DD1311 Programmeringsteknik med PBL
Per Sedholm
26 mars 2008

Debugging

För att avlusa ett program måste man själv sätta sig in i koden (om man inte skrivit den), så att man kan förstå vad den *borde* göra. Alltså, själv göra jobbet som kompilator; det som Python (programmet) gör. Det är måhända inte så roligt, men ifall inte *du* förstår vad programmet gör, hur kan du då förvänta dig att Python ska göra det?

När man förstår vad programmet borde göra, kan man sedan gå igenom det, rad för rad, och se efter så att programmet faktiskt gör det. Oftast genom att lägga in extra utskrifter, och testköra: är jag inte säker på vad listan `lst` innehåller på rad 57, så lägger jag in `print lst` på rad 58, och kör programmet igen.[1]

Antag vi har ett program som läser in en del Monty Python-sketcher, och ska skriva ut dem sorterat. Vi vill sortera dem som engelsk text, dvs ord som ”The”, ”A”, ”An”, etc i början av titeln ska inte påverka ordningen.

Vi börjar med att skriva ett program som åtminstone inte bryr sig om ”The”.

```
class Sketch(object):
    """A Monty Python sketch"""
    def __init__(self, title):
        self.title = title
        self.normal = title
        if (self.normal.startswith("The ")):
            self.normal = self.normal[5:] + ", The"

    def normalized(self):
        return self.normal

    def __str__(self):
        return self.title

def read_file(filename):
    sketches = []
    f = open(filename, "r")
    s = f.readline().strip()
    while s:
        sketches.append(Sketch(s))
    f.close()
    return sketches

sketches = read_file('sketches.txt')
print "Before sorting:"
print sketches

sketches.sort()
print "***"

print "After sorting:"
print sketches
```

(Det här är inte enda sättet att lösa uppgiften, och förmodligen inte det bästa sättet heller. Man kan förmodligen klara sig utan klassen *Sketch*, till exempel. Men ifall vi bara använder de inbyggda klasserna, så har vi inte lika mycket att debugga. . .)

När vi provkör det så händer ingenting. Inga utskrifter, inga felmeddelanden. Varför? Eftersom programmet aldrig ens skriver ut `Before sorting:`, så måste felet uppstå innan dess. Vi får som sagt inga felmeddelanden, vilket vi skulle få ifall filen inte fanns, så förmodligen går det bra åtminstone tills vi öppnat filen. Men lyckas vi läsa något? Vi ser efter ifall vi faktiskt lägger till någonting i listan genom att lägga till en utskrift i slingan (i funktionen `read_file`):

```
    f = open(filename, "r")
    s = f.readline().strip()
    while s:
        print "### Adding sketch for " + s
        sketches.append(Sketch(s))
```

När vi provkör igen får vi massor med utskrifter. Vi bryter med *Ctrl-C*.

[1] För större program kan man även använda en *debugger*, men det går vi inte igenom här.

```
### Adding sketch for The Dead Parrot
### Adding sketch for The Dead Parrot
[...]
### Adding sketch for The Dead Parrot
### Adding sketch for The Dead Parrot
<Ctrl-C>
```

Det verkar som om vi aldrig läser mer än första raden ifrån filen. Och det stämmer ju – vi uppdaterar aldrig variabeln `s` i slingan. Ändra igen, till

```
    while s:
        print "### Adding sketch for " + s
        sketches.append(Sketch(s))
        s = f.readline().strip()
```

Då funkar det:

```
### Adding sketch for The Dead Parrot
### Adding sketch for The Lumberjack Song
### Adding sketch for Spam
### Adding sketch for Nudge Nudge
### Adding sketch for The Spanish Inquisition
### Adding sketch for Upper Class Twit of the Year
### Adding sketch for Four Yorkshiremen sketch
### Adding sketch for Cheese Shop
### Adding sketch for The Ministry of Silly Walks
Before sorting:
[<__main__.Sketch object at 0xa5310>, <__main__.Sketch object at 0xa5350>, <__
main__.Sketch object at 0xa52f0>, <__main__.Sketch object at 0xa5530>, <__main
__.Sketch object at 0xa5550>, <__main__.Sketch object at 0xa5570>, <__main__.S
ketch object at 0xa5590>, <__main__.Sketch object at 0xa55b0>, <__main__.Sketc
h object at 0xa55d0>]
After sorting:
[<__main__.Sketch object at 0xa52f0>, <__main__.Sketch object at 0xa5310>, <__
main__.Sketch object at 0xa5350>, <__main__.Sketch object at 0xa5530>, <__main
__.Sketch object at 0xa5550>, <__main__.Sketch object at 0xa5570>, <__main__.S
ketch object at 0xa5590>, <__main__.Sketch object at 0xa55b0>, <__main__.Sketc
h object at 0xa55d0>]
```

