UPPSALA UNIVERSITET

OU1, numpy, strängar och filer

Carl Nettelblad

2020-02-06

Info

- Gästföreläsning början av andra timmen
- Framsteg uppdaterade i Studentportalen
- Deadline OU2 i dag/i morgon

UPPSALA UNIVERSITET

OU1

- En uppgift där vi ritade flaggor och figurer som vandrade runt i en kvadrat.
- Eller en uppgift om att dela upp kod i funktioner och metoder i flera lager.

UPPSALA UNIVERSITET

Var ligger ansvaret?

- Vanlig effekt i deluppgift 1
  - `pentagram` och `rectangle` gör ungefär samma sak
  - Ofta

```
t = make_turtle(x,y)
t.hideturtle()
t.speed(0)
```

  - Om det är vad vi alltid vill göra borde vi lägga till det i `make_turtle`

UPPSALA UNIVERSITET

Svänga +/- 45 grader

- `move_random` har två fall
  - Gå mot mitten, oavsett befintlig riktning.
    - Aaa
  - Sväng max 45 grader åt vänster/höger.

```
direction = int(t.heading())
newdirection = random.randint(direction - 45,
       direction + 45)
t.setheading(newdirection)
```

UPPSALA UNIVERSITET

Lite långt

- Väldigt mycket ”direction”
- Kanske så här?

```
direction = t.heading()
newdirection = direction +
            random.randint(-45, 45)
t.setheading(newdirection)
```

- Tydligare att slumpvärdet inte beror på gamla vinkeln.
- Säg samma sak en gång.

Men vad finns det för metoder?

- Vi vill egentligen inte veta den gamla vinkeln.
  - Behöver den bara för att svänga relativt gamla vinkeln.
- Insikt: Höger 45 grader är samma sak som vänster -45 grader.

```
t.left(random.randint(-45, 45))
```

- Men går paddan med på att svänga en negativ vinkel?
  - Det *hade* kunnat orsaka problem, men testa!
- Kortare, enklare, tydligare.

UPPSALA UNIVERSITET

Beräkna avstånd

```
math.sqrt((t1.xcor()-t2.xcor())**2
        +((t1.ycor()-t2.ycor())**2) < 50
```

- Långt. Risk för misstag.
- Formeln för euklidiska avstånd är inte svår och kommer inte att ändras, men finns det något färdigt?

```
t1.distance(t2) < 50
```

UPPSALA UNIVERSITET

Hur ritar man tio pentagram?

```
nowx = startx
for i in range(10):
        if i < 5:
                pentagram(nowx, y – height, 50, ’green’)
                nowx += 50
        elif i == 5:
                nowx = startx
                pentagram(nowx, y + height, 50, ’green’)
                nowx += 50
        elif i > 5:
                pentagram(nowx, y + height, 50, ’green’)
                nowx += 50
```

UPPSALA UNIVERSITET

Problem

- `for`-slinga med tre fall
  - INGEN kod gemensam mellan fallen.
  - Kontinuerliga block för olika värden på `i`.
- Variabel som ändras mellan fallen
  - För att förstå fallen måste man följa vad som händer med `nowx`

UPPSALA UNIVERSITET

Snygga sätt

```
for i in range(5):
      pentagram(startx + 50 * i, y – height,
            50, ’green’)
for i in range(5):
      pentagram(startx + 50 * i, y + height,
            50, ’green’)
```

UPPSALA UNIVERSITET

Snygga sätt

```
for row in range(2):
      for i in range(5):
            pentagram(startx + 50 * i,
                  y + height * (-1 + 2 * row),
                  50, ’green’)
```

UPPSALA UNIVERSITET

Snygga sätt

```
for i in range(5):
        pentagram(startx + 50 * i,
                  y - height,
                  50, ’green’)
        pentagram(startx + 50 * i,
                  y + height,
                  50, ’green’)
```

- Kanske det tydligaste om det är okej att rita upp i den ordningen.

