Föreläsning 4
Programmeringsteknik
DD1310

- Felhantering
- Definiering av egna funktioner
- Parametrar
- Lokala och globala variabler
- Retursats
- None

Felhantering

- `try` och `except` är reserverade ord som används för hantering av exekveringsfel.

```
plista= ['s','v','m','mp','fp','c','kd']
try:
   i=int(input(”Ange index: ”))
   parti = plista[i]
except:
   print (”Något fel inträffade”)
```

Syntax

```
try:
```

Kod som kan orsaka något typ av exekveringsfel

```
except:
```

Kod som exekveras om och endast om det blir något fel i kodblocket efter `try`

Så här går det till

Kod i blocket mellan `try:` och `except:` börjar exekvera, men så fort ett fel uppstår i någon rad avbryts exekveringen omedelbart, resterande rader ***i blocket*** kommer ***inte*** exekveras och koden som finns i blocket efter `except:` börjar exekvera i stället.

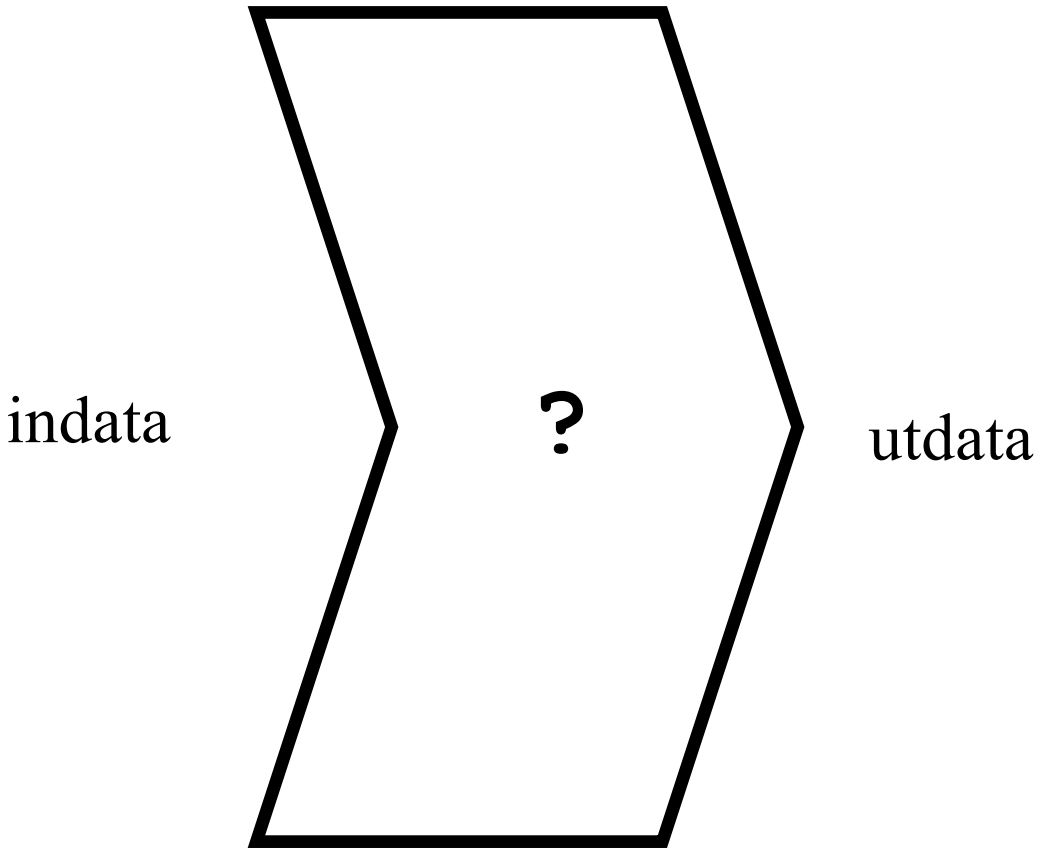
Mer detaljerad

```
plista= ['s','v','m','mp','fp','c','kd']
try:
   indx_str = input('Ange index: ')
   indx = int(indx_str)
   parti = plista[indx]
except IndexError as indx_fel:
   print ('ogilltigt index’ ,indx_fel)
except ValueError:
   print ('index maste var ett tal’)
except:
   print(’uppstod något okänd fel!’)
```

