

Exceptions and file input/output

- try-raise-except-finally
- Exception
- control flow
- file open/read/write
- `sys.stdin`, `sys.stdout`, `sys.stderr`

Exceptions – Error handling and control flow

- **Exceptions** is a widespread technique to handle run-time **errors** / abnormal behaviour (e.g. in Python, Java, C++, JavaScript, C#)
- **Exceptions** can also be used as an **advanced control flow mechanism** (e.g. in Python, Java, JavaScript)
 - *Problem: How to perform a "break" in a recursive function ?*

Built-in exceptions (class hierarchy)

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |   +-- FloatingPointError
        |   +-- OverflowError
        |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        |   +-- ModuleNotFoundError
    +-- LookupError
        |   +-- IndexError
        |   +-- KeyError
    +-- MemoryError
    +-- NameError
        |   +-- UnboundLocalError
    +-- TypeError
    +-- ValueError
        |   +-- UnicodeError
        |       +-- UnicodeDecodeError
        |       +-- UnicodeEncodeError
        |       +-- UnicodeTranslateError
```

```
+-- OSError
    |   +-- BlockingIOError
    |   +-- ChildProcessError
    |   +-- ConnectionError
    |       |   +-- BrokenPipeError
    |       |   +-- ConnectionAbortedError
    |       |   +-- ConnectionRefusedError
    |       |   +-- ConnectionResetError
    +-- FileNotFoundError
    +-- FileNotFoundError
    +-- InterruptedError
    +-- IsADirectoryError
    +-- NotADirectoryError
    +-- PermissionError
    +-- ProcessLookupError
    +-- TimeoutError
+-- ReferenceError
+-- RuntimeError
    |   +-- NotImplementedError
    |   +-- RecursionError
+-- SyntaxError
    |   +-- IndentationError
    |   +-- TabError
+-- SystemError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
    +-- ResourceWarning
```

Typical built-in exceptions

and unhandled
behaviour

Python shell

```
> 7 / 0
| ZeroDivisionError: division by zero
> int("42x")
| ValueError: invalid literal for int() with base 10: '42x'
> x = y
| NameError: name 'y' is not defined
> L = list(range(1000000000))
| MemoryError
> 2.5 ** 1000
| OverflowError: (34, 'Result too large')
> t = (3, 4)
> t[0] = 7
| TypeError: 'tuple' object does not support item assignment
> t[3]
| IndexError: tuple index out of range
> t.x
| AttributeError: 'tuple' object has no attribute 'x'
> x = {}
> x['foo']
| KeyError: 'foo'
> def f(x): f(x + 1)
> f(0)
| RecursionError: maximum recursion depth exceeded
> def f(): x = x + 1
> f()
| UnboundLocalError: local variable 'x' referenced before assignment
```

Catching exceptions – Fractions (I)

```
fraction1.py
```

```
while True:  
    numerator = int(input("Numerator = "))  
    denominator = int(input("Denominator = "))  
    result = numerator / denominator  
    print("%s / %s = %s" % (numerator, denominator, result))
```

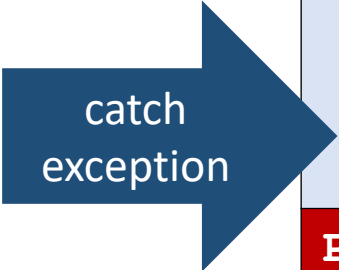
```
Python shell
```

```
| Numerator = 10  
| Denominator = 3  
| 10 / 3 = 3.3333333333333335  
| Numerator = 20  
| Denominator = 0  
| ZeroDivisionError: division by zero
```

Catching exceptions – Fractions (II)

`fraction2.py`

```
while True:
    numerator = int(input("Numerator = "))
    denominator = int(input("Denominator = "))
    try:
        result = numerator / denominator
        print("%s / %s = %s" % (numerator, denominator, result))
    except ZeroDivisionError:
        print("cannot divide by zero")
```



catch
exception

`Python shell`

```
| Numerator = 10
| Denominator = 0
| cannot divide by zero
| Numerator = 20
| Denominator = 3
| 20 / 3 = 6.666666666666667
| Numerator = 42x
| ValueError: invalid literal for int() with base 10: '42x'
```

Catching exceptions – Fractions (III)

`fraction3.py`

