مقدمة في بيرل

نسخة أولية

Aalhanane

aalhananes1@gmail.com
http://www.arabteam2000-forum.com

مقدمة :

1) الخطوات الأولى

2)تعليق (comment)

3) العمل مع القيم البسيطة

4) جمل الشرط والتكرار

Perl

سنتطرق في هذا الدرس إلى المبادئ الأساسية للغة perl

مقدمة :

------- Perl هي لغة برمجة أحدثها Larry Wall (ingénieur système) سنة 1986 بالإضافة إلى أروع مليون مطور، ومن ذلك الحين و perl مشهورة وتعرف تطوير دائم.(5.10.0)

------- P.e.r.l تعني في الأغلب Practical Extraction and Report Language

------ Perlهي لغة غير متخصصة أحدثت في المقام الأول من أجل :
المعالجة التلقائية للنصوص
تطوير الweb
برمجة الشبكات
إحداث الواجهات الرسومية
وأشياء أخرة .....

----لماذا Perl عرفت هذا الإنتشار :
* قابلية النقل (اليوم Perl توجد أو لها إمكانية الوجود على معظم أنظمة التشغيل
(..... Amiga, Atari ,VMS, Mac, Windows, Unix)
* مجاني (متوفر في الإنترنيت بدون قيود)
*البساطة (أيcode من perl يسمح بإنجاز عمل يقوم به برنامج مكون مثلا من 500 سطر مكتوب ب C أو ب Pascal)
.
.
.

ملاحظة :
perl (p minuscule) هو الاسم الذي يطلق على مفسر(interpreter) لغة Perl Perl (P majuscule) هو الاسم الذي يطلق على إسم اللغة

ملاحظة:
Perl4 هي الماضي، Perl5 هي الحاضر، Perl6 هي المستقبل

----- تاريخ موجز ل Perl بواسطة Larry Wall:

```
Perl 0  introduced Perl to my officemates.
Perl 1  introduced Perl to the world, and changed /\(...\|...\)/ to
        /(...|...)/.  \(Dan Faigin still hasn't forgiven me. :-\)
Perl 2  introduced Henry Spencer's regular expression package.
Perl 3  introduced the ability to handle binary data (embedded nulls).
Perl 4  introduced the first Camel book.  Really.  We mostly just
        switched version numbers so the book could refer to 4.000.
Perl 5  introduced everything else, including the ability to introduce
everything else.
```

--1-- الخطوات الأولى

1-1- تركيب أو تنصيب Perl

لنتأكد أولا هل لدينا Perl بجهازنا أم لا، نذهب إلى نافذة الدوس أو terminal ونكتب

```
c:\> Perl  -v
```

فإذا حصلتم على رسالة تبين لكم معلومات عن اللغة (إصدارها وما شابه ذلك) يبين هدا أن Perl مثبتة على جهازكم، أما إذا حصلتم على رسالة تدل على حدوث خطأ مفاده أنه لم يعرف تلك التعليمة (Perl -v) ،يدل ذلك على أن Perl غير مثبت على جهازكم.

ملاحظة: عادتا لا يكون perl موجود على xp ، أما بالنسبة لlinux فمعظم توزيعات لينكس تركّب perl تلقائياً.

------ كيف نثبت Perl على Windows ؟

لكي تتعامل مع Perl بشكل صحيح في Windows يلزمك تثبيت Active Perl، يمكنك تنزيله من هنا

<----- www.activestate.com

<----- /http://downloads.activestate.com/ActivePerl

بعد إنزال Active Perl قم بتثبيته كما تثبت أي برنامج آخر. شيء عادي (التالي، التالي، التالي،تثبيت ثم نهاية)

--- هذا عنوان آخر للتحميل

http://www.softpedia.com/get/Programming/Coding-languages-Compilers/ActivePerl.shtml

---

2-1 أين أكتب codes الخاصة بperl ؟

فقط قم بفتح Notepad ثم قم بكتابة الكود الخاص بك بعدها قم بتسجيله بإسم (أي إسم) فقط إعطيه الإمتداد التالي pl ، مثلا example.pl

كملاحظة فهناك الكثير من البرامج المساعدة التي تتيح لك الكتابة بشكل مرتب وميسر (يمكن إستعمالها بدل Notepad) ، من بينها نجد مثلا:

* برنامج ActiveState Komodo Edit.
*برنامج DzSoft Perl Editor
* EngInSite Perl Editor
Perl Studio 2009*
SannySoft Perl Editor*

3-1 برنامجنا الأول في Perl

```
#!usr/bin/perl
use strict;
use warnings;
print "Marhabann bikom fi 3alam Perl\n";
```

----- #!/usr/bin/perl يدل على المسار الذي يوجد به المفسر Perl ،

كما يمكنك بدل `#!/usr/bin/perl` كتابة التالي:

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

بما أننا نعمل تحت Windows يمكن بدل **`#!/usr/bin/perl`** أن نكتب

```
#!C:\Perl\bin\perl.exe
```

فإن لم تكن لك إمكانية الحصول على المسار الصحيح فقم بأخذ المساعدة من (system administrator) مدير النظام،

أو يمكنك إن كنت متأكد من إصدار Perl الذي لديك (مثلا v5.8.8) أن تكتب التالي في السطر الأول للبرنامج :

```
Use 5.8.8;
```

ملاحظة: إن كان لديك متلا perl دو إصدار أقل من 5.8.8 واخترت أن تكتب الكود السابق فستحصل على خطأ،
أما إن كان لديك perl دو إصدار أكبر من 5.8.8 فلن تحصل على أي خطأ .

تذكير : لتعرف الإصدار الذي لديك إفتح terminal أو نافدة الدوس و أكتب التالي

```
C:\>Perl –v
```

--------بالنسبة ل use stricts فهي بإختصار تلزمك بأن تكون كتابتك للبرنامج تحقق قوانين وشروط لغة Perl

-----أما بالنسبة ل use warnings فهي تقوم بطلب إظهار – رسائل الإنذار - إن حصل أي خطأ أو خلل في syntaxe.

4-1 تنفيذ برنامج في perl

سنتطرق هنا لطريقتين بارزتين لتنفيذ أحد البرامج في perl

------- الطريقة الأولى:
نقوم بكتابة الكود في نافدة الدوس مباشرة
إفتح نافدة الدوسcmd وقم بكتابة مثلا :

```
C:\>perl -e "print \"Marhabann bikom fi 3alam Perl\n\""
```

بعد الضغط على entr
سيتم التنفيذ، و يظهر لنا

```
C:\> Marhabann bikom fi 3alam Perl

C:\>
```

يمكننا أن نظيف w– لإظهار رسائل الإنذار إن حصل أي خطأ– مثل use warnings - ونكتب كالتالي-

```
C:\>perl –w -e "print \"Marhabann bikom fi 3alam Perl\n\""
```

هذه الطريقة ليست متداولة بكثرة وغير مضمونة - إن صح التعبير-، لكن هي طريقة سلسة وسريعة لعمل عملية صغيرة أو نداء لدالة خاصة بـPerl.

ملاحظة:

بالنسبة ل Linux نفتح terminal ونكتب التالي

```
perl -e 'print "Marhabann bikom fi 3alam Perl\n"'
```

ملاحظة:

لمعرفة المزيد من command line الخاصة ب perl مع شرح بسيط لها ، قم بكتابة التالي:

```
C:\>perl –h
```

-------الطريقة الثانية :

إفتح (bloc-notes) Notepad وكتب التالي – مثل ما كتبنا في السكريبت الأول –

```
#!usr/bin/perl
use strict;
use warnings;
print "Marhabann bikom fi 3alam Perl\n";
```

وقم بتسجيله بأي إسم مثلا

file name ===> exemple.pl
Save as type ===> All Files(*.*)

بعدها إذهب إلى نافذة الدوس وقم بكتابة التالي:

```
C:\>perl exemple.pl
```

ثم إضغط على entr ليتم تنفيذ البرنامج ويظهر لك التالي:

```
C:\> Marhabann bikom fi 3alam Perl

C:\>
```

ملاحظة: للتأكد من صحة البرنامج دون تنفيذه أكتب التالي:

```
C:\>Perl –c exemple.pl
```

بعد الضغط على entr إن لم يكن في البرنامج خطأ ما سيظهر في الأغلب التالي:

```
C:/>exemple.pl syntax OK
```

5-1 أنواع البيانات في perl

Perl لها ثلاث أنواع من البيانات – data :-
scalars (كميات غير موجهة)أو (كميات متغيرة)،
arrays of scalars (مصفوفات خاصة بالكميات الغير موجهة) ،

associative arrays of scalars (المصفوفات المتعاونة أوالمشتركة الخاصة بالكميات الغير موجهة) المعروفة بال Hashes
وهناك أيضا في perl نوع بيانات آخر يطلق عليه بـfilehandle .

بشكل عام هذه هي الكتابة العامة للإعلان عن أحد المتغيرات (scalars) :

```
my  $name_scalar;
```

نكتب دائما$ قبل اسم المتغير **(scalars)**

ملاحظات:

* يمكن القول أن scalar مرتبطة دائما ب $.
* تستخدم $ بكثرة للدلالة على 'the' .
*scalar يمكن أن يأخذ قيمة واحدة إما string بسيطة (بدون حدود، مادامت الذاكرة تتيح ذلك) ، كما يمكن أن يأخذ عدد،أو حتى **reference (مرجع) .**
***في perl لا نحتاج أن نعرف نوع المتغير مثلا هل هو int أو char**
مثلا

perl	Vb.net	c
my $var ;	Dim var as char	char var ;
my $var ;	Dim var as integer	Int var ;

Normal arrays

* تتيح عدة قوائم منظمة ومرتبة يمكنك أن تخزن بها قيمة واحد أوعدة قيم مفهرسة بأرقام تبدأ هذه الأرقام ب0.

Hashes

* تتيح تخزين مجموعات من القيم الغير مرتبة ومفهرسة بقيمة تحددها أنت و التي تمثل key (الدليل).

