גירסה 1.00 – 28.9.2003

Perl

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

אנא שלחו תיקונים והערות אל המחבר.

תוכן עניינים

Perl היא שפת סקריפטים שנוצרה ב-1987 על ידי Larry Wall. השפה הינה Open-Source וקיימות גרסאות שלה עבור מערכות הפעלה רבות.

דוגמאות ראשונות

במידה ואנו כותבים סקריפט שרץ תחת UNIX, השורה הראשונה של הסקריפט תהיה המנוע בעזרתו נפענח את הסקריפט. שורת התחלה טיפוסית הינה:

```
#!/usr/bin/perl
```

נניח כי שורה זו מופיעה בכל התוכניות שנכתוב מעתה.

- כאשר נעבוד ב-Windows אין צורך בשורה זו.

שתי שורות נוספות שנכלול תמיד הינן השורות הבאות:

```
use strict;
use warnings;
```

שורות אלו גורמות ל-Perl להציג כל הודעת שגיאה אפשרית וכן לבדוק את הקוד שלנו עבור שגיאות תחביריות, וכך נוכל לאתר שגיאות בקוד יותר בקלות, על ידי כך ש-Perl תדווח לנו על רובן.

דוגמא 1:

התוכנית הקלאסית, המדפיסה את המילים "Hello, World" על המסך תראה ב-Perl כך:

```
use strict;
use warnings;

print "Hello, World";
```

הפקודה print משמשת אותנו כדי להדפיס מידע אל הפלט הסטנדרטי.

הביטוי שבמרכאות הוא הביטוי אותו אנו רוצים להדפיס.

כמו בכל שפה ניתן גם ב-Perl להוסיף הערות לקוד על מנת לתעד אותו.

הערה ב-Perl מתחילה בסימן # והיא נמשכת עד סוף אותה שורה.

נביט כעת בדוגמא נוספת בה נראה מספר פקודות נוספות וכן שימוש בהערות.

דוגמא 2:

```
use strict;
use warnings;

print "What is your name?  ";       # print out the question
my $username;                       # declares the variable
$username = <STDIN>;                # ask for the username
chomp($username);                   # remove new line
print "Hello, $username.\n";        # print out the greeting
```

תוכנית זאת מבקשת מהמשתמש להקיש את שמו, שומרת את שמו במשתנה, ואז מציגה ברכה בה כלול השם שהוקלד.

ניתן לראות שכל הצהרה (statement) מסתיימת ב-; בדומה לשפת C.

המשמעות של <STDIN> היא קריאה מהקלט הסטנדרטי.

chomp היא פונקציה סטנדרטית של Perl המנקה \n מסוף המחרוזת.

ביטויים והצהרות

נגדיר **ביטוי** (expression) בתור כל קטע קוד חוקי של שפת Perl. בדומה ל-C, לכל ביטוי יש ערך, אליו אנו יכולים להתייחס (אבל לא חייבים).

נגדיר **הצהרה** (statement) בתור אפס או יותר ביטויים המסתיימים בנקודה-פסיק.

כל הצהרה, בדומה לביטוי, היא בעלת ערך, וכמו כן יכול להיות לה **אפקט** (side-effect).

למשל במקרה של ההצהרה chomp($username) האפקט הוא קיצוץ התו \n מסוף המחרוזת, והערך המוחזר של ההצהרה הוא כמה תווים קוצצו.

עבודה עם סקלרים

סקלר (Scalar) היא יחידת המידע הבסיסית של Perl. סקלרים הינם מחרוזות, תווים, מספרים ועוד.
דוגמאות לסקלרים: 7, -1, 'word', "Perl" .

<u>מחרוזות</u>

מחרוזת הינה רצף של תווי ASCII.
מחרוזות יכולות להיות בכל אורך ולהכיל אותיות, מספרים, סימנים שאינם אלפאנומריים ועוד.
Perl מכילה מספר דרכים לייצג מחרוזות. נציג אותם כעת.

מחרוזות עם גרש בודד

כאשר אנחנו משתמשים במחרוזות בעלות גרש בודד, המחרוזת מפוענחת כפי שהיא. תווים מיוחדים כגון \n המוכרים גם משפות אחרות, אינם עוברים כל פענוח.
למשל, נביט בביטוי הבא:

```
'i\o';  # The string 'i\o'
```

הביטוי מייצג את המחרוזת בה התו הראשון הוא i, לאחריו מופיע \ ובסיום התו o.

ישנם יוצאי דופן שאינם מתפענחים כפי שהם עקב צורך.
נניח שנרצה לשים גרש בודד כחלק מהמחרוזת. עקרונית לא היה ניתן לעשות זאת, מכיוון שכרגע בודד מסמל את סיום המחרוזת. לפיכך נקבע כי אם נשים לפני גרש בודד את התו \ אז הגרש ייחשב כחלק מהמחרוזת, ולא כמסיים המחרוזת. לדוגמא:

```
'xxx\'xxx';  # xxx, a single-quote character, and then xxx
```

בגלל שלתו \ משמעות מיוחדת, גם אותו צריך לכתוב בדרך מיוחדת.
הדרך לעשות זאת היא על ידי כתיבת שני \ אחד אחרי השני, כך: \\
לדוגמא:

```
'I don\'t think so.';    # Note the ' inside is escaped with \
'a \\ (backslash)';      # The \\ gives us \
```

ניתן להוסיף תווי שורה חדשה לתוך מחרוזת בעלת גרשיים בודדות. נעשה זאת פשוט על ידי פיצול המחרוזת למספר שורות. Perl יוסיף את תווי מעבר השורה כשנלחץ Enter. למשל:

```
'Time to
start a new line.';  # Represents the single string composed of:
                     # 'Time to' followed by a newline, followed by
                     # 'start a new line.'
```

הפונקציה print

כפי שכבר צויין, הפונקציה print משמשת ב-Perl לצורך פלט. נציג פקודה זו כעת על מנת שנוכל להציג פלט על המסך ולראות את תוצאות הפעולות שאנו עושים.

הצורה הבסיסית ביותר של השימוש בפונקציה הינה:

```
print STRING;
```

כאשר STRING זוהי מחרוזת חוקית.

לדוגמא:

```
use strict;
use warnings;

print 'Three \\\'s: "\\\\\"'; # Print first string
print 'xxx\'xxx';             # Print the second
print 'Time to
start a new line.
';    # Print last string, with a newline at the end
```

פלט התוכנית יהיה:

```
Three \'s: "\\\"xxx'xxxTime to
start a new line.
```

נשים לב ש-Perl לא תעבור שורה ללא שנאמר לה לעשות זאת.