```
while True:
    try:
        numerator = int(input("Numerator = "))
        denominator = int(input("Denominator = "))
    except ValueError:
        print("input not a valid integer")
        continue
    try:
        result = numerator / denominator
        print("%s / %s = %s" % (numerator, denominator, result))
    except ZeroDivisionError:
        print("cannot divide by zero")
```

catch
exception

catch
exception

`Python shell`

```
| Numerator = 5
| Denominator = 2x
| input not a valid integer
| Numerator = 5
| Denominator = 2
| 5 / 2 = 2.5
```


Catching exceptions – Fractions (IV)

`fraction4.py`

```
while True:
    try:
        numerator = int(input("Numerator = "))
        denominator = int(input("Denominator = "))
        result = numerator / denominator
        print("%s / %s = %s" % (numerator, denominator, result))
    except ValueError:
        print("input not a valid integer")
    except ZeroDivisionError:
        print("cannot divide by zero")
```

catch
exceptions

`Python shell`

```
| Numerator = 3
| Denominator = 0
| cannot divide by zero
| Numerator = 3x
| input not a valid integer
| Numerator = 4
| Denominator = 2
| 4 / 2 = 2.0
```

Keyboard interrupt (Ctrl-c)

- throws **KeyboardInterrupt** exception

infinite-loop1.py

```
print("starting infinite loop")

x = 0
while True:
    x = x + 1

print("done (x = %s)" % x)
input("type enter to exit")
```

Python shell

```
| starting infinite loop
| Traceback (most recent call last):
|   File "infinite-loop1.py", line 4, in <module>
|     x = x + 1
| KeyboardInterrupt
```

infinite-loop2.py

```
print("starting infinite loop")

try:
    x = 0
    while True:
        x = x + 1
except KeyboardInterrupt:
    pass

print("done (x = %s)" % x)
input("type enter to exit")
```

Python shell

```
| starting infinite loop
| done (x = 23890363) # Ctrl-c
| type enter to exit
```

Keyboard interrupt (Ctrl-c)

- Be aware that you likely would like to leave the Ctrl-c generated **KeyboardInterrupt** exception unhandled, except when stated explicitly

```
read-int1.py
while True:
    try:
        x = int(input("An integer: "))
        break
    except ValueError: # only ValueError
        continue

print("The value is:", x)

Python shell
| An integer:          Ctrl-c
| KeyboardInterrupt
```

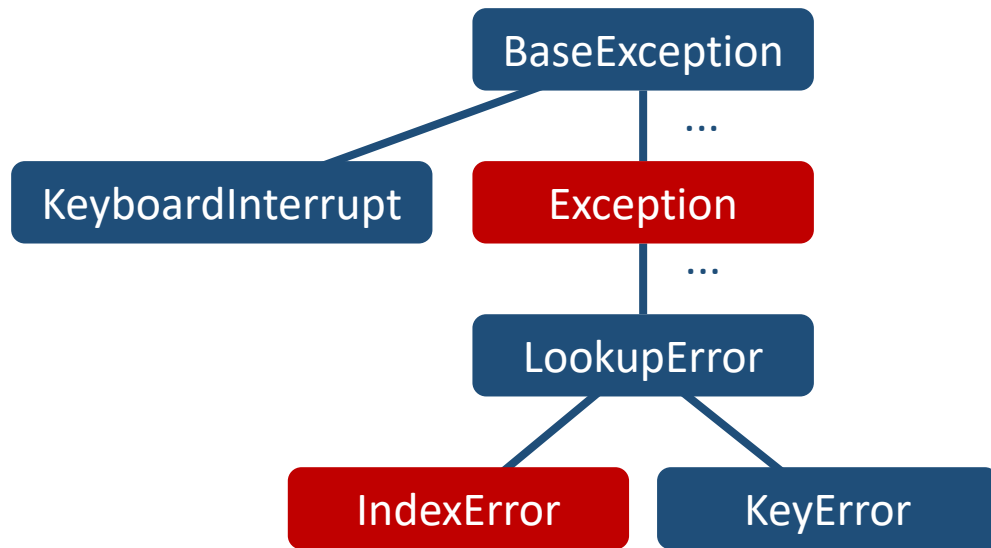
```
read-int2.py
while True:
    try:
        x = int(input("An integer: "))
        break
    except: # all exceptions ← catches KeyboardInterrupt
        continue

print("The value is:", x)

