Pythonkurs, övningssession: Felsökning och exceptions

April 7, 2020

1 Personnummerkollen

1.1 Inledning

Förutom rena fel blir det ofta också problem med ofärdig och otestad kod. Man säger ibland att allt som inte är testat är att betrakta som trasigt, eller åtminstone ofärdigt. Det följs sällan fullt ut i verkligheten, men det är ändå ett användbart tankesätt. I den här uppgiften kommer ni att utgå från en befintlig kod för att dels felsöka, dels se till att den fungerar på ett tillfredställande vis.

1.2 Uppgifter

1. Rätta till programmet som kollar personnummer så att det uppfyller specifikationerna nedan.

1.2.1 Specifikationer

1. Programmet ska fråga efter ett personnummer
2. Programmet ska visa en av två saker beroende på input:
   (a) Om det personnummer som anges innehåller kontrollsiffra så ska denna verifieras. (Enligt Luhn-algoritmen)
   (b) Om det saknar kontrollsiffra så ska en kontrollsiffra beräknas och visas för användaren. (Enligt Luhn-algoritmen)
3. Programmet ska godta personnummer formulerade med och utan bindestreck mellan födelsedatum och födelsenummer
4. Programmet ska godta personnummer både med och utan århundrade angivet.
5. Programmet ska skriva ut ett informativt felmeddelande om angivet personnummer:

(a) Är för kort

(b) Är för långt

(c) Innehåller något annat än siffror och bindestreck

1.2.2 Algoritm

Algoritmen som används för beräkning av kontrollsiffran är den så kallade Luhn-algorithmen. Den går till så här:

1. Multiplicera varannan siffra i de nio första siffrorna av personnumret med två. Börja på den första siffran.
2. Ersätt alla produkter som blir större än 10 med sin siffersumma
3. Summera alla siffror
4. Kontrollsiffran är differensen mellan resultatet och närmast högre tiotal. Om differensen är tio används resultatet 0. Detta kan skrivas som $(10 - (s\%10))\%10$ där s är siffersumman och % betecknar modulusoperatorn.

1.3 Utförande

1. Den här uppgiften löses bäst genom upprepat fixande och testande. Testa om en sak fungerar, om den inte gör det, fixa den och testa igen. Upprepa detta för följande fall:

Input	Resultat
811218-9876	Giltigt
811218-9874	Ogiltigt
811218-987	811218-9876

2. Utöka ovanstående tabell så att den även gäller för alla kombinationer av århundarade och bindestreck (19811218-9876, 8112189876, 198112189876)
3. Utifrån ovanstående information: ta fram testfall och prova dessa för att se att alla specifikationer ovan följs.
4. Givet det du vet om if-satser och exceptions, skulle du kunna ha gjort en del saker på ett annat vis? Varför valde du det sätt du gjorde?

1.4 Kommentarer

Att komma på testcase är en utmaning i sig, och det är viktigt att man täcker in alla eller åtminstone nästan alla möjliga fall. Man får inte heller glömma att testa negativa utfall. I det här fallet så använder vi till exempel ett personnummer som vi vet är felaktigt. Det är lika viktigt att vara säker på att man har rätt som att vara säker på att man har fel.

Som så ofta annars så finns det flera sätt att göra saker. I de allra flesta fall kan man helt låta bli att använda sig av exceptions genom att använda sig

av if-satser. En stor fördel med exceptions är att de kan fånga upp saker som kanske händer längre ner i kedjan. I princip så skulle en variabel kunna ändras efter din if-sats och du fångar det då inte. Efter en "try" så kommer du att fånga allting som händer. Å andra sidan kan if-satser vara smidigare och ibland upplevas som mer intuitiva.

Metoden att testköra för att se var det blir fel och sedan ändra tills det blir rätt är en just nu ganska populär utvecklingsmetod kallad testdriven utveckling. Denna kommer vi att ta upp i samband med automatiserade enhetstester.

2 Input-validering

2.1 Inledning

Som vi var inne på i förra uppgiften så är det bra att kolla att input är riktig. Ett sätt att göra detta på är exceptions.

2.2 Uppgifter

1. Skapa ett enkelt telefonboks-program enligt specifikationer.

2.2.1 Specifikationer

1. Varje post ska innehålla namn, telefonnummer, epostadress och ålder
2. När användare startar programmet så ska hen skriva in ett namn. Då ska det finnas tre fall:
   (a) Namnet finns -> Visa uppgifter
   (b) Namnet finns inte -> Skriv in uppgifter
   (c) Namnet lämnas tomt -> Avsluta programmet
3. När en användare visat eller skrivit in en post så ska ett nytt namn efterfrågas
4. Om en epostadress inte innehåller minst en punkt och precis ett '@' så ska ett nytt efterfrågas
5. Telefonnummer ska sparas som strängar som bara innehåller siffror
6. Ålder ska lagras som ett heltal (integer)
7. Programmet måste byggas enligt principen om att varje funktion endast gör en sak, eftersom det kan komma att byggas ut senare
8. Av samma skäl som ovan så ska all hantering av exceptions ligga i huvudfunktionen

2.3 Utförande

1. Börja med att skriva huvudloopen, men skriv det med funktionsnamn som inte finns. ’

2. Gå ett steg nedåt i taget genom att implementera (skriva) funktionerna som behövs

3. Tips på datastruktur: En dict med namn som nycklar, där varje post är en dict med fältnamn som nycklar

2.4 Kommentarer

Den här uppgiften kan bli mycket lättare med exceptions. Det går som vanligt att konstruera även med if-satser, men det är stor risk att flödeskontrollen för programmet bakas in då.