Akademia Pythona

VII Wyjątki oraz narzędzia

KN Pythona - Kurs Pythona

KN Pythona wita na kursie Pythona.

Plan:

- Podstawy wyjątków
- Szczegółowe informacje dotyczące wyjątków
- Obiekty wyjątków
- Projektowanie z użyciem wyjątków

Podstawy wyjątków

Wyjątki w Pythonie służą do modifikacji przebiegu sterowania w programie. Wyjątęk nie zawsze oznacza błąd.

Instrukcje wyjątków

Instrukcje wyjątków:

- **try/except**
- **try/finally**
- **raise**
- **assert**
- **with/as**

Role wyjątków

Role wyjątków:

- Obsługa błędów
- Powiadomienie o zdarzeniach
- Obsługa przypadków specjalnych
- Działanie końcowe
- Niezwykły przebieg sterowania

Domyślny program obsługi wyjątków

```
def fetcher(obj, index):
    return obj[index]
x = 'abcd'
fetcher(x, 8) # Traceback ... IndexError
```

Przechwytywanie wyjątków

```
try:
    fetcher(x, 8)
except IndexError:
    print("Przechwycono wyjątek")
```

Zgłaszanie wyjątków

```
try:
    raise IndexError
except IndexError:
    print("Przechwycono wyjątek")
```

Zgłaszanie wyjątków

```
assert False, "Assert error here!"
# AssertionError: Assert error here!
```

Definiowanie wyjątków

```
class Bad(Exception):
    pass
def doomed():
    raise Bad()
try:
    doomed()
except Bad:
    print("Przechwycenie Bad")
```

Działania końcowe

```
def after():
    try:
        fetcher(x, 8)
    finally:
        print("Po fetcher")
    print("Po try")
after() # Po fetcher; IndexError
```

Mieszanie instrukcji

```
try:
    assert False, "false"
except:
    print("except")
else:
    print("else")
finally:
    print("finally")
```

Pełna forma instrukcji try

```
try:
    <instrukcje>
except <nazwa1>:
    <instrukcje>
except (nazwa2, nazwa3):
    <instrukcje>
except <nazwa4> as dane:
    <instrukcje>
else:
    <instrukcje>
finally:
    <instrukce>
```

Instrukcja raise

```
raise <instancja>
raise <klasa> # niejawne wywołanie klasa()
raise # ponowne zgłoszenie ostatniego wyjątku
exc = IndexError()
raise exc
```

Dostęp do instancji wyjątku

```
try:
    ...
except IndexError as X:
    ... # użycie X
```

Łańcuchy wyjątków Pythonie

```
try:
    1 / 0
except Exception as E:
    raise TypeError('Źle!') from E
```

Instrukcja assert

```
assert <test>, <dane> # <dane> opcjonalnie
if __debug__: # python -O main.py - zmienia debug na False
    if not <test>:
        raise AssertionError(<dane>)
```

Menedżery kontekstu

```
with wyrażenie [as zmienna]:
    blok_with
```

Menedżery kontekstu

```
with open('file.txt') as in_file:
    for line in in_file:
        print(line)
```

Menedżery kontekstu

```
in_file = open('file.txt')
try:
    for line in in_file:
        print(line)
finally:
    in_file.close()
```

Menedżery kontekstu

```
class TraceBlock:
    def message(self, arg):
        print(f"in message {arg}")
    def __enter__(self):
        return self
    def __exit__(self, exc_type, exc_value, exc_tb):
        if exc_type is None:
            print("normalne wyjście")
        else:
            print("reraising error")
            return False # tylko jeśli False
```

Menedżery kontekstu

```
with TraceBlock as action:
    action.message()
with TraceBlock as action:
    action.message()
    raise TypeError
```

Obiekty wyjątków

Obiekty wyjątków:

- Można je organizować w kategorie
- Dołączają informacje o stanie
- Obsługują dziedziczenie

Kategorie wyjątków

```
class General(Exception): pass
class Specific1(General): pass
class Specific2(General): pass
def raiser0():
    raise General
def raiser1():
    raise Specific1
def raiser2():
    raise Specific2
```

Kategorie wyjątków

```
for func in [raiser0, raiser1, raiser2]:
    try:
        func()
    except General:
        print("Przechwycono")
```

Stan wyjątku

```
try:
    raise IndexError('abc')
except IndexError as E:
    print(E.args)
```