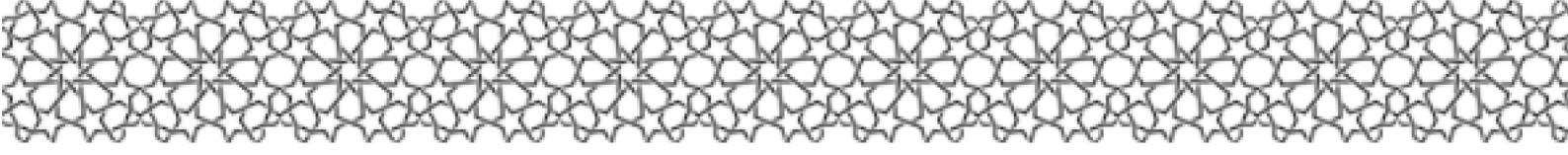


هناك أكثر من طريقة للقيام بذلك



ESSENTIAL IN

PERL

In
Arabic

الأساسي في
البيزل

بن العيد
مروان

النسخة : 09.04

الأساسي في البيزل

ESSENTIAL IN

PERL

In Arabic

هذا العمل المتمثل في مجموعة من الدروس خاصة بلغة البرمجة "بيزل" موجهة للمبتدئين الذين لهم خلفية مبدئية مع البرمجة، مجاني النشر لغرض التعليم و الاستفادة من الميزات التي توفرها هذه اللغة البرمجية لمختلف مجالات المعلوماتية.

تعلم أساسيات البرمجة بالبيزل
بلغة عربية

بن العيد
مروان

27 أكتوبر 2008

آخر تحديث: 14 أبريل 2009 *

هذه الصفحة تُركت فارغة عمداً

الفهرس

1	مقدمة
2	I - المدخل إلى البيرل
2	1- مقدمة
2	2- تعريف البيرل
2	3- لماذا نستعمل البيرل ؟
2	4- اللوازم
2	1-4- المكتبة و المترجم
2	2-4- تحرير الشفرة
3	5- تثبيت البيرل في النظام
3	1-5- على لينوكس
3	2-5- على ويندوز
4	6- قبل البرمجة بالبيرل
5	7- المتغيرات
7	1-7- الإدخال
8	8- المصفوفات و القوائم
8	1-8- المصفوفات
8	تعريف
8	حجز (تعريف) مصفوفة
9	استخراج بعض العناصر كمصفوفة من المصفوفة الأم
10	حجم (طول) المصفوفة
10	استخراج أقصى عنصر في المصفوفة و ترتيبه
10	إضافة و حذف العناصر من المصفوفة
12	استبدال، إضافة و حذف عناصر من المصفوفة
12	ترتيب تصاعدي لعناصر مصفوفة
13	عكس ترتيب عناصر مصفوفة
13	2-8- القوائم
14	9- العمليات الحسابية و النصية
14	1-9- العمليات الرياضية
15	الزيادة و النقصان
15	2-9- العمليات المنطقية
16	10- الجمل الشرطية و الحلقات
16	1-10- الجمل الشرطية
16	الشروط if .. else و elseif
18	الشرط unless
18	المعاملات && و
18	2-10- الحلقات
19	الحلقة التكرارية for
19	الحلقة التكرارية foreach
21	الحلقة الشرطية while
21	الحلقة الشرطية until
21	الحلقة الشرطية do..while و do..until

II – معالجة الكلمات و النصوص 22

1- العمليات الأساسية 23

1-1- جمع الكلمات 23

1-2- تقسيم الكلمات و النصوص 24

1-3- البحث داخل النص 25

2- تكرار الكلمات و النصوص 26

3- العمليات على حجم الحروف 26

4- تحويل النصوص إلى أرقام 27

5- عكس الكلمات و النصوص (المرآة) 28

6- تنفيذ أوامر عبر shell 28

الخاتمة 29

مقدمة

[بن العيد]

يتكرر السؤال حول جدوى تعلم لغة البيرل و استخداماتها لذلك ستكون إجابتي على هذه التساؤلات بمثابة مقدمة عامة لهذه السلسلة من الدروس.