بالاستئناس بتلك الملاحظات لاحظ التالي:

```
print $var ;                              # إظهار قيمة scalar بسيطة
 print "$var[19] " ; # إظهار العنصر الذي يأتي في الرتبة 20 في المصفوفة@var
```

```
print "$var{‘nom’}" ;
                        # إظهار القيمة التي يدل عليها المؤشر nom في %var Hashe
```

```
print  " $#var" ;
                                    إظهار قيمة آخر (Index) في المصفوفة #@var
```

* بالنسبة للمصفوفات arrays نشير لها في perl ب@
* @ تستخدم بكثرة للدلالة على 'these' أو'those'
*

perl	Vb.net	c
my @var ;	**Dim var(9) as char**	**char var[9] ;**
my @var ;	**Dim var(9) as integer**	**Int var[9];**

الفرق بين perl و c و vb.net في الإعلان عن مصفوفة ما.

perl	javascript
my $nom ;	var nom ;
my @nom ; لاتحتاج لتحديد عدد عناصر المصفوفة في perl	var nom=new Array(8) ; تعريف مصفوفة nom مكونة من 8 عناصر
my @nom= ("Mohammad","Hamza","Jamal") ;	var nom=new Array["Mohammad","Hamza","Jamal"] ;

الفرق بين perl و javascript في الإعلان عن متغير ما أو مصفوفة ما.

```
print  @var ;
                                                مماثل لي
print   ($var[0], $var[1],$var[2],$var[3]….,$var[n])
                                    لإظهار جميع القيم الموجودة في المصفوفة @var
```

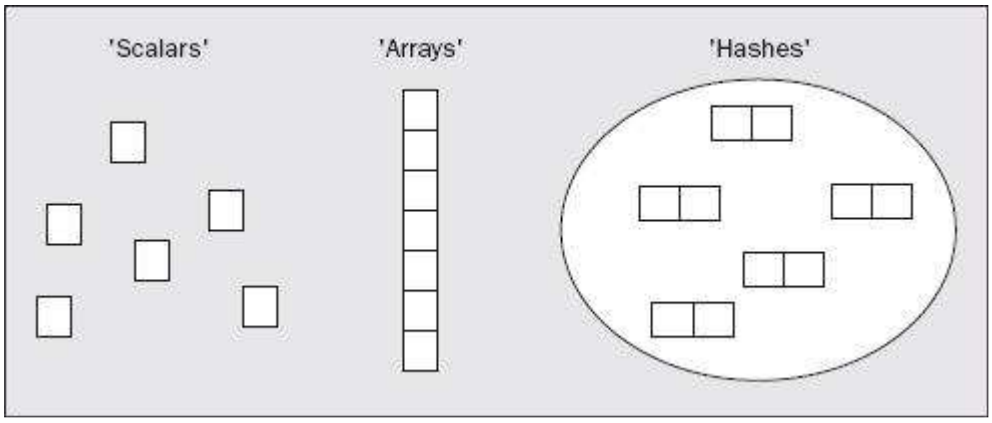
```
print @var[2,6,8] ;          #print these (2,6,8)
                                                مماثل ل
print ($var[2],$var[6],$var[8])    #print the(1),the(2),the(3)
```

يظهر فقط العناصر التي حددنا لها المؤشرات (index) الدالة عليها

* في perl نشير لل Hashe ب %

```
my %var ;       #(key1, val1, key2, val2 ...)
```

خلاصة:



$ ----------> one
@ ----------> many
% ----------> dictionary

إلى هنا أضن أننا أخدنا فكرة أولية عن perl ، وبالتأكيد الآن أصبحت تدور عدة أسئلة في دهنك، هذا جيد ، هدا ما نريد أن تصل إليه في هدا الجزء .

2) تعليق (comment)

هناك إمكانيتان لكتابة تعليق، إما كتابة تعليق في سطر واحد، أو في عدة أسطر.

بالنسبة لكتابة تعليق في سطر واحد نضيف # في أول الكود، مثلا:

```
# print ‘mmmmm’ ;
# Larry Wall
```

المفسر perl سيتجاهل هذان السطر لأنهما مسبوقان ب#

في حالة أننا نريد كتابة تعليق في عدة أسطر، نكتب ما نريد أن يتجاهله perl بين pod= و cut=
مثلا:

```
=pod
Comment
Larry Wall
=cut
```

3)العمل مع القيم البسيطة

numbers (3-1

بالنسبة لperl ، كما رأينا سابقا لاتحتاج أن تقوم بتعريف نوع المتغير أي هل (أو integer أو double أو....).
numbers التي يتعامل معها perl على العموم هي integers و floating-point

```
12345             # integer
-54321            # negative integer
12345.67          # floating point
6.02E+023           # scientific notation تعني 6.02x10 23
0xffff            # hexadecimal
0377              # octal
4_294_967_296     # underline for legibility (من أجل تسهيل القرائة)
```

الآن لنقم ببعض الأمثلة لإظهار أو طباعة الأرقام في perl:

```
#!/usr/bin/perl
# perl nbr1.pl
use warnings ;
use strict ;
print 4,   -6 ;
```

هنا لو رأينا الكود لقلنا أننا سنحصل على:

```
C :>perl nbr1.pl
4   -6
C :>
```

لكن في perl يتجاهل (space) الفراغ ولهدا سنحصل على:

```
C :>perl nbr1.pl
4-6
C :>
```

بدون طباعة (space) فراغ بين 4 و 6-

--- قم بتعديل البرنامج السابق ليكون كالتالي:

```
#!/usr/bin/perl
# perl nbr2.pl
use warnings ;
use strict ;
print 4,"   ",-6 ;
```

لاحظ معي لكي لا تتجاهل print الفراغ قمنا بوضع الفراغ (space) بين (") و (") .

لهذا سنحصل على:

```
C :>perl nbr2.pl
4   -6
C :>
```

--- مثال آخر

```
#!/usr/bin/perl
# perl nbr3.pl
use warnings ;
use strict ;
print 1_234_567_890, "    ",-8  ;
```

سنحصل على:

```
C :>perl nbr3.pl
1234567890   -8
C :>
```

على العموم إستعملنا (_) لتسهيل قراءة الكود

Binary,Hexadecimal, And octal Numbers

هنا سنكتب برنامج بسيط لنوضح عمليا كيف نحول الأنضمة الأعداد الثلاثة إلىdecimale .

```
#!/usr/perl/bin
#nbrSy1.pl
use warnings ;
use strict ;
print 255, "\n" ;           # decimale
print 0377, "\n" ;            # octal
print 0b11111111, "\n";       # binary
print 0xFF, "\n" ;              # hexadecimal
```

بعد تنفيده نحصل على:

```
C:\>perl nbrSy.pl
255
255
255
255

C:\>
```

يمكن أن نستنتج من هذا المثال أنه يمكننا كتابة نفس العدد (لدينا هنا255) إما كعدد عادي إن صح التعبير الذي نكتبه بدون إضافات أو كBinary و Hexadecimalو octal التي نلخص كيفية كتابتها بالتالي:

0 (صفر) من أجل octal.
0b من أجل binary.
0x من أجل hexadecimal.

أو يمكننا إستعمال الدوال التي تتيحها لنا perl :
oct() من أجل octal
hex() من أجل hexadecimal
بالنسبة لbinary لاتوجد دالة خاصة حسب علمي لكن يمكن إستعمال **oct()**

مثال بالنسبة لهذه الدوال:

```
#!/usr/perl/bin
#nbrSy2.pl
use warnings ;
use strict ;
print  'hex("ff")=',hex("ff") ,"\n";        # hexadecimal ----      هنا إظافة x0 غير ملزمة
 print  'oct("377")=',oct("377") ,"\n";              # octal ---------هنا إظافة 0 غير ملزمة
print  'oct("0b11111111")=' ,oct("0b11111111"), "\n";       # binary
                                                                  هنا إظافة 0b ملزمة
```

بعد تنفيذ nbrSy2.pl سنحصل على:

```
C:\>perl nbrSy.pl
hex("ff")=255
oct("377")=255
oct("0b11111111")=255

C:\>
```

String(3-2

1.2.3) المحددتين (") و (')

في perl نضع سلسلة الرموز (string) أو(سلسلة حروف) بين محددتين : **double quoted** (") أو **single quoted (')**،مع العلم أنه ليس لهما نفس الدور أو الوظيفة .
*فالبنسبة للمحددة (")**double quoted** ، المحتوى("الذي يوجد بينهما ") هو مفسر(أو أنه سيفسر من perl أثناء الطباعة).
*أما البنسبة للمحددة (') **single quoted** فالمحتوى (' الذي يوجد بينهما ') ليس بالضرورة مفسر (perl سيعتبره كأنه سلسلة حروف - string –(سيقوم بطباعته في الأغلب)) .
هذا بشكل مختصر ، لنرى معا هذا المثال:

```
#!c:/perl/bin/perl.exe
#string1.pl
use warnings ;
use strict;
 print "\tThis is a double quoted string.\n";
```

```
print '\tThis is a single quoted string\n';
```

عند تنفيذه سيظهر التالي:

```
C:\>perl string1.pl
        This is a double quoted string.
\tThis is a double quoted string.\n
```

نلاحظ أنه تم تفسير كل من \t و\n عند إستعمال (") double quoted بينما لم يتم تفسيرهما عند إستعمال (') single quoted بل تم طباعتهما.

لنلاحظ هذا المثال أيضا:

```
#!/usr/perl/bin/
# string2.pl
use warnings;
use strict;
my $name="mohammad";  # mohammad الإعلان عن متغير سميناه بname وأعطيناه القيمة
print "this is double quoted....Name=$name\n";
print 'this is single quoted....Name=$name\n' ;
```

عند تنفيذه سيظهر التالي:

```
C:\>perl string2.pl
this is double quoted....Name=mohammad
this is single quoted....Name=$name\n
```

المثال واضح: (لاحظ المتغير $name في كلتي الحالتين) (راجع التعريف البسيط فوق)

مثال آخر

```
#!/usr/perl/bin/
# string3.pl
use warnings;
use strict;
print "this is \ "double\" quoted....\\0=\0ss\n";
print 'this is \'single\' quoted....\\0=\0ss\n';
```

عند تنفيذه سيظهر التالي:

```
C:\>perl string3.pl
this is "double" quoted....\0= ss
this is 'single' quoted....\0=\0ss\n
```

\0 تعني **Character null**

الذي نلاحظه هنا ، عند إستعمال double quoted (")وأرضنا أن لايفسر perl (مثلا كما لدينا في المثال) \0 نكتب أممها \ أي هكذا \\0 .

مثلا: لا نريد أن يفسر perl \n (أي سطر جديد line feed) نكتب \\n

نفس الشيء، لكي نظهر أو ليتم طباعة $ (أو @ أو " وهكذا....) بدون أن يفسرها perl نسبقها بي \ هكذا \$.(نحن هنا نتكلم في الحالة التي نستعمل فيها " double quoted ، لأنه عند إستعمال ' single quoted فهي (أي $) ومثيلاتها لا يتم تفسيرها من طرف perl أصلا.

تذكر:

الرمز	تعليق	Octal	Hexadecimal
\0	NULL(NUL)	\0	\0
\n	سطر جديد(LF)	\012	\x0A
\r	Carriage return (CR)	\015	\x0D
\b	Backspace(BS) حذف حرف للوراء (يسار)	\010	\x08
\f	form feed(FF)صفحة جديدة	\014	\x0C
\a	alert تصدر صوت(BEL)	\007	\x07
\t	Horizontal Tab(HT) ;مسافة جدولة	\011	\x09
\s	space	\040	\x20
\u	تحول الحرف الإنجليزي الأول (الذي يلي \u) إلى upper-case أي كبير		
\l	تحول الحرف الإنجليزي الأول (الذي يلي \l) إلى lower-case أي صغير		
\U	تحول مجموعة الحروف الإنجليزية (التي تلي \U) إلى upper-case أي كبيرة حتى أول \E		
\L	تحول مجموعة الحروف الإنجليزية (التي تلي \L) إلى lower-case أي صغيرة حتى أول \E		
\Q	تنصص ال Regular Expr حتى أول \E		
\E	انهاء كل من \U و \L و \Q		

و(\ ) مع أي شيء آخر تعني ذلك الشيء نفسه وليس معناه وفائدة هذا مثلا إذا أردت وضع شيء بين علامة اقتباس مثلا اكتب ;"\tThis is a double quoted string.\n" print كما ذكرنا سابقا.....

مثال : **print "\Ularry wall.\n ";**
سيطبع
LARRY WALL.

مثال : **print "\Ularry w\Eall.\n";**
سيطبع
LARRY Wall.

مثال : **print "\ularry \uwall.\n";**
سيطبع :
Larry Wall.

2.2.3) بديل المحددتين (") (') :

بدل إستعمال المحددات (')و(") يمكننا إستعمال على التوالي ال (/ /q)و (/ /qq) :
أمثلة للشرح:

```
#!/usr/perl/bin/
#string4.pl
use strict ;
```

```
use warnings ;
print qq/Salam ana "Muslim" auhibo 'Rassaul Allah'\n/ ;
print q/Salam ana "Muslim" auhibo 'Rassaul Allah'\n/ ;
```

بعد التنفيد نحصل على:

```
C:\>perl string4.pl
 Salam ana "Muslim" auhibo 'Rassaul Allah'
Salam ana "Muslim" auhibo 'Rassaul Allah'\n
C:\>
```

نلاحظ أنه عند إستعمال //qq أو //q فإننا نستخدم (") و (') بشكل عادي بدون إستعمال(\) على عكس ما كنا نفعل سابقا عندما نستخدم أحد المحددتين (') أو (") في الطباعة . دون هذه الحالة تقريبا يمكن القول أن //q تعوض (') ، بينما //qq تعوض (")
تقريبا هذا هو الفرق بين المحددتين (') (") و //q //qq.

