زبان پایتون را رعایت نکرده اید و مفسر (که بر اساس این قواعد کد شما را اجرا می‌کند)، نمی‌تواند درست عمل کند.
- می‌توانید در مستندات پایتون درباره خطاها و استثناها اطلاعات بیشتری کسب کنید.

- خطاهای زمان اجرا، در اثر یک اتفاق غیر منتظره در کد شما خطا رخ می‌دهند. به عنوان مثال، ممکن است منظور شما این بوده باشد که کاربر یک عدد وارد کند، اما آنها به جای آن یک کاراکتر وارد کنند. برنامه شما ممکن است به دلیل این ورودی غیر منتظره از کاربر، به خطا برخورد کند.
- در پنجره ترمینال خود، `code number.py` را اجرا کنید. در ویرایشگر متن کد زیر را وارد کنید:

```
number.py

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

- توجه داشته باشید که با افزودن `f`، به پایتون می‌گوییم که مقدار متغیر `x` را در محلی که با `{x}` مشخص کرده ایم قرار دهد. با آزمایش کد خود، می‌توانید تصور کنید که چگونه می‌توان به راحتی یک رشته یا یک کاراکتر را به جای یک عدد تایپ کرد. حتی کاربر می‌تواند هیچ چیزی هم تایپ نکند و فقط کلید Enter را بزند.

- به عنوان برنامه نویس، ما باید آماده باشیم و اطمینان حاصل کنیم که کاربران ما آنچه را که انتظار داشتیم، وارد کنند. ممکن است مواردی مانند -1 ، 0 یا cat را وارد کند.
- اگر این برنامه را اجرا کنیم و cat را تایپ کنیم، ناگهان خواهیم دید:

```
ValueError: invalid literal for int() with base 10: 'cat'
```

اساساً، مفسر پایتون دوست ندارد که ما cat را به عنوان ورودی نوشته ایم.

- یک استراتژی موثر برای رفع این خطای بالقوه ایجاد «مدیریت خطا» است تا اطمینان حاصل کنیم که کاربر همانطور که ما قصد داریم رفتار می‌کند.
- می‌توانید در مستندات پایتون درباره خطاها و استثناها اطلاعات بیشتری کسب کنید.

- در پایتون try و except روش‌هایی برای تست کردن ورودی کاربر قبل از اینکه مشکلی پیش بیاید، هستند. کد خود را به صورت زیر تغییر دهید:

number.py

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

- توجه داشته باشید که چگونه با اجرای این کد و وارد کردن 50 مشکلی پیش نمی‌آید. با این حال، تایپ کردن cat یک خطا برای کاربر نمایش می‌دهد و به او راهنمایی می‌کند که چرا ورودی آنها پذیرفته نشده است.

- این هنوز هم بهترین راه برای پیاده سازی این کد نیست. توجه کنید که ما می‌خواهیم دو خط کد بنویسیم. در بهترین حالت، شما باید کمترین تعداد خط کد ممکن را که نگران هستید به مشکل برخورد کند، در try قرار دهید. کد خود را به صورت زیر تغییر دهید:

number.py

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

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

- توجه کنید که با این کار به هدفمان برای قرار دادن کمترین تعداد خط ممکن در try رسیدیم. اکنون با یک خطای جدید روبرو هستیم! با یک NameError که x تعریف نشده است. به این کد نگاه کنید و فکر کنید که چرا x در برخی موارد تعریف نشده است؟
- اگر ترتیب عملیات را در `x = int(input("What's x?"))` بررسی کنید، ممکن است اشتباهی به جای عدد یک کاراکتر وارد شده باشد و سعی کند آن را به عنوان یک عدد صحیح در x قرار دهد. (که گفتیم این کار انجام نمی‌شود) و اگر این کار انجام نشود، تخصیص مقدار به x هرگز اتفاق نمی‌افتد. بنابراین، هیچ x برای چاپ در خط نهایی کد ما وجود ندارد!

- به نظر می‌رسد که راه دیگری برای نوشتن `try` وجود دارد که می‌تواند جلوی خطاهایی از این نوع را بگیرد.
- کد خود را به صورت زیر تنظیم کنید:

number.py

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

- توجه داشته باشید که اگر هیچ خطایی رخ ندهد، بلوک کد داخل `else` اجرا می‌شود .
- با اجرای `python number.py` و وارد کردن 50 به عنوان ورودی متوجه خواهید شد که نتیجه چاپ خواهد شد. در یک تلاش مجدد، این بار با نوشتن cat به عنوان ورودی، متوجه خواهید شد که برنامه شما به خطا برخورد کرده است.

- در این بهبودی که به کد خود داده‌ایم، نسبت به کاربر خود کمی بی‌احترامی کرده‌ایم. در حال حاضر، اگر کاربر ما ورودی اشتباه وارد کند، به سادگی برنامه خود را پایان می‌دهیم. در نظر بگیرید که می‌توانیم از یک حلقه برای درخواست x از کاربر استفاده کنیم. کد خود را به صورت زیر بهبود دهید:

number.py

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

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

- توجه داشته باشید که `while True` همیشه اجرا خواهد شد. اگر کاربر موفق به ارائه ورودی صحیح شود، می‌توانیم از حلقه خارج شویم و سپس خروجی را چاپ کنیم. اکنون، از کاربری که چیزی را به اشتباه وارد می‌کند، مجددا وارد کردن ورودی درخواست می‌شود.

9 ایجاد یک تابع برای گرفتن عدد صحیح

- مطمئناً دفعات زیادی وجود دارد که می‌خواهیم یک عدد صحیح از کاربر خود دریافت کنیم. کد خود را به صورت زیر تغییر دهید:

number.py

