UPPSALA UNIVERSITET

Typer, funktioner och turtlar

Carl Nettelblad

2020-01-23

Labbar och deadlines

- Vill du sitta i kö för att få redovisa?
  - Gå på sista labbpasset som är tillåtet enligt deadline
- Vill du få redovisa mer i lugn och ro och få feedback?
  - Gå på ett tidigare pass!

- Tänk igen på att deadline för OU1 och OU2 ligger **väldigt tätt**
  - Du får mer lagom tid om du är klar med OU1 i början av nästa vecka, inte i slutet

UPPSALA UNIVERSITET

Tilldelning

```
a = 3
b = a
a = a + 1
c = ’Python är skoj’
b > a
c = 5
a > c
b = 6
b > a
```

Om en Jupyter Notebook

- En Notebook är den kod som kördes och de resultat som gavs
- Om du stänger en notebook och öppnar den dagen efter *finns inga variabelvärden kvar*
  - Men koden och all utmatning finns kvar, så om du lagt upp det på rätt sätt blir det lätt att köra igen
  - Medan den är öppen laddar Jupyter en ”kernel” för Python som kör koden och lagrar variabelvärden o.s.v.

UPPSALA UNIVERSITET

Python – ett imperativt språk

- Vi utför instruktioner successivt i ordning
- Funktionella språk
  - Beskriva samband mellan in- och utdata
    - LISP, Haskell
- Deklarativa språk
  - Beskriv vad man vill uppnå
    - SQL för databaser
      - Beskriv den mängd data du arbetar med, inte hur man söker rätt på den

UPPSALA UNIVERSITET

Jämförelse

- Tilldelning `a = b`
  - ”Gör så att variabeln a får det värde variabeln b har just nu”
- Jämförelse `a == b`
  - ”Har a samma värde som `b`, i så fall sanningsvärdet `True`, annars sanningsvärdet `False`”

UPPSALA UNIVERSITET

Programflöde

- Vi vill inte bara stapla kommandon på varandra.

```
if a > b:
      print(’a är störst’)
elif a == b:
      print(’a lika med b’)
else:
      print(’b är störst (kanske?)’)
```

UPPSALA UNIVERSITET

Indentering

- Vanligt i många språk att man gör indrag (indenterar) för att visa vilka rader som hör ihop
- I Python är indenteringen *semantisk*
  - Indraget styr vad koden faktiskt betyder
  - Inget ”end” eller liknande i slutet av ett if-block

Slingor

```
while a < b:
        a = a + 1
        print(’Ökar a med 1’)
```

- En `if`-sats kontrollerar villkoret en gång
  - Utför innehållet om det uppfylls
- En `while`-sats kontrollerar villkoret
  - Utför innehållet om det uppfylls
  - Och igen om det fortfarande uppfylls…
  - Och igen om det fortfarande uppfylls…
  - …

UPPSALA UNIVERSITET

Andra tilldelningar

```
a += 3
a *= 3
```

- Exempel på ett mönster:
  - Säg inte samma sak två gånger
  - Om du säger det två gånger kan du råka ändra ena utan att ändra andra (kopierad kod)

Typer

```
a = 3
b = 'Hej'
a < b
```

- Vad händer?

  `a < b` betyder alltså `3 < 'Hej'`

  Är heltalet 3 mindre än texten (strängen) Hej?

Typer

```
a = ’3’
b = ’Hej’
a < b
```

- Vad händer?

  a < b betyder alltså ’3’ < ’Hej’

  Är strängen 3 mindre än strängen Hej?
- Används för att sortera

UPPSALA UNIVERSITET

Uttrycks typ

- Varje *uttryck* i Python har alltså en typ
  - Beror på typerna på ingående uttryck
  - En variabels typ beror på variabelns värde
  - En variabel kan byta typ vid tilldelning

```
a = 3
a = 'Hej'
```

  - Python är dynamiskt typat (en variabels typ kan ändras under körning, alltså dynamiskt)