مثال أخرى:

```
#!/usr/perl/bin/
# string5.pl
use strict ;
use warnings ;
my $variable= "Mohammad" ;
print qq/ "double quoted"\040'single quoted' \$variable= $variable\n/ ;
print q/"double quoted"\040'single quoted' \$variable= $variable\n / ;
```

بعد التنفيد نحصل على:

```
C:\>perl string5.pl
"double quoted" 'single quoted Mohammad'
"double quoted"\040'single quoted $variable'\n
C:\>
```

أضنه لا يحتاج لتعليق

في حالة أردت طباعة (/) نكتب

```
#!/usr/perl/bin/
# string6.pl
use strict ;
use warnings ;
print qq/ \/  \n/ ;
print q/ \/ \n / ;
```

سيطبع التالي:

```
C:\>perl string6.pl
 /
 / \n
C:\>
```

لكي لا يرتبك مفسر perl (ويعتبر / التي أضفت هي لإغلاق //q)نضيف (\) قبل (/) أي هكذا(\/).
أو تغير هذه الكتابة //qq إلى (مثلا) q|| ، لأن perl تتيح لك تغيير / إلى | أو # أو..

لنرى هذا المثال

```
#!/usr/perl/bin/
# string7.pl
use strict ;
use warnings ;
print qq| /  \n| ;
print q| /  \n| ;

print qq# \nSalam ana "Muslim" auhibo 'Rassaul Allah'\n# ;
print q#Salam ana "Muslim" auhibo 'Rassaul Allah'\n# ;

print qq< \nSalam ana "Muslim" auhibo 'Rassaul Allah'\n> ;
print q<Salam ana "Muslim" auhibo 'Rassaul Allah'\n> ;

print qq(\nSalam ana "Muslim" auhibo 'Rassaul Allah'\n) ;
print q(Salam ana "Muslim" auhibo 'Rassaul Allah'\n) ;
```

تنفيد البرنامج

```
C:\>perl string7.pl
/
/ n\
 Salam ana "Muslim" auhibo 'Rassaul Allah'
Salam ana "Muslim" auhibo 'Rassaul Allah'\n
 Salam ana "Muslim" auhibo 'Rassaul Allah'
Salam ana "Muslim" auhibo 'Rassaul Allah'\n
 Salam ana "Muslim" auhibo 'Rassaul Allah'
Salam ana "Muslim" auhibo 'Rassaul Allah'\n
C:\>
```

Here-Documents(3.2.3

هذه الوسيلة تتيح لك كتابة مجموعة كبيرة من النصوص داخل برنامجك .
لنرى هذا المثال التوضيحي:

```
#!/usr/perl/bin/
#string8.pl
use warnings ;
use strict ;
my $forum=q/ www.arabteam2000-forum.com/ ;
 print<<EOF;
 This is "Here-Documents".
\$forum=$forum .
\\
\\t
\t Hello, world .
EOF
```

بعد التنفيذ:

```
C:\>perl string8.pl
 This is "Here-Documents".
$forum='www.arabteam2000-forum.com'.
```

```
\
 \t.
                Hello, world
```

يمكنك بدل كتابة EOF(End Of File) أي كلمة آخر بدون space.
نرى أن لهذه الطريقة نفس خاصية كل من (") و //qq من حيث أن perl يقوم بتفسير رموز ASCII إذا أسبقتها ب(\) وأيضا الرموز الأخرى مثل $ ، @ .. و أيضا مثل (") و //qq إذا أردت عدم تفسير هذه الرموز من قبل perl أثناء طباعتها قم بوضع (\) قبل أحد الرموز تلك. مثلا $\ ليتم طباعة $، بينما هذا المثال t\\ لطباعة t\، أما هذا \\ فهو لطباعة \.

3-3) الإعلان عن المتغيرات

ملاحظة:

متغير scalar يمكن أن يأخذ مثلا:
6
2.5625985
"hello word"
بدون تحديد نوع المتغير

أولا يجب أن نذكر ثانيتا أن متغير scalar يستطيع أن يحتوي تقريبا على أي شيء، فهو إن صح التعبير لا يفرق بين الأعداد الصحيحة والأعداد ذات الفاصلة أو سلسلة حروف. على الرغم من ذالك لا يطرح أية مشكلة في perl كما سنرى لاحقا.

* للإعلان عن متغير scalar أكتب إسمه (الذي إخترت له) مسبوق ب$.
* في perl أنت غير ملزم بالإعلان مسبقا على متغير ما، يمكنك إستخدامه مباشرة في العمليات التي تقوم بها.
* يوجد خيار يسمح لك بفرض syntax دقيق(strict) ، في هذه الحالة يصبح ملزما لك الإعلان مسبقا عن المتغيرات، من أجل ذلك ومن أجل حماية أكثر، أكتب في بداية البرنامج:

```
#!/usr/perl/bin/
use strict ;
```

ملاحظة:

أمثلة للأسماء الجائز إعطائها للمتغيرات:
$A_Scalar_Variable;
$scalarNo8;
$_private_scalar;

أمثلة للأسماء التي لا يجوز إعطائها للمتغيرات:
لاتضع رقم في أول الإسم # $8bit_characters;
لاتضع أي علامة في آخر الإسم # $cash_in_£;

```
لاتستعمل - في الفصل بين كلمتين ولا تستعمل space  # ;$long-var
```

* هناك تقريبا كلمتي مفتاح للإعلان عن متغير ما (مهما كان Hash,array,scalar) هما my و our [مثل في vb (Dim و public)]. غالبا ما نستخدم my .

مثلا: من أجل الإعلان عن متغير أعطيناه إسم var نكتب التالي:

```
#!/usr/perl/bin/
use warnings;
use strict;
my $var;
```

لإعطاء قيمة للمتغير نكتب

```
#!/usr/perl/bin/
#var1.pl
use warnings;
use strict;
my $var="Mohammad";    # string هنا أعطيناه
print "$var\n";
$var=44;               # int  هنا أعطيناه
print "$var\n" ;
$var=4.1254;     #  float  هنا أعطيناه
print "$var\n" ;
```

بعد التنفيذ:

```
C:\>perl var1.pl
Mohammad
44
4.1254

C:\>
```

نلاحظ أننا قمنا بإعطاء لنفس المتغير ثلاث قيم مختلفة من حيث النوع بدون أن نحدد له نوع المتغير.

مثال آخر

```
#!/usr/perl/bin/
#var2.pl
use warnings;
use strict;
my ($var,$sum) ;
$var=44;
print "44+4=$var+4\n";

print "44+4=", $var+4,"\n" ;
```

```
$sum=$var +4;
print "44+4=$sum\n";
```

بعد التنفيذ:

```
C:\>perl var2.pl
44+4=44+4
44+4=48
44+4=48

C:\>
```

إذا أردنا الإعلان عن متغيرين أو أكثر في وقت واحد نجمعهما بين قوسين ونفصلهما بفاصلة هكذا: **my ($var,$sum) ;**
نفرض أننا أرضنا أن نعطي قيم لهذه المتغيرات في وقت واحد أو في سطر واحد في هذه الحالة نكتب
my ($var,$sum)=(44,0) ; هده الكتابة تكافئ my $sum=0 ; my $var=44 ;
إذا تم وضع عملية حسابية مابين (") أو ما يشبهها سيتم طباعتها بشكل عادي ولن يتم حسابها.

الأن لنلاحظ هذا المثال الذي يعالج إشكالية أخرى:

```
#!/usr/perl/bin/
#var4.pl
# Mohammad من أجل إظهار
# mmad مع Moha بتوحيد
use warnings ;
use strict ;
my $var= "Moha" ;
print "my name is $var mmad\n" ;   # print « my name is $varmmad » ;   is fulse
print "my name is ${var}mmad\n" ;
```

بعد التنفيذ:

```
C:\>perl var4.pl
my name is Moha mmad
my name is $Mohammad

c:\>
```

نلاحظ هنا أننا عندما نريد طباعة نص يكون ملتصق(إن صح التعبير) بالقيمة التي يحتويها متغير ما نضع إسم هذا المتغير بين{}.

ملاحظة:

إعطاء نفس القيمة لعدة متغيرات في نفس السطر

```
my  ($var,$k,$b,$z) ;
$var=($k=$b="Mohammad"),($z="lachg") ;
print("\$var=",$var,"\n\$k=",$k,"\n\$b=",$b,"\n\$z=",$z,"\n") ;
```

سيتم طباعة التالي:

```
$var=Mohammad
$k=Mohammad
$b=Mohammad
$z=lachg
```

$var,$k,$b تأخذ نفس القيمة "Mohammad" بينما $z تأخذ قيمة مخالفة "lachg"

Standard output (4-3

من أجل طباعة قيمة ما على الشاشة نستعمل في الأغلب **print** أو **printf** .

print (1.4.3

print تظهر نص أو سلسلة نصوص. ترجع 1 أي true في حالة النجاح.
مثال:

```
#!/usr/perl/bin/
#output1.pl
use warnings ;
use strict ;
print "Hello world\n" ;
```

```
C:\>perl block1.pl
Hello world

C:\>
```

المثال أضنه واضحا .

printf (2.4.3

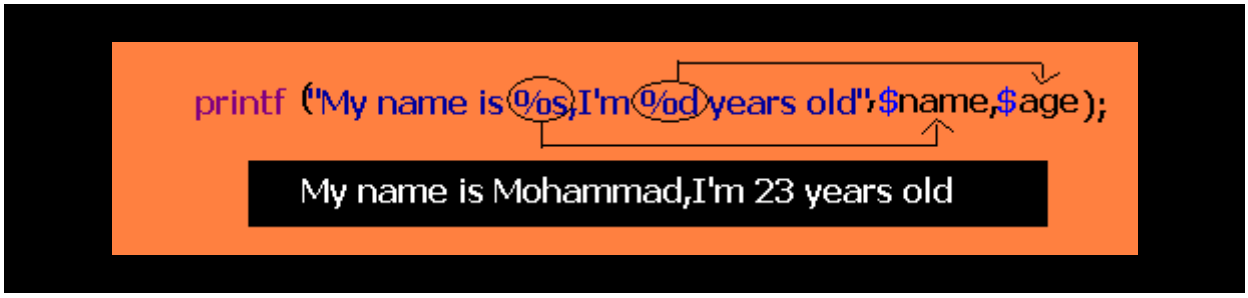
بالنسبة لـ**printf** فهي أيضا لطباعة المعلومات على الشاشة لكن بشكل مرتب أو أكثر تخصيصا (إن صح التعبير)، على العموم فإن كنت مررت على لغة c فإن هذه الدالة ستكون مألوفة لك.
مثال:

```
#!/usr/perl/bin/
#output2.pl
use warnings ;
use strict ;
my ($var,$nbr)=(2.53241,526) ;
printf ("\$var=%.2f \n",$var) ;
printf ("\$var=%.0f \n",$var) ;
printf ("\$nbr=%d\n",$nbr) ;
```

```
C:\>perl output2.pl
$var=2.5
```

```
$var=3
$nbr=526
C:\>
```

لاحظ هذه الصورة التوضيحية:



