```python
def divide(a,b):
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a // b

try:
    print(divide(10, 0))
except ValueError:
    print("Error: Division by zero.")
```

Wyjątki w pythonie

M@я3k Pų𝔻€£kØ

Programowanie w Pythonie

Spis treści

Wyjątek

- Wyjątek (Exception) to zdarzenie, które przerywa normalny przepływ programu, gdy pojawi się sytuacja nieprzewidziana w normalnym toku wykonywania kodu.
- Python natychmiast przerywa program i zgłasza wyjątek
  - Odwołanie do nieistniejącej zmiennej
  - Dzielenie przez zero

Przykład wyjątku

```
'''Wyjątki w Pythonie
'''
a = 0
x = 10 / a
print (x)
```

```
'''
Wyjątki w Pythonie
'''


x = 10 / 0
print ("Błąd dzielenia! Nie da się zrobić")
```

```
Traceback (most recent call last):
  File "/home/main.py", line 5, in <module>
    x = 10 / 0
        ~~~^~~
ZeroDivisionError: division by zero
```

Najczęstsze wyjątki w pythonie

ZeroDivisionError	Dzielenie przez zero
FileNotFoundError	Próba otwarcia nieistniejącego pliku
TypeError	Operacja wykonywana na niewłaściwym typie danych (np. dodawanie tekstu do liczby).
ValueError	Funkcja otrzymuje argument o odpowiednim typie, ale niewłaściwej wartości
IndexError	Próba dostępu do indeksu poza zakresem listy
KeyError	Próba użycia nieistniejącego klucza w słowniku

Hierarchia wyjątków

- Python organizuje wyjątki w hierarchię. Oto schemat:

```
BaseException
 ├── SystemExit          # Wyjście z programu (exit())
 ├── KeyboardInterrupt   # Ctrl+C
 └── Exception           # Typowe wyjątki
      ├── ValueError     # Błędna wartość
      ├── TypeError      # Błędny typ
      ├── IndexError     # Indeks poza zakresem
      ├── KeyError       # Brakujący klucz w słowniku
      ├── FileNotFoundError  # Brak pliku
      ├── ZeroDivisionError  # Dzielenie przez zero
      └── ...            # I inne
```

Instrukcja **try .. except**

- Instrukcja **try .. except** umożliwia zdefiniowanie reakcji na pojawienie się wyjątku.
- Instrukcje, które mogą wywołać pojawienie się błędu są wpisywanie pod **try**.
- Gdy nie wystąpi żaden wyjątek, część **except** jest pomijana i zostaje zakończone wykonywanie instrukcji **try**.
- Jeśli wystąpi wyjątek reszta kodu pod **try** jest pomijana. Wykonywana jest część **except**, a potem program wraca do dalszej pracy.

```
try:
    x = 10 / 0
    print (x)
except:
    print ("Błąd dzielenia! Nie da się zrobić")
print ("Dalsza praca")
```

- Jeśli wystąpi wyjątek nie pasujący do wyjątku nazwanego w **except**, przekazuje się go do zewnętrznych instrukcji **try**; jeśli nie zostanie znaleziona obsługa, jest to nieobsłużony wyjątek i program zatrzymuje się z komunikatem błędu.

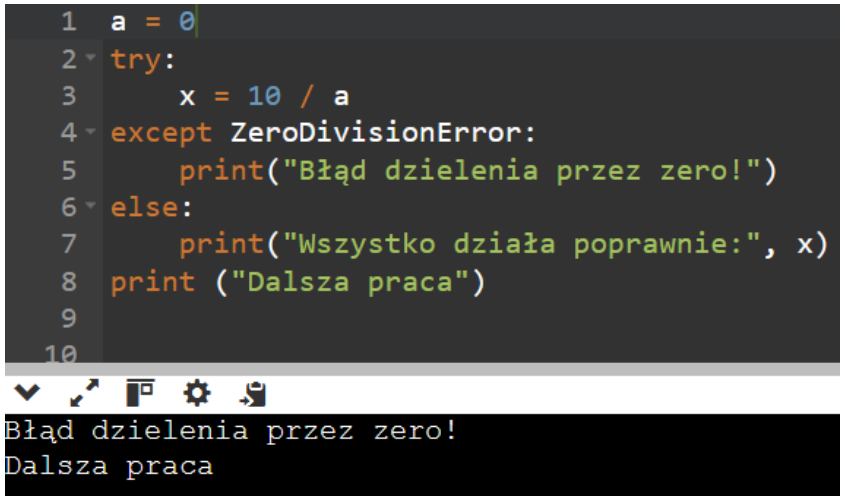
```
try:
    x = 10 / 0
    print (x)
except:
    print ("Błąd dzielenia! Nie da się zrobić")
print ("Dalsza praca")
```

