

Obsługa błędów (wyjątki) - podstawy

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 krokiem 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 błąd się pojawił")

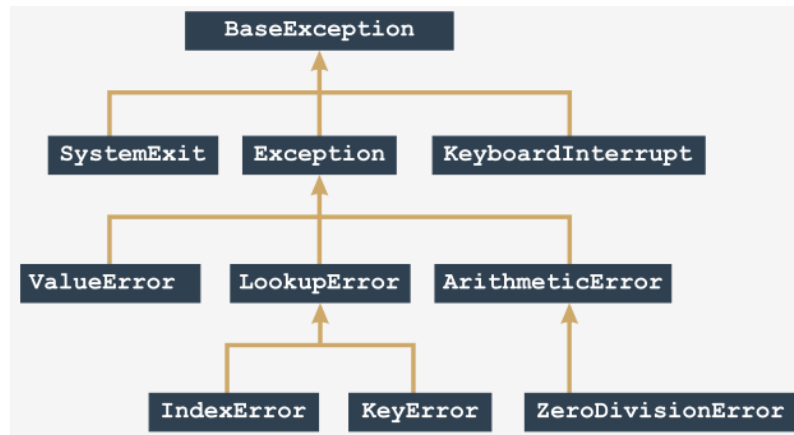
print("KONIEC.")
```

- 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 przejrane;
- 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.

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



`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.")
```

3. Raise i Assert - podstawy

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("Bład")  
  
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 liczbę: "))  
assert x != 0.0  
x = 1/x  
print(x)
```