מחרוזות בעלות מרכאות

ניתן לייצג מחרוזות קבועות ב-Perl גם באמצעות מרכאות. במקרה זה הטקסט בתוך המחרוזת יעבור ניתוח מסויים לפני שהתוכן הסופי של המחרוזת ייקבע.

הניתוח יתבצע אחרי הסימן \. לתווים רבים ישנה משמעות אחרת אם הם מופיעים אחרי תו זה.
הטבלה הבאה תסכם זאת:

Char	Meaning
\\	an actual, single backslash character
\$	a single $ character
\@	a single @ character
\t	tab
\n	newline
\r	hard return
\f	form feed
\b	backspace
\a	alarm (bell)
\e	escape
\033	character represented by octal value, 033
\x1b	character represented by hexadecimal value, 1b

מספרים

סוג הסקלרים השני הוא ערכים מספריים.
להלן מספר דוגמאות לערכים מספריים חוקיים ב-Perl:

```
42;          # The number 42
12.5;        # A floating point number, twelve and a half
101873.000; # 101,873
.005;        # five thousandths
5E-3;        # same number as previous line
23e-100;     # 23 times 10 to the power of -100 (very small)
2.3E-99;     # The same number as the line above!
23e6;        # 23,000,000
23_000_000; # The same number as line above
             # The underscores are for readability only
```

ניתן לראות שערכים מספריים מיוצגים ב-Perl בצורה דומה לאיך שהם מיוצגים בשפת C++.
כמו כן Perl מציעה אפשרות לשיפור בקריאות של מספרים ארוכים, על ידי הוספת קווים תחתונים לחלוקת המספר, וזאת בלי פגיעה במשמעות שלו.

הדפסת ערכים מספריים

ניתן להדפיס ערכים מספריים בעזרת הפונקציה print. עד כה ראינו שימוש ב-print כאשר אנו מעבירים לה ארגומנט אחד - המחרוזת אותה אנו רוצים להדפיס. במידה ונרצה להדפיס מספר ערכים שונים נשתמש ב-print כאשר אחריה מספר פרמטרים, המופרדים ביניהם על ידי פסיקים.

לדוגמא:

```
print 2E-4, ' ', 9.77E-5, " ", 100.00, " ", 10_181_973, ' ', 9.87E9,
      " ", 86.7E14, "\n";
```

הפלט של שורה זו יהיה:

```
0.0002 9.77e-05 100 10181973 9870000000 8.67e+15
```

משתנים סקלריים

ניתן לאכסן נתונים סקלריים בתוך משתנים סקלריים. משתנה סקלרי הוא **מזהה** המתחיל בתו $, לאחריו אות או קו תחתון, ולאחריה רצף של אותיות ומספרים. שמות משתנים הם case-sensitive, כלומר המשתנה $aBc שונה מהמשתנה $abc. כמו כן Perl מגבילה את אורך שם המשתנה ל-255 תווים.

שמות משתנים חוקיים: $abc, $_ab, $a1
שמות משתנים בלתי חוקיים: $123a, $abc%.

הגדרת משתנה נעשית על ידי הפונקציה my. פונקציה זו מצהירה על משתנה מקומי.

דוגמא:

```
use strict;
use warnings;

my $my_var = "Hello";
$my_var = 4.5;
my $my_var2 = $my_var + 5;
print $my_var2
```

נשים לב לכמה נקודות:

- ניתן להגדיר משתנים בכל קטע של הקוד.
- סוג הסקלר השמור במשתנה יכול להשתנות. בתחילה המשתנה $my_var מכיל מחרוזת, ולאחר מכן הוא מכיל מספר.
- האופרטור = הוא אופרטור השמה, שמשמעותו לשים ערך בתוך המשתנה.
- ניתן להציב משתנה, או משתנה עליו מופעלת פעולה, לתוך משתנה אחר.

ניתוח משתנים

קודם הכרנו מספר סוגי מחרוזות - מחרוזות החתומות על ידי גרשיים ומחרוזות החסומות על ידי מרכאות כאשר אנו כותבים שמות של משתנים כחלק ממחרוזת, אנו שמים לב לחשיבות נוספת למרכאות בהן בחרנו להשתמש.

במילה והשתמשנו במרכאות שם המשתנה יעבור פיענוח ובמחרוזת יוצב ערכו של המשתנה. לדוגמא:

```
use strict;
use warnings;

my $friend = 'David';
my $greeting = "Hello, $friend!\n";
# $greeting contains "Hello, David!\n"

my $cost = 25.52;
my $statement = "Please pay \$$cost.\n";
# $statement contains "Please pay $25.52.\n"

print "$greeting";
print "$statement";
```

פלט התוכנית יהיה:

```
Hello, David!
Please pay $25.52.
```

נציג דוגמא נוספת:

```
use strict;
use warnings;

my $owner  = 'Rotem';
my $dog    = 'Zeus';
my $amount = 12.5;
my $what   = 'dog food';

print "${owner}'s dog, $dog, ate $amount kilos of $what.\n";
```

שם המשתנה owner מוקף בסוגריים מסולסלות. הסיבה לכך היא שהגרש המופיע אחרי שם המשתנה עלול לבלבל את Perl. כדי למנוע בעיות נסמן בעזרת הסוגריים המסולסלות את שם המשתנה.

כמו כן, ניתן לראות בדוגמא זו מחרוזת אחת בה שילבנו מספר משתנים. זוהי שיטה נפוצה בעת השימוש ב-Perl.

השיטה של עטיפת המשתנה בסוגריים מסולסלות נפוצה במספר מקרים. אחד מהם הוא המקרה לעיל. מקרה נוסף הוא מקרה בו מיד אחרי המשתנה נכתוב אותיות נוספות - ללא רווחים. Perl לא תוכל להגיד האם זהו חלק משם המשתנה או לא, ולכן גם במקרה כזה נצטרך לעטוף את המשתנה בסוגריים מסולסלות. למשל:

```
use strict;
use warnings;

my $this_data = "Something";
my $that_data = "Something Else ";

print "_$this_data_, or $that_datawill do\n";
# INVALID: actually refers to the scalars $this_data_
#          and $that_datawill

print "_${this_data}_, or ${that_data}will do\n";
# CORRECT: refers to $this_data and $that_data,
#          using curly braces to make it clear
```

משתנים לא מאותחלים

אם נגדיר משתנה בלא לאתחל אותו, הוא יכיל ערך מיוחד - undef, המציין את העובדה כי המשתנה איננו מאותחל. לדוגמא נביט בקטע הקוד הבא:

```
use strict;
use warnings;

my $anEmptyVar;
print $anEmptyVar;
print "Hey";
```

הפלט של קטע זה יהיה:

```
Use of uninitialized value in print at line 2.
Hey
```

Perl מזהה שאנו משתמשים במשתנה בלתי מאותחל ומתריעה על כך.

