Datorövningar i aritmetik och rekursion

Syfte

I den här laborationen får du arbeta praktiskt med modulo-aritmetik, och du får implementera en test för primtal. Du får också arbeta praktiskt med rekursion, inklusive att programmera funktioner som ritar ut träd. Det visar hur rekursion kan användas till att skapa något intressant och inte bara till att beräkna tal. Du har också möjlighet att arbeta med sannolikheter och slumptal. En del av uppgifterna här är frivilliga – lös dem om du har tid.

Den här laborationen innehåller mer programmering än den föregående. Det är dock ganska korta funktioner som du ska skriva.

Aritmetik

När vi arbetar med heltalsaritmetik så finns följande operationer:

x // y – kvoten vid heltalsdivision av x med y

x % y - resten vid heltalsdivision av x med y

Uppgift 1 (redovisas):

Skriv en funktion som tar ett klockslag (timmar och minuter) och adderar ett antal timmar och minuter och returnerar ett nytt klockslag i timmar och minuter. Klockslagen kan representeras som tupler: (*timmar*, *minuter*), t ex (10, 45). Ett exempel på hur funktionen kan fungera:

```
>>> klocka_plus((10, 45), (14, 30))
(1, 15)
```

Att tänka på: om man adderar minuter så kan man hamna på över 60 minuter. Om man tar resten av heltalsdivision med 60 så får man antalet minuter när man han tagit bort eventuellt överskjutande timmar (t ex. `(70 % 60) == 10`), och om man tar kvoten vid samma division så får man de överskjutande timmarna (t ex `(70//60) == 1`). Tänk också på att man inte vill gå över 23 timmar, eller under 0 timmar.

Pröva också om `klocka_plus` fungerar för negativa tider, t ex:

```
>>> klocka_plus((10, 45), (-14, -30))
(20, 15)
```

Uppgift 2 (redovisas):

Skriv en funktion som testar om ett tal är delare till ett annat tal. Exempel:

```
>> delare(3,6)
True
```

```
>>delare(3,7)
False
```

Skriv sedan en funktion som testar om ett tal är ett primtal, och som använder den föregående funktionen.
Tips: använd en for-loop. Du kan hoppa ur loopen (och hela funktionen) med `return`.

Exempel:

```
>> primtal(12)
False

>> primtal(13)
True
```

Om du kör fast, så finns det lite tips på Blackboard under dessa instruktioner.

Rekursion

Läs först igenom kapitel 13 avsnitt 1 och 2 i *Python programming*.

Uppgift 3 (redovisas)

Testa och jämför de två varianterna för exponent (Power) och Fibonacci i kapitel 13. Gör sedan uppgift 1 i boken.

Uppgift 4 (överkurs, behöver ej redovisas)

Gör uppgifterna 3 (palindrom) och 7 (antalet oordnade urval) i kapitel 13.

Uppgift 5 (redovisas)

Ladda ner filen `trees.py`. Den innehåller en rekursiv funktion `draw_spiral` för att rita en spiral i ett fönster. Om du kör filen i Python så öppnas ett fönster där en spiral ritas ut. Experimentera lite med att ändra olika inparametrar i funktionsanropet i slutet av filen. Parametrarna är:

- x, y - spiralens startpunkt
- angle - vinkel för första (yttersta) linjen i spiralen. 0 är uppåt, 90 är åt höger.
- dangle - hur mycket vinkeln förändras för varje steg (”d” står för ”delta”, som brukar indikera förändring).
- length - längden på första (yttersta) linjen i spiralen.
- dlength - ett tal som längden delas med för varje steg.
- steps - hur många steg (dvs linjesegment) som ska ritas ut
- canvas - fönstret som spiralen ska ritas upp i

Börja med att pröva med olika värden för 'step'. Vad händer om 'step' är 0, 1, 2 osv?

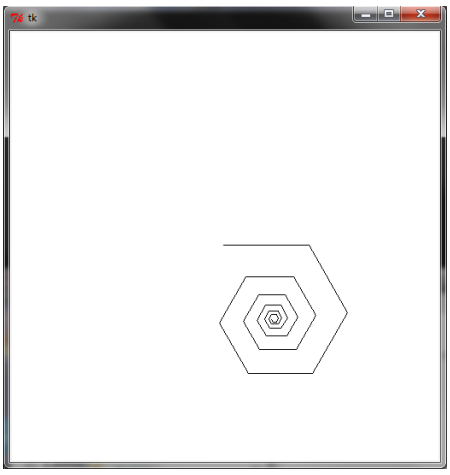
Pröva sedan olika värden på 'dangle'. Vad händer om 'dangle' sätts till ett negativt värde? Vad händer om 'dangle' blir 0?

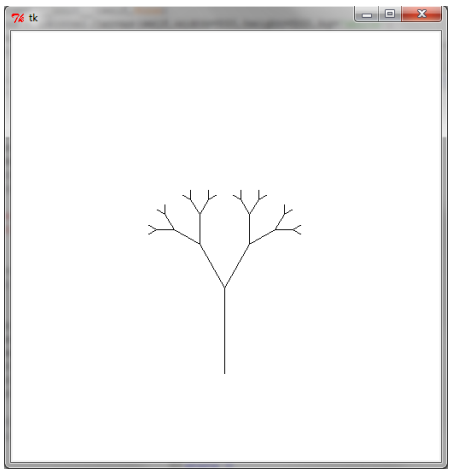
Pröva slutligen olika värden på 'dlength': 1 och 2.

Nu ska du kopiera, döpa om (till `draw_tree`) och modifiera funktionen så att den ritar upp ett träd i stället för bara en spiral. För att göra det behöver du ändra på två saker:

1. Först måste du lägga till en rad till själva funktionen draw_tree, i det rekursiva fallet. En spiral består ju bara av en enda sammanhängande kurva. Det är därför som draw_spiral anropas sig själv rekursivt en enda gång. Ett träd däremot måste innehålla förgreningar. Därför måste draw_tree anropa sig själv två gånger. Eftersom grenarna ska gå åt olika håll, så måste också de två förgreningarna ha olika vinklar.
2. Sedan måste också funktionsanropet i slutet av filen ändras, bl a så att vinkeln i början är 0 (dvs uppåt).

Se här – draw_spiral till vänster och draw_tree till höger:





Prova din draw_tree-funktion med lite olika parametrar, precis som du gjorde med draw_spiral. Se dock till att parametern 'steps' inte har för höga värden: antalet linjer att rita ut fördubblas varje gång 'steps' ökas med 1.

Gör sedan en funktion `draw_tree3` (kopiera, döp om och modifiera) som skapar ett träd med 3 förgreningar i stället för 2.

Om du tycker det är svårt så hittar du lite fler tips på Blackboard under dessa instruktioner.

Sannolikheter

Som förberedelse, läs igenom kapitel 9 i *Python programming*.

Uppgift 6 (överkurs, behöver ej redovisas)

Gör någon/några av uppgifterna 3 + 4 + 5 (tillsammans, räknas som en), 7, 8, 12 + 13 från kapitel 9 i *Python programming*.

Redovisning

Övningarna visas för assistenten och redovisas skriftligt. Om ni är två personer som har arbetat ihop, så ska ni ange det i redovisningen. Ni får gärna diskutera med andra personer/grupper under laborationen, men varje grupp ska lämna in sin egen lösning.

Checklista

[ ] Visat övningar för assistenten.

[ ] Skriftlig redovisning i en Python-fil med de uppgifter som behövs (klocka_plus, delare, primtal, Power, Fibonacci, draw_tree, draw_tree3)

[ ] Era namn står i en kommentar i början av filen.