

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

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

Try & Except

Repetition

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

Try & Except

Repetition

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

- ▶ `EEOFError`, End Of File
- ▶ `KeyboardInterrupt`, ctrl+c

Try & Except

Repetition

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

- ▶ `EEOFError`, End Of File
- ▶ `KeyboardInterrupt`, ctrl+c
- ▶ `ValueError`, värdefel

Try & Except

ValueError

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

Try & Except

Exceptions

Vilka *Exceptions* finns det?

- ▶ EOFError
- ▶ KeyboardInterrupt
- ▶ ValueError

Try & Except

Exceptions

Vilka *Exceptions* finns det?

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

Try & Except

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
Övningar

Raise exception

Exempel

```
1 def dela(a,b):  
2     return a/b  
3  
4 while True:  
5     tal = input("Skriv två heltal: ")  
6     tal = tal.split()  
7     svar = dela(int(tal[0]),int(tal[1]))  
8     print("Kvoten mellan " + tal[0] + " och " + tal[1] + " är "  
          + str(svar))
```

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:

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

raise

Exempel

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

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

raise

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

```
1 python "mitt_program.py"  
2 python -O "mitt_program.py"
```

raise

Förutse fel

I exemplet från tidigare:

```
1 def dela(a,b):  
2     if b == 0:  
3         raise ZeroDivisionError("Du får inte dela med noll")  
4     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?

raise

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 #.

raise

Fel som inte är fel men ändå fel

Sänka skepp:

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

raise

Fel som inte är fel men ändå fel

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

Outline

Try & Except
Repetition
Exceptions
raise exception

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