## استفاده از `try`, `except`, `else`, `finally` برای مدیریت استثناها

در پایتون، برای مدیریت استثناها از ساختار `try`, `except`, `else` و `finally` استفاده می‌شود. این ساختار به شما این امکان را می‌دهد که به‌طور مؤثری خطاها را شناسایی و مدیریت کنید و رفتارهای مختلف برنامه را برای مواقع مختلف کنترل کنید.

### 1. ساختار کلی `try`, `except`, `else`, `finally`

- `try` : بخش اصلی که کد ممکن است در آن خطا ایجاد شود.
- `except` : این بخش برای مدیریت استثناهایی است که ممکن است در بلوک `try` رخ دهند.
- `else` : این بخش تنها در صورتی اجرا می‌شود که هیچ استثنایی در `try` رخ ندهد.
- `finally` : این بخش همیشه اجرا می‌شود، چه استثنایی رخ دهد یا نه. برای کارهایی که باید پس از اجرای کد اصلی انجام شوند (مثل بستن فایل‌ها یا آزاد کردن منابع).

### 2. مثال‌های کاربردی

#### استفاده از `try`, `except`, `else` و `finally`

```
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError as e:
        print(f"Error: Cannot divide by zero! {e}")
    else:
        print(f"The result is {result}")
    finally:
        print("This block always runs, whether an error occurred or not.")

# تست تابع با مقادیر مختلف
divide(10, 2)  # بدون خطا
divide(10, 0)  # خطای تقسیم بر صفر
```

**خروجی:**

```
The result is 5.0
This block always runs, whether an error occurred or not.
Error: Cannot divide by zero! division by zero
This block always runs, whether an error occurred or not.
```

#### توضیح کد:

1. `try` : تابع `divide` سعی می‌کند که `a` را بر `b` تقسیم کند.
2. `except` : در صورتی که خطای `ZeroDivisionError` رخ دهد، پیام خطا چاپ می‌شود.
3. `else` : اگر خطایی در کد `try` وجود نداشته باشد، نتیجه محاسبه شده نمایش داده می‌شود.
4. `finally` : این بخش همیشه اجرا می‌شود و برای انجام کارهایی مانند آزادسازی منابع (مثلاً بستن فایل‌ها) یا چاپ پیغام‌های تکمیلی استفاده می‌شود.

## 3. استفاده از `else` و `finally`

- **`else`**: این بخش تنها زمانی اجرا می‌شود که هیچ استثنایی در بلوک `try` رخ ندهد. این مفید است زمانی که می‌خواهید کدهایی را اجرا کنید که فقط در صورت موفق بودن کد `try` باید انجام شوند (مانند چاپ نتایج یا ذخیره‌سازی داده‌ها).
- **`finally`**: این بخش همیشه اجرا می‌شود. حتی اگر در بلوک `try` استثنای دیگری رخ دهد، یا اگر خطا مدیریت شود، `finally` همچنان اجرا می‌شود. این برای کارهایی مثل بستن فایل‌ها، آزاد کردن منابع یا چاپ پیغام‌های پایانی مفید است.

### مثال: استفاده از `finally` برای بستن فایل

```
def read_file(file_name):
    try:
        file = open(file_name, 'r')
        content = file.read()
        print("File content:", content)
    except FileNotFoundError as e:
        print(f"Error: {e}")
    else:
        print("File read successfully.")
    finally:
        if 'file' in locals():
            file.close()  # بستن فایل پس از اتمام کار
            print("File has been closed.")

# تست با فایل موجود
read_file('example.txt')

# تست با فایل غیرموجود
read_file('non_existent_file.txt')
```

**خروجی نمونه:**

```
File content: This is an example file.
File read successfully.
File has been closed.

Error: [Errno 2] No such file or directory: 'non_existent_file.txt'
File has been closed.
```

## 4. جمع‌بندی

- **`try`**: برای قرار دادن کدی که ممکن است استثنا ایجاد کند.
- **`except`**: برای مدیریت خطاهایی که در `try` رخ می‌دهند.
- **`else`**: برای اجرای کدی که فقط در صورتی که خطا رخ نداده باشد اجرا می‌شود.
- **`finally`**: برای انجام کارهایی که باید همیشه انجام شوند (مثل بستن فایل‌ها، آزادسازی منابع و غیره).

استفاده از این ساختارها کمک می‌کند تا کدتان پایدارتر و قابل اعتمادتر شود و بتوانید استثناها و خطاها را به‌طور مؤثر مدیریت کنید.