Wyjątki

Po co są wyjątki?

Wyjątki stosowane są wtedy, gdy pewna metoda albo funkcja nie może wykonać się poprawnie. Na przykład kiedy dane wprowadzone od użytkownika są nieprawidłowe albo jest problem z dostępem do zasobu (np. pliku). Wyjątek jest wtedy podnoszony, żeby powiadomić program, że funkcja nie jest w stanie sobie poradzić z napotkanym problemem. Program może wtedy albo próbować poradzić sobie z wyjątkiem, albo przekazać go wyżej, dochodząc ostatecznie do warstwy systemu.

Wyjątki nie powinny być stosowane przy normalnym użytkowaniu projektowanej aplikacji. Wystąpienie wyjątka oznacza błąd programu!

Podnoszenie wyjątków

```
1  def bar():
2      raise NameError
3
4  if __name__ == '__main__':
5      bar()
```

Tworzenie własnych wyjątków

```
1  class CtgDoesNotExistsError(ArithmeticError):
2      strerror = 'Cotangens dla kąta 90 nie istnieje'
3      errno = 10
4
5  def ctg(deg):
6
7      if deg == 90:
8          raise CtgDoesNotExistsError
9
10     return "wylicz cotangens kąta"
```

Przechwytywanie wyjątków

Python spróbuje najpierw wykonać to co będzie zaprogramowane w ramach słowa kluczowego `try`. Jeżeli w trakcie wykonywania tego fragmentu kodu interpreter napotka na wyjątek, kod zostanie przerwany i zostanie wykonany fragment zaprogramowany w ramach słowa kluczowego `except`, odnoszący się do danego wyjątku, albo do wszystkich pozostałych `else`. Na koniec, niezależnie od tego czy wyjątek wystąpił czy nie, zostanie wykonany fragment zawarty w `finally`.

- `try`
- `except`
- `else`

- `finally`

```
def bar():
    raise NameError


def foo():
    try:
        bar()
    except NameError:
        print('Błąd nazwy zlapany')
    except SyntaxError:
        print('Błąd składni zlapany')


if __name__ == '__main__':
    foo()
```

Najpopularniejsze wyjątki

Nazwa wyjątku Opis

AttributeError Raised when an attribute reference (see Attribute references) or assignment fails. (When an object does not support attribute references or attribute assignments at all, TypeError is raised.) ImportError Raised when an import statement fails to find the module definition or when a from ... import fails to find a name that is to be imported. IndexError Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not an integer, TypeError is raised.) KeyError Raised when a mapping (dictionary) key is not found in the set of existing keys. KeyboardInterrupt Raised when the user hits the interrupt key (normally Control-C or Delete). During execution, a check for interrupts is made regularly. The exception inherits from BaseException so as to not be accidentally caught by code that catches Exception and thus prevent the interpreter from exiting. NameError Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found. NotImplementedError This exception is derived from RuntimeError. In user defined base classes, abstract methods should raise this exception when they require derived classes to override the method. RuntimeError Raised when an error is detected that doesn't fall in any of the other categories. The associated value is a string indicating what precisely went wrong. SyntaxError Raised when the parser encounters a syntax error. This may occur in an import statement, in a call to the built-in functions exec() or eval(), or when reading the initial script or standard input (also interactively). IndentationError Base class for syntax errors related to incorrect indentation. This is a subclass of SyntaxError. TypeError Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

Hierarchia wyjątków

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
```

```
+-- Exception
     +-- StopIteration
     +-- StopAsyncIteration
     +-- ArithmeticError
     |    +-- FloatingPointError
     |    +-- OverflowError
     |    +-- ZeroDivisionError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- EOFError
     +-- ImportError
     +-- LookupError
     |    +-- IndexError
     |    +-- KeyError
     +-- MemoryError
     +-- NameError
     |    +-- UnboundLocalError
     +-- OSError
     |    +-- BlockingIOError
     |    +-- ChildProcessError
     |    +-- ConnectionError
     |    |    +-- BrokenPipeError
     |    |    +-- ConnectionAbortedError
     |    |    +-- ConnectionRefusedError
     |    |    +-- ConnectionResetError
     |    +-- FileExistsError
     |    +-- FileNotFoundError
     |    +-- InterruptedError
     |    +-- IsADirectoryError
     |    +-- NotADirectoryError
     |    +-- PermissionError
     |    +-- ProcessLookupError
     |    +-- TimeoutError
     +-- ReferenceError
     +-- RuntimeError
     |    +-- NotImplementedError
     |    +-- RecursionError
     +-- SyntaxError
     |    +-- IndentationError
     |         +-- TabError
     +-- SystemError
     +-- TypeError
     +-- ValueError
     |    +-- UnicodeError
     |         +-- UnicodeDecodeError
     |         +-- UnicodeEncodeError
     |         +-- UnicodeTranslateError
     +-- Warning
          +-- DeprecationWarning
          +-- PendingDeprecationWarning
          +-- RuntimeWarning
          +-- SyntaxWarning
```

```
            +-- UserWarning
            +-- FutureWarning
            +-- ImportWarning
            +-- UnicodeWarning
            +-- BytesWarning
            +-- ResourceWarning
```

Przykład wyjątków przy czytaniu plików

```
import logging
log = logging.getLogger(__name__)

log.info('Rozpoczynam program')

try:

    log.debug('Próbuję odczytać plik')

    with open(FILENAME, 'w') as file:
        content = file.read()
        print(content)

    log.debug('Plik odczytany i zamknięty')

except PermissionError as e:
    log.error(e)
    print(e.strerror)

except OSError as e:
    log.error(e)
    print(e.strerror)

except Exception as e:
    log.error(e)
    print(e.strerror)

else:
    log.info('Operacje na pliku zakończyły się powodzeniem')

finally:
    log.debug('Zakończenie pracy nad plikiem')

log.info('Kończymy program')
```

Korzystanie z `warnings`

```
import warnings

def sumuj(a, b):
    warnings.warn('Nie stosuj tego', PendingDeprecationWarning)
    return a + b


sumuj(1, 2)
```