مدیریت استثناها در پایتون با `try`, `except`, `else`, `finally`

مدیریت استثناها (Exception Handling) در پایتون یکی از مهم‌ترین مهارت‌هایی است که به برنامه‌نویسان کمک می‌کند تا از متوقف شدن ناگهانی برنامه به دلیل وقوع خطا جلوگیری کنند. برای این کار از بلوک‌های `try`, `except`, `else` و `finally` استفاده می‌شود.

---

1. بلوک `try` و `except`

بلوک `try` برای اجرای کدهایی استفاده می‌شود که ممکن است باعث بروز خطا شوند. اگر خطایی رخ دهد، برنامه وارد بلوک `except` شده و از کرش شدن جلوگیری می‌شود.

مثال ساده:

```
try:
    num = int(input("یک عدد وارد کنید: "))
    print(f"عدد وارد شده: {num}")
except ValueError:
    print("لطفاً یک عدد صحیح وارد کنید!")
```

🔹 **توضیح:**

- اگر ورودی عدد صحیح باشد، برنامه بدون مشکل ادامه پیدا می‌کند.
- اگر کاربر مقدار غیرعددی وارد کند (مثل `"hello"`)، خطای `ValueError` رخ می‌دهد و پیام داخل `except` اجرا می‌شود.

---

2. استفاده از `except` برای انواع مختلف خطاها

می‌توان چندین بلوک `except` برای مدیریت انواع مختلف خطاها تعریف کرد.

مثال:

```
try:
    x = 10 / int(input("یک عدد وارد کنید: "))
except ZeroDivisionError:
    print("خطا: تقسیم بر صفر مجاز نیست!")
except ValueError:
    print("خطا: مقدار وارد شده باید یک عدد باشد!")
except Exception as e:
    print(f"یک خطای غیرمنتظره رخ داد: {e}")
```

🔹 **توضیح:**

- اگر کاربر `0` وارد کند، خطای `ZeroDivisionError` رخ می‌دهد.
- اگر مقدار غیرعددی وارد کند، خطای `ValueError` رخ می‌دهد.
- `Exception as e` برای گرفتن تمام خطاهای دیگر و نمایش توضیحات مربوط به خطا استفاده شده است.

---

3. بلوک `else` در مدیریت خطاها

بلوک `else` در صورتی اجرا می‌شود که هیچ استثنایی در `try` رخ ندهد.

مثال:

```
try:
    num = int(input("یک عدد وارد کنید: "))
except ValueError:
    print("خطا: مقدار وارد شده باید عدد صحیح باشد!")
else:
    print(f"شما عدد {num} را وارد کردید.")
```

🔹 **توضیح:**

- اگر ورودی معتبر باشد، `else` اجرا شده و مقدار دریافت‌شده نمایش داده می‌شود.
- اگر خطا رخ دهد، `except` اجرا شده و `else` نادیده گرفته می‌شود.

---

4. بلوک `finally` برای عملیات پایانی

بلوک `finally` همیشه اجرا می‌شود، چه خطایی رخ دهد و چه ندهد. این بخش معمولاً برای آزادسازی منابع مانند بستن فایل‌ها، قطع اتصال پایگاه داده و غیره استفاده می‌شود.

مثال:

```
try:
    file = open("test.txt", "r")
    content = file.read()
    print(content)
except FileNotFoundError:
    print("خطا: فایل موردنظر یافت نشد!")
finally:
    print("برنامه به پایان رسید، در صورت باز بودن فایل، آن را ببندید.")
    if 'file' in locals():
        file.close()
```

🔹 **توضیح:**

- اگر فایل موجود نباشد، خطای `FileNotFoundError` گرفته می‌شود.
- `finally` همیشه اجرا می‌شود تا عملیات پایانی مانند بستن فایل انجام شود.

