Obsługa błędów (wyjątki)

1. Try – Except

Blok **try** umożliwia przetestowanie bloku kodu pod kątem błędów.

Blok **except** pozwala obsłużyć błąd.

Blok **else** umożliwia wykonanie kodu, gdy nie ma błędu.

Blok **finally** umożliwia wykonanie kodu, niezależnie od wyniku prób i z wyjątkiem bloków.

Pierwszym kroku język Python próbuje wykonać wszystkie instrukcje umieszczone pomiędzy twierdzeniami try: i except:

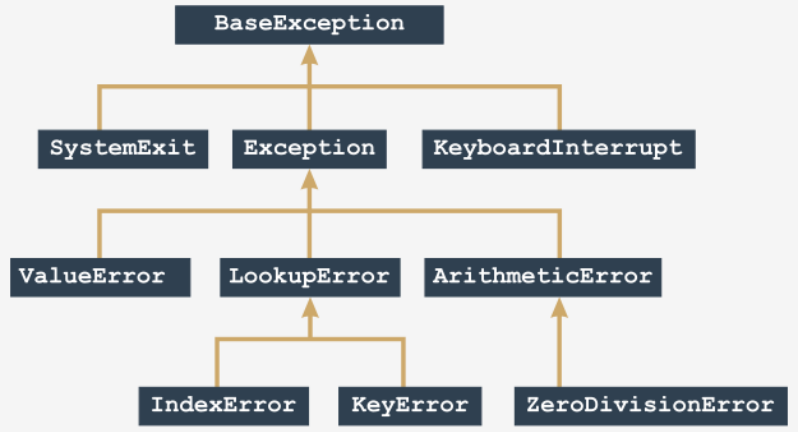
jeśli nie wystąpi błąd wykonania i wszystkie instrukcje zostaną wykonywane poprawnie, przechodzi on do punktu za ostatnią linią bloku **except**:, a wykonanie bloku uważa za zakończone;

```
#Przykład 1
liczba1 = int(input("Wprowadź pierwszą liczbę: "))
liczba2 = int(input("Wprowadź drugą liczbę: "))

try:
    print(liczba1 / liczba2)
except:
    print("Ta operacja nie może być wykonana.")

print("KONIEC.")

#Przykład 2
try:
    x = int(input("Wprowadź liczbę: "))
    y = 1 / x
    print(y)
except ZeroDivisionError:
    print("Nie możesz dzielić przez zero.")
except ValueError:
    print("Musisz wpisać wartość całkowitą.")
except:
    print("jakiś bład się pojawił")

print("KONIEC.")
```

- gałęzie except są przeszukiwane w tej samej kolejności, w jakiej pojawiają się w kodzie;
- nie wolno używać więcej niż jednej gałęzi except o danej nazwie wyjątku;
- liczba różnych gałęzi except jest dowolna - jedynym warunkiem jest to, że jeśli używasz try, musisz umieścić po nim co najmniej jeden except (nazwany lub nie);
- słowo kluczowe except nie może zostać użyte bez poprzedzającego try;
- jeśli zostanie wykonana którakolwiek z gałęzi except, inne gałęzie nie zostaną przejrzane;
- jeśli żadna z określonych gałęzi except nie pasuje do zgłoszonego wyjątku, wyjątek ten pozostanie nieobsłużony
- jeśli istnieje gałąź except bez nazwy (bez nazwy wyjątku), to musi ona być określona jako ostatnia.

2. Wyjątki

Język Python 3 definiuje kilkadziesiąt wbudowanych wyjątków. Wbudowane wyjątki, a wszystkie one tworzą hierarchię w kształcie drzewa.



`ZeroDivisionError` to specjalny przypadek bardziej ogólnej klasy wyjątku zwanej ArithmeticError;

`ArithmeticError` to specjalny przypadek bardziej ogólnej klasy wyjątku zwanej Exception;

`Exception` to specjalny przypadek bardziej ogólnej klasy wyjątku zwanej BaseException;

niewłaściwe użycie

```
try:
    y = 1 / 0
except ArithmeticError:
    print("Problem arytmetyczny!")
except ZeroDivisionError:
    print("Dzielenie przez Zero!")

print("KONIEC.")
```

właściwe użycie

```
try:
    y = 1 / 0
except ZeroDivisionError:
    print("Dzielenie przez Zero!")
except ArithmeticError:
    print("Problem Arytmetyczny!")

print("KONIEC.")
```