أيضا أنضر إلى هذا الجدول :

رمز	الوصف
%d	عدد صحيح دو إشارة (+) و(-) (decimal)
%u	عدد صحيح بلا إشارة (-) (unsigned decimal،(integer
%f	عدد كسري،(عدد الأرقام التي تأتي بعد الفاصلة هو ثابت). لكن يمكنك أن تحدد عدد الأرقام التي ستطبع أو ستظهر على الشاشة بعد الفاصلة هكذا: %.2f حيث الرقم 2 يمثل عدد الأرقام بعد الفاصلة التي تريدها أن تطبع ملاحظة: في المثال السابق كتبنا %.0f الذي بسببه تمت طباعة 3 بدل 2 الذي كان منطقيا لأننا أردنا طباعة صفر رقم بعد الفاصلة. الرقم الذي لدينا هو 2.53241 ،لاحظ أن الرقمين الذين بعد الفاصلة هما 53 (بسبب أن هذا الرقم أكبر من 50) لهذا تمت طباعة 3
%e	مثل %f لكنه سيطبع على الشكل الهندسي
%g	عدد كسري(عدد الأرقام بعد الفاصلة يطبع كما أدخلته) عدد الأرقام التي تأتي بعد الفاصلة في كل من %f و %e أحسن أوأكثر من %g (على العموم).
%o	عدد صحيح بلا إشارة(–) ، بالoctal
%x	عدد صحيح بلا إشارة (-) ، بالHexadecimal
%b	عدد صحيح بلا إشارة (–) بال binary
%X	مثل %x لكن بحرف كبير
%E	مثل %e لكن بحرف كبير
%s	سلسلة نصية. يمكنك تحديد عدد الحروف التي تريدها أن تطبع (حتى space يحسب). مثلا لدينا السلسلة النصية الثالية "Hello" عند إستعمال %.2s سيطبع He ، إذا الرقم 2 يدل على عدد الأرقام التي يجب أن تطبع

%%	لطباعة إشارة النسبة المئوية. (لإلغاء عمل% (إن صح التعبير))
%p	المؤشر أو العنوان (Pointer)

هنا سنتطرق لهذا المثال لنستنتج منه بعض الخاصيات:

```
#!/usr/perl/bin/
# output3.pl
use warnings ;
use strict ;

 printf "(% d)\n",88 ;        #---------->( 88)      يطبع فراغ واحد بين القوس و88
 printf "(%     d)\n\n",88 ; #---------->( 88)     نفس الشيئ هنا

printf "(%+d)\n",88 ;         #---------->(+88)
printf "(%+++d)\n\n",88 ;     #---------->(+88)

printf "(%o)\n",88 ;              #---------->(130)
printf "(%#o)\n\n",88 ;           #---------->(0130)     عند إظافة # يتم طباعة 0 في أول العدد
                                                        الذي يدل على أن هذا العدد ينتمي إلى octal

printf "(%x)\n",88 ;              #---------->(58)
 printf "(%#x)\n\n",88 ;          #---------->(0x58)     هنا يتم طباعة العدد 58 مع الرمز0x الذي
                                                        يدل على أن 58 ينتمي إلى hexadecimal

printf "(%#X)\n\n",88 ;       #---------->(0X58)  (upper-case) نفس شيء فقط X حرف كبير

printf "(%b)\n",88 ;              #---------->(1011000)
printf "(%#b)\n\n",88 ;           #---------->(0b1011000)  binary هنا للدلالة على


 printf "(%10s) \n","Hello" ;  #---------->(     Hello)     هنا بما أن سلسلة النصية تحتوي فقط على
                                5 حروف ونحن أعطينا 10 سيتم إظافة 5 مسافات أو فراغات أوspaces لتكملة الرقم 10 وهذا من جهة يمين
                                السلسلة النصية  لأن 10 موجبة

printf "(%-10s) \n","Hello" ;        #---------->(Hello     ) نفس الشيء إلا أن ذلك سيكون على
                                                        يسار السلسلة النصية وهذا لأن العدد 10 سالب

 printf "(%010s) \n","Hello" ; #---------->(00000Hello) نفس الشيئ لكن هنا أضفنا 0 قبل10،
                                في هذه الحالة بدل طباعة الفراغ سيتم طباعة 0 على يمين السلسلة النصية
```

عند التنفيذ نحصل على:

```
C:\>perl output3.pl
( 88)
( 88)
```

```
(+88)
(+88)

(130)
(0130)

(58)
(0x58)

(0X58)

(1011000)
(0b1011000)

(Hello      )
(      Hello)

C:\>
```

من خلال هذا المثال يمكن أن نكتب :

فراغ (Space) ==========> بادئة خاصة بالأعداد الموجبة (space واحد فقط)

\+ ==========> بادئة خاصة بالأعداد الموجبة

\# ==========> بادئة خاصة بضمان جلب (0) من أجل أي octal ، (0x) من أجل Hexadecimal و (0b) من أجل Binary

\- ==========> بادئة تتحكم في هامش اليسار

0 ==========> يعني إستخدم الأصفار ولا تستخدم spaces من أجل التحكم في هامش اليمين

الوصول إلى البرامتر عن طريق إختيار علامة أو مؤشر(index) دال عليه

لنرى هذا المثال :

```
#!/usr/perl/bin/
# output4.pl
use warnings ;
use strict ;

 printf ("(%2\$d    %1\$d)\n\n",6,8) ; #------> (8     6)     لتحديد البارامتر الذي تريده أن
يطبع بدون أن تغير ترتيبهم  في الكود ، نكتب $2 حيث 2 يدل على الرتبة التي يحتلها البارامتر المطلوب،
هنا قمنا بإضافة ( \ ) لكوننا إستعملنا (") ولهذا كتبنا(\$2)

printf ('(%2$d    %1$d)\n',6,8) ;   #---------->(8     6)
نفس الشيء ،  فقط هنا بما أننا إستعملنا (') فلهذا لم نحتاج إلى إضافة (\)
```

```
print "\n\n" ;      #لترك سطرين فارغين
printf ("(%3\$d   %d     %1\$d) \n",6,8,44) ;   #--------->(44     6      6)    لاحظ
```

تنفيذ

```
C:\>perl output4.pl
(8      6)

(8    6)\n

(44     6       6)

C:\>
```

ملاحظة:هناك أيضا الدالة sprintf

Standard input (5-3

من أجل القراءة من لوحة المفاتيح نستعميل **<STDIN>**
مثال:

```
#!/usr/perl/bin/
#input1.pl
use warnings ;
use strict ;
my $in ;
print "Please enter value :    " ;
$in=<STDIN> ;
chomp($in) ;
print "\$in=$in " ;
```

```
C:\>perl input1.pl
Please enter value :   44
$in=44
C:\>
```

هذا البرنامج هو بسيط ، يطلب من المستخدم بإدخال قيمة ما بعدها يقوم بطباعة القيمة التي قام المستخدم بإدخالها أليا.
نفرض أن المستخدم قام بكتابة الرقم44 فبعدما يقوم بالضغط على entr في لوحة المفاتيح، عندها سيأخذ المتغير(scalar) $in في الحال القيمة **"44\n"**، إفترضنا أننا لا نحتاج إلى رمز الإنتقال إلى سطر جديد(**\n**) لهذا إستعملنا الدالة **chomp()** التي تقوم بإزالة الرمز الذي يؤدي إلى سطر جديد و الموجود في آخر الجملة.

ملاحظة : نحصل على رمز سطر جديد بالضغط على المفتاح entr في لوحة المفاتيح.
(يمكن إستعمال **chop** بدل **chomp**)

Blocks (6-3

التعامل مع ال**Blocks**
مثال:

```
#!/usr/perl/bin/
#block1.pl
use warnings ;
use strict ;
my $var=4 ;          #1
print "\$var =$var\n" ;
{
print "{\n" ;
 print "This is block scoped\n";
my $var=8 ;          #2
print "\$var =$var\n}\n" ;
}
print "\$var =$var\n" ;     #3
```

```
C:\>perl block1.pl
4
{
This is block scoped
8
}
4
C:\>
```

المتغيرات التي يتم الإعلان عنها داخل block فهي وحدها (إن صح التعبير) ينحصر عملها داخل block و ليست قابلة للإستعمال إذا وجدت خارج مجال الblock .

```
#!/usr/perl/bin/
#block2.pl
use warnings ;
use strict ;
{
print "{\n" ;
 print "This is block scoped\n";
my $var=4 ;
print "\$var=$var\n}\n" ;
}
print "\$var=$var\n" ;     # $var  لايوجد هنا لأننا قمنا بالإعلان عنه أو بتعريفه داخل block فقط
```

هنا يوجد خطأ، لم يتم التعرف على المتغير، الذي قمنا بالإعلان عنه فقط داخل مجال block.

```
#!/usr/perl/bin/
#block3.pl
use warnings ;
use strict ;
my $var=4 ;
print "\$var=$var\n" ;      #1
{
print "{\n" ;
 print "This is block scoped\n";
$var=8 ;            #2
print "\$var=$var\n}\n" ;
}
print "\$var=$var\n" ;      #3
```

```
C:\>perl block3.pl
$var=4
}
This is block scoped
$var=8
{
$var=8
C:\>
```

هنا لم نقم بالإعلان عن المتغير داخل مجال block و إنما قمنا بإعطائه قيمة جديدة فقط ،لهذا أخذ المتغيرvar$ القيمة 8 عند طباعة var$ في المرة الثالثة.

7.3)القيمة undef

هذه قيمة خاصة تعني(undefined) أي (غير محدد أو مبين ....).
هي كذالك (default value) القيمة الإفتراضية التي تأخذها المتغيرات scalars إن لم تكن قد أعطيت المتغيرات قيمة ما مسبقا. مثلا

```
my $var ;
```

هذه الكتابة مماثلة ل

```
my $var=undef ;
```

نستطيع أن نعطي هذه القيمة لمتغير ما بعد أن نكون أعطيناه قيمة مسبقة مثلا:

```
#!/usr/perl/bin/
#undef1.pl
use warnings ;
use strict ;
```

```
my $var=44 ;
print $var ;
$var=undef ;   # undef($var) أو هذه الكتابة
print $var ;
```

عند التنفيذ

```
C:\>perl undef1.pl
Use of uninitialized value in print at C:\p.pl line 9.
44
C:\>
```

الإنذار الأول الذي يذل على أن المتغير الذي تريد طباعة محتواه لم تقم بإعطائه قيمة ما
هذا الإنذار ظهر لكوننا قمنا باستعمال **use warnings**

إذا إحتجت أن تفحص متغير scalar ما هل به قيمة لاتساوي قيمة undef ، إذا في هذه الحال نستعمل هذه الدالة **defined مثلا**
if(defined($var))..... هذا الفحص يعطي صحيح (**true**) إذا كان **$var** به قيمة ما تخالف **undef**

ملاحظة:

If($var !=undef)..... هذه الكتابة خاطئة

Operators (8.3

تعتبر المعاملات هي الوسيط الأساسي للتعامل بين قيم المتغيرات.
وتنقسم المعاملات إلى عدة أقسام كما يلي:

Arithmetic Operators
Bitwise Operators
Concatenation Operators
Comparison Operators

Arithmetic Operators (1.8.3

الرمز	دلالة
+	من أجل عملية الجمع ($x=$y+85)
-	من أجل عملية الطرح . (-) أيضا تستخدم للدلالة على عدد سالب ($x= -26-$y)
*	من أجل عملية الضرب ($x=$y*$z)
/	من أجل عملية القسمة ($x=2/22)----> 0.0909090909090909
%	من أجل module (باقي قسمة عدد)($x=2%22) -----> 2
**	من أجل exponentiation ($x=4**2) ------> 16
++	Auto-increment تزايد بواحد قبل إرجاع النتيجة. مثال ($y++) .

