Výjimky a aserce, debugování a zpracování chyb

Výjimky

Výjimky v programovacích jazycích představují mechanismus pro signalizaci a zpracování chyb nebo abnormálních situací během běhu programu. Fungují na principu objektově orientovaného přístupu k ošetření chyb.

Při detekci chyby Python vytvoří objekt výjimky obsahující informace o nastalé chybě. Následně přeruší normální běh programu a hledá příslušný blok kódu, který tuto výjimku dokáže zpracovat. Pokud není nalezen, program se ukončí s chybovou hláškou.

Hierarchie výjimek

Všechny výjimky v Pythonu jsou odvozeny od základní třídy `BaseException`. Běžně používané výjimky dědí od třídy `Exception`. Hierarchie zahrnuje kategorie
ArithmeticError - matematické chyby při výpočtech (dělení nulou, přečtení)
TypeError - situace kdy funkce dostane argument nesprávného typu (text místo čísla atd.)
ValueError - správná hodnota, ale nevhodná hodnota (odmocnina pro záporné číslo atd.)
IOError - práce se soubory (soubor nelze otevřít, není oprávnění…)

Mechanismus try-except

try - obsahuje kód, kde může nastat výjimka
except - definuje zpracování konkrétních typů výjimek
else - volitelné, provede se pokud v try nenastala žádná výjimka
finally - volitelné, provede se i když nastala výjimka

K vyvolání se používá klíčové slovo raise.

```
while True:
    # Základní příklad zpracování výjimek v Pythonu
    try:
        # Načtení vstupu a převod na číslo
        cislo = int(input("Zadej číslo: "))

        # Pokus o výpočet s tímto číslem
        vysledek = 100 / cislo

        # Zobrazení výsledku
        print(f"Výsledek dělení 100 / {cislo} = {vysledek}")

    except ValueError:
        # Zachycení chyby při převodu na číslo
        print("Chyba: Zadaná hodnota není platné číslo")

    except ZeroDivisionError:
        # Zachycení chyby při dělení nulou
```

```
        print("Chyba: Nelze dělit nulou")

    except Exception as e:
        # Zachycení všech ostatních chyb
        print(f"Nastala neočekávaná chyba: {e}")

    else:
        # Kód, který se provede, pokud nenastala žádná výjimka
        print("Výpočet proběhl úspěšně!")

    finally:
        # Kód, který se provede vždy, bez ohledu na výjimky
        print("Konec programu")
```

Vlastní výjimky

Jazyky nám umožňují vytvářet i vlastní typy výjimek odvozením od třídy Exception nebo jejich podtříd. Vlastní výjimky nám umožňují vytvořit hodně konkrétní vlastní, která třeba by dělala jiným.

Aserce

Kontrolní mechanismus pro ověření předpokladů v kódu. Kontrolují, že program funguje podle očekávání programátora.

Výjimky řeší očekávatelné chyby za běhu, aserce odhalují programátorské chyby a chyby v logice programu

V pythonu jsou implementovány pomocí klíčového slova assert, které evaluuje boolean výraz a v případě, že je true, tak je vše good, když je false tak vyvolá AssertionError

```
def start_program(data: dict):
    assert isinstance(data, dict), "Invalid data"
    assert data, "Empty..."

    print("Load successfully")

if __name__ == '__main__':
    json = {'user' : 123}
    start_program(data=json)
```

Máme metodu start_program, která má na vstupu slovník. První assert řeší to, jestli to je vůbec slovník, pokud ne tak vyhodí raise vlastní error - Invalid data. Druhý assert řeší to, jestli je vůbec něco v tom seznamu, pokud ne tak vlastní error - Empty….. Pokud je všechno v chillu tak normálně se vypíše Load successfully.

Debugování
Systematický proces identifikace, analýzy a odstranění chyby v programu. V kódu se může vyskytnout syntaktické, logické a běhové chyby.

Základní techniky debugování

1. Využití print

Nejjednodušší technika debugování spočívá ve strategickém umístění příkazů `print()` do kódu pro zobrazení hodnot proměnných a sledování toku programu.

```
def vypocet_ceny(cena, mnozstvi, sleva):
    print(f"Vstupní hodnoty: cena={cena}, množství={mnozstvi}, sleva={sleva}")

    mezivysledek = cena * mnozstvi
    print(f"Mezivýsledek (před slevou): {mezivysledek}")

    vysledek = mezivysledek * (1 - sleva / 100)
    print(f"Výsledek (po slevě): {vysledek}")

    return vysledek
```

2. Python debugger (pdb)

Python obsahuje vestavěný modul `pdb`, který poskytuje interaktivní prostředí pro debugování.

```
import pdb

def problematicka_funkce(data):
    vysledek = 0
    pdb.set_trace()  # Zde se spustí debugger
    for i in range(len(data)):
        vysledek += data[i]
    return vysledek
```

3. IDE a grafické debuggery

Prostředí jako PyCharm, VS Code) nabízejí debugging přes breakpointy

Zpracování chyb

Zpracování chyb je klíčovou součástí tvorby spolehlivého a robustního kódu. V reálném světě programy často naráží na problémy – soubory neexistují, síťová spojení selhávají, uživatelé zadávají neočekávaná data. Dobře navržené zpracování chyb umožňuje programu tyto situace zvládnout elegantně. Používají se try/except bloky