Python shell
| An integer:          Ctrl-c
| An integer:          Ctrl-c
| An integer:
```

- (left) KeyboardInterrupt is unhandled (right) it is handled (intentionally?)

Exception class hierarchy




`except-twice1.py`

```
try:
    L[4]
except IndexError: # must be before Exception
    print("IndexError")
except Exception:
    print("Fall back exception handler")
```

`except-twice2.py`

```
try:
    L[4]
except Exception: # and subclasses of Exception
    print("Fall back exception handler")
except IndexError:
    print("IndexError") # unreachable
```



try statement syntax

try:

code

except ExceptionType1:

code # executed if raised exception instance of
ExceptionType1 (or subclass of ExceptionType1)

except ExceptionType2:

code # executed if exception type matches and none of
the previous except statements matched

...

else:

code # only executed if no exception was raised

finally:

code # always executed independent of exceptions
typically used to clean up (like closing files)

arbitrary number of except cases

except variations

`except:` # catch all exceptions



`except ExceptionType:` # only catch exceptions of class `ExceptionType`
or subclasses of `ExceptionType`

`except (ExceptionType1, ExceptionType2, ..., ExceptionTypek):`
catch any of k classes (and subclasses)

`except ExceptionType as e:`
catch exception and assign exception object to `e`
`e.args` contains arguments to the raised exception

User exceptions

- New exception types are created using **class inheritance** from an existing exception type (possibly defining `__init__`)
- An exception is raised using one of the following (the first being an alias for the second):

`raise ExceptionType`

`raise ExceptionType()`

`raise ExceptionType(args)`

`tree-search.py`

```
class SolutionFound(Exception): # new exception
    pass

def recursive_tree_search(x, t):
    if isinstance(t, tuple):
        for child in t:
            recursive_tree_search(x, child)
    elif x == t:
        raise SolutionFound # found x in t

def tree_search(x, t):
    try:
        recursive_tree_search(x, t)
    except SolutionFound:
        print("found", x)
    else:
        print("search for", x, "unsuccessful")
```

`Python shell`

```
> tree_search(8, ((3,2),5,(7,(4,6))))
| search for 8 unsuccessful
> tree_search(3, ((3,2),5,(7,(4,6))))
| found 3
```

3 ways to read lines from a file

Steps

1. Open file using `open`
2. Read lines from file using
 - a) `filehandler.readline`
 - b) `filehandler.readlines`
 - c) `for line in filehandler:`
3. Close file using `close`

try to open file for reading filename

```
reading-file1.py
f = open('reading-file1.py')
for line in f:
    print(">", line[:-1])
f.close()
```

filehandle

iterate over lines in file

close file when done

```
reading-file2.py
f = open('reading-file2.py')
lines = f.readlines()
for line in lines:
    print(">", line[:-1])
f.close()
```

read all lines into a list of strings

```
reading-file3.py
f = open('reading-file3.py')
line = f.readline()
while line != "":
    print(">", line[:-1])
    line = f.readline()
f.close()
```

read single line (terminated by '\n')

3 ways to write lines to a file

- Opening file:
`open(filename, mode)`
where *mode* is a string, either 'w' for opening a new (or truncating an existing file) and 'a' for appending to an existing file
- Write single string:
`filehandle.write(string)`
Returns the number of characters written
- Write list of strings strings:
`filehandle.writeline(list)`
- Newlines ('\\n') must be written explicitly

```
factorial.py
f = open('output-file.txt', 'w')
f.write('Text 1\\n')
f.writelines(['Text 2\\n', 'Text 3 '])
f.close()

g = open('output-file.txt', 'a')
g.write('Text 4\\n')
g.writelines(['Text 5 ', 'Text 6'])
g.close()

output-file.txt
Text 1
Text 2
Text 3 Text 4
Text 5 Text 6
```

try to open file for writing

write mode

write single string to file

write list of strings to file

append to existing file

Exceptions while dealing with files

- When dealing with files one should be prepared to handle errors / raised exceptions, e.g. `FileNotFoundError`