كان تعلمي للبيرل نتيجة طبيعة عملي السابق، حيث كنت مطالب بتنفيذ تعليمات متوالية على الواجهة console في نظام اليونيكس، بعض هذه التعليمات يتطلب الكثير من الوقت يفوق بعضها 12 ساعة و بديهي أن تنفيذ أمر كل مرة و انتظار نهايته للمرور إلى ما يليه أمر يستدعي السخرية. لذلك فإن الـ scripting يعد الحل الأمثل إلا أن الـ batch و الـ ksh لم يكونا الحلين الأمثلين، فكانت هناك حاجة للغة أخرى تكون أكثر سلاسة و أبسط و هو ما توفره لغة البرمجة البيرل.

لم تكن مهمتي صعبة إنما روتينية و تتطلب الكثير من التركيز لأن أي خطأ في إحدى المراحل سيغير النتيجة النهائية التي عادة ما تكون متوقعة؛ و هي عدد المكونات التي تغيرت في نهاية ترجمة شفرة نظام التشغيل. ساعدني مجددا البيرل في اجتنب أخطاء المنجرة عن قلة التركيز عبر تحويل الأوامر و التعديلات اليدوية إلى شيفرة بيرل تنفذ المطلوب و تتأكد من صحة النتيجة.

مسألة أخرى هامة جدا و حلها البيرل، متمثلة في رفع نسخة من نظام التشغيل في الموقع الخاص للشركة نهاية عملية الترجمة و إبلاغ الروس أوتوماتيكيا بوجود النسخة، باستخدام مكتبة TelNet للربط بالمستخدم و المكتبة FTP لنقل الملفات. فبينما نحن نحظر أنفسنا للذهاب للفراش يكون الروس قد حملوا النسخة و شرعوا في تجربة الإصدار الموجود من نظام التشغيل، و في الصباح نجد تقرير الاختبار في الانتظار... و لكم تصور كم من الوقت تم ربحه.

مجال آخر هام جدا يتخصص فيه البيرل و بدون منازع يتمثل في القرصنة. يوفر البيرل العديد من المكتبات التي توفر إمكانية التعامل مع الشبكات و بطريقة يسيرة مما جعلها لغة القرصنة المحبذة. يعد الكتاب " Breaking into computer networks from the Internet " لـ Roelof Temmingh أبرز داعم لما ذكرت.

حاليا استخدم مجددا البيرل لتقييم جودة الشفرة التي طورها فريق العمل المنتمي له إثر فرض استخدام سكريبت بيرل لتغطية نقائص الأداة ¹ splint، فغير منطقي البحث يدويا في ملفات تحتوي الآلاف من الأسطر عن متغيرات مسماة بحرف واحد أو إذا استخدم أحد المطورين تعليمات مثل #endif، goto ؛ ييسر هذا السكريبت عملية البحث الآلي و اكتشاف التجاوزات في قواعد كتابة الشفرة.

¹ Splint is a tool for statically checking C programs for security vulnerabilities and programming mistakes.

I – المدخل إلى البيرل

1- مقدمة

نظرا لأهمية لغة البرمجة بيرل في تطوير البرمجيات و ما تنسم به من ثراء معرفي أشارككم في هذه الأسطر خبرتي المتواضعة في هذا المجال. هذا الدليل، إن صحت تسميته كذلك، يخص بالأساس الجانب التطبيقي في أساسيات البيرل و هو موجه للمبتدئين و إن شاء الله في الإصدارات القادمة سيكون هناك ما يستفيد به حتى ذوي الخبرة.

2- تعريف البيرل

بيرل لغة نصية، أنشأها "لاري وال" (Larry Wall) سنة 1987، كان الهدف منها معالجة النصوص (في نظام يونيكس) حيث هناك مسائل من الصعب حلها بالوسائل المستعملة حينها (awk).¹ أما الآن فقد توسع استعمالها لتصبح الأكثر طلبا لأتمتة نظام إدارة المهام (automating system administration tasks).

معنى التسمية هو اختصار لـ Practical Extraction and Report Language، و اختفاء "a" منها سببه وجود لغة بالاسم PEARL لذلك اكتفى كاتبها بالتسمية perl.

