Grundläggande Programmering

Föreläsning 3 - for-loopar, iteration, flödeskontroll

Agenda för idag

Repetition
for
range
break
continue
nästling
match
(om vi hinner)
funktioner (mjukstart med konkreta exempel)
klasser (mjukstart med konkreta exempel)

Repetition

Tilldelningar

Vad skriver detta program ut?

```
1  a = 13
2  b = 17
3  a = b
4  b = a
5  print(a, b)
6
```

Vad skriver detta program ut?

```
left = 13
right = 42
middle = left
left = right
right = middle
print(left, right)
```

Syntaktiskt socker för swap i Python

```
left, right = right, left
print(left, right)
```

Några frågor på en enkel for-loop

For-loopens syntax

```
for loopvariabler in Uttryck vi kan iterera över :
    Indenterat block som körs om och om igen.
    Varje nytt varv får loopvariabeln/ loopvariablerna ett nytt värde
    från “Uttryck vi kan iterera över” på samma sätt som vid tilldelning
```

Loopvariablernas datatyp(er)

Tillämpningar på for-loopen

Du har en bit av universum fångat i ett tillstånd i datorn.
Simulera universum i N steg och se vad som händer.
Utvärdera simulationen och se om den stämmer överens med verkligheten.
Utvärdera utfallet och se om det stämmer överens med vad du vill uppnå.

Matematiskt sett, utför numeriska beräkningar för att nå resultat som är svåra eller omöjliga att lösa analytiskt.

Vardagligt: Stega igenom en lista med indata och kolla att det är giltigt.

Dina första for-loopar använder bara i och range.

for i in range([start], [end], [stride])

Indenterat block som körs om och om igen.
Varje nytt varv får i ett nytt värde från sin range

Motivation för for-loopar (på sikt)

for liksom while används för att repetera en bit kod om och om igen med små variationer.

Det kan användas för att utföra numeriska beräkningar med Newton-Rhapson.

Kan användas för att processa stora mängder data med ganska lite kod, till exempel att kolla hur väl verkligheten stämmer överens med en modell: Ta 1 miljard mätpunkter, jämför med en kort simulation och mät skillnaden.

Skriv dem smart och undvik koduppprepning.

(Oftast) Mer förutsägbara än while eftersom de (oftast) itererar över något ändligt.

Funktionen range

- Tar upp till tre parametrar. Start (inklusive), End (exklusive), Stride (steglängd)
- Om du anger två kommer det att bli Start och End, steglängden 1.
- Om du bara anger en så kommer det att bli End.
- Om du missar målet så spelar det ingen roll.
- Negativa steglängder går bra.
- Syntaxen påminner om slices (från förra föreläsningen)

Experimentera med range.

```
In [9]: for i in range(1, 11, 1): print(i, end = ", ")
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
In [10]: for i in range(1, 11, 2): print(i, end = ", ")
1, 3, 5, 7, 9,
In [11]: for i in range(2, 11, 2): print(i, end = ", ")
2, 4, 6, 8, 10,
In [12]: for i in range(2, 10, 2): print(i, end = ", ")
2, 4, 6, 8,
In [13]: i
Out[13]: 8

In [14]: for i in range(8): print(i, end = ", ")
0, 1, 2, 3, 4, 5, 6, 7,
In [15]:
```

Fler range-experiment

```
In [14]: for i in range(8): print(i, end = ", ")
0, 1, 2, 3, 4, 5, 6, 7,
In [15]: for i in range(1, 5): print(i, end = ", ")
1, 2, 3, 4,
In [16]: for i in range(3, 7): print(i, end = ", ")
3, 4, 5, 6,
In [17]: for i in range(10, 0, -1): print(i, end = ", ")
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
In [18]: for i in range(11, 0, -2): print(i, end = ", ")
11, 9, 7, 5, 3, 1,
In [19]: i
Out[19]: 1

In [20]:
```

Nollindexering

Varför börjar vi så ofta räkna på 0 istället för 1?

Om du har flera saker av samma storlek som ligger efter varandra i minnet så hittar datorn dessa genom följande formel:
Addressen_till_första_elementet + index* storlek_av_elementet

Assembler, C, C++, C#, Java, Javascript, Rust och Haskell

Visual Basic och Matlab är 1-indexerade.

Tänk om alla hade samma standard...