Olika typ av fel

Typ	**Beskrivning**
IOError	uppstår när man vill öppna en fil som inte finns.
IndexError	Uppstår vid tillgång till ett index i en lista som inte finns
KeyError	Uppstår när en uppslagslista inte har ett element med den nyckel som har använts i koden
TypeError	Uppstår när en inbyggd operation eller funktion tillämpas på ett objekt av olämplig typ
ValueError	Uppstår när funktion används med ett parameter med korrekt typ men fel värde
ZeroDivisionError	Uppstår när divisor i en division är 0

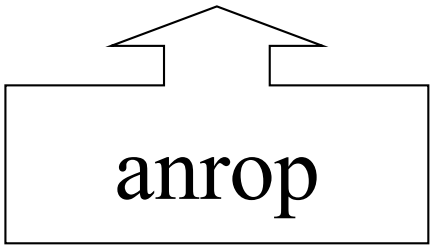
Program

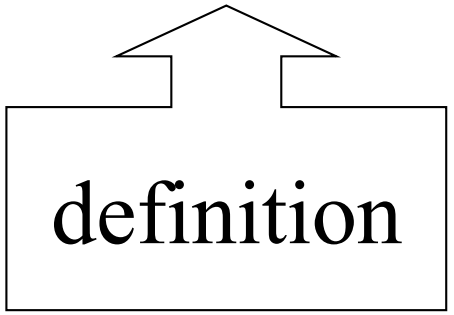


Exempel

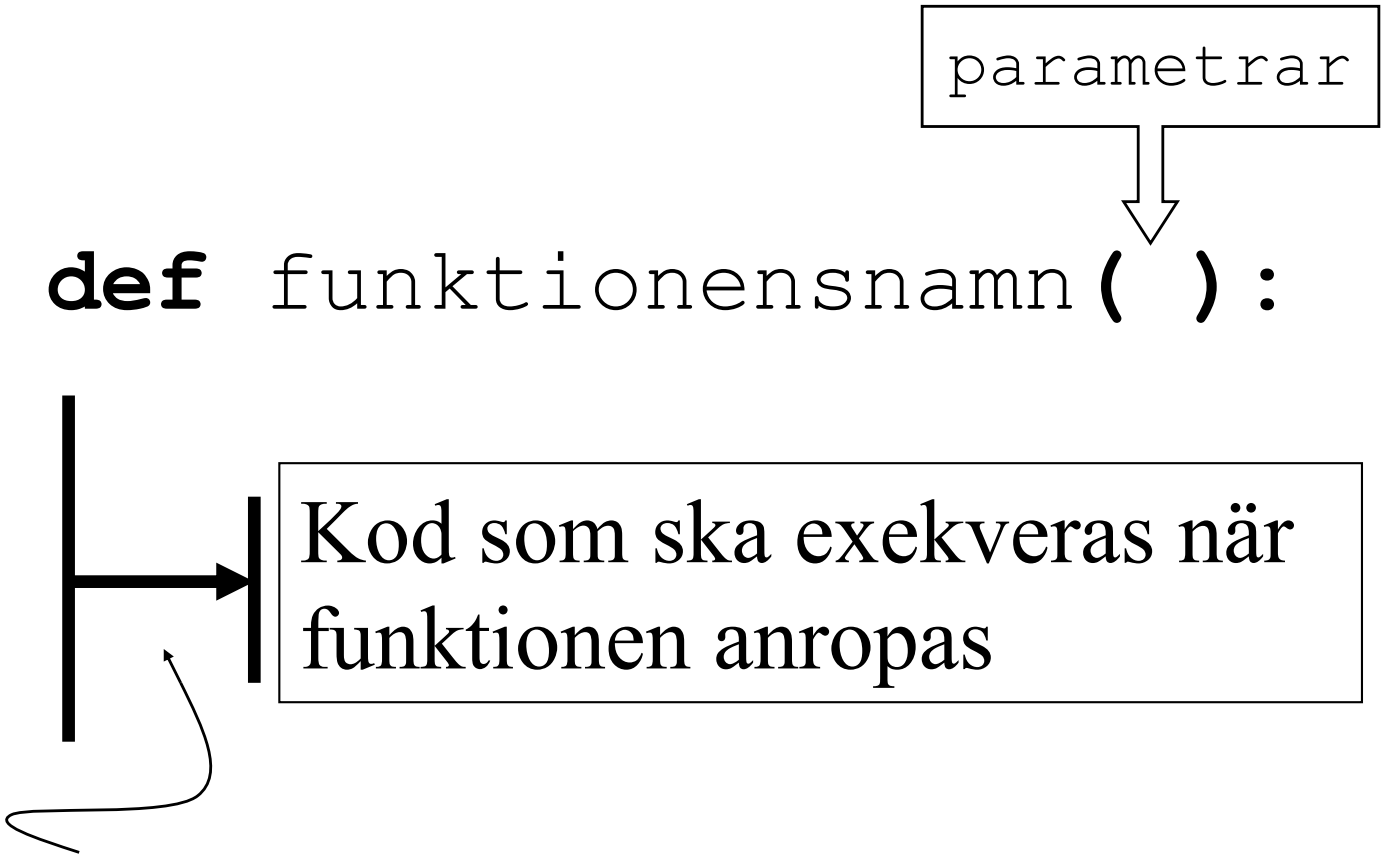
```
def meny():
  print("""1. Spela
           2. Visa spelregler
           3. Avsluta
           Välj:""",end="")
```

```
meny()
```