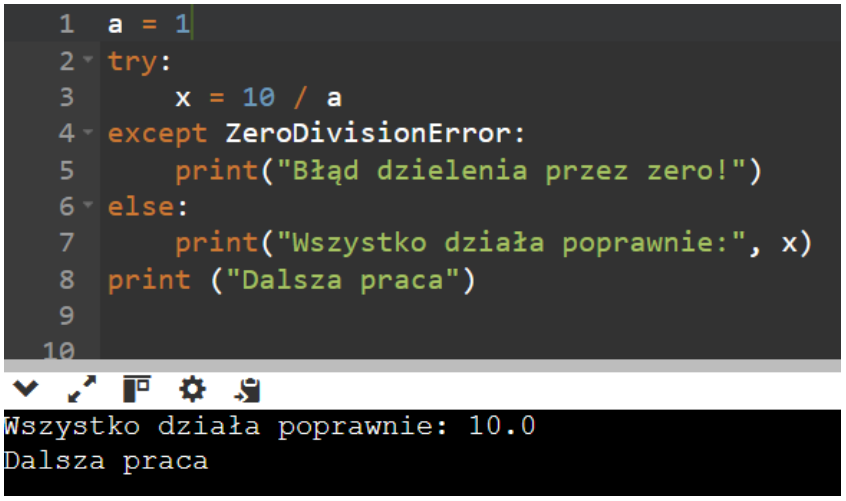
```
Błąd dzielenia! Nie da się zrobić
Dalsza praca
```

Instrukcja **else**

- Blok **else** wykona się tylko wtedy, gdy w **try** nie wystąpi żaden wyjątek:
- Jest to przydatne do odseparowania kodu, który może wywołać wyjątek (w **try**), od kodu, który powinien działać tylko po pomyślnym zakończeniu operacji (w **else**).

```
a = 0
try:
        x = 10 / a
except ZeroDivisionError:
    print("Błąd dzielenia przez zero!")
else:
    print("Wszystko działa poprawnie:", x)
print ("Dalsza praca")
```