يمكن أيضا كتابتها على هذا الشكل ($y++) لكن هنا لا يتم إرجاع القيمة التي أخذها المتغير ($y). تكافئ تقريبا هذه الكتابة $y=$y+1	
Auto-decrement تتناقص بواحد قبل إرجاع النتيجة. مثال:(--$y) or ($y--) (نفس الملاحظة السابقة عند إستعمال++) تكافئ تقريبا هذه الكتابة $y=$y-1	--

لنرى معا هذه الأمثلة:

```
#!/usr/perl/bin/
# Numeric Operators1.pl
use warnings ;
use strict ;

my $var=4 ;

print "++\$var=", ++$var, "\n" ;
```

```
C:\>perl Numeric Operators1.pl
++$var=5

C:\>
```

المثال واضح تمت زيادة واحد أي تم تطبيق هذه العملية $var=$var+1 أي $var=4+1 .
وبهذا تم إختصار الكتابة السابقة بالتالي ++$var

مثال لحالة ثانية:

```
#!/usr/perl/bin/
# Numeric Operators2.pl
use warnings ;
use strict ;

my $var=4 ;

print "\$var++=", $var++, "\n" ;
```

```
C:\>perl Numeric Operators2.pl
$var++=4

C:\>
```

هنا ربما كنت تنتظر أن يطبع العدد رقم 5 ، لكن كما قلنا سابقا ، الذي حصل هو أن المتغير $var أخذ العدد 5 لكن لم يتم إرجاعه. قم بطباعة $var مرة ثانية لترى النتيجة.
أنظر الأمثلة التالية لتتضح الفكرة

نفس المثال لكن هنا سنطبع المتغير مرتين متتاليتين ، لنرى النتيجة

```
#!/usr/perl/bin/
# Numeric Operators2.pl
use warnings ;
use strict ;

my $var=4 ;

print "\$var++=", $var++, "\n" ;
print "\$var++=", $var++, "\n" ;
```

```
C:\>perl Numeric Operators2.pl
$var++=4
$var++=5

C:\>
```

بالنسبة للمرة الأولى التي قمنا فيها بطلب الطباعة وقع كما قلنا أي عند تنفيذ التالي

```
print "\$var++=", $var++, "\n" ;
```

يحصل التالي سيتم طباعة العدد 4 وفي نفس الوقت سيأخذ المتغير $var القيمة 5

أما بالنسبة للمرة الثانية أي عن تنفيذ التالي:

```
print "\$var++=", $var++, "\n" ;
```

يحصل التالي سيتم طباعة العدد 5 وفي نفس الوقت سيأخذ المتغير $var القيمة 6
وهكذا.....

مثال لحالة آخرى:

```
#!/usr/perl/bin/
# Numeric Operators3.pl
use warnings ;
use strict ;



my ($var,$num)=(4,2) ;

$num=++$var ;
print "\$num=", $num, "  \$var=", $var, "\n" ;
```

```
C:\>perl Numeric Operators3.pl
$num=5   $var=5
C:\>
```

المثال واضح . ما الذي حصل؟

عند تنفيذ التالي:

```
$num=++$var ;
```

سيتم إضافة العدد واحد إلى القيمة التي توجد لدا المتغير $var ومنه $var سيساوي 5 في نفس الوقت سيتم إرجاع النتيجة ، إذا هذه النتيجة التي تم إرجاعها هي التي سيأخذها المتغير $num إذا $num سيساوي 5

مثال لحالة آخرى:

```
#!/usr/perl/bin/
# Numeric Operators4.pl
use warnings ;
use strict ;

my ($var,$num)=(4,2) ;

$num=$var++ ;
print "\$num=", $num, "  \$var=", $var, "\n" ;
```

```
C:\>perl Numeric Operators4.pl
$num=5   $var=5
C:\>
```

المثال واضح . ما الذي حصل؟

سيتم إضافة العدد واحد إلى القيمة التي توجد لدا المتغير $var ومنه $var سيساوي 5، بينما لن يتم إرجاع النتيجة ، إذا بما أنه لم يتم إرجاع أي قيمة، فإن المتغير $num سيساوي 4

مثال لحالة آخرى:

نناقش هنا الحالة التي نستعمل فيها ++ مع سلسلة رموز أو حروف

```
#!/usr/perl/bin/
# Numeric Operators5.pl
use warnings ;
use strict ;

my $var ;

print "++(\$var=\" 99\")", ++($var= "99"), "\n" ;   #100
print "++(\$var=\"a0\")", ++($var= "a0"), "\n" ;    #a1
print "++(\$var=\"Az\")", ++($var= "Az"), "\n" ;    #Ba
print "++(\$var=\"zz\")",  ++($var= "zz"), "\n" ;     #aaa
```

```
C:\>perl Numeric Operators5.pl
++(\$var=\" 99\") =100
++(\$var=\"a0\")=a1
++(\$var=\"Az\")=Ba
++(\$var=\"zz\")=aaa

C:\>
```

المثال واضح . ما الذي حصل؟

لتنفيذ

```
print "++(\$var=\" 99\")", ++($var= "99"), "\n" ;
```

يتم زيادة واحد إلى 9 نحصل على 10 نكتب 0 ونحتفظ بواحد، هذا الواحد نضيفه إلى 9 التي توجد على اليسار فنحصل على 10 إذا المجموع هو 100

لتنفيذ
```
print "++(\$var=\"a0\")", ++($var= "a0"), "\n" ;    #a1
```
يتم زيادة واحد إلى 0 أي سنحصل على 1 ، لا يوجد أي إحتفاظ إذا لن نغير a
ومنه النتيجة الأخيرة هي a1

لتنفيذ
```
print "++(\$var=\"Az\")", ++($var= "Az"), "\n" ;    #Ba
```
يتم زيادة واحد ،أي ننتقل إلى الحرف الذي يلي z ،
نحن نعلم أن z هو الحرف الأخير من حيث الترتيب ،
إذا في هذه الحالة نعود إلى الحرف الأول من حيث الترتيب الأبجدي (a) ونحتفظ بواحد، هذا الواحد نضيفه إلى الحرف الثاني(A) أي - ننتقل إلى الحرف الذي يلي (A) - أي (B)
ومنه نحصل على Ba
ملاحظة: نحافظ على حالة الحرف إذا كان كبير أو صغير

لتنفيذ
```
print "++(\$var=\"zz\")",  ++($var= "zz"), "\n" ;
```
يتم زيادة 1 أي ننتقل إلى الحرف الذي يلي z الذي هو a ونحتفظ ب1
هذا الإحتفاظ يتيح الإنتقال إلى الحرف الذي يلي الحرف الثاني (z) أي سيصبح a ونحتفظ ب 1
هذا الإحتفاظ الأخير جعلنا نظيف حرف a آخر
منه نحصل على aaa
أرجو أن الفكرة وصلت. هذا شرح غير متعمق.
ربما لتتضح الفكرة أكثر راجع جدول ASCII

ملاحظة: هذه الميزة ليست متوفرة في Auto-decrement (--) ، فعند إستعمالها مع سلسة حروف ترجع في الأغلب -1

ملاحظة:

عند إستعمال (++) مع متغير به قيمة undef فا undef يأخذ القيمة 0 في الأغلب
مثلا
```
Undef($var) ;
print  $var++ ;
```
عند التنفيذ سيطبع 0 ، وكما ذكرنا سابقا فإن $var سيأخذ القيمة 1

كل هذه الأمثلة صالحة بالنسبة لAuto-decrement (--) إلا في بعض الحالات التي تم الإشارة إليها.

Bitwise Operators (2.8.3

p	q	P&q	P|q	P^q	~p
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

الإحتمالات الشائعة لكل عملية

1=صحيح=True
0=خطأ=False

P&q <---- p وq

P|q <---- p أوq
P^q <---- p exclusive or q
~p <---- ليس p (أو عكس p)

الرمز	دلالة
~	من أجل النفي(not) (أنظر للأمثلة في الأسفل)
&	and
|	or
^	(xor) exclusive or
<<	Shift Operators(نقل) نحو اليسار
>>	Shift Operators نحو اليمين

تلخيص بسيط

مثال بسيط لكل من(~ & | ^ << >>)

The '&' operator

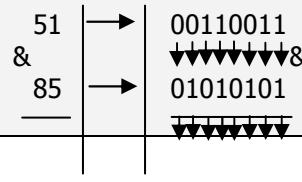
```
#!/usr/perl/bin/
#and1.pl
use warnings ;
use strict ;

print "51 \& 85=" ,51&85 ;  #17
```

```
C:\>perl and1.pl
51 & 85=17
C:\>
```

51 & 85 تساوي 17
يعني :
51إذا قمنا بتحويلها إلى النظام BInary نحصل على 00110011
85إذا قمنا بتحويلها إلى النظام BInary نحصل على 01010101
بعدها قم بإجراء العملية التالية (01010101 & 00110011) (إستعمل الجدول الأول فوق كمؤنس)
حسب الجدول المذكور فإن :
1و1 يعطي 1
1و0 يعطي 0
0و1 يعطي 0
0و0 يعطي 0

النتيجة التي نحصل عليها هي00010001 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي17.

يمكن تلخيص ذلك في:



```
17  ←  00010001
```

The '|' operator

```
#!/usr/perl/bin/
#or1.pl
use warnings ;
use strict ;

print "51\|85=" ,51|85 ;  #119
```

```
C:\>perl or1.pl
51|85=119
C:\>
```

51|85 تساوي 119

(51 | 85)

قم بإجراء العملية التالية (01010101 | 00110011) (إستعمل الجدول الأول فوق كمؤنس)

النتيجة التي نحصل عليها هي01110111 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي119.

يمكن تلخيص ذلك في:

```
51  →  00110011
|              |
85  →  01010101
---            ---
119 ←  01110111
```

The '^' operator

```
#!/usr/perl/bin/
#xor1.pl
use warnings ;
use strict ;

print "51\^85=" ,51^85 ;  #102
```

```
C:\>perl xor1.pl
51^85=102
```

```
C:\>
```

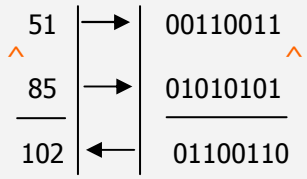
85^51 تساوي 102

(51 ^ 85)

قم بإجراء العملية التالية (01010101 ^ 00110011) (إستعمل الجدول الأول فوق كمؤنس)

النتيجة التي نحصل عليها هي01100110 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي102.

يمكن تلخيص ذلك في:



The '~' operator

```
#!/usr/perl/bin/
#not1.pl
use warnings ;
use strict ;

print "\~85=" , ~85 ;   #4294967210
```

```
C:\>perl not1.pl
~85=4294967210
C:\>
```

85~ تساوي 102

(~85)

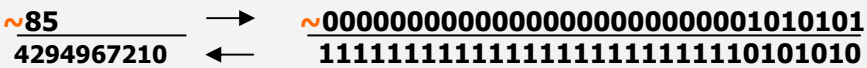
قم بإجراء العملية التالية (01010101 ~) (إستعمل الجدول الأول فوق كمؤنس)

النتيجة التي نتوقعها حسب الجدول هي10101010 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي170. لكن هذه النتيجة خاطئة بمأننا في الأغلب نستخدم processeur 32 bits إذان :

العملية التي نقوم بها هي (01010101~) أي (00000000000000000000000001010101~)

النتيجة التي نحصل عليها هي 11111111111111111111111110101010بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي4294967210.