נשים לב שלמרות זאת ריצת התוכנית לא נעצרה בגלל שהשתמשנו במשתנה בלתי מאותחל. Perl סלחנית לגבי השימוש במשתנים בלתי מאותחלים. אם נשתמש במשתנה בלתי מאותחל כמחרוזת, הוא ייחשב כמחרוזת ריקה, ואם נשתמש בו כמספר, הוא ייחשב למספר 0.

עם זאת, מכיוון שאמרנו ל-Perl להציג כל הערה אפשרית על ידי use warnings, אנו מקבלים התרעה בכל פעם בה נשתמש במשתנה בלתי מאותחל.

אפשרות שימושית ש-Perl מספקת היא בדיקה האם משתנה מאותחל או לא. הבדיקה נעשית בעזרת הפונקציה defined. דוגמא לשימוש בפונקציה:

```
use strict;
use warnings;

my $anEmptyVar;
my $NotEmpty = "a value";
print defined $anEmptyVar;
print defined $NotEmpty;
```

הפונקציה defined מחזירה 1 במידה והמשתנה מאותחל, או מחרוזת ריקה "" במידה והוא איננו מאותחל.

ניתן לגרום למשתנים שהיו מאותחלים להפוך שוב למשתנים לא מאותחלים, וזאת על ידי הצבת undef לתוכם, למשל:

```
$myvar = undef;
```

אופרטורים

Perl מציעה מגוון של אופרטורים שבעזרתם ניתן לבצע מניפולצות שונות על סקלרים.
נציג את האופרטורים השונים לפי קטגוריות:

אופרטורים מתמטיים

Perl תומכת באופרטורים מתמטיים בדומה לשפות עיליות אחרות. היא תומכת בסדר פעולות החשבון, ובפעולות כגון מודולו וחזקה.
התוכנית הבאה תדגים פעולות אריתמטיות שונות הניתנות לביצוע ב-Perl:

```
use strict;
use warnings;

my $a = 4 + 7 * 2;              # a = 18
my $b = $a - 3;                 # b = 15
my $c = $a * ($b - 5);          # c = 180
my $d = 2 ** 10;                # d = 1024
my $e = 59 % 5;                 # e = 59 mod 5 = 4

print "a = $a\tb = $b\tc = $c\td = $d\te = $e\n";
```

בנוסף Perl תומכת באופרטורים =- ,=+ וכו', בדומה לשפת C/C++, לדוגמא:

```
my $abc = 5;
$abc += 4;
```

ניתן להשתמש בצורת כתיבה מקוצרת זו עבור כל אופרטור בינארי הקיים בשפה.

אופרטורים של השוואה

ב-Perl הערכים "", 0 ו-undef מוגדרים כ-False וכל ביטוי אחר מוגדר כ-True. (הגדרה מדוייקת יותר תינתן בהמשך).

ב-Perl ניתן להשתמש באופרטורים גם על ידי סימני האופרטורים >, < וכו' וגם על ידי מילים בעלות אותה משמעות.

הטבלה הבאה מרכזת את אופרטורי ההשוואה השונים של השפה.

אנו מניחים בטבלה כי אנו מריצים את הביטוי $left <OP> $right, כאשר OP היא הפעולה הנדונה בכל שורה בטבלה.

האופרטור	סימון	סימון מילולי	ערך מוחזר
קטן מ-	<	lt	1 אמ"מ $left קטן מ-$right
קטן או שווה	<=	le	1 אמ"מ $left קטן או שווה מ-$right
גדול מ-	>	gt	1 אמ"מ $left גדול מ-$right
גדול או שווה	>=	ge	1 אמ"מ $left גדול או שווה מ-$right
שווה	==	eq	1 אמ"מ $left שווה ל-$right
שונה	!=	ne	1 אמ"מ $left שונה מ-$right
השוואה	<=>	cmp	-1 אמ"מ $left קטן מ-$right, 0 אמ"מ $left שווה ל-$right 1 אמ"מ $left גדול מ-$right

דוגמא:

```
use strict;
use warnings;


my $a = 5; my $b = 50;
$a < $b;                # evaluates to 1
$a >= $b;               # evaluates to ""
$a <=> $b;              # evaluates to -1

my $c = "hello"; my $d = "there";
$d cmp $c;              # evaluates to 1
$d ge  $c;              # evaluates to 1
$c cmp "hello";         # evaluates to ""
```

אופרטורי הוספה והפחתה

אופרטורי ההוספה וההפחתה ב-Perl עובדים בצורה זהה לאופרטורים אלו בשפת C/C++, ולכן נסתפק בהצגת דוגמא לשימוש בהם:

```
use strict;
use warnings;

my $abc = 5;
my $efg = $abc-- + 5;       # $abc is now 4, but $efg is 10
my $hij = ++$efg - --$abc;  # $efg is 11, $abc is 3, $hij is 8
```

אופרטורים הפועלים על מחרוזות

Perl כוללת מספר אופרטורים לפעולה על מחרוזות.

האופרטור . הוא אופרטור שירשור המחרוזות, והאופרטור x הוא אופרטור שכפול המחרוזות.

נדגים שימוש בהם:

```
use strict;
use warnings;

my $greet = "Hi! ";
my $longGreet  = $greet x 3;         # $longGreet is "Hi! Hi! Hi! "
my $hi = $longGreet . "To You :).";  # $hi is "Hi! Hi! Hi! To You
:)."

print "$hi";
```

ניתן לבצע שירשור מחרוזות בצורה מקוצרת באופן הבא:

```
my $greet = "Hi! ";
$greet  .= "Everyone\n";
$greet  = $greet . "Everyone\n"; # Does the same operation
                                 # as the line above
```

הפונקציה printf

עד כה ראינו פונקציה אחת לפלט ב-Perl והיא הפונקציה print.

פונקציה נוספת לפלט היא הפונקציה printf, הדומה מאוד לפונקציה זו בשפת C.