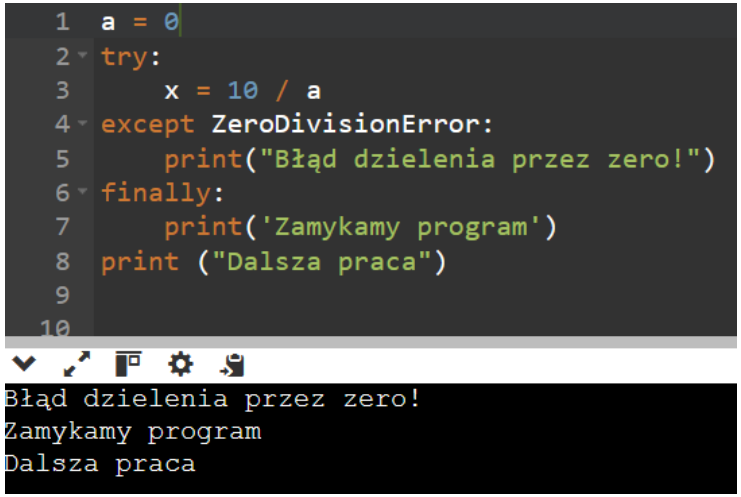


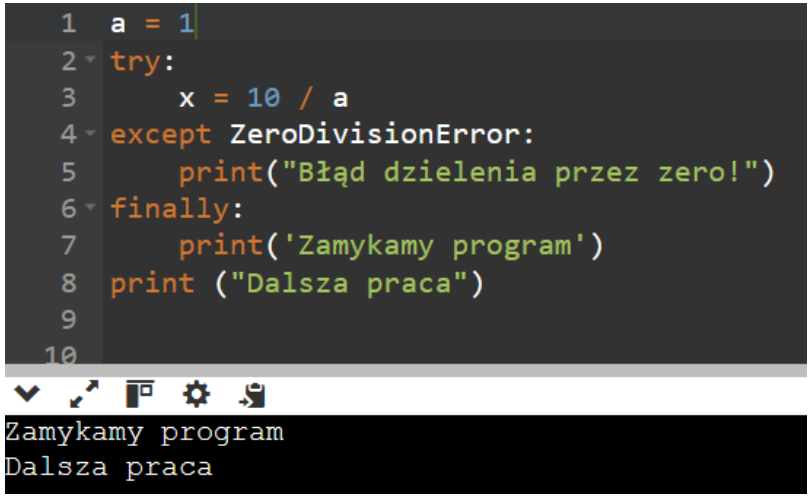


Instrukcja **finally**

- Blok finally w obsłudze wyjątków służy do wykonania określonego kodu niezależnie od tego, czy wyjątek został zgłoszony, czy nie.
- Kluczowe miejsce do umieszczania kodu porządkującego, takiego jak zamykanie plików, zwalnianie zasobów sieciowych czy bazodanowych

```
a = 0
try:
        x = 10 / a
except ZeroDivisionError:
    print("Błąd dzielenia przez zero!")
finally:
    print("zamykamy program")
print ("Dalsza praca")
```





Obsługa wielu wyjątków w jednym bloku

Możliwa jest obsługa obsłużyć kilka typów błędów w jednym **except**:

```
try:     wynik = 10 / int("0")
except (ZeroDivisionError, ValueError) as e:
    print(f"Wystąpił błąd: {e}")
```

```
Wystąpił błąd: division by zero
```

```
try:     wynik = 10 / int("a")
except (ZeroDivisionError, ValueError) as e:
    print(f"Wystąpił błąd: {e}")
```

```
Wystąpił błąd: invalid literal for int() with base 10: 'a'
```

Rzutowanie wyjątków

- Rzutowanie (zgłaszanie) wyjątków w Pythonie odbywa się za pomocą instrukcji **raise**.
- Pozwala na wywołanie konkretnego typu wyjątku (np. ***ValueError***, ***TypeError***) wraz z opcjonalnym komunikatem opisującym problem.
- Dzięki temu programista może kontrolować przepływ programu w sytuacjach awaryjnych

Rzutowanie wyjątków

```
b = 0
if b == 0:
    raise ZeroDivisionError("Dzielenie przez zero zakazane")
try:
    print(b)
except ZeroDivisionError as e:
    print("Błąd:", e)
```

Tworzenie własnych wyjątków

- Własne wyjątki w Pythonie definiuje się, tworząc nową klasę dziedziczącą po wbudowanej klasie Exception (lub jej podklasach).
- Pozwala to na precyzyjne przechwytywanie błędów domenowych w aplikacjach.
- Najprostsza definicja używa słowa kluczowego pass, a nazwa powinna kończyć się na „Error”

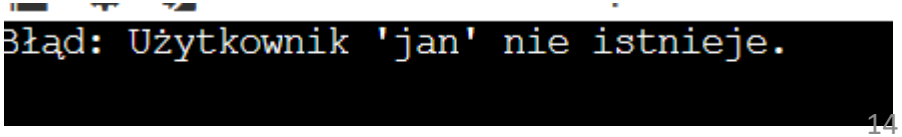
Tworzenie własnych wyjątków

```
# 1. Definiowanie własnego wyjątku
class BazaDanychError(Exception):
    """Wyjątek zgłaszany, gdy wystąpi błąd w bazie danych."""
    pass

class UzytkownikNieistniejeError(BazaDanychError):
    """Wyjątek zgłaszany, gdy użytkownik nie istnieje."""
    def __init__(self, login):
        self.login = login
        super().__init__(f"Użytkownik '{login}' nie istnieje.")

# 2. Użycie wyjątku
def pobierz_uzytkownika(login):
    if login != "admin":
        raise UzytkownikNieistniejeError(login) # Rzucenie wyjątku
    return "Dane admina"

# 3. Obsługa wyjątku
try:
    pobierz_uzytkownika("jan")
except UzytkownikNieistniejeError as e:
    print(f"Błąd: {e}")
```



- # Najczęstsze błędy przy obsłudze wyjątków

1. Używanie **except:** bez typu

Taka konstrukcja wyłapuje wszystkie wyjątki, włącznie z błędami systemowymi.