UPPSALA UNIVERSITET

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 str2."""
        if len(str1) > len(str2):
                return str1
        else:
                return str2
```

UPPSALA UNIVERSITET

Värden och referenser

- 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
- Samma sak gäller element i listor, tupler
- Viktigt att hålla reda på när man skickar ett original och en lista

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      for j,i in enumerate(x):
            x[j] = i * i
      return x

a = [3,5]
b = square(a)
```

UPPSALA UNIVERSITET

Kopiera en lista

- Ibland behöver vi skapa en kopia på en lista

```
a = [1,2]
b = a.copy()
```

- Om vi skivar hela listan får vi också en kopia

```
a = [1,2]
b = a[:]
```

- Om vi skapar en lista med `list` får vi en kopia

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

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      x = x.copy()
      for j,i in enumerate(x):
            x[j] = i * i
      return x

a = [3,5]
b = square(a)
```

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      x = x.copy()
      for j,i in enumerate(x):
            x[j] = i * i
      return x

a = [3,5]
b = square(a)
```

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      return [i * i for i in x]

a = [3,5]
b = square(a)
```

UPPSALA UNIVERSITET

numpy

- Listor, lexikon och tupler är enormt flexibla
- Det har en kostnad
  - Långsammare än det behöver vara
  - Använder mer minne än det behöver göra
  - Spelar nästan aldrig roll för hundra tal
  - Spelar nästan alltid roll för miljarder tal
- `numpy` är ett bibliotek specifikt för vektor- och matrisberäkningar
  - Egen typ `numpy.ndarray`
  - Stöder skivning för indexering på samma sätt som listor

UPPSALA UNIVERSITET

numpy-exempel

```
import numpy
import numpy.random

ettor = numpy.ones((5,5)) # matrisstorlek som tupel
nollor = numpy.zeros((5,5))
ettsq1 = ettor * ettor # elementvis multiplikation
ettsq2 = ettor ** 2 # elementvis kvadrat
ettsq3 = numpy.matmul(ettor, ettor) # matrismultiplikation
ettsq4 = ettor @ ettor # matrismultiplikation
slumpmatris = numpy.random.randint(10, size=(5,5))
# 5x5-matris med värden 0-9
summa = numpy.sum(ettor)
# skalär summa över hela matrisen
listatillmatris = numpy.array([[1,2,3], [4,5,6]])
```

UPPSALA UNIVERSITET

Kvadrera elementen i en lista eller matris med numpy

```
def square(x):
      x = numpy.array(x)
      return x * x


a = [3,5]
b = square(a)
```

UPPSALA UNIVERSITET

Kopior av numpy-arrayer

- En numpy-array är ett objekt precis som en lista
- Dessutom flexiblare skivning
- Om man faktiskt vill ändra på värden och ha ett orört original bör man ta en kopia med `copy()`-metoden

UPPSALA UNIVERSITET

`numpy` är mycket mer

- Vanliga algoritmer för linjär algebra, lösa differentialekvationer, med mera, med mera.
- Underpaket `numpy.matlib` innehåller funktioner som ger extra ”Matlab”-lik syntax.

Lexikon

- Speciell typ för att lagra tvåtupler (par) av *nycklar* och *värden*
  - `dict` i Python (från *dictionary*)
- En nyckel kan förekomma högst en gång i ett lexikon
  - Så vi får hoppas att inga studenter har samma förnamn i vårt exempel

```
betyg = [('Amina', 5), ('Johanna', 4),
         ('Gregor', 3), ('Carl', None)]
betygDict = dict(betyg)
```

eller skapa den direkt

```
betygDict = {'Amina' : 5, 'Johanna' : 4,
             'Gregor' : 3, 'Carl' : None}
```

Lexikon fungerar som listor

- I stället för index använder vi nycklarna

```
betyget = betygDict['Gregor']
```

- Nyckeln måste finnas (ger fel)

```
betyget = betygDict['Max']
```

- Nyckeln måste vara exakt lika, följande värden finns *inte* i lexikonet

```
betygDict['gregor']
betygDict['Gregor ']
```

- Vi kan ändra eller lägga till med indexering

```
betygDict['Carl'] = 3
betygDict['Elsa'] = 4
```

- Vi kan ta bort en nyckel (och dess värde)

```
del betygDict['Carl']
```

UPPSALA UNIVERSITET

for-slingor med lexikon

- Bara nycklar (se upp med ordningen!)