Enkelt men vackert siffersamband

dicander@beauty: ~/Documents/gruprog/gruprog_lecture_notes/F02_if_while_for

```
1 left = 0
2 for i in range(1, 10):
3     left = 10*left + i
4     print(f"{left:10}*8 + {i} = {left*8 + i}")
~
~
```

```
         1*8 + 1 = 9
        12*8 + 2 = 98
       123*8 + 3 = 987
      1234*8 + 4 = 9876
     12345*8 + 5 = 98765
    123456*8 + 6 = 987654
   1234567*8 + 7 = 9876543
  12345678*8 + 8 = 98765432
 123456789*8 + 9 = 987654321
```

Samma exempel i BASIC på en Commodore 64 (C64), en hemdator från 1982.

```
ready.
list

10 for i=1 to 9
20 left = left*10 + i
30 print spc(9-i)left"* 8 +"i"="left*8+i

40 next
ready.
run
        1 * 8 + 1 = 9
       12 * 8 + 2 = 98
      123 * 8 + 3 = 987
     1234 * 8 + 4 = 9876
    12345 * 8 + 5 = 98765
   123456 * 8 + 6 = 987654
  1234567 * 8 + 7 = 9876543
 12345678 * 8 + 8 = 98765432
123456789 * 8 + 9 = 987654321

ready.
```

Bildkälla för datorn : Wikipedia

Python har inte goto (finns i extensions, men dessa rekommenderas inte att använda), utan uppmanar istället till att skriva break, continue, early exit (return) eller sys.exit()

```
In [43]: i = 0

In [44]: while i<10:
    ...:     i += 1
    ...:     if i == 7:
    ...:         break
    ...:

In [45]: i
Out[45]: 7

In [46]:
```

Break i mer detalj

Hoppa ur nuvarande loop direkt utan att köra resten av sekvensen.

break ligger oftast inuti en if-sats i loopen.

Om det finns en else-sats på while så skippas den vid break.

Villkoret för while-loopen testas inte.

För att hoppa ur nästlade loopar: använd en extra variabel som är False från början men blir True när det är dags att lämna alla loopar. Testa den efter att innerloopen har lämnats och gör break igen. Alternativt om det är en funktion använd return, alternativt om programmet ska avslutas, använd sys.exit()

Ett annat litet hopp: continue

continue börjar om från loopens början och testar villkoret igen.

```
In [46]: i = 0

In [47]: j = 0

In [48]: while i < 10:
    ...:     i += 1
    ...:     if j >= 3:
    ...:         continue
    ...:     j += 1
    ...:

In [49]: i
Out[49]: 10

In [50]: j
Out[50]: 3

In [51]:
```

break och continue i mer detalj

Äldre och mer systemnära programmeringsspråk har ibland en instruktion som heter goto som kan användas för friare hopp. Python har inte denna instruktion (Om man inte använder extensions, men dessa rekommenderas inte i grundkursen). Istället uppmanar python till att använda mer specifika instruktioner som break och continue för att kunna avbryta ett varv i en loop snabbt (continue) eller avbryta loopen helt och hållet (break).

break och continue bor oftast inuti if-satser (men inte vilka if-satser som helst).

break och continue kan inte användas utanför loopar: använd sys.exit() för att hoppa ur ett program och early exit (return) för att avsluta en funktion tidigt (funktoner kommer senare i kursen i detalj, ni som läste kapitel 4 i Pythons tutorial fick se lite exempel)

Nästlade for-loopar

Du kan lägga in en for-loop i en for-loop

Gå igenom alla pixlar på en skärm

Matrisoperationer (Se Linjär Algebra)

För varje lista, gå igenom elementen i listan

Kombinationer! För varje element, gå igenom alla andra element.

Match-syntax (Ni behöver inte kunna http-status-koderna utantill!)