يمكن تلخيص ذلك في:



The '<<' operator

```
#!/usr/perl/bin/
# Shift2.pl
use warnings ;
use strict ;

print "51\<<2=" ,85<<2 ;   #340
```

```
C:\>perl Shift2.pl
85<<2=340
C:\>
```

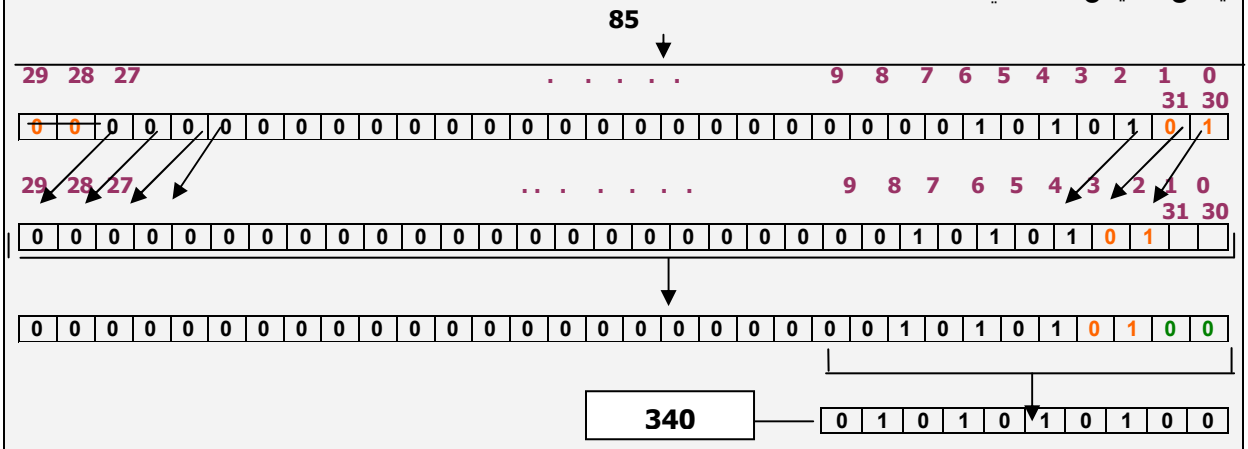
85<<2 تساوي 340

(85<< 2)

قم بإجراء العملية التالية (00000010 << 01010101)

النتيجة التي نحصل عليها هي 0101010100 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي340. (تذكر أننا نفترض أننا نستخدم processeur 32 bits)

يمكن تلخيص ذلك في:



رسم بياني يفسر عملية الإنتقال نحو اليسار(حسب المثال طلبنا أن يتم الإنتقال بخانتين(2bits) نحواليسار)

The '>>' operator

```
#!/usr/perl/bin/
# Shift2.pl
use warnings ;
use strict ;
my $var=85 ;
print "85\>>3=" ,$var>>3 ;   #10
```

```
C:\>perl Shift2.pl
85>>3=10
C:\>
```

3>>85 تساوي 340

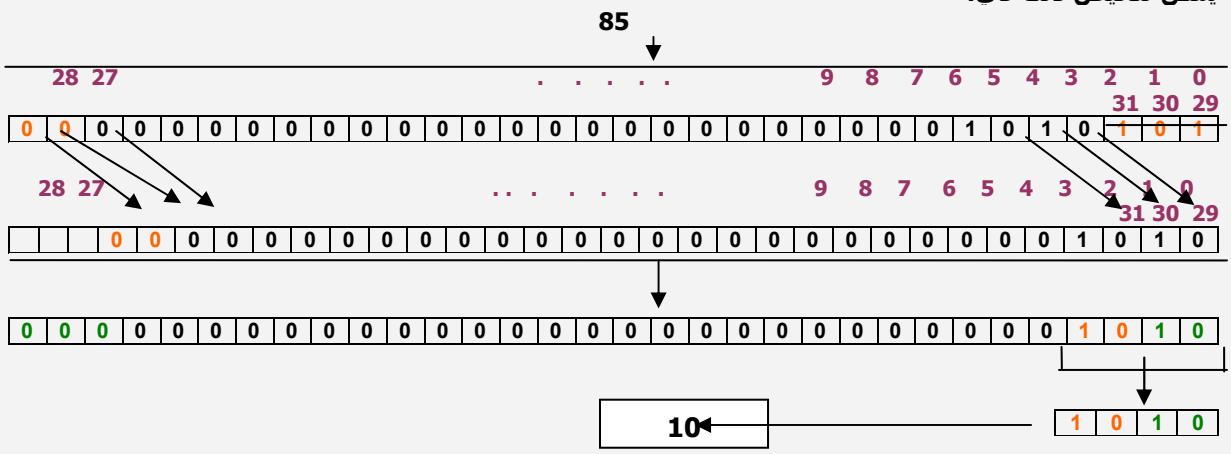
(85>> 3)

قم بإجراء العملية التالية (00000011 >> 01010101)

النتيجة التي نحصل عليها هي 00001010 بعدها نحولها إلى النظام العشري لتكون النتيجة الأخيرة هي10.

(تذكر أننا نفترض أننا نستخدم processeur 32 bits)

يمكن تلخيص ذلك في:



رسم بياني يفسر عملية الإنتقال نحو اليمين(حسب المثال طلبنا أن يتم الإنتقال ب3 خانات (3bits) نحو اليمين)

Concatenation Operators(3.8.2

(**concatenation**)للتوحيد بين العمليات وسلاسل نصية التي سيتم طباعتها نضع الفاصلة(,) بينهما ولا نضع علامة زائد (+) .
لنضع (+) بدل (,) ونرى النتيجة:

```
#!/usr/perl/bin/
# concatenation1.pl
use strict;
my ($var,$sum) ;
$var=44;
print "44+4=", $var+4,"\n" ;    #1

print "44+4="+ $var+4,"\n" ;    #2

print "44Hi4="+ $var+4,"\n" ;   #3

print "Hi44+4="+ $var+4,"\n" ;   #4
```

بعد التنفيذ:

```
C:\>perl concatenation1.pl
44+4=48
92
92
48

C:\>
```

1: في هذه الحالة نحصل على نتيجة عادية
2: هنا وعند إستعمال (+) بدل (,) نحصل على 92 وهي نتيجة غير متوقعة ، حيث تم إنجاز هذه العملية 4+44+44،
حيث تم إعتبار "44+4=" كأنه العدد 44 ففي هذه الحالة يتم أخذ العدد الأول من سلسلة النصوص.
3: نفس ماوقع في(2) ، أخذ العدد الأول من سلسلة النصوص("44Hi4=") أي 44 و إضافتها ل 48 (**$var+4**).
4: هنا لايوجد عدد في أول سلسلة النصوص ("Hi44+4=") ولهذا يعتبر 0 هو العدد الأول أي سيتم إضافة 0 ل48 .

ملاحظة: لو إستعملت (use warnings;) فسيتم إظهار تحذير يبين لك أن الذي قمت بإضافته للعدد 48 هو ليس عدد
مثال للتحذير الذي سيظهر لك:

Argument "44+4=" isn't numeric in addition (+) at C:\ concatenation1.pl line 25

أما بالنسبة للتوحيد بين سلسلة رموز مع سلسلة رموز أخرى نضع نقطة(.) بينهما.
لنرى هذا المثال:

```
#!/usr/perl/bin/
# concatenation2.pl
use warnings;
use strict;
my ($part1,$part2)=( "Hello","world") ;
print $part1." ".$part2."\n" ;
```

```
C:\>perl concatenation2.pl
Hello world

C:\>
```

أظن أن المثال واضح

مثال آخر:

```
#!/usr/perl/bin/
# concatenation3.pl
                #    --->     يعني     سيطبع
use warnings;
use strict;
my ($part1,$part2)=( "Hello","world") ;
print $part1." ".$part2," \n \n" ;  # "Hello world"
```

```
print $part1.$part2.3+2,"\n " ;  #"Hello world3"+2 ---> 0+2-->2 (مع إظهار تحذير )

print $part1.$part2,3+2,"\n\n" ;  #"Hello world",3+2 -----> Hello world5

($part1,$part2)=( 4,6) ;      #إعطاء قيم جديدة لكل من المتغيرين
print $part1.$part2.$part1+$part2."\n" ;     #    "464"+6     ------> 470
print $part1,$part2, $part1+$part2."\n\n" ;  #   "4","6",6+4 يطبع ----> 4610

print $part1.$part2."  ".$part1+$part2."\n" ;  #  "46  4"+6 ------> 52(مع إظهار تحذير)
print $part1.$part2."  ", $part1+$part2."\n" ;  # "46  ",4+6 -----> 46  10
```

```
C:\>perl concatenation2.pl
Hello world

Argument "Helloworld3" isn't numeric in addition (+) at C:\
concatenation3.pl line 9.
2
Helloworld5

470
4610

Argument "46  4" isn't numeric in addition (+) at C:\ concatenation3.pl line
17.
52
10  46

C:\>
```

أظن أن المثال واظح:

ملاحظة : **هنا إستعملنا (use warnings;) لهذا تم إظهار التحذيرات.**

أرجو أن نكون قد إستوعبنا الفرق بين إستخدام الفصلة (,) و (+) و نقطة(.) عند إستخدامهما كمعاملات **Concatenation**

Comparison Operators(4.8.3

تستخدم لإجراء عمليات المقارنة

للمقارنة بين الأرقام	للمقرنة بين الحروف	الدلالة
==	eq	يرجع صحيح إذا كان المعامل الموجود يسارا [(==) أو (eq)] هو يساوي المعامل الموجود على اليمين
<	lt	يرجع صحيح إذا كان المعامل الموجود يسارا [(<) أو (lt)] هو

أصغر قطعا من المعامل الموجود على اليمين		
يرجع صحيح إذا كان المعامل الموجود يسارا [(<=) أو (le)] هو أصغر من أو يساوي المعامل الموجود على اليمين	le	<=
يرجع صحيح إذا كان المعامل الموجود يسارا [(>) أو (gt)] هو أكبر قطعا من المعامل الموجود على اليمين	gt	>
يرجع صحيح إذا كان المعامل الموجود يسارا [(>=) أو (ge)] هو أكبر أو يساوي المعامل الموجود على اليمين	ge	>=
يرجع صحيح إذا كان المعامل الموجود يسارا [(= !) أو (ne)] هو لا يساوي المعامل الموجود على اليمين	ne	!=
يرجع 1 أو 0 أو 1- إذا كان المعامل الموجود يسارا [(<=>) أو (cmp)] هو على التوالي (أصغر أو يساوي أو أكبر) من المعامل الموجود على اليمين	cmp	<=>

lt == less *than* == أصغر من
gt == greater *than* == أكبر من
le == less *than or equal to* == أصغر من أو يساوي
ge == greater *than or equal to* == أكبر من أو يساوي

مثال:

```
#!/usr/perl/bin/
# Comparison1.pl
use warnings;
use strict;


print 1==1, "\n" ;  # 1
print 1==2, "\n\n " ;  #

print "m" eq "m" , "\n" ;  # 1
print "m" eq "f" , "\n\n " ;   #

print 2<4, "\n" ;  # 1
print ("a" lt "b" , "\n") ;  # 1

print 8>3, "\n" ;  # 1
print ("c" gt "a" , "\n\n ") ;   # 1

print 3 !=1, "\n" ;  # 1
print ("z" ne "k" , "\n\n ") ;   # 1

print 3<=>3 , "\n" ;  # 0
print 3<=>2 , "\n" ;  # -1
```

```
print 2<=>3 , "\n" ; # 1
print ("m" cmp "m" , "\n ") ;  # 0
print ("m" cmp "t" , "\n ") ;  # -1
print ("t" cmp "m" , "\n ") ;  # 1
```

```
C:\>perl Comparison1.pl
1


1


1
1
1
1

1
1

0
1
1-
0
1-
1

C:\>
```