Men utskrifterna är fula. Vi ser i och för sig att det är *olika* instanser av klassen *Sketch* (olika minnesadresser), men vi vill ju ha en lista med sorterade titlar. Vi ändrar från `print sketches` till

```
for s in sketches:
    print s
```

så att print skriver ut varje enskilt element i listan istället. (Då kommer det elementet att göras om till en sträng, och vår `__str__`-metod kommer att användas.)

```
Before sorting:
The Dead Parrot
The Lumberjack Song
Spam
Nudge Nudge
The Spanish Inquisition
Upper Class Twit of the Year
Four Yorkshiremen sketch
Cheese Shop
The Ministry of Silly Walks
***
After sorting:
Spam
The Dead Parrot
The Lumberjack Song
Nudge Nudge
The Spanish Inquisition
Upper Class Twit of the Year
Four Yorkshiremen sketch
Cheese Shop
The Ministry of Silly Walks
```

Men ordningen blir fel. Ja, vi har ju inte berättat vad instanserna av *Sketch* ska sorteras på! Ändra till

```
sketches.sort(key=Sketch.normalized)
```

```
Before sorting:
The Dead Parrot
The Lumberjack Song
Spam
Nudge Nudge
The Spanish Inquisition
Upper Class Twit of the Year
Four Yorkshiremen sketch
Cheese Shop
The Ministry of Silly Walks
***
After sorting:
Cheese Shop
Four Yorkshiremen sketch
Nudge Nudge
Spam
Upper Class Twit of the Year
The Dead Parrot
The Ministry of Silly Walks
The Spanish Inquisition
The Lumberjack Song
```

Men det blev ju jättekonstigt?! Alla ”The XXX” kommer sist, till och med efter ”Upper Class...”. Vad returnerar metoden `normalized`? Gå igenom listan efter inläsningen, och skriv ut.

```
sketches = read_file('sketches.txt')
for s in sketches:
    print "'" + s.normalized() + "' from " + str(s)
```

```
'ead Parrot, The' from The Dead Parrot
'umberjack Song, The' from The Lumberjack Song
'Spam' from Spam
'Nudge Nudge' from Nudge Nudge
'panish Inquisition, The' from The Spanish Inquisition
'Upper Class Twit of the Year' from Upper Class Twit of the Year
'Four Yorkshiremen sketch' from Four Yorkshiremen sketch
'Cheese Shop' from Cheese Shop
'inistry of Silly Walks, The' from The Ministry of Silly Walks
```

Vi har visst plockat bort för mycket i början av titeln. Ändra i `__init__` så att vi plockar bokstav 4 och framåt (istället för 5):

```
    def __init__(self, title):
        self.title = title
        self.normal = title
        if (self.normal.startswith("The ")):
            self.normal = self.normal[4:] + ", The"
```

```
Before sorting:
The Dead Parrot
The Lumberjack Song
Spam
Nudge Nudge
The Spanish Inquisition
Upper Class Twit of the Year
Four Yorkshiremen sketch
Cheese Shop
The Ministry of Silly Walks
***
After sorting:
Cheese Shop
The Dead Parrot
Four Yorkshiremen sketch
The Lumberjack Song
The Ministry of Silly Walks
Nudge Nudge
Spam
The Spanish Inquisition
Upper Class Twit of the Year
```

Då funkar det. Slutgiltigt program, alltså:

```
class Sketch(object):
    """A Monty Python sketch"""
    def __init__(self, title):
        self.title = title
```

```
        self.normal = title
        if (self.normal.startswith("The ")):
            self.normal = self.normal[4:] + ", The"

    def normalized(self):
        return self.normal

    def __str__(self):
        return self.title

def read_file(filename):
    sketches = []
    f = open(filename, "r")
    s = f.readline().strip()
    while s:
        #print "### Adding sketch for " + s
        sketches.append(Sketch(s))
        s = f.readline().strip()
    f.close()
    return sketches

sketches = read_file('sketches.txt')
#for s in sketches:
#    print "'" + s.normalized() + "' from " + str(s)

print "Before sorting:"
for s in sketches:
    print s

sketches.sort(key=Sketch.normalized)
print "***"

print "After sorting:"
for s in sketches:
    print s
```

Nu kan vi (eventuellt) plocka bort de bortkommenterade debug-utskrifterna.

Med tiden så blir man mer van, och skulle förmodligen kunna hitta åtminstone ett par av de här felen utan utskrifter. Man blir också mer van vid att göra ”vanliga” saker, som att skriva ut alla element i en lista, och kan snabbt skriva dit de rader som behövs för att skriva ut den. Dessutom kommer man att vänja sig vid Pythons felmeddelanden, och kunna använda dem för att snabbare hitta felen.

Men i det här fallet fick vi ju aldrig något felmeddelande från Python – all syntax var ju rätt. Det var bara fel på *vad* programmet gjorde, inte *hur*. Alltså måste man kunna gå igenom det, vid behov.

Till sist: För att dessutom ta hand om ”A”, ”An” och så vidare, så finns det en del verktyg man kan sätta sig in i – exempelvis något som kallas *reguljära uttryck* (eng. *regular expression*; förkortas ofta *regexp*). Men känner man inte till de verktygen, så går det ju alltid att ta det fall för fall. I princip

```
if (börjar med 'The'):
    (ta bort 'The' från normaliseringen)
elif (börjar med 'A'):
    (ta bort 'A' från normaliseringen)
elif ...
```