```
for el in betygDict:
    print(el)
```

- Bara värden

```
for el in betygDict.values():
    print(el)
```

- Tupler med nycklar och värden

```
for el in betygDict.items():
    print(el)
for namn, betyg in betygDict.items():
    print(namn, betyg)
```

UPPSALA UNIVERSITET

Finns nyckeln?

`betyg['gregor']` ger fel om 'gregor' inte finns

Booleskt villkor om nyckeln finns:

```
if 'gregor' in betyg:
```

Booleskt villkor om nyckeln inte finns:

```
if 'gregor' not in betyg:
        betyg['gregor'] = 3
```

Uppslag med metoden `get` ger inte fel för saknade nycklar:

```
print(betyg.get('gregor'))
print(betyg.get('gregor', 3))
```

UPPSALA UNIVERSITET

Allt är ett lexikon

- Alla objekt har också ett lexikon med sin information
  - Det kan vara intressant att titta på `t.__dict__` för en padda

- En modul är också ett lexikon

```
import math
print(math.__dict__)
```

- Man ska normalt INTE modifiera eller titta på dessa
  - Men flexibelt i vissa fall

UPPSALA UNIVERSITET

Lösa problem och felsöka

- Börja bygga steg för steg
- Vad behöver jag ta in?
- Vad ska skickas ut?
- Vilka ”enkla fall” finns det?
  - ”Ett tal är ett primtal om det inte finns delare så `return true` sist”
- Testa!
  - `print` (vad händer med min lista i första steget, i varje steg av slingan)
  - Debugger (se bilder från förra föreläsningen)
    - Går att följa vilka rader som körs i vilken ordning, med vilka variabelvärden
- Regressionstesta!
  - Gjorde du en ändring? Testa att gamla och nya koden gör samma sak på de fall som inte borde ändras.

UPPSALA UNIVERSITET

Strängar

- Strängar är sekvenser, precis som tupler och listor
- Indexering, inklusive negativa index (`a[-1]`) och skivning (`a[5:9]`), fungerar
- Konkatenering med + och upprepning med * fungerar
- Strängar är oföränderliga, precis som tupler
- Du kan *inte* skriva `a[3] = 'Q'` för att byta ut ett tecken

UPPSALA UNIVERSITET

Användbara metoder i `str`

`str.count(substr)` räkna efter hur många gånger `substr` finns i `str`

`str.find(substr)` returnerar index till första förekomsten av `substr` (`-1` om den ej finns)

`str.index(substr)` ger ett fel om `substr` inte finns i stället

`str.lower()` returnerar en *ny* sträng där `str` är skriven med enbart gemener

`str.upper()` returnerar en *ny* sträng där `str` är skriven med enbart versaler

`str.split(delim)` returnerar en lista med ”ord” ur `str`, där orden separeras med `delim`

`str.join(seq)` returnerar en konkatenering av strängarna i sekvensen `seq`, med `str` inskjuten emellan, alltså `seq[0] + str + seq[1] + …`

UPPSALA UNIVERSITET

Använda filer

- Vi kan arbeta med filer i Python.
- Antingen speciella moduler för olika filformat, eller att läsa och skriva filer direkt (text eller binära data).
- Viktigt att skilja på filnamn, filer och filobjekt
  - Filer finns på datorn (i ”filsystemet”)
  - Filnamn är strängar
    - Antingen bara ett namn eller en sökväg (med / eller \ beroende på operativsystem)
    - Pythonprogrammet körs i en viss katalog
      - Måste ange sökväg om inte filen finns i samma katalog

UPPSALA UNIVERSITET

Filobjekt

- Ett *filobjekt* beskriver hur vi arbetar med en fil i Python
  - Har:
    - Olika metoder
    - En koppling till den verkliga filen
    - En ”markör”
      - Läsning eller skrivning går igenom ett antal tecken och flyttar markören
    - Ibland en buffert
      - Filen måste *stängas* när man är klar med den
      - Särskilt om man skriver
        - » Även om man bara läser

UPPSALA UNIVERSITET

Öppna en fil

- Vi använder funktionen `open` för att skapa våra filobjekt

```
fil = open("filen.txt", "r")
helafilen = fil.read()
fil.close()
print(helafilen)
```

- `read` utan fler parametrar läser så mycket det går från där markören står. Den börjar först i filen, så variabeln `helafilen` innehåller hela filen som en sträng.
- `"r"` betyder read.

Stänga automatiskt

- Det är lätt hänt att man glömmer att stänga.
  - Särskilt om det kan inträffa något fel, eller man returnerar eller liknande.
  - Med ett `with`-block i Python kan man skapa ett objekt som ska ”slängas bort” korrekt när blocket är slut.