דוגמא לשימוש בפונקציה:

```
use strict;
my $str  = "Howdy, ";
my $name = "Joe.\n";
print $str, $name;    # Prints out: Howdy, Joe.<NEWLINE>
my $f = 3e-1;
printf "%2.3f\n", $f; # Prints out: 0.300<NEWLINE>
```

מערכים ורשימות

אחד מהטיפוסים הבסיסיים החשובים ביותר של Perl הוא מערכים.
מערכים הם רשימות של סקלרים.
בדומה לשפת C, ניתן להתייחס אל כל אחד מהאיברים במערך על ידי אינדקס ולגשת אליו ישירות.
בניגוד לשפת C הגודל של המערכים ב-Perl איננו קבוע. גודל המערך יכול לגדול כאשר מכניסים אליו איברים חדשים, ולקטון כאשר מוציאים ממנו איברים. בצורה זו Perl מספקת למתכנת מערכים המספקים גם את הפונקציונליות של רשימה וגם של מערך.
חסרון אחד במערכים של Perl הוא שהביצועים שלהם עלולים להיות גרועים במידה ומכניסים אליהם מספר גדול של איברים ברצף (כך שגודלם כל הזמן גדל). במקרה כזה Perl מאפשרת גם להגדיר מערכים בעלי גודל קבוע מראש, ולחסוך את זמן ההגדלה הדינאמי.

מערכים ב-Perl מכילים סקלרים, אולם הם אינם חייבים להכיל את אותו סוג סקלרים בכל התאים. אין מניעה שבמערך חלק מהאיברים יהיו מספרים וחלק יהיו מחרוזות.
נשים לב להבדל משמעות בין המילים מערך לרשימה. **מערך** הוא משתנה המכיל קבוצת איברים. **רשימה** יכולה להיות אוסף קבוע של איברים.

<u>רשימות קבועות</u>

Perl מציעה שתי שיטות להגדיר רשימות. דרך אחת היא בעזרת האופרטור סוגריים, כאשר בתוכו נשים סקלרים שיהוו את אברי הרשימה.
הדרך השניה היא באמצעות האופרטור qw ולאחריו תו מפסיק. הרשימה תכיל את כל המחרוזות שיופיעו בהמשך הקוד עד ההופעה הראשונה של התו המפסיק. היתרון של שימוש באופרטור זה הוא במידה ויש לנו מחרוזות רבות, ואנחנו רוצים לחסוך את הצורך לעטוף כל אחת מהן במרכאות.
לדוגמא:

```
();                    # this list has no elements; the empty list
qw//;                  # another empty list
("a", "b", "c",
  1,  2,  3);          # a list with six elements
qw/hello world
  how are you today/; # another list with six elements
```

אופרטור נוסף הקשור למחרוזות הוא האופרטור ..

במידה ויש לנו שני ערכים סקלרים ואנו רוצים להכניס לרשימה את כל הסקלרים הנמצאים ביניהם נוכל להשתמש באופרטור זה כדי ליצור את הרשימה. לדוגמא:

```
(1 .. 100);     # a list of 100 elements: the numbers from 1 to 100
('A' .. 'Z');   # a list of 26 elements: the uppercase letters From A
to Z
('01' .. '31'); # a list of 31 elements: all possible days of a month
                #    with leading zeros on the single digit days
```

משתנים מסוג מערך

כל משתנה ב-Perl מתחיל עם תו מיוחד המזהה את הסוג של המשתנה. ראינו כבר משתנים סקלריים המתחילים תמיד בתו $. באופן דומה, כל משתנה מסוג מערך מתחיל בסימון @. חוקי בחירת השם זהים לאלו של בחירת שמות לסקלריים.

ניתן לשים ערכים בתוך מערכים בדומה לדרך בה שמים ערכים בתוך משתנים סקלרים בעזרת האופרטור שווה (=), כאשר בצד הימני מופיעה מחרוזת.

לדוגמא:

```
use strict;
use warnings;

my @stuff  = qw/a  b  c/;            # @stuff a three element list
my @things = (1, 2, 3, 4);           # @things is a four element list
my $oneThing = "all alone";
my @allOfIt = (@stuff, $oneThing,
               @things);             # @allOfIt has 8 elements!
```

נשים לב למשהו חדש שניתן לראות בדוגמא זו. כאשר אנו משתמשים באופרטור סוגריים, אנו יכולים לשים סקלרים וכן אנו יכולים לשים משתנים אחרים מסוג רשימה. במקרה זה Perl תיצור רשימה חדשה שהיא שרשור של הרשימות והסקלרים השונים המופיעים בין הסוגריים.

סקלרים הקשורים למערכים

בכל פעם שאנו מגדירים מערך מוגדרים ביחד איתו סט של סקלרים המתאימים אליו, והמשתנים עם השתנות המערך.

נניח שנתון לנו מערך בשם @array, בעל n איברים.

מוגדרים אוטומטית עבורו הסקלרים $array[0], $array[1], ..., $array[n-1] המכילים את האיבר הראשון, השני וכו' של המערך. אלו משתנים סקלרים איתם אנו יכולים לעשות כל פעולה שמותרת על משתנה סקלרי רגיל - כולל שינוי ערכם. שינוי משתנים אלו ישנה את האיברים עצמם במערך.

סקלר נוסף שמוגדר הוא $#array. משתנה זה מכיל את האינדקס של האיבר האחרון במערך.

במילים אחרות, $array[$#array] הינו תמיד האיבר האחרון במערך. אורך המערך עצמו הינו $#array + 1. שוב, ניתן להתייחס אל משתנה זה כמו אל כל סקלר. ניתן לשנות את ערכו או לקרוא אותו. עם זאת, יש לשמור תמיד שערכו יהיה 1- או יותר. כאשר אנו משנים את ערך משתנה זה גודל המערך עצמו משתנה. Perl מקצה זיכרון למערך בהתאם לגודלו של משתנה זה. לפיכך אם ידוע כי אנו הולכים להכניס משתנים רבים למערך נגדיל ראשית את משתנה זה לערך גדול, במקום לתת ל-Perl לעשות זאת אוטומטית פעמים רבות במהלך ההכנסה.

דוגמא:

```
my @someStuff = qw/Hello and
                   welcome/;     # @someStuff: an array of 3 elements
$#someStuff = 0;                 # @someStuff now is simply ("Hello")
$someStuff[1] = "Joe";           # Now @someStuff is ("Hello", "Joe")
$#someStuff  = -1;               # @someStuff is now empty
@someStuff   = ();               # does same thing as previous line
```

<u>מניפולציות על מערכים</u>

חתיכת מערכים

לעתים נרצה ליצור מערך חדש המבוסס על תת קבוצה של איברים ממערך אחר. כדי לעשות זאת נשתמש באפשרות לחתוך מערך. החתך מוגדר על ידי קבוצת מספרים שלמים, המציינים את האינדקסים אותה אנו רוצים לקחת מהמערך.

