**Programmera med Perl**

Perl är ett programmeringsspråk som kan användas för bland annat hantering av stora och komplicerade textfiler – eller för att ladda ner data från webben, sk scraping.

Vid installationen av Perl så placeras programmet oftast på C: i biblioteket C:\Perl64. Ett tips är att skapa en folder som heter scripts eller liknande under detta bibliotek och där spara alla program man gör.

VARIABLER

I Perl börjar de flesta variabler med $, och om de uppträder för första gånger så talar man om det med my, dvs:

my $namn=’Helena’;

my $alder=48;

Alla rader i programmet avslutas med ;

Öppna en text-editor, exempelvis Anteckningar eller UltraEdit eller TextWrangler och skriv:

#!/usr/local/bin/perl

print "Helena\n";

Spara filen som pgm1.pl och tänk på att placera den i biblioteket scripts under Perl64. Öppna sedan kommandoförstret genom att trycka på Start-knappen och sedan skriva cmd. Då öppnas ett svart förster. Förmodligen står det C:\Users\Dittnamn eller något liknande. Skriv cd.. (två punkter) och tryck enter och upprepa detta tills du kommit till roten C:\. Skriv då cd perl64 och tryck enter och sedan cd scripts. I ditt svarta fönster ska det nu stå C:\Perl64\scripts> Där skriver du:

perl pgm1.pl

Vad har vi gjort? Vi har skrivit (print) en text (”Helena\n”) och texten innehåller ett returtecken (\n). Den första raden i programmet finns alltid där och talar om för perl var den ska köra programmet.

ADDERA OCH LÄGGA IHOP TEXT

Gå tillbaka till texteditorn och ändra:

#!/usr/local/bin/perl

my $alder = 47;

print "Helena is $alder years old\n";

$alder=$alder+1;

print "No, Helena is $alder years old\n";

Välj Spara som i menyn och spara filen som pgm2.pl. Gå sedan till kommandofönstret och skriv perl pgm2.pl.

Det verkar alltså som om man kan räkna med variabler på vanligt sätt. Men, hur lägger man ihop text?

Tillbaka till texteditorn och skriv in följande:

#!/usr/local/bin/perl

my $fornamn = 'Helena';

my $efternamn = 'Bengtsson';

my $namn = $fornamn." ".$efternamn;

print "Jag heter $namn\n";

Spara som pgm3.pl och gå tillbaka till svarta fönstret och kör programmet. Ändra sedan i progarmmet så att du fyller i ditt egetnamn.

LISTOR

Ibland vill man spara listor. Listor i Perl heter array och betecknas av att variabeln börjar med ett @ istället. Vi tester genom att skriva följande, spara som pgm4.pl och sedan köra programmet:

#!/usr/local/bin/perl

my @personer = ('Helena', 'Katarina', 'Lotta');

print "$personer[0] is the oldest\n";

Vad betyder då detta? Vi matar in tre namn i en lista. Ur denna lista kan vi sedan hämta ett av namnen. För att krångla till det lite extra så är första namnet nummer 0, andra är nummer 1 och tredje är nummer2. Notera att när du hämtar ett enskilt namn så förvandlas listan till en vanlig variabel, en sådan som börjar på $.

Testa att ändra programmet så att det också skriver ut att Katarina är i mitten och Lotta är den yngsta

REPETITION - LOOPAR

En viktig beståndsdel i programmering är möjligheten att repetera uppgifter – att låta något gå i en loop. Det finns olika sorters loopar, men de två vanligaste går ut på att antingen göra något i X antal gånger – eller så gör man något så länge som ett villkor är öppet. Vi börjar med det förstnämnda och går tillbaka till texteditorn och skriver:

#!/usr/local/bin/perl

for $i (1, 2, 3, 4, 5) {

 print "$i\n";

}

Spara som pgm5.pl och kör programmet. Fundera sedan på hur du skulle kunna ändra ditt program så att det istället stegar igenom listan från pgm4 (@personer) och skriver ut ett namn på varje rad.

VILLKOR

En annan viktig beståndsdel är villkor – att kolla om något är sant, då gör man X annars gör man Y. Testa genom att skriva detta lilla program:

#!/usr/local/bin/perl

my $a = 3;

my $b = 5;

if ($a > $b) {

 print "$a is larger than $b\n";

} else {

 print "$b is larger than $a\n";

}

Spara som pgm6.pl och kör programmet. Studera noga hur man skriver en OM-sats. Man måste ha parenteser kring villkoret. Man måste rama in med { och } – och man har inget ; i slutet, utan bara en sista }. Här ser man tydligt att det underlättar om man skjuter in delar av programmet, sk indentering, för att visa vad som tillhör vilken del av villkoret.