Dozwolone to jest tylko przy debugowaniu lub logowaniu, nigdy w gotowym kodzie.

2. Ignorowanie błędów

except: pass to najgorsze rozwiązanie. Program działa „pozornie” dobrze, ale ukrywana jest przyczyna błędu.

3. Zbyt rozległe **try-except**

Otoczenie całego kodu jednym **try** powoduje, że nie znana jest **przyczyna** błędu. Lepiej rozdzielić je na ryzykowne fragmenty, by wiedzieć, gdzie naprawdę wystąpił problem.

Dobre praktyki obsługi błędów

- Przechwytuj tylko te błędy, które rozumiesz. Jeśli nie wiesz, jak go poprawić – nie łap go.
- Loguj wyjątki. Używaj modułu logging, aby zapisywać błędy zamiast je wyświetlać użytkownikowi.
- Nie ukrywaj błędów – kontroluj je. Lepiej, by program zakończył się z komunikatem, niż działał niepoprawnie.
- Twórz własne wyjątki dla logiki biznesowej. Zwiększa to czytelność i utrzymanie kodu.

Przykład programu

```
a = 0
try:
    x = 10 / a
except ZeroDivisionError:
    print ("Błąd dzielenia przez zero")
else:
    print("Wszystko działa poprawnie:", x)
finally:
    print("zamykamy program")
print ("Dalsza praca")
```

Ćwiczenie 1

1. Napisz program, który: pobiera od użytkownika dwie liczby, wykonuje dzielenie, obsługuje wyjątki:
   - dzielenie przez zero,
   - wpisanie nie-liczby.
2. Napisz funkcję wczytaj_liczbe(), która:
   - prosi użytkownika o podanie liczby całkowitej,
   - jeśli użytkownik poda błędne dane, prosi ponownie (użyj pętli i try-except).
3. Utwórz listę 5 elementów. Następnie:
   - poproś użytkownika o indeks,
   - wyświetl element z listy,
   - obsłuż wyjątek wyjścia poza zakres listy.
4. Zdefiniuj własny wyjątek **ZaMalyWiekError**. Następnie:
   - napisz funkcję, która przyjmuje wiek użytkownika,
   - jeśli wiek < 18, zgłasza wyjątek,
   - w przeciwnym razie wypisuje „Dostęp przyznany”.

Zadanie 1

- Stwórz prosty kalkulator.
- Ma obsługiwać 4 podstawowe działania wczytywane jako znak (+, -, *, /).
- Wczytuje 2 liczby i wykonuje na nich działanie.
- Obsługuje błędy:
  - Dzielenie przez zero
  - Nieznana operacja arytmetyczna
  - Błędne dane wejściowe (nie liczba)

Zadanie 2

- Napisz książkę teleadresową, która pobiera od użytkownika:
  - imię (nie może być puste),
  - wiek (liczba całkowita 0–120),
  - email (musi zawierać @),
- Dla każdego błędu zgłasza inny wyjątek (własne klasy!),
  - zbiera wszystkie błędy i wyświetla je na końcu (nie przerywa po pierwszym błędzie)..

Powtórzenie

- Co to jest wyjątek?
- Dlaczego stosujemy wyjątki w programie?
- Omów hierarchię wyjątków.
- Jakie zabezpieczenie daje instrukcja **try** .. **Except**?
- Instrukcja **try** .. **except**
- Po co stosujemy instrukcję **else**?
- Jakie korzyści zapewnia instrukcja **finally**?
- Jak obsłużyć wiele wyjątków w jednym bloku?
- Jaki cel ma rzutowanie wyjątków?
- Omów proces tworzenie własnych wyjątków.