Syntax för funktion



Funktioner

- Används för att dela upp ett program i **lätthanterliga** och **återanvändbara** delar
- En funktion tar oftast **indata** och ger **utdata**, men en funktion kan även utföra operationer
- Man kan undvika upprepning av kod genom att använda egna funktioner och parametrar
- Med hjälp av egna funktioner inför man abstraktion i sitt program

Parametrar

En funktion med parametrar har större användningsområde än en funktion utan parametrar. Varför?

Vilken av följande har större användningsområde?

```
def calc():
  print(2*3+3)


calc() # 9
calc() # 9
```

```
def calc(a,b):
  print(a*b+b)


calc(2 , 3) #9
Calc(9 , 5) #50
```

return

En funktion med parametrar som ***returnerar*** ett värde har ännu större användningsområde.

```
def calc(a,b):
  print(a*b+b)


calc(2,3)
calc(5,4)
```

```
def calc(a,b):
  return a*b+b


r = calc(2,3)
print(r)
q = calc(r,9)
```

parametrar, `return` och `None`

- Indata skickas till funktioner via funktionens parametrar.
- Funktioner returnerar utdata med hjälp av `return`-satsen.
- Om en funktion inte har `return`-sats i kroppen kommer då funktionen att returnera `None`.
  *(`None` **vilket betyder ingenting i python)***

Exempel

```
def calc(a,b):
  c = a*b + b
  return c


m=calc(2 , 3)
```

Variabeln m kommer att få värdet 9 vilket är av typen integer

```
def calc(a,b):
  c = a*b + b



m=calc(2 , 3)
```

Variabeln m kommer att få värdet `None`

Logiskt fel

```
def BMI( height , weight):
    return weight/(height*height)
```

Vilket av följande anrop är fel?

`BMI(1.70,65)` 22.49	`BMI(65,1.70)` 0.000402

Exekveringsfel och logiskt fel

- Exekveringsfel: det är ett fel som gör att programmet *kraschar* innan det hinner *avslutas normalt*
- Logiskt fel: det är då programmet *avslutas* med felaktig utdata. (denna typ av fel är oftast svårare att spåra)

Globala och lokala variabler

En variabel kan antingen vara global eller lokal, kan alltså inte vara både samtidigt:

- Globala variabler: är variabler som lever tills programmet avslutas:
  - Globala variabler definieras med nyckelordet `global`
- Lokalvariabel: är variabler som oftast lever en kort stund.
  - Formella parameter
  - Lokala variabler som skapas i funktioner
  - Lokala variabler är känd endast i funktionen och har kortare livslängd än globala variabler

Exempel

Följande kod är korrekt:

```
def change():
    global x
    x = x/2
x = 10
print(x)
change()
print(x)
```

Följande kod ger fel:

```
def change():
    x = x/2

x=10
change()
```

Exempel

Följande kod ger fel:

```
def change(x):
    global x
    x = x/2
```

Följande kod är rätt:

```
def change(x):
    x = x/2

x=10
print(x)
change(x)
print(x)
```

Sammanfattning

- Genom att deklarera egna funktioner inför man abstraktion i sitt program, vilket gör att programmet blir lättare att läsa och förstå, dessutom koden blir mer överskådlig
- Man kan undvika kod upprepning genom att deklarera egna funktioner
- Se till att dina funktioner är fristående och självständiga
- Beroenden mellan funktioner gör det svårt att modifiera programmet
- Undvik globala variabler