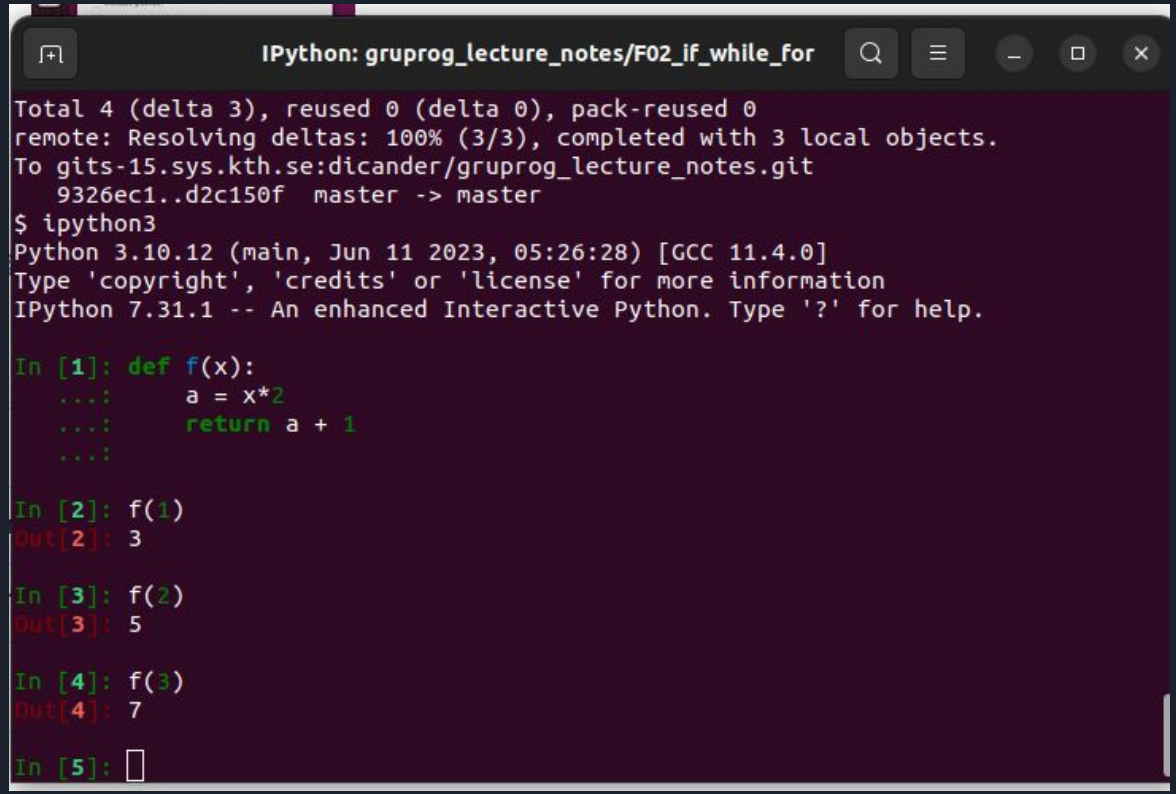
Vanliga if-satser

```
In [1]: status_code = 200

In [2]: if status_code == 200:
   ...:     print("OK")
   ...: elif status_code == 403:
   ...:     print("Forbidden")
   ...: elif status_code == 404:
   ...:     print("Not Found")
   ...:
OK

In [3]:
```

```
In [3]: match status_code:
   ...:     case 200:
   ...:         print("OK")
   ...:     case 403:
   ...:         print("Forbidden")
   ...:     case 404:
   ...:         print("Not Found")
   ...:
OK

In [4]:
```

Minimal funktion

Enkel funktion, detaljer



Den minimala klassen

```
In [11]: class Minimal:
    ...:     pass
    ...:

In [12]: m1 = minimal()

In [13]: m2 = minimal()

In [14]: m1 == m2
Out[14]: False

In [15]: m1 is m2
Out[15]: False
```

Lite mer avancerad klass

```
In [16]: class Person:
    ...:     def __init__(self, name, age):
    ...:         self.name = name
    ...:         self.age = age
    ...:

In [17]: p1 = Person("Greta Thunberg", 20)

In [18]: p2 = Person("Malala Yousafzai", 26)

In [19]: p1.name
Out[19]: 'Greta Thunberg'

In [20]: p1.age
Out[20]: 20

In [21]: p2.name
Out[21]: 'Malala Yousafzai'

In [22]: p2.age
Out[22]: 26
```

Smidigare syntax för ovanstående

```
In [25]: from dataclasses import dataclass
```

```
In [26]: @dataclass
    ...: class Person:
    ...:     name: str
    ...:     age: int
    ...:

In [27]: p1 = Person("Greta Thunberg", 20)

In [28]: p2 = Person("Malala Yousafzai", 26)

In [29]:

In [29]: p1.age
Out[29]: 20

In [30]: p1.name
Out[30]: 'Greta Thunberg'

In [31]: p2.name
Out[31]: 'Malala Yousafzai'

In [32]: p2.age
Out[32]: 26
```