Exemplu de realizare și exploatare a subrutinelor în PERL

```
use constant LUNGIME => 60;
print "=" x LUNGIME, "\n";
print "-oOo-" x 12, "\n";
print "-" x 50, "\n";
print ">>==<<==" x 8, "\n";

------------------------------------------------------------------------

use constant LUNGIME => 60;
sub deseneaza
     {
          print "-" x LUNGIME, "\n";
     }
deseneaza;
deseneaza;


------------------------------------------------------------------------


use constant LUNGIME => 60;
sub deseneaza
     {
          print $_[0] x $_[1], "\n";
     }
print "Introduceti modelul de desen: ";
$model = <STDIN>;
chomp $model;
print "\n","De cate ori trebuie afisat modelul: ";
$lungime = <STDIN>;
chomp $lungime;
deseneaza ($model, $lungime);


------------------------------------------------------------------------


use constant LUNGIME => 50;
sub deseneaza
     {
          $caracter = shift || "-";
          $contor = shift || LUNGIME;
          print $caracter x $contor, "\n";
     }
print "Introduceti modelul de desen: ";
$model = <STDIN>;
chomp $model;
print "\n","De cate ori trebuie afisat modelul: ";
$lungime = <STDIN>;
chomp $lungime;
deseneaza ($model, $lungime);
```

Commented [A1]: Exemplul pornește de la utilizarea repetată a funcției *print*, aceasta fiind utilizată ori de câte ori este necesară afișarea pe ecran.

Commented [A2]: Primul exemplu introduce subrutina în scopul de a ușura afișarea pe ecran a unor caractere doar prin apelarea numelui subrutinei, ori de câte ori este nevoie. În acest stadiu, în ciuda faptului că nu mai trebuie scrise nenumărate instrucțiuni *print*, o problemă rămâne nerezolvată: subrutina este inflexibilă și afișează întotdeauna același lucru.

Commented [A3]: Un prim pas în scopul de a flexibiliza subrutina constă în inițializarea și apelarea acesteia cu parametri. De aceea, valorile acestor parametri trebuie transferate în interiorul subrutinei prin intermediul unei variabile predefinite de tip *array* @_

Commented [A4]: Acest *array* predefinit (@_) se inițializează doar la apelarea subrutinei. Numărul de componente ale *array*-ului este similar cu numărul și ordinea parametrilor indicați la apelarea subrutinei. În interiorul subrutinei, valorile acestor parametri vor fi exploatate prin intermediul numelui acestui *array* predefinit conform regulilor de manipulare a datelor dintr-o variabile de tip *array*.

Commented [A5]: Acest exemplu îmbunătățește subrutina cu posibilitatea de a afișa pe ecran o soluție, chiar și în cazul în care este apelată fără parametri. Acest fapt dă posibilitatea programatorului să o configureze în așa fel încât rezultatul ei pe ecran, *by default*, să fie și cel mai des folosit.

```
use constant LUNGIME => 50;
sub deseneaza
     {
          %argumente = @_;
          $caracter = $argumente{Model} || "-";
          $contor = $argumente {Contor} || LUNGIME;
          print $caracter x $contor, "\n";
     }
print "Introduceti modelul de desen: ";
$model = <STDIN>;
chomp $model;
print "\n","De cate ori trebuie afisat modelul: ";
$lungime = <STDIN>;
chomp $lungime;
deseneaza (Model => "$model", Contor => $lungime);
```

Commented [A6]: Acest exemplu se comportă identic cu cel anterior, dar folosește alt mod de a prelua parametri. În acest caz se utilizează o variabila *hash* care se interpune între *array*-ul @_ și variabilele interne ale subrutinei. La apelare, parametri trebuie să aibă forma unor componente caracteristice unei variabile de tip *hash*.

Commented [A7]: Această instrucțiune apelează subrutina cu doi parametri. În acord cu tipul unei variabile *hash*, primul parametru este constituit din două elemente: dintr-o cheie (identificator) și o valoare corespunzătoare (*Model* fiind cheia iar *$model* este valoarea corespunzătoare, preluată prin intermediul variabilei respective. *Contor* este ce-a de-a doua cheie căreia îi va corespunde valoarea adusă de variabila *$lungime*). Astfel, *array*-ul @_ se inițializează cu patru componente. În cadrul *array*-ului acestea își pierd identitatea de chei/valori, însă își păstrează ordinea, astfel încât, atunci când sunt transferate din nou într-o variabilă *hash* (%argumente) acestea își recapătă identitatea și *hash*-ul astfel inițializat poate fi folosit.

```
use strict;
use constant LUNGIME => 50;
sub deseneaza
     {
          my %argumente; my $caracter; my $contor;
          %argumente = @_;
          $caracter = $argumente{Model} || "-";
          $contor = $argumente {Contor} || LUNGIME;
          return $caracter x $contor;
     }
my $model; my lungime;
print "Introduceti modelul de desen: ";
$model = <STDIN>;
chomp $model;
print "\n","De cate ori trebuie afisat modelul: ";
$lungime = <STDIN>;
chomp $lungime;
print "\n modelul este: ",deseneaza (Model => "$model", Contor =>
$lungime);
```

Commented [A8]: Ultima versiune a subrutinei introduce funcția *return* care specifică ce variabile (purtătoare de valori) vor fi întoarse de către subrutină.

Commented [A9]: În acest caz utilizarea funcției *print* din programul principal se poate face fără probleme, altfel, în cazul în care *print* exista și în subrutină, apare situația în care două funcții *print* trebuie să împartă simultan accesul la memoria video, ceea ce nu este permis într-o astfel de situație.
De asemenea, modulul *strict* impune o serii de condiții în ceea ce privește declararea tuturor variabilelor de tip local.

De precizat că, ultimul exemplu, în care s-a optimizat subrutina se poate adapta și la exemplul 3 sau 4, exemple care nu folosesc o variabilă *hash* în scopul preluării parametrilor. Acestea sunt de preferat, datorită simplității și mai ales a vitezei de execuție (nu se mai pierde timp în transferul parametrilor de la @_ la *hash*, se pot folosi direct valorile cu care deja s-a inițializat @_).