En viktig sak att tänka på är att likhetstecknet, =, betyder tilldela. Alltså $a=3; betyder att variabeln $a får värdet 3. Villman kolla om två variabler är lika måste man använda två likhetstecken, ==.

LÄSA IN FILER

Något som Perl är extra bra på är att läsa och hantera filer. Man kan öppna filer på två sätt – öppna dem för att läsa – eller öppna dem för att skriva. Vi ska börja med att läsa en fil.

Skriv följande i texteditorn:

#!/usr/local/bin/perl

open TEXTFIL, "<C:/Perl64/scripts/Fojo/artikel.txt";

while (<TEXTFIL>) {

 my $line = $\_;

 chomp($line);

 print "$1ine\n";

}

Och spara som pgm7.pl. Vad gör vi då här? Vi öppnar en fil för läsning. Att det är för läsning styrs av att det står < allra först innan filnamnet. Sedan använder vi oss av den andra sortens loop, WHILE, dvs gör detta tills filen tar slut. Vi läser en rad åt gången, använder ett kommando som rensar raden från onödiga tecken och så skriver vi ut raden, samt ett radslut. Och när alla rader tagit slut så avslutas programmet. Notera särskilt variabeln $\_ som är en av Perls standardvariabler – varje gång du läser en textfil hamnar raden du läst in i $\_.

REGULJÄRA UTTRYCK

I Perl har man stora möjligheter att hantera text och använda sig av mönster i text. För att göra detta använder man något som heter reguljära uttryck, eller Regular Expressions. Det är ett slags programmering i programmet och innebär att Perl känner igen olika text-mönster. Det finns massvis med böcker skrivna om RegEx och vi ska bara nosa på en liten del av dem. Ett sådant uttryck är [A-ZÅÄÖ] som betyder att alla tecken mellan A-Ö matchas av uttrycket. Men bara stora bokstäver. Vill jag matcha små så måste jag skriva [a-zåäö] istället. Slutligen så betyder [a-zåäö]+ att det kan finnas en eller många små bokstäver och alla dem matchas i uttrycket.

Säg då att vi skulle vilja hitta alla namn i artikeln ovan. Namn kännetecknas ju ofta av att de består av minst två ord efter varandra där båda inleds med stor bokstav. Och det finns inget annat skiljetecken än mellanslag. (Den här premissen gör att vi inte får med Anna-Karin eller Karl-Erik, men det är ok just nu). Så ändra pgm7.pl genom att lägga till följande:

#!/usr/local/bin/perl

open TEXTFIL, "<C:/Perl64/scripts/Fojo/artikel.txt";

while (<TEXTFIL>) {

 my $line = $\_;

 chomp($line);

 if ( $line =~ m/([A-ZÅÄÖ][a-zåäö]+ [A-ZÅÄÖ][a-zåäö]+)/ ) {

 print "$1\n";

 }

}

Var noga när du skriver in så att du får med alla parenteser och andra tecken, såsom ~. Det kan kännas knepigt, men är bara det sätt som Perl har valt att göra en matchning med reguljära uttryck. Parentesen runt det vi matchar, ([A-ZÅÄÖ][a-zåäö]+ [A-ZÅÄÖ][a-zåäö]+), har ett särskilt syfte. Den gör att en ny standardvariabel uppstår, $1, och vi kan använda den för att bara skriva ut matchningen, inte hela raden. Använd Spara som och spara programmet som pgm8.pl. Kör och se vad som blir resultatet.

Fundera på vad en matchningstext av typen ([0-9] [0-9] [0-9] [0-9] [0-9] [0-9]- [0-9] [0-9] [0-9] [0-9]) skulle kunna användas till.

SKRIVA TILL FILER

Om man har en mängd filer att gå igenom blir det naturligtvis ohållbart att skriva till kommandofönstret. Istället vill man samla sina matchningar i en ny fil. Ändra det tidigare programmet så här:

#!/usr/local/bin/perl

open TEXTFIL, "<C:/Perl64/scripts/Fojo/artikel2.txt";

open UTFIL, ">C:/Perl64/scripts/Fojo/utfil.txt";

while (<TEXTFIL>) {

 my $line = $\_;

 chomp($line);

 if ( $line =~ m/([A-ZÅÄÖ][a-zåäö]+ [A-ZÅÄÖ][a-zåäö]+)/ ) {

 print UTFIL "$1\n";

 }

}

close TEXTFIL;

close UTFIL;

Spara programmet som pgm9.pl och kör. Om allt har gått rätt så skapas en ny fil som heter utfil.txt. Öppna den och se efter vad som är resultatet – jämför med filen artikel. Vad funkar? Vad funkar inte? Vad skulle vi kunna ändra i programmet?

