Flik 6: Undantag

Typer av fel

- Fel - tre typer
  - Logiska fel; "feltänk" - fel i algoritmer etc
  - Syntaxfel; Python-tolken vägrar köra programmet
  - Runtime-fel (exekveringsfel); Under körning - fel indata, indexeringsfel etc

- Runtime-fel
  - Pythontolken avbryter exekveringen och skriver ut en dump (traceback)
  - Varför och var felet uppstod presenteras

- Undantag
  - Hur man "fångar" dessa runtime-fel och hanterar felet på ett kontrollerat sätt istället för att avbryta programexekveringen

- Exempel på runtime-fel

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: int division or modulo by zero

>>> open('finnsej.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or
  directory: 'finnsej.txt'
```

try - except

- Upptäcka och hantera undantag

```
try:
    try_block
except undantagstyp as orsak:
    except_block
```

- Om ett undantag sker i `try_block` går exekveringen direkt över i `except_block` förutsatt att undantaget matchar *`undantagstyp`*
  - `except_block` ska ”hantera” felet
  - Man får inte generera nya exceptions i felhanteringsblocket
- Exempel

```
try:
    fh = open('fil.txt', 'r')
except OSError:
    print("Filen fil.txt kunde inte öppnas")
    ....
```

Klasshierarkin

```
BaseException
    SystemExit
    KeyboardInterrupt
    GeneratorExit
    Exception
      StopIteration
      ArithmeticError
        ZeroDivisionError
      AssertionError
      AttributeError
      EOFError
      OSError
        FileNotFoundError
        PermissionError
      TypeError
      ValueError
      ...
```

try - except - bifoga orsak

- Bifoga orsak

```
try:
    fh = open('fil.txt', 'r')
except OSError as orsak:
    print('open fil.txt:', orsak)
    ....

# T.ex:
# open fil.txt: [Errno 2] No such file or directory: 'fil.txt'
```

- Om inte undantaget hanteras "propagerar" felet ett steg upp.

```
try:
    fh = open('fil.txt', 'r')
    ....
except Exception as orsak:
    print('Något gick fel:', orsak)
    ....
```

- Undvik att ha generella undantag (t.ex. *Exception* eller *except:*)
  - **Fånga bara fel du kan hantera!**

Flera except

- Flera except:

```
def safe_float(obj):
    try:
        retval = float(obj)
    except ValueError:
        retval = 'could not convert non-number to float'
    except TypeError:
        retval = 'object type cannot be converted to float'
    return retval

>>> safe_float([])
'object type cannot be converted to float'
>>> safe_float('xyz')
'could not convert non-number to float'
```

- Flera except - samma block

```
    except (ValueError, TypeError) as e:
        ...
```

Vad sker här?

```
try:
    L = []
    n = 1
    while True:
        n += 1
        L.extend(list(range(n)))
except Exception:
    print("Kraschade")
except KeyboardInterrupt:
    print("Stoppad av användaren")
```

Övning

- **ovn0601/matte2.py** Som i övning 3.1, men lägg till felhantering så att programmet inte kraschar utan bara frågar igen ifall man råkar mata in något som inte är ett heltal. Ifall det inte går att läsa mer från stdin, så ska programmet avslutas (dvs. EOFError ska inte fångas, bara ValueError).

```
$ python3 matte2.py
What is 2+2: vet ej
What is 2+2: 5
Wrong! What is 2+2: 4
Correct!
$
```

Övning

- **ovn0602/gissatal.py** Som i övning 3.3, men lägg till felhantering så att programmet inte kraschar ifall man råkar mata in något som inte är ett heltal. (Gissningar som inte är heltal ska inte räknas in i det totala antalet gissningar som skrivs ut i slutet, se exempel nedan.)

```
$ python3 gissatal.py
Gissa tal (1-1000): 500
För stort! Gissa tal (1-1000): g300
inget heltal, försök igen
Gissa tal (1-1000): 300
För stort
Gissa tal (1-1000): 100
För litet
Gissa tal (1-1000): 213
OK efter 4 gissningar.
```

Övning

- **ovn0603/ordfrekvens2.py** Lägg till felhantering i övning 5.6 så att programmet inte kraschar, utan ger vettiga felmeddelanden till stderr och sedan avslutar med SystemExit, ifall:
  - filen inte går att öppna, eller
  - filen inte är kodad i utf-8.

```
$ python3 ordfrekvens2.py finns.ej
Kan inte öppna filen
$ python3 ordfrekvens2.py /usr/bin/python3
Filen är inte i utf-8
$
```

else

- *try - except - else* - Kör `else_block` om inget fel inträffat

```
try:
    try_block
except ex as e:
    except_block
else:
    else_block
```

- Egentligen inte särskilt användbart...

finally

- *finally* körs alltid, vad som än händer
  - Mycket användbart, t.ex. för ”uppstädning” (stänga fil e.d.)

```
try:
    try_block
except ex as e:
    except_block
finally:
    finally_block
```

Två roller

- Ifall man skriver kod som är tänkt att anropas av någon annan (en ”klient”) så går man in i rollen som ”implementatör”.
- Skilj mellan
  - **Klientkod**
  - **Implementationskod**
- Funktioner och klasser är typisk implementationskod.
  - Även om man anropar sin egen kod är det *olika roller*.
- Det är implementationen som genererar undantagen.
- `try` och `except` används av klienten.
- Ett undantag är ett *meddelande från implementationen till klienten*
  - ”Hör du, jag grejar inte det här.”

Generera undantag: raise

```
def månadsnamn(n):
    L = "X Jan Feb Mar Apr Jun Jul Aug Sep Okt Nov Dec".split()
    if not isinstance(n, int):
        raise TypeError("Numret måste vara ett heltal")
    if 1 <= n <= 12:
        return L[n]
    else:
        raise ValueError("Numret måste vara mellan 1 och 12")
```

```
>>> månadsnamn(3)
'Mar'
>>> månadsnamn(13)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in månadsnamn
ValueError: Numret måste vara mellan 1 och 12
>>>
```

Mer om raise

```
# Exempel 1: avsluta programmet
raise SystemExit

# Exempel 2: avsluta programmet med slutstatus 3
raise SystemExit(3)

# Exempel 3: kunde inte åtgärda felet
try:
    try_block
except ex as e:
    try_to_recover_block
    if failed_to_recover:
        raise              # "Återaktivera" undantaget
```

assert

- *assert* - Kontrollera att uttryck returnerar True - annars *AssertionError*

```
try:
    ...
    assert len(my_list) <= 10, \
        'längden får inte överstiga 10'
    ...
except AssertionError as e:
    print('Fel i test:', e)
```

Egna undantagstyper

- Skapa egna undantag - ärv från befintligt undantag

```
class MyOwnError(Exception):
    pass


# ...

raise MyOwnError

# ...

raise MyOwnError("ogiltigt värde")
```