---

5. ترکیب `else`, `except`, `try` , و `finally` در یک مثال کامل

```
try:
    num1 = int(input("عدد اول را وارد کنید: "))
    num2 = int(input("عدد دوم را وارد کنید: "))
    result = num1 / num2
except ZeroDivisionError:
    print("خطا: تقسیم بر صفر مجاز نیست!")
except ValueError:
    print("خطا: لطفاً فقط عدد وارد کنید!")
else:
    print(f"نتیجه تقسیم: {result}")
finally:
    print("عملیات به پایان رسید.")
```

🔷 **توضیح:**

1. درون `try` دو عدد از کاربر دریافت شده و تقسیم می‌شوند.
2. اگر عدد دوم `0` باشد، `except ZeroDivisionError` اجرا می‌شود.
3. اگر مقدار غیرعددی وارد شود، `except ValueError` اجرا می‌شود.
4. اگر خطایی رخ ندهد، `else` اجرا شده و نتیجه چاپ می‌شود.
5. در هر صورت، `finally` اجرا شده و پیام "عملیات به پایان رسید" نمایش داده می‌شود.

جمع‌بندی

بلوک	وظیفه
`try`	اجرای کدهایی که ممکن است خطا داشته باشند
`except`	مدیریت خطاها و جلوگیری از کرش شدن برنامه
`else`	اجرای کد فقط در صورتی که هیچ خطایی رخ ندهد
`finally`	اجرای کد در هر شرایطی (حتی در صورت وقوع خطا)

مدیریت استثناها یکی از بهترین روش‌ها برای جلوگیری از خرابی برنامه است و باعث افزایش پایداری و امنیت کد شما می‌شود.

مدیریت خطاهای فایل‌ها با `try`, `except`, `else`, `finally`

هنگام کار با فایل‌ها، ممکن است خطاهایی مانند **عدم وجود فایل، عدم دسترسی، یا مشکلات در خواندن و نوشتن اطلاعات** رخ دهد. استفاده از `try` , `except` , `else` , و `finally` به ما کمک می‌کند تا این خطاها را مدیریت کرده و از کرش شدن برنامه جلوگیری کنیم.

---

1. بررسی وجود فایل قبل از خواندن آن

اگر فایلی که قصد خواندن آن را داریم وجود نداشته باشد، خطای `FileNotFoundError` رخ می‌دهد.

مثال: خواندن یک فایل و مدیریت خطای `FileNotFoundError`

```
try:
    file = open("data.txt", "r")  # تلاش برای باز کردن فایل
    content = file.read()
except FileNotFoundError:
    print("خطا: فایل موردنظر یافت نشد!")
else:
    print("محتوای فایل:")
    print(content)
finally:
    print("پایان عملیات خواندن فایل.")
    if 'file' in locals():  # اگر فایل باز شده باشد، آن را ببند
        file.close()
```

🔷 **توضیح:**

- اگر فایل **وجود داشته باشد**، محتوای آن خوانده و نمایش داده می‌شود.
- اگر فایل **وجود نداشته باشد**، `except` اجرا شده و از کرش شدن جلوگیری می‌شود.
- `finally` همیشه اجرا شده و در صورت باز بودن فایل، آن را می‌بندد.

---

2. مدیریت خطاهای نوشتن در فایل

هنگام نوشتن در یک فایل، ممکن است دسترسی به فایل نداشته باشیم یا مسیر فایل اشتباه باشد.

مثال: نوشتن در یک فایل با مدیریت خطاها

```
try:
    file = open("output.txt", "w")  # باز کردن فایل در حالت نوشتن
    file.write("این یک تست برای مدیریت خطاها در فایل‌ها است.\n")
except PermissionError:
    print("خطا: مجوز لازم برای نوشتن در فایل را ندارید!")
except Exception as e:
    print(f"یک خطای غیرمنتظره رخ داد: {e}")
else:
    print("نوشتن در فایل با موفقیت انجام شد.")
finally:
    print("پایان عملیات نوشتن در فایل.")
    if 'file' in locals():
        file.close()
```

🔷 **توضیح:**

- اگر فایل بدون مشکل باز شود، متن در آن نوشته می‌شود.
- اگر مجوز نوشتن در فایل وجود نداشته باشد، `PermissionError` اجرا می‌شود.
- `Exception as e` برای گرفتن هر نوع خطای دیگر استفاده شده است.
- در `finally` فایل بسته می‌شود.

3. مدیریت خطا هنگام خواندن از یک فایل غیرقابل خواندن

اگر فایلی که باز کرده‌ایم فقط برای نوشتن (`"w"`) باز شده باشد، نمی‌توانیم از آن بخوانیم.

