Python exceptions

JANUSZ JABŁONOWSKI, MIMUW

Exception handling

- World is far from perfect
- Even our programs
- Expect the unexpected - use exception handling

Raising exceptions

- Mostly they are raised somewhere else
- We can raise them too
  - `raise`
  - `raise <expression>`
  - `raise <expression> from <expression>`
- First form reraises last active exception (or raises RuntimeError if none is active)

- Both last forms calculate the first expression, it should give a subclass or an instance of BaseException (in case of a class an instance is created with parameterless constructor)

- The last form allows for exception chaining: the last expression also has to be an exception which will be the cause for the raised exception

Raising exceptions

```
raise RuntimeError("Division by zero occurred")
```

Exception handling

- Much more important than raising is exception catching

```
try:
     instructions
except [expression [as identifier]]:  # may be many such clauses
     instructions
[else:
      instructions]
[finally:
      instructions]
```

Exception handling

- If no exception within **`try`** is raised then no exception handler is called
- If an exception is raised then the execution of try statements is abandoned, and an exception handler is looked for
- Handlers are examined one by one till the first matching is found
- Its instructions are then executed (and no other handlers are examined)
- Expressions in clauses, when tested, are calculated; their values match the exception object if
  - value’s base classes is the same as that of exception object, or
  - value’s base class is base class of the exception object, or
  - value is a tuple containing a matching object
- If there is an expression-free clause then it matches any exception and must be the last clause

Exception handling

- The **else** clause is executed if no exception was raised and no `return`, `continue`, or `break` statement was executed.
- If there is an exception raised in the else clause it is searched outside of the try statement
- The **finally** clause is always executed
- If there was an exception in exception handling in the **try** statement, then it is suspended on the time of the execution of finally, if **finally** does not execute a **return**, **break** or **continue** statement this exception is then reraised (otherwise discarded)
- If try block ended because of the **return** statement and **finally** did so too, then the return value of a function is that of finally

Exception handling

- If no exception handler is found within the try instruction, then the search for exception continues in usual way - up the invocation stack
- If the evaluation of a clause expression raises exception, then the original exception object is abandoned and the search for a new exception continues in the normal way outside of the try statement

Exception

- It is an object
- Object of a subclass of the Exception class
- Contains a traceback - important to find causes of the problem
- May contain a cause - other exception which handling caused this one
- When the exception handler is being searched details of the exception can be examined by calling `sys.exc_info()`. Its result is a 3-tuple: exception class, the exception instance and a traceback object.

Own exceptions

- It is possible to define them
- They have to be subclasses of Exception
- (see example on the next page)

Own exceptions

```python
class NotNumber(Exception):
    pass

def add1(x):
    if type(x) == int or type(x) == float or type(x) == complex:
        return x + 1
    else:
        raise NotNumber("x must be a number")
```

Built-in exceptions

- There are many exceptions built-in into Python
- Here we list only the most interesting ones
- Exception - all user defined exceptions should inherit it, also all non-system-exiting exceptions inherit from this class.
- ArithmeticError - superclass for e.g.:
  - OverflowError (rather not used, for ints it should be MemoryError, some functions expecting a value from a range may generate it, most parts of floats implementation does not generate it)
  - ZeroDivisionError
  - FloatingPointError (currently is not used)
- AssertionError - generated by the assert command
- (cont...)

Built-in exceptions

- (...cont)
- IndexError - an index of a sequence id out of range
- KeyError - a key was not found in a dictionary
- MemoryError - the program ran out of memory
- NameError - local/global name was not found
- OSError - error connected to op. syst. (like problem with a file), has many subclasses, e.g. FileNotFoundError
- RecursionError - too deep recursion (use sys.set/getrecursionlimit())
- TypeError - an operation got an operand of a wrong type
- ValueError - an operation got an operand of a right type but wrong value

Thank you for your attention!