Pythonkurs, övningssession 1-3: Strängar

17 mars 2020

1 Grundläggande strängoperationer

1.1 Inledning

Python är ett språk som är mycket väl lämpat för att hantera och manipulera just textsträngar. Det medför att även de grundläggande operationerna är relativt kraftfulla. Vilket i sin tur gör det till än viktigare att behärska de här funktionerna för att kunna utnyttja många av fördelarna med Python.

1.2 Uppgifter

1. Skapa ett program som frågar om två strängar och sedan skriver ut dem på varsin rad, omgivna av citationstecken med ett och samma printkommando. Outputen ska se ut exempelvis så här:

   ```
   'Första strängen'
   "Andra strängen"
   ```

2. Ta reda på vilka escape-sekvenser som stöds av just din terminal, samt hur de reagerar.

1.3 Utförande

1. Öppna en ny fil enligt tidigare övningar.
2. Skriv ett program som tar emot två strängar från användaren
3. Lägg till enkla citationstecken i början och slutet av den första strängen, till exempel så här:

   ```
   new_first_string = "'" + first_string + "'"
   ```

4. Lägg till dubbla citationstecken i början och slutet av den andra strängen
5. Lägg ihop första och andra strängen med tecknet för ny rad mellan.
6. Skriv ut den nya strängen som svar.

7. Provkör och se hur resultatet blir.

8. Testa dig igenom alla escape-sekvenser nämnda i presentationen genom en serie print-kommandon. (Teckenkoder för att testa de escapesekvenserna kan du exempelvis hitta här: https://www.fileformat.info/info/unicode/block/miscellaneous_symbols). För att se resultatet bör du ha ett tecken före och ett tecken efter, exempelvis:

   ```
   print("A\nB")
   ```

1.4 Kommentarer

Escape-sekvenserna bygger på en standard som är mycket äldre än Python, och modellen med backslash som escape-tecken är även den gammal. Det här gör att vana programmerare känner igen sig, men det kan också tyvärr göra så att vissa saker känns märkliga, och framförallt att vissa beteenden inte stöds. Det lär exempelvis vara relativt få terminaler idag som stöder \a. Rent tekniskt så är det ett tecken som skickas till terminalen, som dock väljer att inte göra något av det.

2 Stränginterpolering

2.1 Inledning

Stränginterpolering innebär att lägga in andra värden i strängar. Det här kan givetvis åstadkommas genom strängkonkatenering, att helt enkelt foga in värden i strängen, som i uppgift 1. Det här blir dock ganska snart både svårläst och onödigt krångligt. Då det är en väldigt vanlig sak att vilja göra, och python är väldigt starkt i sin stränghantering, så finns det ett flertal sätt att göra det på. Anledningen till att det finns flera olika metoder är att det har utvecklats efter hand, men för att behålla bakåtkompatibilitet, samt för att inte alla som kodar ska behöva lära om, så har man behållt de gamla metoderna.

2.2 Uppgifter

1. Skapa ett program som frågar efter två tal och skriver ut summan, differensen, produkten och kvoten av talen.

2.2.1 Specifikationer

1. Det ska vara tydligt för användaren vilken input som förväntas, samt vilket svar som är vilket

2. Presentationen av resultaten för de olika räknesätten ska använda sig av olika metoder för stränginterpolation, det vill säga:

   (a) Strängkonkatenering

(b) %-operatorn

(c) .format-sträng

(d) f-sträng

3. Koden ska vara lättläst och begriplig även för någon som inte på förhand känner till vad koden ska göra.

2.3 Utförande

1. Öppna ett en ny fil.
2. Låt användaren skriva in tal genom att använda input(). Se till att skriva en begriplig uppmaning till användaren.
3. Gör beräkningar av summa, differens, produkt och kvot. För att kunna det behöver du konvertera strängarna till tal genom att använda funktionen int.
4. Gör strängar av typen "Summan av [tal] och [tal] är [resultat]" där programmet lagt in rätt värden genom metoderna som beskrivs ovan.
5. Skriv ut strängarna
6. Testkör programmet. Testa att mata in olika konstiga värden och se om du kan få programmet att krascha.
7. Undersök om det med någon eller några metoder är möjligt att utföra beräkningarna i strängen. Fundera på om det är en bra idé eller inte.