```
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()
```

- توجه داشته باشید که ما در حال افزودن تعداد زیادی ویژگی خوب هستیم. اول، ما توانایی گرفتن یک عدد صحیح را از بقیه برنامه جدا کردیم. حال، کل این برنامه به سه خط اول برنامه خلاصه می‌شود.

- حتی هنوز هم می‌توانیم این برنامه را بهبود ببخشیم. در نظر بگیرید که چه کارهای دیگری می‌توانید برای بهبود این برنامه انجام دهید. کد خود را به صورت زیر تغییر دهید:

number.py

```
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()
```

توجه داشته باشید که `return` نه تنها شما را از یک حلقه خارج می‌کند، بلکه یک مقدار را نیز بر می‌گرداند.

- برخی افراد ممکن است استدلال کنند که شما می‌توانید به این صورت کد قبل را خلاصه تر کنید:

number.py

```
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()
```

- توجه داشته باشید که این همان روند کد ما را انجام می‌دهد، فقط تعداد خطوط کمتری دارد.

- می‌توانیم کد را طوری بنویسیم که به کاربر هشداری ندهد، بلکه به سادگی با تغییر کد خود به صورت زیر، سؤال ما را دوباره از آنها بپرسد:

number.py

```
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()
```

- توجه داشته باشید که کد ما همچنان کار می‌کند، اما به طور مکرر کاربر را از خطای به وجود آمده مطلع نمی‌کند. در برخی موارد، باید کاملا برای کاربر واضح باشد که چه خطایی ایجاد شده است. در مواقع دیگر، ممکن است تصمیم بگیرید که می‌خواهید فقط دوباره از آنها ورودی را درخواست کنید.

- یک اصلاح پایانی که می‌تواند اجرای تابع `get_int` را بهبود بخشد. توجه کنید که تابع `get_int` صرفا برای گرفتن مقدار `x` از کاربر طراحی شده است. بهتر است پیغامی که برای دریافت مقدار از کاربر چاپ می‌کنیم را به شکل زیر به `get_int` بدهیم تا هر جای برنامه به هر شکلی خواستیم از آن استفاده کنیم:

number.py

```
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()
```

- در مستندات پایتون می‌توانید اطلاعات بیشتری درباره pass کسب کنید.

در این هفته مباحث زیر را مورد بررسی قرار دادیم:

- ✓ Exceptions
- ✓ Runtime Errors
- ✓ `try` و `except`
- ✓ `else`
- ✓ ایجاد یک تابع برای گرفتن عدد صحیح
- ✓ `pass`

حالا یک توانایی به لیست رو به افزایش توانایی های شما در زبان پایتون اضافه شد. این رشد ادامه داره...

CS50x Iran

Harvard's Computer Science 50x Iran