مثال: باز کردن یک فایل و تلاش برای خواندن آن در حالت نوشتن

```
try:
    file = open("output.txt", "w")  # باز کردن فایل در حالت نوشتن
    content = file.read()  # (خطا رخ می‌دهد) تلاش برای خواندن
except UnsupportedOperation:
    print("خطا: این فایل در حالت نوشتن باز شده و امکان خواندن وجود ندارد!")
except Exception as e:
    print(f"یک خطای غیرمنتظره رخ داد: {e}")
else:
    print("محتوای فایل خوانده شد.")
finally:
    print("بستن فایل برای جلوگیری از مشکلات احتمالی.")
    if 'file' in locals():
        file.close()
```

🔹 **توضیح:**

- `UnsupportedOperation` وقتی رخ می‌دهد که یک عملیات نامعتبر روی فایل انجام شود (مثلاً خواندن از فایلی که فقط برای نوشتن باز شده).
- در `finally` فایل بسته می‌شود تا از مشکلات بعدی جلوگیری شود.

---

4. مدیریت چندین خطا هنگام باز کردن فایل

گاهی ممکن است چندین خطا به‌طور هم‌زمان رخ دهند، مانند **عدم وجود فایل، نداشتن دسترسی، یا سایر مشکلات مرتبط با فایل**.

مثال: مدیریت چند خطای احتمالی

```
try:
    file = open("data.txt", "r")  # تلاش برای باز کردن فایل
    content = file.read()
except FileNotFoundError:
    print("خطا: فایل وجود ندارد!")
except PermissionError:
    print("خطا: شما اجازه دسترسی به این فایل را ندارید!")
except Exception as e:
    print(f"یک خطای غیرمنتظره رخ داد: {e}")
else:
    print("فایل با موفقیت خوانده شد.")
finally:
    print("بستن فایل در صورت باز بودن.")
    if 'file' in locals():
        file.close()
```

🔹 **توضیح:**

- این کد چندین نوع خطا را پوشش می‌دهد:

- **اگر فایل وجود نداشته باشد:** `FileNotFoundError`
- **اگر دسترسی لازم را نداشته باشیم:** `PermissionError`
- **اگر خطای دیگری رخ دهد:** `Exception as e`
- در نهایت، اگر فایل باز باشد، بسته خواهد شد.

---

5. ترکیب خواندن و نوشتن با مدیریت خطاها

اگر بخواهیم داده‌ای را از یک فایل بخوانیم و در فایل دیگری بنویسیم، بهتر است خطاهای احتمالی را در هر دو مرحله مدیریت کنیم.

مثال: خواندن از یک فایل و نوشتن در فایل دیگر

```
try:
    with open("input.txt", "r") as infile:  # باز کردن فایل ورودی
        data = infile.read()

    with open("output.txt", "w") as outfile:  # باز کردن فایل خروجی
        outfile.write(data)

except FileNotFoundError:
    print("خطا: فایل ورودی یافت نشد!")
except PermissionError:
    print("خطا: مجوز لازم برای دسترسی به یکی از فایل‌ها وجود ندارد!")
except Exception as e:
    print(f"یک خطای غیرمنتظره رخ داد: {e}")
else:
    print("داده‌ها با موفقیت از یک فایل به فایل دیگر منتقل شدند.")
finally:
    print("عملیات فایل‌ها به پایان رسید.")
```

🔹 **توضیح:**

- `with open()` استفاده شده تا نیاز به `finally` برای بستن فایل‌ها حذف شود.
- خطاهای احتمالی شامل **عدم وجود فایل ورودی، نداشتن مجوز نوشتن، یا خطاهای دیگر** مدیریت شده‌اند.
- اگر هیچ خطایی رخ ندهد، `else` اجرا شده و پیام موفقیت نمایش داده می‌شود.

نتیجه‌گیری

✅ مدیریت خطاها در فایل‌ها باعث جلوگیری از کرش شدن برنامه و بهبود تجربه کاربری می‌شود.
✅ ترکیب `else`، `except`، `try` و `finally` در عملیات فایل‌ها تضمین می‌کند که منابع به درستی مدیریت شوند.
✅ استفاده از `with open()` برای باز کردن فایل‌ها، نیاز به بستن دستی آن‌ها را کاهش می‌دهد.

📌 **پیشنهاد:** همیشه قبل از خواندن یا نوشتن فایل‌ها، بررسی کنید که فایل وجود دارد و دسترسی‌های لازم داده شده است.