8

Robusnost i performanse

Kada jednom napišete koristan program na Pythonu, naredni korak je da vaš kod dobije *upotrebnu* vrednost i da bude otporan na greške. Podjednako je važno da programi budu pouzdani kada naiđu na neočekivane okolnosti koliko i da imaju odgovarajuću funkcionalnost. Python ima ugrađene mogućnosti i module koji pomažu u kaljenju programa tako da budu robusni u raznim situacijama.

Jedna dimenzija robusnosti je skalabilnos i performansa. Kada implementirate Pythonove programe koji se bave netrivijalnom količinom podataka, često ćete primetiti usporenja izazvana složenošću algoritma u kodu ili drugim tipovima opterećenja pri izračunavanju. Srećom, Python sadrži mnoge algoritme i strukture podataka koji su vam potrebni da dostignete visoke performanse uz minimalan napor.

Tema 65: Iskoristite prednosti svakog `try/except/else/finally` bloka

Postoje četiri uočljive situacije u kojim biste mogli poželeti da preduzmete nešto tokom rada sa izuzecima u Pythonu. One su obuhvaćene funkcijom blokova `try`, `except`, `else` i `finally`. Svaki blok ima jedinstvenu namenu u složenoj naredbi, a njihove razne kombinacije su korisne (Tema 87: Definišite korenski `Exception` da biste izolovali pozivaoce od API-ja, sadrži još jedan primer).

Blokovi `finally`

Koristite `try/finally` kada želite da izuzeci budu preneti naviše, ali i da izvršite kod za raščišćavanje čak i kada se jave izuzeci. Jedna uobičajena namena blokova `try/finally` jeste pouzdano zatvaranje identifikatora datoteka (Tema 66: Razmotrite naredbe `contextlib` i `with` za ponovljivo ponašanje `try/finally`, za još jedan – verovatno bolji – pristup):

```
def try_finally_example(filename):
    print('* Opening file')
    handle = open(filename, encoding='utf-8') # Možda OSError
    try:
```

```
        print('* Reading data')
        return handle.read()  # Možda UnicodeDecodeError
    finally:
        print('* Calling close()')
        handle.close()        # Uvek se izvršava nakon bloka try
```

Bilo koji izuzetak koji generiše metoda `read` uvek će se preneti naviše do pozivnog koda, ali metoda `close` za `handle` u bloku `finally` prva će se izvršiti:

```
filename = 'random_data.txt'

with open(filename, 'wb') as f:
    f.write(b'\xf1\xf2\xf3\xf4\xf5')  # Neispravan utf-8

data = try_finally_example(filename)

>>>
* Opening file
* Reading data
* Calling close()
Traceback ...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in
➥position 0: invalid continuation byte
```

Morate pozvati `open` pre bloka `try` jer izuzeci koji se javljaju pri otvaranju datoteke (kao što je `OSError` ako datoteka ne postoji) treba potpuno da preskoče blok `finally`:

```
try_finally_example('does_not_exist.txt')

>>>
* Opening file
Traceback ...
FileNotFoundError: [Errno 2] No such file or directory:
➥'does_not_exist.txt'
```

Blokovi else

Koristite blokove `try/except/else` da bi bilo jasno koje će izuzetke rešiti vaš kod, a koji će biti preneti naviše. Kada blok `try` ne generiše izuzetak, izvršava se blok `else`. Blok `else` pomaže da minimizujete količinu koda u bloku `try`, što je dobro za izolovanje potencijalnih problema sa izuzecima i poboljšava čitljivost. Na primer, ako želim da učitam podatke JSON rečnika iz znakovnog niza i da vratim vrednost ključa koji on sadrži:

```
import json

def load_json_key(data, key):
    try:
        print('* Loading JSON data')
        result_dict = json.loads(data)  # Moguć ValueError
    except ValueError as e:
        print('* Handling ValueError')
        raise KeyError(key) from e
    else:
        print('* Looking up key')
        return result_dict[key]         # Moguć KeyError
```

U uspešnom slučaju, JSON podaci se dekodiraju u bloku `try`, a potom se traži ključ u bloku `else`:

```
assert load_json_key('{"foo": "bar"}', 'foo') == 'bar'
```

```
>>>
* Loading JSON data
* Looking up key
```

Ukoliko uneti podaci nisu ispravni JSON, dekodiranje pomoću `json.loads` generiše izuzetak `ValueError`. Izuzetak će uhvatiti i obraditi blok `except`:

```
load_json_key('{"foo": bad payload', 'foo')
```

```
>>>
* Loading JSON data
* Handling ValueError
Traceback ...
JSONDecodeError: Expecting value: line 1 column 9 (char 8)

The above exception was the direct cause of the following
➥exception:

Traceback ...
KeyError: 'foo'
```