לדוגמא:

```
use strict;
use warnings;

my @stuff = qw/everybody wants a rock/;
my @rock  = @stuff[1 .. $#stuff];      # @rock is qw/wants a rock/
my @want  = @stuff[ 0 .. 1];           # @want is qw/everybody wants/
@rock     = @stuff[0, $#stuff];        # @rock is qw/everybody rock/
```

כאשר השתמשנו באופרטור .. הגדרנו תחום במערך אותו אנו רוצים לחתוך. בעזרת פסיקים בחרנו איברים ספציפיים בהם אנו מעוניינים.

פונקציות

Perl מספקת פונקציות הפועלות על מערכים. נציג כעת כמה מהן.

<u>מערך כמחסנית</u>

ניתן לחשוב על מחסנית כעל מערך בגודל לא חסום. צורת חשיבה זו גרמה להוספת שתי פונקציות ל-Perl - pop ו-push, לשליפת והוספת איברים מהמערך.

הנה דוגמא לשימוש בפונקציות אלו:

```
use strict;
use warnings;

my @stack;
push(@stack, 7, 6, "go");   # @stack is now qw/7 6 go/
my $action = pop @stack;    # $action is "go", @stack is (7, 6)
my $value = pop(@stack) +
            pop(@stack);    # value is 6 + 7 = 13, @stack is empty
```

מערך כתור

ניתן לממש גם תור בעזרת מערך. נעשה זאת בצורה הבאה:

נשתמש בפונקציות unshift ו-pop, כאשר הפונקציה unshift משמשת כ-enqueue והפונקציה pop משמשת כ-dequeue.

דוגמא:

```
use strict;
use warnings;

my @queue;
unshift (@queue, "Customer 1"); # @queue is now ("Customer 1")
unshift (@queue, "Customer 2");
            # @queue is now ("Customer 2" "Customer 1")
unshift (@queue, "Customer 3");
            # @queue is now ("Customer 3" "Customer 2" "Customer 1")

my $item = pop(@queue);
            # @queue is now ("Customer 3" "Customer 2")
print "Servicing $item\n";      # prints:  Servicing Customer 1\n
$item = pop(@queue);            # @queue is now ("Customer 3")
print "Servicing $item\n";      # prints:  Servicing Customer 2\n
```

אם זאת, יש להזהר מהשימוש בפונקציה unshift במובן של תור. הדוגמא לעיל נכונה מכיוון ש-unshift כל פעם הוסיפה איבר בתחילת המערך, ו-pop הוציאה איבר מסופו. אולם יש לזכור תכונה של unshift והיא שפונקציה זו מכניסה איברים למערך בסדר בה הם נכתבים, דבר הנוגד את אופי התור.

נביט בדוגמא הבאה:

```
use strict;
use warnings;

my @notAqueue;
unshift(@notAqueue, "Customer 0", "Customer 1");
# @queue is now ("Customer 0", "Customer 1")

unshift (@notAqueue, "Customer 2");
# @queue is now ("Customer 2", "Customer 0", "Customer 1")
```

התוצאה כאשר אנו משתמשים ב-unshift עם יותר מפרמטר אחד איננה תור. לכן - יש להזהר כאשר משתמשים בה למימוש תור.

ההקשר - סקלר מול רשימה

כאשר אנו מתכנתים ב-Perl אנו משתמשים לעתים בסקלרים ולעיתים ברשימות. כמו כן, יתכנו מקרים, הוראות לשפה, בהם נוכל להשתמש הן בסקלרים והן ברשימות, ותוצאת הפעולה תשתנה בהתאם לסוג הנתונים בו נשתמש. סוג הנתונים עליו פועלת פעולה הוא **ההקשר** של הפעולה.

לדוגמא:

```
use strict;
use warnings;

my @things = qw/a few of my favorite/;
my $count  = @things;              # $count is 5
my @moreThings = @things;          # @moreThings is same as @things
```

כאשר אנו מכניסים את ערכה של רשימה אל תוך סקלר, הסקלר מקבל את מספר האיברים של הרשימה.

כאשר אנו מכניסים את ערכה של רשימה אל תוך רשימה אחרת, תוכן הרשימה מועתק אל הרשימ השניה.

פענוח מערכים

בדומה לסקלרים, אם נשים מערך בין מרכאות תוכנו יפוענח ויוצב במקומו. Perl תחבר בין כל האיברים למחרוזת ארוכה אחת, כאשר בין כל שני איברים יפריד רווח.

דוגמא:

```
use strict;
use warnings;

my @saying = qw/these are a few of my favorite/;
my $statement = "@saying things.\n";
         # $statement is "these are a few of my favorite things.\n"
my $stuff = "@saying[0 .. 1] @saying[$#saying - 1, $#saying]
things.\n"
         # $stuff is "these are my favorite things.\n"
```

הדפסת מערכים

ניתן להדפיס מערכים על ידי שימוש בפונקציה print.

נוכל לאמר ל-print ישירות להדפיס את המערך. במקרה זה איברי המערך יודפסו אחד אחרי השני ברצף.

נוכל לעטוף את המערך במרכאות, ואז הוא ראשית יפוענח למחרוזת, ואז יודפס.

כמו כן, נוכל להתייחס למערך כאל סקלר.

הדוגמא הבאה תבהיר את האפשרויות השונות להדפסת מערך:

```
use strict;
use warnings;

print @food;      # By itself
print "@food";    # Embedded in double quotes
print @food."";   # In a scalar context
```

הפלט של קטע זה יהיה:

```
Iwantfoodnow:)
I want food now :)
5
```

מבני בקרה

מבני בקרה הם אחד החלקים החשובים ביותר של כל שפה. נציג את מבני הבקרה ש-Perl תומכת בהן.

בלוק

בדומה לשפת C/C++ ניתן לחלק את הקוד שלנו לבלוקים. בלוק הוא קטע העטוף בסוגריים מסולסלות.
בלוק מתחיל ב-{ ונגמר ב-}.
לדוגמא:

```
use strict;
use warnings;
{
      my $var;
      Statement;
      Statement;
      Statement;
}
```

בדומה לבלוקים בשפת C, משתנים המוגדרים בתוך בלוק קיימים רק בתוכו.

תזכורת + הרחבה - ערכים בוליאנים ב-Perl

כל ביטוי ב-Perl הוא אמת, מלבד הערכים הבאים:

- המחרוזות "0" ו-"", או כל ביטוי המתפענח לאחת משתי מחרוזות אלו.
- כל ביטוי מתמטי שערכו הוא 0.
- כל ביטוי שערכו הוא undef.