UPPSALA UNIVERSITET

Funktioner

- Vi har redan sett funktionen `print` i Python
- Vi använder funktioner för att organisera vår kod
- En funktion kan returnera värden, eller låta bli…

- Några speciella funktioner i Python används för typomvandling, exempel:
  - `str` för att skapa en sträng
  - `int` för att skapa ett heltal (tar heltalsdelen om invärdet är ett decimaltal)
  - `float` för att skapa ett decimaltal (mer exakt flyttal)

UPPSALA UNIVERSITET

Exempel

```
4 < 39
str(4) < 39
str(4) < str(39)
str(4 < 39)
```

- Vad är typerna för de olika (del)uttrycken?
- Vilka jämförelser kan utföras? Vad blir resultaten?

UPPSALA UNIVERSITET

Heltalsdivision och heltalsomvandling

- I nätlektion 1 såg ni `-5 // 2`
  - Vad blir det?
  - Varför? (Vad blir `5 // 2`? Vad blir `5 % 2`? Vad blir `-5 % 2`?)

- Vad blir `int(-2.5)`? `int(-2.01)`? `int(2.5)`? `int(2.01)`?

UPPSALA UNIVERSITET

Listor

- Python har två sorters typer
  - Primitiva typer
    - ”Värden”
    - Heltal, flyttal, sanningsvärden och strängar
  - Alla andra typer
    - Till exempel listor

- En variabel innehåller antingen ett värde
  - Inklusive det speciella värdet None
  - Eller en *referens* till ett *objekt* av en annan typ

UPPSALA UNIVERSITET

Arbeta med listor

```
a = list(1, 2, 3)
a = [1, 2, 3]
```

- Samma effekt, den senare är snyggare

```
b = a
a[0]
a[1]
a[2] = ’Hej’
b[-1]
b[0:2]
```

UPPSALA UNIVERSITET

Arbeta med listor

```
a.append('Nästa värde')
len(a)
```

- `append` är en *metod*
  - Den finns definierad för listor i det här fallet
  - a är en lista och alltså kan vi använda List-metoden `append` genom att skriva `a.append`
- `len` är en funktion
  - Den tar en lista som en parameter och returnerar längden

UPPSALA UNIVERSITET

for-slinga

- Vi har sett `while`
- Loopa över en lista kunde skrivas

```
i = 0
while i < len(a):
      print(a[i])
      i += 1
```

- Finns ett bättre alternativ!

```
for x in a:
      print(x)
```

UPPSALA UNIVERSITET

break, pass, continue

```
for x in [3, 9, 52]:
        if x == 9:
                break
        print(x)

for x in [3, 9, 52]:
        if x == 9:
                continue
        print(x)

for x in [3, 9, 52]:
        if x == 9:
                pass
        print(x)
```

UPPSALA UNIVERSITET

`break`, `pass`, `continue`

- `break` avslutar en slinga (i förtid)
  - Återstående värden passeras inte, `while`-villkoret kontrolleras inte igen för en `while`-slinga
- `continue` hoppar till *nästa* steg i en slinga
  - Resten av kodet i blocket körs inte för den nuvarande *iterationen*
  - Hämtar nästa värde och början om från början för `for`, testar villkoret igen för `while`
- `pass` gör **ingenting**, används i tomma block

UPPSALA UNIVERSITET

for-slinga

- `for` kan gå över olika sorters itererbara typer (*iterables*)
- Vanliga fall:

```
a = [3, 9, 52]
for x in a:

for x in range(5):

for x in range(1,6):

for (i, x) in enumerate(a):
```

Vilka metoder finns det?

```
a = [1,2,'Hej']
```

- Vilka metoder har `a`? Vilka metoder har `a[2]`? Vilka typer har de båda objekten?

- Hur tar du reda på det?
  - ipython/Jupyter/IDLE/Visual Studio Code kan ge en lista
  - `dir`-funktionen
  - Officiella Pythondokumentationen
    - https://docs.python.org/3/library/stdtypes.html

UPPSALA UNIVERSITET

Kombinera metoder

- ”Hur läser man in en lista med heltal med input?”
  - Det går inte!