أظن أن المثال واضح:
عندما لا يتم طباعة شيء فالأغلب يعني ذلك أن هناك خطأ ما (False)

Logical Operators (5.8.3

المعاملات المنطقية تستخدم لإجراء مقارنة منطقية، وغالبا ما تكون أطراف المقارنة إحدى القيم 1 (True) أو 0 (False).
عند إجراء مقارنة يكون ناتج هذه المقارنة إحدى القيم إما True أو False

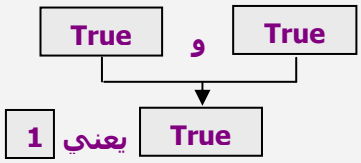
الرمز	الدلالة
&& أو and	تسمح بتحقيق عملية بين شرطين (و)
|| أو or	تسمح بتحقيق عملية بين شرطين (أو)
xor	يسمح بتحقيق عملية بين شرطين (exclusive or)
! أو not	يسمح بتحقيق عملية النفي لشرط ما

مثال:

```
#!/usr/perl/bin/
# Logical1.pl
use warnings;
use strict;


print (((1==1) && (2==2)), "\n") ;  # 1
print (((1==1) and (2==2)), "\n") ;  # 1
print (((1==1) && (2==7)), "\n ") ;  #
print ((("t" eq "m") && ("t" eq "m")) , "\n\n") ;   #

print (((1==1) || (2==2)), "\n");  # 1
print (((1==1) or (2==2)), "\n") ;  # 1
print (((1==1) || (2==7)), "\n ") ;  # 1
print ((("t" eq "m") || ("t" eq "m")) , "\n\n") ;   #


print (((1==1) xor (2==1)), "\n\n") ;  # 1

print ((not (2==1)), "\n") ;  # 1
print (( ! (2==1))) ;  # 1
```

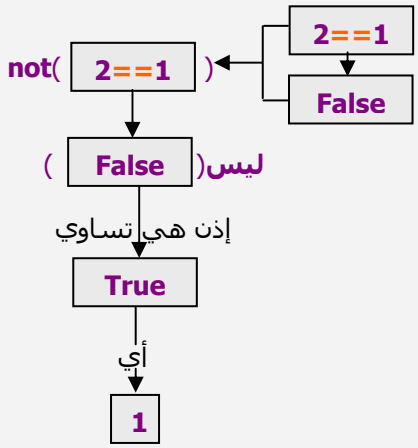
```
C:\>perl Logical.pl
1
1



1
1
1


1

1
1
C:\>
```

بالنسبة ل((1==1) && (2==2))



للإستعاب أكثر إستعن بالجدول السابق
True أي 1
False أي 0

بالنسبة ل (not (2==1))



ملاحظة:
عندما لا يتم طباعة شيء فالأغلب يعني ذلك أن هناك خطأ ما (False)

String Operators (6.8.3

الرمز	الدلالة
.	(النقطة) خاص بتوحيد سلسلة (concatenation) رموز(string) مع أخرى . مثال: $str="abc"."defgh" ; عند طباعة $str نحصل على abcdefgh
x	(حرفx صغير) يسمح بتكرار سلسلة رموز معينة لعدد من المرات أنت تحدده مثال: $str="a"x6 ; عند طباعة $str نحصل على aaaaaa نلاحظ أنه تم تكرار حرف a ست مرات
=~	تستخدم على العموم في

التعبير المنتظم(Regular Expression) (سنتطرق لهذا لاحقا)	

ملاحظة:

يوجد كذالك الأشكال المختصرة للمعاملات الرئيسية (مثل c) . نجد في هذا الصدد :
=<<, .=, /=, *=, -=, +=, وهكذا.....

ملاحظة:

(..) (نقطتين) تدل إلى مجال ما
مثلا:

```
print (0..6) ;
```

سيتم طباعة:

0123456

إذا 0..6 تدل على المجال التالي [0,6]
مثال أخر:

```
print ("a".."g") ;
```

سيتم طباعة:

Abcdefg

9.3) بعض الدوال الخاصة ب String

الدالة	تفسير
uc()	**(upper case) تمكن من تحويل الحروف إلى حروف كبيرة** **مثال:** print (uc("Palestine")) **سيطبع :** PALESTINE
lc()	**(lower-case) تمكن من تحويل الحروف إلى حروف صغيرة** **مثال:** print (lc("PALESTINE ")) **سيطبع :** palestine
ucfirst()	**تمكن من تحويل الحرف الأول من جملة ما إلى حرف صغيرة** **مثال:** print (ucfirst("PALESTINE ")) **سيطبع :** pALESTINE
lufirst()	**تمكن من تحويل الحرف الأول من جملة ما إلى حرف كبيرة** **مثال:** print (lcfirst("palestine")) **سيطبع :** Palestine
chr()	تستعمل لتحويل رقم ما إلى سلسلة رموز (**string**) **مثال:** print (chr(112)) **سيطبع :** p
ord()	تستعمل لتحويل سلسلة رموز (**string**) إلى ASCII **مثال:** print (ord("p")) **سيطبع :** 112 **ملاحظة: النتيجة هي بDecimal**
length()	هذه الدالة تقوم بإرجاع عدد، هذا العدد يدل على طول سلسلة الرموز (**string**) **مثال:** print (length("Palestine")) **سيطبع :** 9

index()	هذه الدالة تقوم بإرجاع رقم، هذا العدد يدل على موقع الحرف الذي نبحث عن موقعه **مثال:** `print (index("Palestine","e"))` **سيطبع :** 3 **ملاحظة: يمكننا أن نحدد موقع بداية البحث.** **مثلا:** `print (index("Palestine","e",1))` **سيطبع :** 3 **نفرض أننا حددنا له أن يبدأ البحث من الموقع رقم 4** **مثلا:** `print (index("Palestine","e",4))` **سيطبع :** 8 بدأنا البحث من الحرف رقم 4 أي من الحرف s إذا بدأنا البحث بعد e الأولى لهذا تمت طباعة 8 التي تدل على حرف e الذي يوجد في الموقع ذات الرتبة 8. **ملاحظة: إذا لم يتم العثور على الحرف الذي نبحث عليه سترجع الدالة -1** 0123 78 **ملاحظة:موقع أول حرف هو الرقم 0 . إذا** Palestine
rindex()	على العموم هذه الدالة مثل index() ، الإختلاف الملاحظ بينهما هو أن **index()** تبدأ البحث ابتداء من أول حرف إلى الحرف الأخير ، بينما **rindex()** تقوم بالبحث من أخر حرف إلى أول حرف **مثال:** `print (rindex("Palestine","e"))` **سيطبع :** 3 **e توجد في موقعين، الموقع الذي يأتي في الرتبة رقم3 و الموقع الذي يأتي في الرتبة رقم 8 .** **بما أن الدالة تبدأ البحث من أخر حرف و e توجد في الموقع الأخير من الكلمة الذي يأتي في الرتبة رقم 8 . لهذا تم طباعة 8 ولم يتم طباعة 3 الذي يدل على موقع e الثاني** **ملاحظة: يمكننا أن نحدد موقع بداية البحث.** **مثلا:** `print (rindex("Palestine","e",4))` **سيطبع :** 3 قمنا بتحديد موقع إنطلاق البحث وهو 4 . (كما ذكرنا الدالة تبدأ البحث من أخر حرف في إتجاه أول حرف)، إذا تم إعتبار الحرف الموجود في الموقع ذات الرتبة الرابعة هو آخر حرف . لهذا تم طباعة 3 لأنه هو أول موقع تم العثور فيه على e **ملاحظة: perl يفرق بين الحروف الكبيرة والحروف الصغير ، لهذا وجب عليك إتخاذ ذلك في الحسبان أثناء البحث عن موقع أي حرف.**
substr()	تمكننا من تحديد جزأ من سلسلة رموز (string) ما الذي سيطبع أو الذي سيبدل بسلسلة رموز(string) آخرى مثلا أو يمككنا من تحديد المكان الذي سنزيد فيه سلسلة رموز (string) جديدة ضمن سلسلة الرموز الموجودة مسبقا. يمكننا أن نلخص بارامتر هذه الدالة بالتالي: **substr(a,b,c,d)** **a**---> بدلا منها نضع التعبير المراد العمل عليه **b**-----> بدلا منه نضع الرقم الذي يدل على الموقع الذي سيكون بداية التحديد **c** ---- -> بدلا منه نضع طول أو عدد الرموز المراد تحديدها

chomp()	هذه الدالة تستخدم بكثرة لمسح و إزالة (القفز لسطر جديد) (**\n**) الموجود في آخر الجملة
	مثلا :
	`my $d="abcd\n defg\n end\n" ;`
	`chomp ($d) ;`
	`print $d ;`
	`print "DELETE" ;`
	
	سيطبع:
	abcd
	defg
	endDELETE
	
	نفرض أننا لم نستعمل الدالة chomp() النتيجة المتوقعة في هذه الحالة هي:
	abcd

defg
end
DELETE
منه نستنتج أنه عندما إستعملنا chmop() فقد تم إزالة (\n) الموجودة في الرتبة الآخيرة
 	
ملاحظة :chomp() تقوم بإرجاع عدد ما قامت بحذفه
 	
لنلاحظ هذا المثال:
`my $d="varible1_line1\nvarible1_line2\nvarible1_line3\n" ;`
`my $z=" VARIABLE2\n";`
`print "Return", chomp ($d,$z),"\n" ;  #1`
`print $d ;`
`print $z ;`
`print " DELETE" ;`
 	
سيطبع:
Return2
varible1_line1
varible1_line2
varible1_line3 VARIABLE2 DELETE
#1-----> تم حذف (**\n**) الموجود في الموقع الأخير في كل من المتغيرين **$d** و **$z** وذلك في وقت واحد، و في نفس الوقت طلبنا طباعة ما سيتم إرجاعه من طرف الدالة **chomp() .**
هنا تم إرجاع 2 ويدل ذلك أن الدالة قد قامت بحذف (**\n**) مرتين حيث الأولى تم حذفها من المتغير **$d** والثانية من المتغير **$z** .	

هذه الدالة تقوم بحذف رمز أو حرف أو رقم من أخر سلسلة رموز ما (**string**)
مثلا:
`my $d="abcd" ;`
`print "Return ",chop ($d),"\n" ;`
`print $d ;`
 	
سيطبع:
Return d
abc
قمنا بطلب حذف الحرف الأخير من الكلمة التي يحتويها المتغير **$d**، وفي نفس الوقت قمنا بطلب طباعة ما سيتم إرجاعه من طرف الدالة **chop()** ، وفي الأخير طلبنا بطباعة المحتوى النهائي للمتغير **$d** .
 	