דוגמאות:

ערך	סוג	תוצאה
0	number	false
0.0	number	false
0.0000	number	false
""	string	false
"0"	string	false
"0.0"	string	true
undef	N/A	false
42 - (6 * 7)	number	false
"0.0" + 0.0	number	false
"foo"	string	true

if / unless

משפט if מתפקד ב-Perl בדומה למשפט זה בשפת C/C++. להלן התחביר של משפט if בשפת Perl:

```
if (expression)
{
      Statement1;
      Statement2;
      Statement3;
}
elsif (another_expression)
{
      Expression_Elseif_Statement1;
      Expression_Elseif_Statement2;
      Expression_Elseif_Statement3;
}
else
{
      Else_Statement1;
      Else_Statement2;
      Else_Statement3;
}
```

בדומה לשפת C, החלקים elsif ו-else הם אופציונליים בלבד.

בניגוד לשפת C, הסוגריים המסולסלות הן חובה אחרי כל אחד מהמילים if, elsif, else, אפילו אם נרצה לכתוב הצהרה אחת בלבד.

הביטוי unless זהה לביטוי if, מלבד העובדה שהבלוק שלו מתבצע רק אם התנאי אינו מתקיים. במילים אחרות - הביטוי unless (expression) {}; זהה לביטוי if (!expression) {};.

while/until

הביטוי while זהה לביטוי while של שפת C. תחבירו:

```
while (expression)
{
      While_Statement1;
      While_Statement2;
      While_Statement3;
}
```

הביטוי until(expression) זהה לביטוי while(!expression).

do... while/until

הביטוי do... while דומה לביטוי while, מלבד העובדה שהקוד רץ פעם אחת לפחות, ורק אז נבדק תנאי הלולאה.

תחביר:

```
do
{
      DoWhile_Statement;
      DoWhile_Statement;
      DoWhile_Statement;
} while (expression);
```

גם כאן, הביטוי until(expression) זהה לביטוי while(!expression).

לולאת for

לולאת ה-for עובדת ב-Perl בצורה זהה ללולאת for בשפת C.

תחביר לולאת ה-for הינו:

```
for(Initial_Statement; expression; Increment_Statement)
{
      For_Statement;
      For_Statement;
      For_Statement;
}
```

לולאת foreach

לולאת foreach מקבלת סקלר, רשימה ובלוק. בכל איטרציה של הלולאה הסקלר מקבל את ערכו של אחד מאברי הרשימה, ועליו מתבצע הבלוק.

לדוגמא:

```
use strict;
use warnings;

my @collection = qw/hat shoes shirts shorts/;
foreach my $item (@collection)
{
      print "$item\n";
}
```

נשים לב שמותר לנו להגדיר משתנה בתוך הצהרת ה-foreach. משתנה זה קיים רק לאורך ריצת הלולאה.

מערכים אסוצייאטיבים / Hash Tables

מערכים אסוציאטיבים הם סוג נתונים עיקרי שלישי ב-Perl, לצד סקלרים ומערכים. הם למעשה טבלאות ערבול, אשר Perl מציעה למתכנת אותם כחלק בסיסי של השפה, במקום שיהא עליו לתכנת HashTable בעצמו.

לעתים רבות נכנה את המערכים האסוציטיביים פשוט בשם hash.

משתנים מסוג מערך אסוציאטיבי

כפי שראינו, לכל סוג משתנים בשפת Perl יש קידומת משלו. הקידומת של מערכים אסוציטיביים הינה התו %. הגישה אל מערכים אסוציטיביים נעשית באופן דומה לגישה אל מערכים דינמיים רגילים, אולם במקרה של מערכים אסוציטיביים איננו ניגשים אל תוכן התאים בעזרת אינדקס מספרי, אלא בעזרת כל סקלר כלשהו. הגישה עצמה נעשית על ידי סוגריים מסולסלים ולא על ידי סוגריים מרובעים המאפיינים מערכים רגילים. דוגמא:

```
use strict;
use warnings;

my %table;
$table{'schmoe'} = 'joe';
$table{7.5}  = 2.6;
```

בדוגמא זו יצרנו מערך אסוציאטיבי בעל שתי כניסות: כניסה בשם 'schmoe' שערכה 'joe' וכניסה 7.5 שערכה 2.6.

כפי שניתן לבצע מניפולציות על הסקלרים שהם אברי המערך, כך ניתן גם על hash.

נביט בדוגמא הבאה, הבאה כהמשך לדוגמא לעיל:

```
print "$table{'schmoe'}\n";    # outputs "joe\n"
--$table{7.5};                 # $table{7.5} now contains 1.6
```

תכונה נוספת של hash הינה שהם יכולים להתפרש בהקשר של רשימה. במקרה כזה כל האיברים בעלי האינדקסים הזוגיים של הרשימה יהיו המפתחות של ה-hash, והאיברים בעלי האינדקסים האי-הזוגיים יהיו הערכים.

לדוגמא:

```
my @tableListed = %table;  # @tableListed is qw/schmoe joe 7.5 1.6/
```

אם ננסה לשים מערך אסוציטיבי בהקשר של סקלר, אזי תוכן הסקלר יהיה undef אם המערך האסוציאטיבי ריק מאיברים, או true אם יש בו.

קבועי hash

קבועי hash לא קיימים בפני עצמם בשפה, אולם אנו יכולים לנצל את הקשר הרשימה על מנת לייצר קבועי hash. כדי להגדיר קבוע hash נגדיר פשוט רשימה שבה כל שני איברים יהוו צמד מפתח+ערך בטבלת ה-hash. לדוגמא:

```
use strict;
my %table = qw/schmoe joe 7.5 1.6/;
```

קוד זה ייצר את אותו מערך אסוציאטיבי שראינו קודם.

פונקציות הפועלות על מערכים אסוציאטיביים

כל הפונקציות הפועלות על מערכים יפעלו גם על מערכים אסוציאטיביים. כפי שראינו, המערך האסוציאטיבי יתורגם למערך המכיל זוגות של איברים, ואז תופעל הפעולה המבוקשת על מערך זה.

מלבד הפונקציות המיועדות למערכים, ישנן גם פונקציות הפועלות במיוחד על מערכים אסוציטיביים, אותן נציג כעת.