- Vi kan däremot läsa in en sträng och tolka den till en lista med heltal.
  - Till exempel ’1,421,12,99’
  - Hur?
  - Vi kan stega över alla tecknen
    - Antingen ett komma
    - Eller en siffra

Lista med heltal

A:

```
val = 0
listan = []
for char in data:
    if char == ',':
        listan.append(val)
        val = 0
    else:
        val *= 10
        val += int(char)
listan.append(val)
```

B:

```
listan = []
for el in data.split(','):
    listan.append(int(el))
```

C:

```
listan = list(map(int,
       data.split(’,’)))
```

D:

```
listan = [int(x) for x in
          data.split(’,’)]
```

RÖSTA på www.menti.com, ange kod 43 86 01

Bra och dåligt med A?

- Bra:
  - Fungerar
    - Men inte med negativa tal
    - Eller en tom lista
    - Eller…
- Dåligt:
  - Lång
  - Detaljinriktad
  - Hela ansvaret att tänka på olika fall ligger på dig

UPPSALA UNIVERSITET

Vad vill vi göra?

- Dela upp strängen i en sträng för varje element som separeras med komma
- Tolka varje sådant element som ett heltal
- Lagra dem i en lista

- Finns det något i Python som kan hjälpa oss med de stegen?

- Titta bland metoderna för strängar…
  - `split`!

UPPSALA UNIVERSITET

Lista med heltal

```
listan = []
for el in data.split(','):
    listan.append(int(el))
```

UPPSALA UNIVERSITET

Bra och dåligt med koden?

- Bra
  - Fungerar
- Dåligt
  - Fortfarande detaljinriktad
    - Vi fokuserar på att bygga upp listan

- Tänk om vi kunde säga ”anropa `int` för varje element i listan vi får från `split`”

UPPSALA UNIVERSITET

`map` till vår räddning

- `map(func, x)`
  - Ta in ett itererbart objekt x (som en lista)
  - Returnera ett nytt itererbart objekt
  - I det nya objektet är varje värde returvärdet från funktionen `func` på motsvarande element i `x`
- Så om `x` ger elementen `'39'`, `'12'`, `'68'` ger `map(func, x)` elementen `func('39')`, `func('12')`, `func('68')`

UPPSALA UNIVERSITET

map

```
result = map(int, data.split(’,’))
listan = list(result)

for el in map(int, data.split(’,’)):
      print(el)
```

Vad är typen för result?

UPPSALA UNIVERSITET

List comprehensions

```
listan = [int(x) for x in data.split(’,’)]
```

- Snabbt sätt att skapa en lista genom att beskriva hur varje element i listan ska skapas

UPPSALA UNIVERSITET

En egen funktion

```
def longestString(str1, str2):
        if len(str1) > len(str2):
                return str1
        else:
                return str2
```

- Inled med `def`, funktionsnamn och noll, en eller flera parametrar
  - Glöm inte kolon och indrag
- Resultatet av funktionen skrivs med `return`
  - Värdet skickas tillbaka/returneras och funktionen avslutas

UPPSALA UNIVERSITET

En funktion utan returvärde

```
def printLongestString(str1, str2):
    if len(str1) > len(str2):
        print(str1)
        return
    print(str2)
```

- Man kan avsluta en funktion utan att returnera något
  - Genom att använda `return` utan värde
  - Genom att komma till sista raden, utan `return`

UPPSALA UNIVERSITET

Kommentarer

- När vi skriver längre kod vill vi ofta förklara lite grann vad koden gör

- Kod är kommunikation
  - Bra kod är i stort sett självförklarande
  - Bra namn på funktioner och parametrar

- Men en del saker vill vi förklara
  - Skriv # följt av det du vill förklara

```
print(x) # Jag borde egentligen snygga till här…
```

Docstrings

- I *början* av varje egen funktion vill vi förklara hur den ska användas.
  - Är det något man måste veta om vad den gör? Vad parametrarna betyder? När den inte bör användas?
  - Använd tre citationstecken.

```
def longestString(str1, str2):
        """Returnerar den längsta strängen av två.
           Om lika långa returneras str1."""
        if len(str1) > len(str2):
                return str1
        else:
                return str2
```

Observera att docstringens info här är **fel**!

