Felhantering

Programmering 2

2024/25

Outline

Try & Except Repetition Exceptions

raise exception

Ett exempel
assert vs. raise
Använda raise exception
Fel som inte är fel men ändå fel
Övningar

Try & Except Repetition

```
while True:
    tal = int(input("Ange ett heltal: "))
svar = tal**2
print("Kvadraten på " + str(tal) + " är " + str(svar))
```

Repetition

```
while True:
       try:
           tal = int(input("Ange ett heltal: "))
           svar = tal**2
4
           print("Kvadraten på " + str(tal) + " är " + str(svar))
       except(EOFError):
6
           print("Programmet avbröts")
           auit()
       except(KeyboardInterrupt):
           print("Du försökte döda programmet")
10
```

Try & Except Repetition

Vi har garderat oss mot två undantag (fel). Vilka fler kan vi räkna med?

- ► EOFError, End Of File
- ► KeyboardInterrupt, ctrl+c

Try & Except Repetition

Vi har garderat oss mot två undantag (fel). Vilka fler kan vi räkna med?

- ► EOFError, End Of File
- ► KeyboardInterrupt, ctrl+c
- ► ValueError, värdefel

ValueError

```
while True:
       trv:
           tal = int(input("Ange ett heltal: "))
4
           svar = tal**2
           print("Kvadraten på " + str(tal) + " är " + str(svar))
       except(EOFError):
6
           print("Programmet avbröts")
           quit()
       except(KeyboardInterrupt):
           print("Du försökte döda programmet")
10
       except(ValueError):
11
           print("Du maste skriva in ett heltal")
12
```

Exceptions

Vilka Exceptions finns det?

- ► EOFError
- ► KeyboardInterrupt
- ► ValueError

Exceptions

Vilka *Exceptions* finns det?

- ► EOFError
- ► KeyboardInterrupt
- ► ValueError
- ZeroDivisionError
- ► TypeError
- ► IndexError
- NameError
- ▶ UnboundLocalError

Exceptions

Vilka *Exceptions* finns det?

- ► EOFError
- ► KeyboardInterrupt
- ► ValueError
- ZeroDivisionError
- ► TypeError
- ► IndexError
- NameError
- ▶ UnboundLocalError

För att hitta alla inbyggda exceptions kan du klicka här.



Outline

Try & Except Repetition Exceptions

raise exception

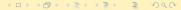
Ett exempel
assert vs. raise
Använda raise exception
Fel som inte är fel men ändå fel

Ovningar

Raise exception

Exempel

Om vi stoppar in talen 5 och 0 får vi ZeroDivisionError.



Assert

Exempel

Eftersom vi vet att funktionen inte fungerar om vi skickar med en nolla kan vi förebygga ett fel på två sätt:

```
def dela(a,b):
      assert b != 0, "Du får inte dela med noll"
      return a/b
4
  while True:
      tal = input("Skriv två heltal: ")
      tal = tal.split()
      svar = dela(int(tal[0]),int(tal[1]))
      print("Kvoten mellan "+ tal[0] + " och " + tal[1] +
          + str(svar))
```

Exempel

Eftersom vi vet att funktionen inte fungerar om vi skickar med en nolla kan vi förebygga ett fel på två sätt:

```
def dela(a,b):
      if b == 0:
           raise ZeroDivisionError("Du får inte dela med noll")
       return a/b
4
5
6
  while True:
       tal = input("Skriv två heltal: ")
       tal = tal.split()
       svar = dela(int(tal[0]),int(tal[1]))
      print("Kvoten mellan "+ tal[0] + " och " + tal[1] +
10
           + str(svar))
```

assert vs. raise

assert ska användas för att kontrollera koden under utveckling. Den ska hitta fel som utvecklaren gör. Om ett fel kan uppstå på grund av en slarvig användare så ska det istället täckas av raise exception. En anledning till detta är att när man skeppar en färdig produkt så stänger man av debug-läget som skriver ut errors. Testa skillnaden med ett program med assert i kommandotolken

```
python "mitt program.py"
python -0 "mitt program.py"
```

Förutse fel

I exemplet från tidigare:

```
def dela(a,b):
    if b == 0:
        raise ZeroDivisionError("Du får inte dela med noll")
    return a/b
```

Så är vi förutseende. Vi förutser att användaren kommer att göra fel och vi tar upp det **innan** det kan leda till problem längre ner i koden. Säg att vi har en stor funktion där divisionen med noll hade kommit sent, då är det fortfarande god sed att fånga den direkt. Varför?

Fel som inte är fel men ändå fel

Ibland täcker inte de undantag som finns något som händer i ens kod.

Exempelvis i Sänka skepp, om man skrev ett negativt tal.

Eller så har man ett program där man tar emot input där första tecknet måste vara en #.

Fel som inte är fel men ändå fel

Sänka skepp:

```
def fråga_rad():
   rad = input("Skriv en rad: ")
   rad = int(rad)
   if rad < 1 or rad > 11:
       raise ValueError("Ett tal måste vara mellan 1 och 10")
   return rad
```

Fel som inte är fel men ändå fel

```
färgkod = input("Skriv en färgkod: ")
if färgkod[0] != "#":
    raise ValueError("Alla färgkoder börjar med #")
```

Outline

Try & Except Repetition Exceptions Ett exempel
assert vs. raise
Använda raise exception
Fel som inte är fel men ändå fel
Övningar

Övningar

Utgå från filen felhantering 2.py

- 1. Kontrollera att du förstår vad koden gör i varje steg.
- 2. Se till att funktionen dela inte kraschar
- 3. Se till att inget i filen gör att den kraschar
- 4. Se till att funktionen list_delare inte accepterar listor som har färre än två element
- 5. Se till att funktionen färgväljare lyfter ett felmeddelande när man anger en annan än de åtta färgerna som finns inprogrammerade