Instrukcja **raise** zgłasza określony wyjątek o nazwie **exc**, tak jakby został on zgłoszony w normalny (naturalny) sposób:

```
raise exc
```

Instrukcja ta umożliwia symulację zgłaszania rzeczywistych wyjątków (np. w celu przetestowania ich obsługi)

```
def badFun(n):
    raise ZeroDivisionError

try:
    badFun(0)
except ArithmeticError:
    print("Blad?")

print("KONIEC działania")
```

Instrukcja **raise** może być również użyta bez nazwy wyjątku (wyłącznie wewnątrz gałęzi except)

```
def badFun(n):
   try:
        return n / 0
    except:
        print("Mamy bład")
        raise

try:
    badFun(0)
except ArithmeticError:
    print("Bład Arytmetyczny!")

print("KONIEC działania")
```

Instrukcja **assert**.

- Analizuje wyrażenie;
- jeśli wyrażenie zwraca wartość True, lub niezerową wartość liczbową, lub niepusty łańcuch znaków, lub jakąkolwiek wartość inną niż None, nie zrobi niczego więcej;
- w przeciwnym razie, automatycznie i natychmiast zgłosi wyjątek o nazwie **AssertionError** (w tym przypadku mówimy, że instrukcja assert się nie powiodła)

```
x = float(input("Wprowadz liczbe: "))
assert x != 0.0
x = 1/x
print(x)
```

3. Wybrane wbudowane wyjątki

ArithmeticError

Lokalizacja: BaseException ← Exception ← ArithmeticError

Wyjątek, w którego skład wchodzą wszystkie wyjątki spowodowane przez arytmetyczne operacje takie jak dzielenie przez zero lub inne….

AssertionError

Lokalizacja: BaseException ← Exception ← AssertionError

IndexError

Lokalizacja: BaseException ← Exception ← LookupError ← IndexError

Wyjątek wywołany, kiedy program próbuje uzyskać dostęp do nieistniejącego elementu sekwencji (np. elementu listy)

```
lst = [1, 2, 3, 4, 5]
i = 0
ok = True

while ok:
    try:
        print(lst[i])
        i += 1
    except IndexError:
        ok = False

print('koniec')
```

KeyboardInterrupt

Lokalizacja: BaseException ← KeyboardInterrupt

Wyjątek wywołany kiedy użytkownik korzysta ze skrótu klawiaturowego przeznaczonego do zakończenia wykonywania programu (Ctrl-C w przypadku większości systemów operacyjnych); jeśli obsługa tego wyjątku nie prowadzi do zakończenia działania programu, działanie programu jest kontynuowane.

LookupError

Lokalizacja: BaseException ← Exception ← LookupError

Abstrakcyjny wyjątek, do którego zaliczają się wszystkie wyjątki spowodowane przez błędy wynikające z nieprawidłowych odniesień do innych kolekcji (list, słowników, krotek, itp.)

MemoryError

Lokalizacja: BaseException ← Exception ← MemoryError

Wyjątek wywołany kiedy operacja nie może zostać ukończona powodu braku wolnej pamięci

OverflowError

Lokalizacja: BaseException ← Exception ← ArithmeticError ← OverflowError

Wyjątek wywołany kiedy wynikiem operacji jest liczba, która jest za duża.

```
try:
      liczba = 1E250 ** 2
except OverflowError:
    print('Liczba jest za duza.')
```

ImportError

Lokalizacja: BaseException ← Exception ← StandardError ← ImportError

Wyjątek wywołany przez nieudaną operację importowania

```
try:
    import math
    import time
    import randomize
except:
    print('błąd importu modułu' )
```

KeyError

Lokalizacja: BaseException ← Exception ← LookupError ← KeyError

Wyjątek wywołany, kiedy próbujesz uzyskać dostęp do nieistniejącego elementu kolekcji (np. elementu słownika)

```
dct = { 'a' : 'b', 'b' : 'c', 'c' : 'd' }
ch = 'a'
try:
    while True:
        ch = dict[ch]
        print(ch)
except KeyError:
    print('Brak klucza:', ch)
```

https://docs.python.org/3.6/library/exceptions.html