UPPSALA UNIVERSITET

Docstrings

- Olika verktyg kan automatiskt läsa och visa docstrings.
- Pythonkommandot `help` är ett exempel

```
help(math.cos)
help(longestString)
```

UPPSALA UNIVERSITET

Använda vår funktion

- Tänk nu att vi har två listor med strängar

```
swedish = [’hund’, ’hej’, ’katt’]
english = [’dog’, ’hello’, ’cat’]
```

Vi vill skapa listan `longest` som väljer de längsta orden i varje par från de två listorna.

UPPSALA UNIVERSITET

Försök 1

```
listan = []
for i in range(len(swedish)):
        listan.append(longestWord(swedish[i], english[i]))
```

- Men vi gillade inte att behöva skriva ut index när vi faktiskt bara vill iterera över hela listan
- Lite problem att vi har TVÅ listor vi vill iterera över samtidigt

UPPSALA UNIVERSITET

Försök 2

```
listan = []
for (w1, w2) in zip(swedish, english):
        listan.append(longestWord(w1, w2))
```

- `zip` skapar en iteration över par från två itererbara objekt (tänk tänderna i ett blixtlås som möts)

UPPSALA UNIVERSITET

Försök 3

```
listan = list(map(longestWord, swedish, english))
```

- `map` kan ta flera itererbara objekt samtidigt och var och en av dem bidrar då med en parameter till funktionen som anropas

- Att använda färdiga verktyg från Python gör koden
  - Kortare
  - Mer entydig
  - Lättare att förstå (när man känner igen mönstren)

UPPSALA UNIVERSITET

Paddgrafik

- Det finns mängder av paket för Python
- En del används för att skapa användargränssnitt eller grafik

- Vi använder det väldigt enkla paketet (modulen) `turtle` för att:
  - Testa att använda ett enkelt paket
  - Testa att skriva kod som visar något på skärmen

UPPSALA UNIVERSITET

Moduler

`import turtle`

- Då kan vi sedan använda delar från turtle genom att skriva `turtle.`*`namn`*, till exempel `turtle.Turtle()` för att skapa ett `Turtle`-objekt
- Turtle är alltså en ny typ som definieras i modulen turtle

`from turtle import *`

- Då kan vi skriva `Turtle()` direkt (inget `turtle.` först)

`from turtle import Turtle`

- Vi kan ange exakt vilka namn vi vill använda, bra om modulen är stor, eller om flera moduler man använder har delar med samma namn

`import turtle as t`

- Så kan man skriva `t.Turtle()`, `t` blir ett alias för `turtle`
- En del modulnamn är väldigt långa, koden kan bli tjatig

Våra paddor

```
import turtle
t = turtle.Turtle()
t2 = t
t.forward(100)
t2.left(90)
t.forward(50)
t.color('green', 'red')
t.begin_fill()
for i in range(6):
    t.right(60)
    t.forward(50)
t.end_fill()
```

Två variabler, t och t2, refererar till samma objekt.

Vi anropar metoder för *en* turtle och bara *en* turtle ritar. Om vi i stället hade skrivit `t2 = turtle.Turtle()` hade vi haft två olika.

UPPSALA UNIVERSITET

Några tips

`t.speed(0)` gör allt snabbare

`t.hideturtle()` kan vara bra om man bara vill rita

`turtle.mainloop()` kan behövas för att se till att fönstret visas/inte stängs (hör till modulen, inte objektet)

- När ett Pythonprogram tar slut avslutas det och allt försvinner
- Om man kör direkt från Pythontolken kan det ligga kvar
- Kan också starta Pythontolken med `python3 -i` *`filnamn`*

- Kalla inte ditt program `turtle.py`
  - Import `turtle` börjar med att leta efter Pythonfiler i samma katalog
    - Då blir `turtle`-modulen ditt program och den riktiga modulen hittas nite