ملاحظة : الدالة chop() ترجع ما قامت بحذفه
	chop()

3.10) بعض الدوال الخاصة ب Numbers

الدالة	تفسير
abs	يرجع أي قيمة إلى القيمة المطلقة مثال my $var1 = abs(-3.4); # var1 is 3.4 my $var2 = abs(5.9); # var2 is 5.9

int	تستخدم من أجل تحويل الأعداد العشرية إلى الأعداد الصحيحة
	مثال
	my $price = 9.95;
	my $dollars = int ($price);
	Dollars هو 9 ، ليس 10 وهذا خاطأ من أجل الإعلان عن السعر
	
	يمكن أن تكتب التالي
	my $price = 9.95;
	my $dollars = sprintf("%.0f", $price);
	في هذه الحالة Dollars هو 10 (كما رأينا في (Standard output

sqrt	هذه الدالة تستعمل في حساب الجدر المربع مثال my $var1 = sqrt(16); # var1 is 4
**	تستعمل للدلالة على الأس مثال my $var1 = 4**2; # var1 is 16
exp	ترجع e أس القيمة المعطاة بحيث e= 2.71828182845905 مثال my $var1 = exp(0); # var1 is (e**0=1) my $var2 = exp(1); # var2 is (e**1=2.71828182845905) my $var3 = exp(8); # var3 is (e**8= 2980.95798704173)
log	ترجع عكس الدالة ()exp مثال لاحظ : my $var1 = exp(0); # var1 is 1 my $var1 = log(1); # var1 is 0)
rand	ترجع قيمة عشوائية لكن إذا أعطيتها قيمة محددة فالقيمة العشوئية التي ترجعها هذه الدالة تكون محصورة بين 0 و القيمة المعطات (القيمة المعطات <= القيمة العشوائية التي ترجعها الدالة <= 0) أما في حالة لم تعطي لها أي قيمة محددة فالقيمة العشوئية التي ترجعها هذه الدالة تكون محصورة بين 0 و واحد (1 > القيمة العشوائية التي ترجعها الدالة <= 0)

hex	يمكن القول أن هذه الدالة تقوم بتحويل القيمة المعطاة لها من النظام الست عشري إلى النظام العشري.. مثال hex("E5")=229
oct	يمكن القول أن هذه الدالة تقوم بتحويل القيمة المعطاة لها من octal إلى النظام العشري

ملاحظة:هناك دوال أخرى مثل .....cos,sin,tan

4) جمل الشرط والتكرار

سنتعلم في هذا الجزء كيف نستخدم perl في تنفيذ تعليمات محددة دون غيرها وفق شروط معينة، وأيضا تكرار لعدد معين من المرات لتعليمة ما .

* جمل الشرط **Conditional Statements**
- **If**
- **unless**
- **switch**

*جمل التكرار والحلقات **Iteration Statements**
- **while**
- **do while**
- **for**
- **foreach**

4.1) الجمل الشرطية

4.1.1) if

للإنجاز إختبار بسيط

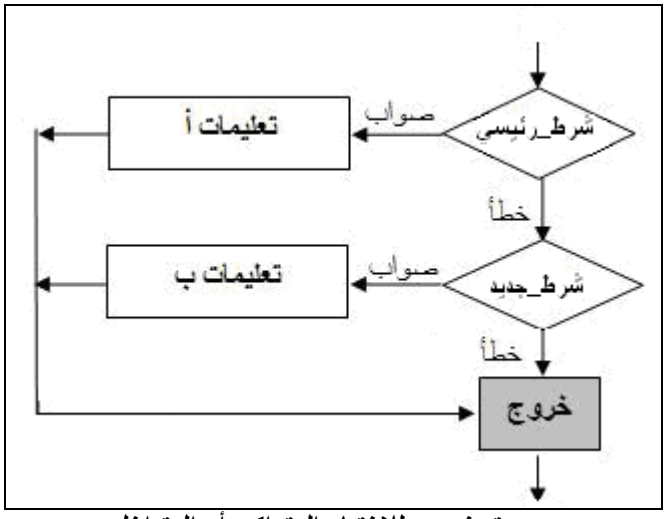
```
If (شرط_رئيسي ){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي صحيح
}
```

للإنجاز إختبار مركب

```
If (الشرط_رئيسي){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي صحيح
}
  else {
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي خاطئ
}
```

لإنجاز إختبار متراكب أو متداخل

```
If (شرط_رئيسي){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_الرئيسي صحيح
}
  elsif (شرط_جديد){
        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_الرئيسي خاطئ
        والشرط_الجديد صحيح
}
else {
        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_الرئيسي خاطئ
        والشرط_الجديد خاطئ
}
```



رسم توضيحي للإختبار المتراكب أو المتداخل

unless (4.1.2

وظيفة **unless** تقريبا مثلها مثل وظيفة **if** لكن بشكل معكوس (إن صح التعبير):

Unless تعني بالعربية: ما لم، إلا إذا

بشكل مختصر أنظر للتالي:

if	unless
if (1>2){ **print ("if")** **}** **هنا لن يطبع شيء**	**Unless (1>2){** **print ("unless")** **}** **هنا سيطبع unless**

من هذا يمكن أن نكتب التالي:

للإنجاز إختبار بسيط

```
unless (شرط_رئيسي){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي خاطئ
}
```

للإنجاز إختبار مركب

```
unless (شرط_رئيسي ){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي خاطئ
}
  else {
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يكون فيها الشرط_رئيسي صحيح
}
```

ملاحظة:

unless (condition)... <====تكافئ====> If (!(condition))...

مع العلم أن **condition** المستخدم في **unless** نفسه المستخدم في **if**

ملاحظة:

توجد كتابة آخرى يمكن إستعمالها في perl :

command if (condition) <-----

مثال `print ("free=Palestine") if($palestine==$mokawama)`
سيطبع:
free=Palestine
وذلك في حالة أن الشرط هو صحيح

command unless (condition) <-----
مثال `print ("free#Palestine") unless($palestine==$mokawama)`
سيطبع :
free#Palestine
وذالك في حالة أن الشرط هو خاطئ

ملاحظة:

يوجد في perl كتابة مختصرة يمكن إستعمالها في حالة الجمل الشرطية:

`condition ? command : command`
مثال:
`(1<2) ? (print "true") : (print "false")`
سيطبع:
true
وذلك لكون الشرط صحيح فلو كان الشرط خاطئ فسيطبع false

مثال آخر:
`$i=($a==$b) ? $a : $b ;`
$i ستأخذ القيمة التي توجد لدى $a إذا كان $a و $b لهما نفس القيمة،
أما في الحالة التي يكون فيها $a و $b لهما قيمتين مختلفتين فإن $i ستأخذ القيمة التي توجد لدى $b .

switch (4.1.3

عندما نريد إختبار متغير ما عدة مرات متتالية في شروط مختلفة من أجل التحقق من مختلف القيم الممكنة، يمكن أن نستعمل (**if**/**elsif**/**else**) لكن هده الطريقة طويلة ومملة .
مثال:

```
If($choice==0){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 0
}
elsif($choice==1){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 1
}

elsif($choice==2){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 2
}

elsif($choice==3){
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 3
}
else{
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي لن يتم فيها إختيار أي من الإختيارات السابقة
أي ليس 0 و 1 و 2 و 3
}
```

بدل هذه الطريقة المملة يتيح لنا perl **switch** وذلك مثل معظم اللغات

```
switch($choice){
                case 0 {
                        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 0
                      }
                case 1 {
                        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 1
                      }
                case 2 {
                        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 2
                      }
                case 3 {
                        التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 3
                      }
                else{
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي لن يتم فيها إختيار أي من الإختيارات السابقة
أي ليس 0 و 1 و 2 و 3
                      }
                  }
```

ملاحظة عند إستعمال **switch** يلزمنا إظافة **use Switch 'perl5'**

مثال:

```
#!/usr/bin/perl -w
use strict ;
use Switch 'perl5'; # S حرف كبير

print "entrez une valeur entre 0 et 4" ;
my $val;
$val=<STDIN>;
 chop($val);

switch ($val) { # هنا نكتب s  حرف صغير
                 case 0 {
          print "0" ;
                        }
                case 1 {
                        print "1" ;
                        }
                case 2 {
                        print "2" ;
                        }
                case 3 {
                         print "3" ;
                        }
                else{
                      print "erreur , entre une valeur entre 0 à 4 " ;
                        }
                   }
system "pause";
```

هنا سنتكلم عن بديل آخر وهو **given** الذي يعتبر من syntaxe المستقبلي الذي سنجده في perl6 ،
نصل إلى هذه الكتابة بكتابة perl6 بدل perl5
أي نكتب **use Switch 'perl6'**

```
given($choice){
        when 0 {
            التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 0
        }
        when 1 {
            التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 1
        }
        when 2 {
            التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 2
        }
        when 3 {
            التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي يتم فيها إختيار 3
        }
        default{
التعليمات الموجودة داخل هذا الbloc يتم تنفيذها في الحالة التي لن يتم فيها إختيار أي من الإختيارات السابقة
أي ليس 0 و 1 و 2 و 3 (بمعنى أنه هنا يتم وضع التعليمة أو البديل الإفتراضي)
        }
    }
```

ملاحظة: في حالة أننا نريد إستعمال كل من switch وgiven نكتب: `use Switch 'perl5', 'perl6' ;`

4.2) الجمل التكرارية

(4.2.1 while , until

while

```
while (الشرط) {
سيتم إعادة تنفيذ التعليمات التي ستكتب هنا
ما دام الشرط صحيح
}
```

until

```
until (الشرط) {
سيتم إعادة تنفيذ التعليمات التي ستكتب هنا
إلى أن يتحقق الشرط (أو حتى يكون الشرط صحيح)
}
```

ملاحظة: كتابة أخرى عند إستعمال تعليمة واحدة

```
instruction while (condition);
instruction until (condition);
```

do while , do until (4.2.2

do while

```
do{
سيتم إعادة تنفيذ التعليمات التي ستكتب هنا
ما دام الشرط صحيح
} while (الشرط) ;
```

do until

```
do{
سيتم إعادة تنفيذ التعليمات التي ستكتب هنا
 إلى أن يتحقق الشرط  (أو حتى يكون الشرط صحيح)
} until (الشرط) ;
```

for (4.2.3

يوجد مجموعة من الكتابات الممكنة بالنسبة لfor

أولا هناك الطريقة المعهودة في لغة C

```
for(instruction ;condition ; augmentation){
تعليمات
}
```

مثال:

```
for(my $i=0 ;$i<4 ;$i++){
printf("%d\n",$i) ;
}
```

سيطبع:

```
0
1
2
3
```

مثال آخر

```
for(my ($i,$j)=(0,4) ;$i<4 ;$i++,$j--){
print("[$i,$j]\n") ;
}
```

سيطبع:

```
[0,4]
[1,3]
[2,2]
[3,1]
```

طريقة آخرى:

```
my $i ;
for(0..3){
          $i+=1 ;
          print "$i" ;
         }
```

سطبع

```
1
2
3
4
```

ملاحظة: النقطتان (..) تكتبان لتوضيح وتعريف سلسة ما تبدأ بالقيم التي تسبق النقطتين (0) و تنتهي بالقيمة التي تأتي بعد النقطتين (3)
إذن يمكن القول أن هذه الكتابة(0..3) هي إختصار لكتابة سلسلة مكونة من أربعة أرقام وهي كالتالي0,1,2,3

foreach (4.2.4

أولا

```
foreach(list){
 تعليمات
}
```

متال.

```
foreach(0..2){
              print "M\n" ;
              }
```

سيطبع:

```
M
M
M
```

ثانيا

```
foreach (array){
 تعليمات
}
```

مثال

```
foreach element (array){
 تعليمات
}
```

ملاحظة: سنتطرق لأمثلة توضيحية عندما نتكلم عن المصفوفات

ملاحظة: العبارة last تستعمل في الحالة التي نريد الخروج من أحد الجمل التكرارية سابقة الذكر

مثال:

```
#! c:/perl/bin/perl.exe
use strict ;

my $i;
for (0..10){
   $i += 1;
   if($i==6){
      print "\n$i";
      last;
   }
}
  print "\n$i";
```

سيطبع :

```
6
11
```

Aalhanane

ملاحظات أو تعليقات أو تصحيح لأخطاء ترسل إلى
aalhananes1@gmail.com

http://www.arabteam2000-forum.com