2.4 Kommentarer

Med så här pass korta strängar så torde ingen av interpolationsmetoderna vara några egentliga problem, men det kan ändå bli klart redan nu vilka fördelarna med de senare är. Om inte annat när man försöker utföra operationer i strängkonkateneringen. Här måste man dock fundera över vad som egentligen är bäst för läsbarheten. Alltför långa operationer inne i en sträng tenderar snarast att göra det svårläsligt. Ett ännu olöst problem är att programmet förväntar sig siffror, och en sträng som inte omdelebart kan konverteras till en siffra kommer att orsaka stora problem. Ett program som kraschar för att en användare skriver in fel sorts värde är inte ett exempel på bra design, och vi kommer senare i kursen att återkomma till hur man hanterar den typen av problem.

3 Strängmetoder

3.1 Inledning

Strängmetoder är egentligen en del av objektorientering, som vi kommer att återkomma till senare i kursen. Det är dock ett så pass kraftfullt, men samtidigt

lättanvänt, verktyg att vi använder det redan här. Fullständiga listor på vilka metoder som finns, finns det gott om på nätet. Ett exempel är https://www.w3schools.com/python/python_ men fler går säkerligen att hitta med en enkel websökning. De som presenteras här är sådana som används för direkt strängmanipulation, det vill säga resultatet av de blir en ny sträng. Det finns även andra som ger andra resultat och som exempelvis kan användas för att fatta beslut ifrån.

3.2 Uppgifter

1. Skriv ett program som börjar med strängen "Jag tYcker om äGg" och i slutändan skriver ut strängen "jAG tYCKER iNTE oM SPAM"

3.2.1 Specifikationer

1. Följande strängar är de enda som får hårdkodas:
   (a) "Jag tYcker om äGg"
   (b) "inte"
   (c) "SPAM"
   (d) " " (Alltså ett mellanslag)
2. Programmet ska inte ta någon användarinput
3. Följande strängmetoder får användas:
   (a) capitalize()
   (b) join()
   (c) lower()
   (d) upper()
   (e) swapcase()
   (f) replace()
   (g) split()
   (h) rsplit()
   (i) title()
   (j) format()
4. Programmet ska skriva ut vad som gjorts samt hur strängen ser ut efter varje genomförd rad.

3.3 Utförande

I följande utförande kommer det inte beskrivas när du ska göra programmet och när du ska köra det. Du förväntas vid det här laget kunna avgöra sådant själv. Men givetvis går det bra att säga till en lärare om du behöver hjälp.

1. Tilldela de hårdkodade strängarna till varsing variabel.
2. Kör någon av metoderna på strängen, exempelvis

   ```
   inital_string.capitalize()
   ```

3. Skriv ut strängen genom att följa raden ovan med en print-rad. Vad hände? Varför?
4. Skriv in hela uttrycket från punkt 2 i print-kommandot. Fungerade det annorlunda nu? Varför?
5. Skriv det som en tilldelning, det vill säga:

   ```
   capitalized_string = initial_string.capitalize()
   ```

6. Skriv ut den nya strängen. Vad blev resultatet nu?
7. Använd det du lärt dig i stegen ovan för att skriva ett program som uppfyller specifikationerna. För att ta reda på vad metoderna göra så kan du antingen googla, eller helt enkelt testa dig fram.

3.4 Kommentarer

Strängar i python är vad som kallas för immutable, det betyder att de inte kan förändras. Sträng-metoderna ändrar därför inte en sträng, utan de skapar en. Man säger att de returnerar en sträng. Den returnerade strängen kan sedan läggas i en variabel eller för den delen direkt skickas till utskrift. Men om inget görs med den, då kommer den att försvinna. Att en sträng returneras innebär också att man kan köra nya metoder på den. Följande är till exempel en giltig rad:

```
changed_string = original_string.lower().title()
```

Raden läses från vänster till höger. Så först kommer en sträng av gemener skapas genom metoden lower(). Sedan kommer den strängens metod title() att köras. Det innebär att i just det här fallet så är metoden lower överflödig. I andra tillfällen kan det dock finnas en poäng med att skriva en hel serie punkter. Desvärre blir det en kod som är ganska svårläst. Fenomenet kallas ibland för "train wreck", sannolikt för att orden ser ut som tågvagnar kopplade med punkter. Det anses inte vara best-practice, även om det är en sak man tyvärr förhållandevis ofta ser i exempel.