```
reading-file4.py
```

```
try:
    f = open('reading-file4.py')
except FileNotFoundError:
    print("Could not open file")
else:
    try:
        for line in f:
            print("> ", line[:-1])
    finally:
        f.close()
```

Opening files using `with`

- The Python keyword `with` allows to create a *context manager* for handling files
- *Filehandle will automatically be closed, also when exceptions occur*
- Under the hood: filehandles returned by `open()` support `__enter__()` and `__exit__()` methods

`f` = result of calling `__enter__()`
on result of `open` expression,
which is the file handle

reading-file5.py

```
with open('reading-file5.py') as f:  
    for line in f:  
        print("> ", line[:-1])
```

Does a file exist?

- Module `os.path` contains a method `isfile` to check if a file exists

```
checking-files.py
```

```
import os.path

filename = input("Filename: ")
if os.path.isfile(filename):
    print("file exists")
else:
    print("file does not exists")
```

module `sys`

- Module `sys` contains the three standard file handles
 - `sys.stdin` (used by the `input` function)
 - `sys.stdout` (used by the `print` function)
 - `sys.stderr` (error output from the Python interpreter)

`sys-test.py`

```
import sys
sys.stdout.write("Input an integer: ")
x = int(sys.stdin.readline())
sys.stdout.write("%s square is %s" % (x, x**2))
```

Python shell

```
| Input an integer: 10
| 10 square is 100
```

print(..., file=output file)

sys-print-file.py

```
import sys

def complicated_function(file):
    print("Hello world", file=file) # print to file or STDOUT

while True:
    file_name = input("Output file (empty for STDOUT): ")

    if file_name == "":
        file = sys.stdout
        break
    else:
        try:
            file = open(file_name, "w")
            break
        except Exception:
            pass

complicated_function(file)

if file != sys.stdout:
    file.close()
```

PEP8 on exceptions

- For all try/except clauses, limit the try clause to the absolute **minimum amount of code** necessary.
- The class naming convention applies (**CapWords**)
- Use the **suffix "Error"** on your exception names (if the exception actually is an error)
- A bare **except:** clause will catch **SystemExit** and **KeyboardInterrupt** exceptions, making it harder to interrupt a program with Control-C, and can disguise other problems. If you want to catch all exceptions that signal program errors, use **except Exception:**

sudoku.py

```
class Sudoku:
    def __init__(self, puzzle):
        self.puzzle = puzzle

    def solve(self):
        def find_free():
            for i in range(9):
                for j in range(9):
                    if self.puzzle[i][j] == 0:
                        return (i, j)
            return None

        def unused(i, j):
            i_, j_ = i // 3 * 3, j // 3 * 3
            cells = {(i, k) for k in range(9)}
            cells -= {(k, j) for k in range(9)}
            cells -= {(i, j) for i in range(i_, i_ + 3)
                    for j in range(j_, j_ + 3)}
            return set(range(1, 10)) - {self.puzzle[i][j] for i, j in cells}

        class SolutionFound(Exception):
            pass

        def recursive_solve():
            cell = find_free()
            if not cell:
                raise SolutionFound
            i, j = cell
            for value in unused(i, j):
                self.puzzle[i][j] = value
                recursive_solve()
            self.puzzle[i][j] = 0

        try:
            recursive_solve()
        except SolutionFound:
            pass
```

sudoku.py (continued)

```
    def print(self):
        for i, row in enumerate(self.puzzle):
            cells = [' %s ' % c if c else ' . ' for c in row]
            print(''.join([''.join(cells[j:j+3]) for j in (0,3,6)]))
            if i in (2,5):
                print('-----+-----+-----')

        with open("sudoku.txt") as f:
            A = Sudoku([[int(x) for x in line.strip()] for line in f])

        A.solve()
        A.print()
```

sudoku.txt

```
517600034
289004000
346205090
602000010
038006047
000000000
090000078
703400560
000000000
```

Python shell

```
| 5 1 7 | 6 9 8 | 2 3 4
| 2 8 9 | 1 3 4 | 7 5 6
| 3 4 6 | 2 7 5 | 8 9 1
|-----+-----+-----|
| 6 7 2 | 8 4 9 | 3 1 5
| 1 3 8 | 5 2 6 | 9 4 7
| 9 5 4 | 7 1 3 | 6 8 2
|-----+-----+-----|
| 4 9 5 | 3 6 2 | 1 7 8
| 7 2 3 | 4 8 1 | 5 6 9
| 8 6 1 | 9 5 7 | 4 2 3
```