Keys, Values

כאשר אנו מתייחסים אל hash בהקשר של רשימה, perl מספקת לנו רשימה של זוגות.
אם זאת, לפעמים נרצה רשימה של המפתחות בלבד, או של הערכים.
Perl מאפשרת לנו לעשות זאת בעזרת שתי פונקציות: keys ו-values:

```
use strict;
my %table = qw/schmoe joe smith john simpson bart/;
my @lastNames  = keys %table;    # @lastNames is: qw/schmoe smith
simpson/
my @firstNames = values %table;  # @firstNames is: qw/joe john bart/
```

Each

הפונקציה Each שימושית כדי לעבור על איברי טבלת ערבול אחד אחרי השני. הפונקציה מחזירה כל פעם זוג של מפתח+ערך, ומאפשרת מעבר על כל איברי ה-hash. לאחר שהשתמשנו בה על מנת לעבור על כל איברי המערך, היא מחזירה undef.

ניתן להשתמש בה למשל בצורה הבאה:

```
use strict;
my %table = qw/schmoe joe smith john simpson bart/;
my($key, $value);  # Declare two variables at once
while ( ($key, $value) = each(%table) )
{
    # Do some processing on $key and $value
}
```

יש להיזהר ולשים לב שכל שינוי של ה-hash "יתאחל" את Each ויגרום לה להתחיל מחדש במעבר על אברי המערך.

ביטויים רגולריים

(הערה: קטע זה הינו חלק ממסמך מורחב יותר בנושא המופיע באתר שלי)

ביטויים רגולריים (Regular Expressions) הם דרך נוחה כדי לתאר תבניות מורכבות בתוך טקסט.

אנחנו יכולים להיעזר בהם כדי לאתר תבניות בתוך טקסט. לאחר שאיתרנו את התבניות אנו מסוגלים להחליפן באחרות, ולבצע מניפולציות על הטקסט.

ביטויים רגולריים הם כלי רב עוצמה המאפשר לפתור בעיות רבות הקשורות לניתוח מחרוזות במהירות.

כאשר נדגים ביטויים רגולריים, וכאשר נשתמש בפונקציות הקשורות לביטויים רגולריים, נשתמש במונחים **טקסט** (text) **ותבנית** (pattern).

טקסט זהו הטקסט בו אנחנו מחפשים מחרוזת או מחרוזות כלשהן. זאת המחרוזת אותה אנו רוצים לנתח.

בדרך כלל נקבל את הטקסט מהמשתמש בתוכנית שלנו.

התבנית היא ביטוי המציין את התבנית של תת המחרוזת אותה אנו מחפשים בטקסט.

כאשר אנחנו מוצאים תת מחרוזת המתאימה לתבנית בתוך הטקסט, אנחנו אומרים כי מצאנו **התאמה** (match).

נציג כעת מעט מהתיאוריה של ביטויים רגולריים, ולאחר מכן נראה כיצד אנו משתמשים בהם ב-Perl.

תחביר בסיסי

התבנית הפשוטה ביותר שתשמש כביטוי רגולרי היא אות, או רצף של אותיות.

כל מופע של האות/האותיות ייחשב כהתאמה.

למשל, עבור התבנית `"a"`, סומנו ההתאמות שימצאו בטקסט הבא:

M**a**ry h**a**d **a** little l**a**mb.
And everywhere th**a**t M**a**ry
went, the l**a**mb w**a**s sure
to go.

נשים לב שאותיות קטנות אינן שוות לאותיות גדולות. עבור התבנית "a" לא מצאנו את האות הגדולה A שבתחילת המילה And.

עבור התבנית `"Mary"` ימצאו בטקסט ההתאמות הבאות:

Mary had a little lamb.
And everywhere that **Mary**
went, the lamb was sure
to go.

הסימן ^ מסמל "מתחיל ב" – לדוגמא, התבנית `"^The"` תמצא התאמה בכל מחרוזת המתחילה במילה The.

הסימן $ מסמל "מסתתים ב" – לדוגמא, התבנית `"final$"` תמצא התאמה בכל מחרוזת שתסתיים במילה final.

ניתן לשלב בין סימנים אלו. התבנית `"^abc$"` תמצא כל מחרוזת המתחילה ומסתיימת ב-abc. זוהי יכולה להיות רק המחרוזת "abc" עצמה.

תווי בריחה: נחפש את התבנית `".*"` במחרוזת הבאה:

Special characters must be escaped.*

ההתאמה שנקבל תהיה כל המחרוזת. הסיבה: . ו-* הם תווים מיוחדים, בדומה ל-^ ול-$. כאשר אנו רוצים לחפש תווים אלו בטקסט, עלינו להשתמש בתו בריחה מיוחד שהוא \ כדי לומר שאנו מתכוונים לתווים אלו כתווים, ולא כאל סימנים מיוחדים שיש לפרשם.

נחפש כעת באותה מחרוזת את התבנית `"\.\*"` ונראה מה תהיה ההתאמה:

Special characters must be escaped.*

רק שני התווים בסוף המחרוזת מהווים הפעם את ההתאמה.

התו . (נקודה) משמש כ-wildcard בתבניות. קוראים המכירים את מערכת ההפעלה DOS יכולים לזהות אותו עם התו ? ב-DOS. משמעותו – "כל תו".

למשל, עבור התבנית `".a"` נסמן את ההתאמות בטקסט הבא:

Mary **ha**d **a** little **la**mb.
And everywhere t**ha**t **Ma**ry
went, the **la**mb **wa**s sure
to go.

כאשר אנו מסתכלים על ביטוי רגולרי, כל אות בו מכונה אטום. נראה בהמשך אופרטורים הפועלים על אטום. כדי להגדיר אטום בן יותר מאות אחת, נשתמש בסוגריים מעוגלות.

לדוגמא עבור `"(Mary)( )(had)"` נקבל את ההתאמה הבאה:

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.

ניתן להגדיר מחלקות תווים. במקום לדרוש שאות מסוימת אחת תופיע, נוכל לדרוש שאות אחת מבין קבוצה שנגדיר היא זו שתופיע. למשל, התבנית `"[abc]"` תתאים לכל מחרוזת הכוללת אחת מהאותיות הקטנות a, b או c. ניתן להגדיר טווח של אותיות, על ידי כתיבת האות הראשונה, לאחריה מקף, ואז את האות האחרונה בטווח. התבנית `"[A-Z]"` תתאים לכל אות גדולה באלף בית האנגלי. דוגמא: עבור התבנית `"[a-z]a"` נקבל את ההתאמה:

Mary **ha**d a little **la**mb.
And everywhere t**ha**t Mary
went, the **la**mb **wa**s sure
to go.

