Undtagelseshåndtering

Undtagelseshåndtering: Forstå, hvordan du kan sikre robust kode gennem korrekt håndtering af fejl og undtagelser, så dine programmer ikke går ned uventet.

Try-Except Blokke

Lær at fange og håndtere fejl i Python ved brug af `try-except` blokke. Dette giver mulighed for at kontrollere, hvordan programmet reagerer på uventede situationer, som f.eks. filen ikke eksisterer, eller der er problemer med netværksforbindelse. Ved at bruge `try-except` kan man sikre, at programmet ikke blot fejler, men i stedet giver brugeren en meningsfuld besked eller forsøger en alternativ løsning.

Håndtering af Specifikke Fejl

Få erfaring med at identificere og håndtere specifikke undtagelsestyper, såsom `FileNotFoundError`, `ValueError`, eller `ZeroDivisionError`. Dette gør din fejlhåndtering mere præcis og sikrer, at dit program kan håndtere specifikke problemer uden at fange alle fejl, hvilket ellers kan skjule uforudsete bugs.

Else og Finally

Forståelsen af brugen af `else` og `finally` blokke i fejlhåndtering hjælper med at skabe en mere detaljeret og robust fejlhåndteringsstruktur. `else` bruges, når der ikke opstod nogen undtagelse, hvilket gør det lettere at differentiere succesfulde operationer fra dem, der har brug for fejlhåndtering. `finally` bruges til at sikre, at bestemte opgaver udføres uanset om en undtagelse opstod eller ej, f.eks. at lukke en fil eller frigive ressourcer.

Oprettelse af Brugerdefinerede Undtagelser

Lær hvordan og hvorfor man opretter brugerdefinerede undtagelser for at skabe mere meningsfulde fejlmeddelelser til komplekse applikationer. Ved at oprette egne undtagelser kan du sørge for, at din kode kommunikerer præcis, hvad problemet er, hvilket gør fejlsøgning lettere og forbedrer programmets læsbarhed. Dette er særligt nyttigt i større programmer, hvor standardfejl kan være utilstrækkelige til at beskrive specifikke, domæne-relaterede fejltilfælde.

Eksempel

```
def divide_numbers(a, b):
    try:
```

```
        result = a / b
        print(f"Result: {result}")
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
    except TypeError:
        print("Error: Invalid input type. Please use numbers.")
    except Exception as e:
        print("Error: An unexpected error happened.")
        print(e)
    finally:
        print("Execution completed.")

# Example usage
divide_numbers(10, 2)  # Normal case
divide_numbers(10, 0)  # Division by zero
divide_numbers(10, "a")  # Invalid input type
```

Ressourcer

Her er en samling af forskellige ressourcer som du kan bruge i forbindelse med din læring og løsning af opgaverne i denne uge.

- Python's officielle dokumentation om undtagelseshåndtering:
  https://docs.python.org/3/tutorial/errors.html
  https://docs.python.org/3/library/exceptions.html
- Real Python - Python Exception Handling: https://realpython.com/python-exceptions/
- Geeks for Geeks - Python Exception Handling:
  https://www.geeksforgeeks.org/python-exception-handling/
- Programiz - Python Try Except:
  https://www.programiz.com/python-programming/exception-handling
- Pythonforbeginners - Python Try Except:
  https://www.pythonforbeginners.com/error-handling/python-try-and-except