```
with open("filen.txt", "r") as fil:
      helafilen = fil.read()
print(helafilen)
```

Problem med åäö

- De flesta datorer är numera överens om hur det engelska alfabetet och andra grundläggande tecken skrivs (ASCII-standarden).
  - Inte lika självklart för åäö
  - Ännu mindre självklart för ryska
  - Ännu mindre självklart för kinesiska
  - Vad Python tar som standard beror på operativsystem och språkinställningar
  - Problem om din fil inte sammanfaller med det
- Ibland kan man behöva ange `encoding` (teckenkodning) manuellt till `open`
- `De två vanligaste för svenska:`

```
fil = open("filen.txt", "r", encoding="latin-1")
fil = open("filen.txt", "r", encoding="utf-8")
```

UPPSALA UNIVERSITET

Läsa lite i taget

- Inte praktiskt att hantera en fil som en enda sträng.
  - Tänk om filen är gigantisk.
- För textfiler ofta rimligt att arbeta rad för rad.

  `fil.readlines()` skapar en lista med alla rader
  - Behöver ändå vänta på att hela filen har lästs in innan man kan fortsätta
    - Läsa filer är ofta *mycket* långsammare än allt annat datorn gör
- Ett filobjekt är ett itererbart objekt, så du kan skriva:

  ```
  for line in fil:
  ```
- Läser en rad i taget, kör innehållet i slingan

Skriva till filer

```
with open('filen.txt', 'w') as fil:
      print(1,2,3,file=fil)
      a = [1,2]
      fil.write(f'En lista: {a}')
```

- 'w' i stället för 'r'
- VARNING!
  - Att bara öppna en fil för skrivning innebär att filens storlek ändras till 0 byte och markören ställs i början av filen.
  - Filens gamla innehåll tas bort.

UPPSALA UNIVERSITET

Reguljära uttryck

- Man vill ofta hitta specifika strängar på olika sätt
- ”Hitta alla ord”
- ”Hitta alla rader som börjar med !, slutar med blaj och innehåller ett mellanslag följt av mer än 6 siffror i följd”

- Vi kan skriva utryck för det i Python

UPPSALA UNIVERSITET

Hitta ett ord, jobbiga sättet

```
ordlista = []
alfabet = ’abcdefghijklmnopqrstuvwxyzåäöABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖ’
ord = ’’
instr = ’här har vi några ord’
for tecken in instr:
        if tecken in alfabet:
                ord += tecken
        else:
                if ord != ’’:
                        ordlista.append(ord)
                ord = ’’
if ord != ’’:
        ordlista.append(ord)
```

Hitta ett ord, elegant

```
import re
instr = ’här har vi några ord’
ordlista = re.findall(r’[a-zA-ZåäöÄÖÖ]+’, instr)
```

- Ett reguljärt uttryck är en ”mall” för hur en sträng ska se ut. Här använder vi hakparentes för att ange flera möjliga tecken. Inuti hakarna anger - ett intervall av tecken.
- + har den speciella betydelsen att innebära en eller förekomster av föregående tecken. (ba+b matchar bab, baab, baaab o.s.v.)
- * innebär 0 eller flera förekomster (så ba*b matchar bb, bab, baab o.s.v.)
- Parentes kan innebära grupperingar ((ba)+ innebär ba, baba, bababa, o.s.v.
- ? innebär 0 eller 1 förekomst (så b(aa)?b innebär bb eller baab)

UPPSALA UNIVERSITET

Olika funktioner

- `findall` returnerar en lista med alla icke-överlappande träffar
- `search` söker efter en match (`None` om ingen finns)
- `finditer` returnerar ett itererbart objekt med alla icke-överlappande träffar
  - Om du ändå ska köra i en for-slinga kan `finditer` vara bättre än `findall`

UPPSALA UNIVERSITET

Filtrera rader

- ”Hitta alla rader som börjar med !, slutar med blaj och innehåller ett mellanslag följt av mer än 6 siffror i följd”

```
with open(’filen.txt’, ’r’) as fil:
      for ln in fil:
            if re.search(r’^!.*[0-9]{7,}.*blaj$’, ln):
                  print(ln)
```

- `^` början på strägen/raden
- `.` valfritt tecken
- `$` slutet på strängen/raden

- Jämför med att skriva ett eget villkor för att kolla detta…

Reguljära uttryck inte bara i Python

- Många olika språk och bibliotek har stöd för reguljära uttryck
  - Lite olika exakt vilken syntax
- Reguljära uttryck innehåller ofta specialtecken
  - I Python sätter vi r före strängen för att markera ”rå” sträng, så många specialtecken tappar sin betydelse
- I terminalen kan vi använda kommandot grep för att söka i filer med reguljära uttryck

```
grep "^.*[0-9]\{7,\}.*blaj$" test.txt
```

- Notera citationstecken och \ före { för att inte specialtecken ska feltolkas