SCRAPING

För att kunna scrapa måste man ladda ner en extra modul till Perl. Gå till kommandofönstret och skriv PPM för Perl Package Manager. Ett program startas som gör det möjligt att ladda ner nya moduler. Skriv mech i sökfönstret och vänta tills PPM har hittat WWW Mechanize. Markera detta och klicka på det lilla paketet med ett plus. (Mark for install). Tryck sedan på gröna pilen och godkänn att modulen hämtas och installeras. När programmet är klart, välj File och Exit i menyn och avsluta programmet. Du har nu installerat en modul som gör det enklare att hämta saker från nätet.

För Mac-användare: Skriv sudo cpan i terminalfönstret. Ange ditt lösenord. Om du aldrig har startat cpan förut så kommer du först att installera cpan. Svara bara enter på alla frågor och vänta tills det är klart. Nu ser prompten ut så här cpan[1]>. Skriv då install WWW::Mechanize för att installera modulen. Exit lämnar cpan.

Öppna texteditorn och skriv följande program:

#!/usr/local/bin/perl

use WWW::Mechanize;

my $link=’ <http://www.allabolag.se/5561001123/verksamhet>’;

open UTFIL, ">C:/Perl64/scripts/Fojo/utfil2.txt";

my $mech = WWW::Mechanize->new();

 $mech->get( $link );

 my $page = $mech->content;

 print UTFIL $page;

close UTFIL;

Spara som pgm10.pl och kör. Titta i filen utfil2.txt och se vad du samlat in. Det är mycket text, men vi vet ju hur man hämtar specifik text från en textfil:

#!/usr/local/bin/perl

open HTMLFIL, "<C:/Perl64/scripts/Fojo/utfil2.txt";

open UTFIL, ">C:/Perl64/scripts/Fojo/klarfil.txt";

while (<HTMLFIL>) {

 my $line = $\_;

 chomp($line);

 if ( $line =~ m/(\/verksamhet\/)/ ) {

 print UTFIL "$line\n";

 }

}

close HTMLFIL;

close UTFIL;

Spara filen som pgm11.pl och kör. Öppna filen klarfil.txt och se resultatet.

Nu kan vi kombinera det vi lärt oss om att läsa in filer och scraping. I filen bolag.txt finns tio adresser till bolag som jag skapat genom att ta en lista över organisationsnummer och så lägga till ”http://www.allabolag.se/” före och ”/verksamhet” efter. Programmet ser ut så här:

#!/usr/local/bin/perl

use strict;

use WWW::Mechanize;

open BOLAG, "< C:/Perl64/scripts/Fojo/bolag.txt";

while (<BOLAG>) {

 open UTFIL, ">> C:/Perl64/scripts/Fojo/utfil2.txt";

 my $line = $\_;

 chomp($line);

 my $link = $line;

 print "$link\n";

 sleep(5);

 my $mech = WWW::Mechanize->new();

 $mech->get( $link );

 my $page = $mech->content;

 print UTFIL"$link\n";

 print UTFIL $page;

 close UTFIL;

} #endwhile

close BOLAG;

Spara som pgm12.pl och kör. Notera att vi skriver två << när vi öppnar UTFIL – det gör vi för att vi lägger till data för varje länk. Vi måste även ändra lite i pgm11.pl för att den ska kunna tala om för oss vilket bolag och vilken SNI. Notera att vi skriver även in länken i UTFIL. Det använder vi sedan när vi samlar den väsentliga informationen i klarfil.txt.

#!/usr/local/bin/perl

open HTMLFIL, "<C:/Perl64/scripts/Fojo/utfil2.txt";

open UTFIL, ">C:/Perl64/scripts/Fojo/klarfil.txt";

while (<HTMLFIL>) {

 my $line = $\_;

 chomp($line);

 if ( $line =~ m/^http:\/\/www\.allabolag\.se/ ) {

 print UTFIL "$line\n";

 }

 if ( $line =~ m/(\/verksamhet\/)/ ) {

 print UTFIL "$line\n";

 }

}

close HTMLFIL;

close UTFIL;

Här måste vi göra två saker i matchningen som är nya. Först använder vi tecknet ^som talar om att vi bara ska matcha om programmet hittar vår text i början av en rad. Det andra är att vissa tecken har speciell betydelse i reguljära uttryck – och då måste man skriva \ före för att matchningen ska matcha en faktisk punkt eller /.

Vi kan sedan ta texten som samlats i klarfil.txt och kopiera in den i Excel och rensa bort html-kod så att vi får fram bara SNI-koderna för de tio företagen.

Det finns många som programmerar i perl – och därför finns det oändliga resurser på nätet. Att googla vad man vill göra samt perl gör att man ofta hittar hjälp med att skriva nya program. Ett tips att börja är Perls egen kurs på nätet: <http://www.perl.com/pub/2008/04/23/a-beginners-introduction-to-perl-510.html>