הסימן ^ מציין לרוב התחלה של מחרוזת, אולם כאשר הוא מופיע בתוך מחלקת תווים הוא מקבל משמעות אחרת – שלילה. הכוונה של ^ בראש מחלקת תווים היא הפיכת משמעות המחלקה – כלומר – "כל תו שאינו מופיע במחלקת התווים". לדוגמא, עבור התבנית `"[^a-z]a"` נקבל את ההתאמה הבאה:

Mary had **a** little lamb.
And everywhere that **Ma**ry
went, the lamb was sure
to go.

<u>חזרות</u>

התווים "*", "+" ו-"?" מציינים תו/תווים החוזרים על עצמם מספר פעמים.

"*" – התו חוזר אפס פעמים או יותר.

"+" – התו חוזר פעם אחת או יותר.

"?" – התו מופיע אפס פעמים או פעם אחת בלבד.

דוגמאות:

- התבנית `"ab*"` מתאימה מחרוזות שיש בהן a, ולאחריו אפס או יותר b-ים: "a", "ab", "abbb" וכו'.
- התבנית `"ab+"` מתאימה מחרוזות שיש בהן a, ולאחריו לפחות b אחד: "ab", "abbb" וכו'.
- התבנית `"ab?"` מתאימה מחרוזות שיש בהן a, ולאחריו ייתכן שיש b או לא.

Perl וביטויים רגולריים

ראשית נציין כי שפת הביטויים הרגולריים כוללת חוקים נוספים שלא הוצגו כאן. הקורא המתעניין יכול להרחיב את ידיעותיו באמצעות מקורות נוספים.

נראה כעת כיצד אנו יכולים להשתמש בביטויים רגולריים ב-Perl.

השימוש בביטויים רגולריים נעשה על ידי האופרטור =~ המשווה בין ביטוי רגולרי לבין סקלר כלשהו.

ביטוים רגולריים ב-Perl עטופים בין צמד //. הביטוי // =~ $scalar יתפרש ל-1 אם קיימת התאמה, או ל-undef אם לא.

נביט בדוגמא הבאה:

```
use strict;
while ( defined($currentLine = <STDIN>) )
{
	if ($currentLine =~ /^a.*/)
	{
		print $currentLine;
	}
}
```

הקוד ייקרא שורות מהקלט כל עוד קיימות כאלו, וידפיס את כל השורות המתחילות בתו a.

פרוצדורות

כמו כל שפה, Perl מאפשרת לנו לכתוב קוד מודולרי, בעזרת פרוצדורות.
הפרוצדורות של Perl מתפקדות למעשה כמו פונקציות בשפת C או שפות אחרות.
נכנה אותן פרוצדורות, ולא פונקציות, על מנת להבדיל בין הפונקציות שראינו שהן חלק מהשפה, לבין פרוצדורות, אותן כותב המשתמש.
ניתן להשתמש בפרוצדורות המוגדרות על ידי המשתמש בכל מקום בו ניתן להשתמש בפונקציות של השפה עצמה.

הגדרת פרוצדורות

הגדרת פרוצדורה נעשית על ידי המילה sub, לאחריה שם הפרוצדורה, ולאחריה סוגריים מסולסלים הפותחים בלוק. דוגמא:

```
use strict;
sub MyFirstProcedure
{
   print "Hello, World\n";
}
```

בכל מקום בו נרצה לקרוא לפונקציה, נכתוב:

```
&MyFirstFunction
```

יש לציין כי ה& ברוב המקרים הוא אופציונאלי בלבד.

ערכים מוחזרים

פרוצדורה יכולה להחזיר ערך, וזאת על ידי המילה return. לדוגמא:

```
use strict;
sub MySecondProcedure
{
   return "Hello, World\n";
}
print &MySecondProcedure;
```

פרמטרים לפרוצדורות

כל פרוצדורה ב-Perl יכולה לקבל פרמטרים. כאשר אנו נכנסים לתוך פרוצדורה, Perl מגדירה מערך מיוחד המכיל את הפרמטרים, בשם _@.

ניתן לגשת ישירות לאברי המערך על ידי הקבועים $_[0 .. $#_] אולם מקובל להגדיר סקלרים שיקבלו את ערכי הפרמטרים, ועליהם לבצע את הפעולות, לדוגמא:

```
use strict;
sub HowdyEveryone
{
   my($name1, $name2) = @_;
   return "Hello $name1 and $name2.\n" .
          "Where do you want to go with Perl today?\n";
}

print &HowdyEveryone("bart", "lisa");
```

יש לציין שמשתנים המוגדרים בתוך פונקציה הם משתנים לוקליים, הקיימים רק בטווח הבלוק שלה.

עבודה עם קבצים

העבודה עם קבצים ב-Perl דומה לעבודה עם קבצים בשפת C: ראשית אנו פותחים קובץ, לאחר מכן אנחנו מבצעים עליו פעולות, ובסוף אנו סוגרים אותו.

הדוגמא הבאה פותחת קובץ, קוראת את תוכנו ומדפיסה אותו על המסך:

```
use strict;
$file = '/etc/passwd';          # Name the file
open(INFO, $file);              # Open the file
@lines = <INFO>;                # Read it into an array
close(INFO);                    # Close the file
print @lines;                   # Print the array
```

הפונקציה open פותחת קובץ. הפרמטר הראשון שלה הוא handle, שם באמצעותו Perl תתייחס לקובץ בהמשך. הפרמטר השני הוא שמו של הקובץ.

הפקודה close אומרת ל-Perl שסיימנו לעבוד עם אותו קובץ.

בדוגמא פתחנו את הקובץ לקלט, אולם ניתן לפתוח את הקובץ גם לפלט, או הוספה.

נציג את התחביר הנדרש:

```
open(INFO, $file);      # Open for input
open(INFO, ">$file");   # Open for output
open(INFO, ">>$file");  # Open for appending
open(INFO, "<$file");   # Also open for input
```

אם פתחנו קובץ לפלט, נכתוב לתוכו על ידי הפונקציה print, כאשר הפרמטר הראשון יהיה ה-handle, למשל:

```
print INFO "This line goes to the file.\n";
```

ניתן לגשת אל ערוץ הקלט הסטנדרטי וערוץ הפלט הסטנדרטי בצורה הבאה:

```
open(INFO, '-');  # Open standard input
open(INFO, '>-'); # Open standard output
```

כאשר כתבנו את הביטוי ;<INFO> = @lines נקרא כל הקובץ לתך המשתנה, כרשימה, וזה מכיוון שהתייחסנו לקובץ בהקשר של מערך.

אם היינו מחליפים את @lines בסקלר $line, רק שורה אחת הייתה נקראת בכל פעם. בכל מקרה כל שורה נקראת מהקובץ בשלמות כולל התו '\n'.

EOF