Ukoliko traženje ključa generiše izuzetke, oni će se proširiti do pozivaoca jer su izvan bloka `try`. Klauzula `else` se stara da ono što sledi posle `try/except` bude vizuelno odvojeno od bloka `except`. Ovo čini ponašanje rasprostiranja izuzetka jasnim:

```
load_json_key('{"foo": "bar"}', 'does not exist')

>>>
* Loading JSON data
* Looking up key
Traceback ...
KeyError: 'does not exist'
```

Sve zajedno

Koristite blokove `try/except/else/finally` kada hoćete sve da uradite u jednoj složenoj naredbi. Na primer, ako hoću da iz datoteke učitam opis posla koji treba da se uradi, da ga obradim i potom ažuriram datoteku na licu mesta. Ovde se blok `try` koristi za čitanje i obradu datoteke; blok `except` se koristi za obradu očekivanih izuzetaka iz bloka `try`; blok `else` se koristi za ažuriranje datoteke na licu mesta i omogućavanje da povezani izuzeci budu preneti naviše; a blok `finally` raščišćava identifikator datoteke:

```
UNDEFINED = object()

def divide_json(path):
    print('* Opening file')
    handle = open(path, 'r+')   # Moguć OSError
    try:
        print('* Reading data')
        data = handle.read()    # Moguć UnicodeDecodeError
        print('* Loading JSON data')
        op = json.loads(data)   # Moguć ValueError
        print('* Performing calculation')
        value = (
            op['numerator'] /
            op['denominator'])  # Moguć ZeroDivisionError
    except ZeroDivisionError as e:
        print('* Handling ZeroDivisionError')
        return UNDEFINED
    else:
        print('* Writing calculation')
        op['result'] = value
        result = json.dumps(op)
        handle.seek(0)          # Moguć OSError
        handle.write(result)    # Moguć OSError
        return value
    finally:
        print('* Calling close()')
        handle.close()          # Uvek se izvršava
```

U uspešnom pokušaju, izvršavaju se blokovi `try`, `else` i `finally`:

```
temp_path = 'random_data.json'

with open(temp_path, 'w') as f:
    f.write('{"numerator": 1, "denominator": 10}')

assert divide_json(temp_path) == 0.1

>>>
* Opening file
* Reading data
* Loading JSON data
* Performing calculation
* Writing calculation
* Calling close()
```

Ukoliko je izračunavanje neispravno, izvršavaju se blokovi `try`, `except` i `finally`, a blok `else` se ne izvršava:

```
with open(temp_path, 'w') as f:
    f.write('{"numerator": 1, "denominator": 0}')

assert divide_json(temp_path) is UNDEFINED

>>>
* Opening file
* Reading data
* Loading JSON data
* Performing calculation
* Handling ZeroDivisionError
* Calling close()
```

Ako su JSON podaci neispravni, blok `try` se izvršava i generiše izuzetak, blok `finally` se izvršava, a potom se izuzetak prenosi naviše do pozivaoca. Blokovi `except` i `else` se ne izvršavaju:

```
with open(temp_path, 'w') as f:
    f.write('{"numerator": 1 bad data')

divide_json(temp_path)

>>>
* Opening file
* Reading data
* Loading JSON data
```

```
* Calling close()
Traceback ...
JSONDecodeError: Expecting ',' delimiter: line 1 column 17
➥(char 16)
```

Ovakav raspored je naročito koristan jer svi blokovi rade zajedno na intuitivan način. Na primer, ovde to simuliram izvršavajući funkciju `divide_json` u trenutku kada mom disku ponestaje prostora:

```
with open(temp_path, 'w') as f:
    f.write('{"numerator": 1, "denominator": 10}')

divide_json(temp_path)

>>>
* Opening file
* Reading data
* Loading JSON data
* Performing calculation
* Writing calculation
* Calling close()
Traceback ...
OSError: [Errno 28] No space left on device
```

Kada je izuzetak generisan u bloku `else` dok su se ponovo upisivali rezultujući podaci, blok `finally` se ipak izvršio i zatvorio identifikator datoteke kao što se očekivalo.

Zapamtite

- Kombinovana naredba `try/finally` omogućava da izvršite kod za raščišćavanje, bez obzira na to da li su generisani izuzeci u bloku `try`.
- Blok `else` pomaže da minimizujete količinu koda u blokovima `try` i da vizuelno odvojite uspešan slučaj od blokova `try/except`.
- Blok `else` se može koristiti da bi se obavile dodatne akcije nakon uspešnog bloka `try`, ali pre opšteg raščišćavanja u bloku `finally`.

Tema 66: Razmotrite naredbe `contextlib` i `with` za ponovljivo ponašanje `try/finally`

Naredba `with` se u Pythonu koristi za naznačavanje da se kod izvršava u posebnom kontekstu. Na primer, uzajamno isključivo zaključavanje (Tema 54: Koristite `Lock` da biste sprečili utrkivanje podataka u nitima) može se koristiti u naredbama `with` da bi se naznačilo da se uvučeni blok koda izvršava samo dok je zaključavanje aktivno: