Wyjątki

- służą do informowania o **niepowodzeniu** pewnego elementu programu,
- dawniej używano tzw. wartości osobliwej (takiej, która czymś się wyróżniała) do informowania, że funkcja się nie powiodła,
- w Pythonie można by zwracać dwie wartości (jedną właściwą, drugą z kodem błędu),
  - ale to bardzo **ZŁY** pomysł, gwarancja zaciemnienia kodu,
- więc już dawno (w pewnej formie przed 1970 rokiem) wymyślono wyjątki,

Wyjątki

- w skrócie, w momencie wystąpienia wyjątku, wykonywanie kodu jest przerywane i następuje skok do najbliższego miejsca obsługi wyjątku
  - w Pythonie jest to najbliższe pasujące except:
  - najbliższe oznacza najbliżej położone w kodzie aktualnej funkcji/metody,
  - jeśli w aktualnie wykonywanej funkcji nie ma except: to przeszukiwana jest funkcja ją wywołująca, zaczynając od momentu wywołania aktualnej funkcji,
  - jeżeli tam też nie ma to są przeszukiwane kolejne wywołania w górę, aż do najwyższego poziomu
- jeżeli nic (pasujące except:) nie zostanie znalezione, to komunikat jest wyświetlany (w uproszczeniu, bo to zależy jeszcze w jaki sposób uruchomiliśmy interpreter Pythona).

Wyjatki - przykłady

```
def dzielenie(x, y=0):
    print(x/y)

dzielenie(5, 5)
dzielenie(5, 0)
```

```
1.0
Traceback (most recent call last):
  File "dzielenie.py", line 5, in <module> dzielenie
    (5, 0)
  File "dzielenie.py", line 2, in dzielenie print(x/
    y)
ZeroDivisionError: division by zero
```

Wyjatki - przykłady

```
def dzielenie(x, y=0):
    return x/y
def wypisz_dzielenie(x, y):
    print(dzielenie(x))

wypisz_dzielenie(5, 5)
```

```
Traceback (most recent call last):
  File "dzielenie.py", line 6, in <module>
    wypisz_dzielenie(5, 5)
  File "dzielenie.py", line 4, in wypisz_dzielenie
    print(dzielenie(x))
  File "dzielenie.py", line 2, in dzielenie
    return x/y
ZeroDivisionError: division by zero
```

Wyjatki - przykłady

Coś złapmy w końcu:

```
def dzielenie(x, y=0):
    try:
        return x/y
    except TypeError:
        print ("Zly typ")
def wypisz_dzielenie(x, y):
    try:
        print(dzielenie(x))
    except ZeroDivisionError:
        print ("Dzieli przez 0 w wypisz_dzielenie")

try:
    wypisz_dzielenie(5, 5)
except ZeroDivisionError:
    print ("Dzieli przez 0 w glownym kodzie")
```

Jaki wynik?

Wyjatki - przykłady

Coś złapmy w końcu: Jaki wynik?

```
Dzieli przez 0 w wypisz_dzielenie
```

Wyjatki - przykłady

A teraz:

```
def dzielenie(x, y=0):
    try:
        return x/y
    except TypeError:
        print ("Zly typ")
def wypisz_dzielenie(x, y):
    try:
        print(dzielenie(x))
    except ZeroDivisionError:
        print ("Dzieli przez 0 w wypisz_dzielenie")
        raise
try:
    wypisz_dzielenie(5, 5)
except ZeroDivisionError:
    print ("Dzieli przez 0 w glownym kodzie")
```

Wyjatki - przykłady

A teraz jaki wynik?

```
Dzieli przez 0 w wypisz_dzielenie
Dzieli przez 0 w glownym kodzie
```

Instrukcja raise powoduje rzucenie wyjątku, a kiedy występuje w bloku po except: to "przerzucenie" wyjątku dalej.

Wyjatki - przykłady

Jeszcze inny wynik:

```
def dzielenie(x, y=0):
    try:
        return x/y
    except TypeError:
        print ("Zly typ")


def wypisz_dzielenie(x, y):
    try:
        print(dzielenie(x))
    except ZeroDivisionError:
        print ("Dzieli przez 0 w wypisz_dzielenie")

try:
    wypisz_dzielenie("5", 5)
except ZeroDivisionError:
    print ("Dzieli przez 0 w glownym kodzie")
```

Wyjatki - przykłady

Jeszcze inny wynik:

```
Zly typ
None
```

Po wypisaniu komunikatu w funkcji dzielenie wykonywanie kodu jest kontynuowane a więc funkcja dzielenie zwraca None. Nie jest to najlepszy sposób obsługi wyjątków.

Wyjątki - przykłady - finally

```
def dzielenie(x, y=0):
    try:
        return x/y
    except TypeError:
        print ("Zly typ")


def wypisz_dzielenie(x, y):
    try:
        print(dzielenie(x, y))
    except ZeroDivisionError:
        print ("Dzieli przez 0 w wypisz_dzielenie")
```

Wyjątki - przykłady - finally

```
try:
    wypisz_dzielenie(5, 5)
except ZeroDivisionError:
    print ("Dzieli przez 0 w glownym kodzie")
finally:
    print("Mi bez roznicy, jak poszlo")

try:
    wypisz_dzielenie(5, 0)
except ZeroDivisionError:
    print ("Dzieli przez 0 w glownym kodzie")
finally:
    print("Mi bez roznicy, jak poszlo")
```

Wyjatki - przykłady

```
1.0
Mi bez roznicy, jak poszlo
Dzieli przez 0 w wypisz_dzielenie
Mi bez roznicy, jak poszlo
```

Kod z bloku finally wykona się zawsze, niezależnie od tego, czy wystąpi wyjątek, czy nie.

Wyjątki - problemy

- Wyjątków trzeba się "spodziewać", a ...
- ... Nikt nie spodziewa się hiszpańskiej inkwizycji.

Wyjątki - problemy

- Wyjątków trzeba się "spodziewać", a ...
- ... Nikt nie spodziewa się hiszpańskiej inkwizycji.

Wyjątki - problemy

- Dużo programistów używa wyjątków do "naprawiania" problemów.
- często ignorowane jest informowanie użytkownika, a można (a w zasadzie trzeba) to zrobić na 2 sposoby:
  - poprzez wyświetlenie użytkownikowi informacji w bardziej czytelnej formie niż komunikat wyjątku,
  - zapisanie informacji w pewnym miejscu (np. do logu programu).

Wyjątki - własne

- Wyjątki są normalnymi obiektami klas,
  - większość dziedziczy po klasie Exception (SystemExit i KeyboardInterrupt po BaseException),
- stąd prosty wniosek, że można zrobić własną klasę i rzucać jej obiekty

Wyjątki - własne - przykład

```
class NiedobreNieparzyste(Exception):
    def __init__(self, komunikat, lista_zlych):
        super().__init__("Nieparzyste: " + komunikat
            )
        self.lista_zlych = lista_zlych
    def wypisz_zle(self):
        for zly in self.lista_zlych:
            print("Zly: " + str(zly))
```

cdn.

Wyjątki - własne - przykład

```
class SumatorParzystych:
    def __init__(self, lista):
        self.lista = lista
    def sumuj(self):
        suma = 0
        lista_zlych = []
        for liczba in self.lista:
            if liczba % 2:
                lista_zlych.append(liczba)
            else:
                suma += liczba
        if lista_zlych:
            raise NiedobreNieparzyste("Blad w sumuj"
                , lista_zlych)
        return suma
```

cdn.

Wyjątki - własne - przykład

```
x = SumatorParzystych([2, 3, 4, 5, 6, 8])
print(x.sumuj())
```

Wyjątki - własne - przykład

```
Traceback (most recent call last):
  File "SumatorParzystych.py", line 24, in <module>
    print(x.sumuj())
  File "SumatorParzystych.py", line 20, in sumuj
    raise NiedobreNieparzyste("Blad w sumuj",
        lista_zlych)
NiedobreNieparzyste: Nieparzyste: Blad w sumuj
```

Wyjątki - własne - przykład

```
try:
    x = SumatorParzystych([2, 3, 4, 5, 6, 8])
    print(x.sumuj())
except NiedobreNieparzyste as e:
    print("Tu mozna wolac funkcje z klasy wyjatku")
    e.wypisz_zle()
```

Zwracam uwagę na użycie słowa kluczowego "as" i nazwę obiektu wyjątku "e" (popularna konwencja).

Wyjątki - własne - przykład

```
Tu mozna wolac funkcje z klasy wyjatku
Zly: 3
Zly: 5
```