3- لماذا نستعمل البيرل ؟

كلنا نعلم أننا عند بداية أعمالنا عادة ما نميل إلى استعمال أكثر الأدوات انتشارا. و بما أن البيرل متعدد الاستعمالات، بكل تأكيد فهو ليس من أجل كل الأعمال. لذلك سنرى أبرز نقاط القوة التي من أجلها كانت هذه اللغة متنفس الكثير من المبرمجين إذا كنت:²

- ✓ مدير نظام تحتاج للغة برمجة نصية للأعمال الروتينية و أيضا للحماية.
- ✓ مدير نظام تريد كتابة إجراءات لأشخاص غير متخصصين في نظام التشغيل.
- ✓ مبرمج ويب و المطلوب منك التعامل مع العديد من الإجراءات CGI.
- ✓ مدير موقع إلكتروني أو مسؤول حمايته.
- ✓ تحتاج إلى توسيع مداركك للتعلم و التحكم في نظام التشغيل.
- ✓ مبرمج تحتاج إلى لغة نموذجية أكثر تعقيدا و مردودية.
- ✓ تريد استبدال تعبيرات Shell بتعابير أسهل قراءة و نتائجها أغنى و أدق.
- ✓ التعامل مع آلاف الملفات سواء بالتعديل أو بتصحيح المسار.
- ✓ تتعامل مع قواعد البيانات و تريد السرعة و الخفة لرؤية التقارير المطلوبة بدون الحاجة إلى الواجهات الرسومية المكلفة للذاكرة.
- ✓ تريد أن تكتب برنامج (إجراء) واحد لأغلب أنظمة التشغيل المعروفة.
- ✓ لا تريد أن تضيع الوقت في عملية التجميع (compiling) – مثل C- لكتابة إجراءات أساسية، و تريد التنفيذ و التعديل أسرع على المصدر (source).

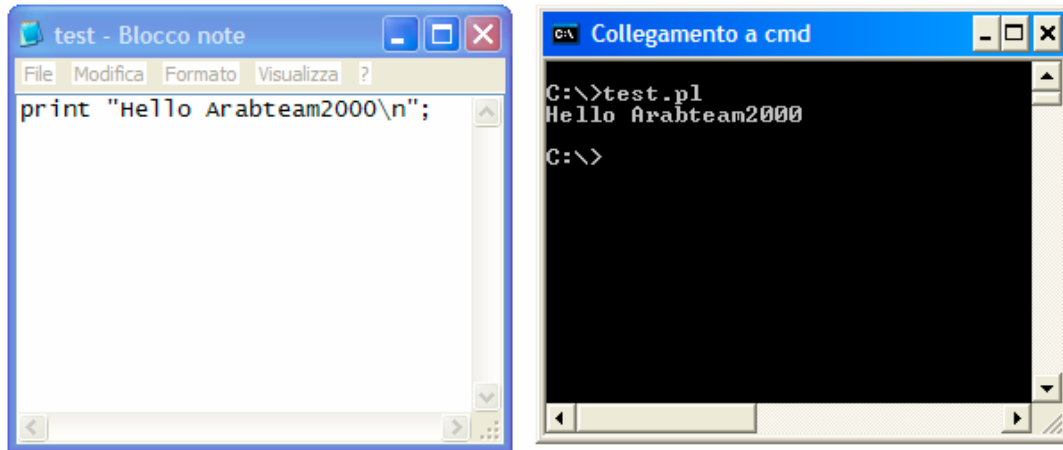
4- اللوازم

4-1- المكتبة و المترجم

إذا كنت من مستعملي نظام يونيكس (أو الأنظمة المعتمدة عليه) فلن تحتاج إلا أي إضافة حتى تتمكن من استعمال لغة البيرل، لأنه منبتها و الفلسفة القائمة عليها³؛ أما إذا كنت مستعمل ويندوز فيوجد ActivePerl و تتعدد إصداراته وفق نظام التشغيل و المعالج أيضا.

4-2- تحرير الشفرة

لكاتبة الشفرة يكفي استخدام محرر النصوص المرفق مع نظام التشغيل (الصورة 1ب- تمثل محرر النصوص الخاص بالويندوز) و للترجمة فعليك فتح "سطر الأوامر" (cmd بالنسبة للويندوز) و كتابة اسم السكريبت من مسار تواجده أو كتابة المسار كاملا (الصورة 1أ).



ب1

أ1

محررات متوافقة مع الويندوز: يمكن استعمال Notepad و Wordpad، و للمزيد من الميزات هناك NEdit، UltraEdit و NotePAD++، يوجد أيضا Komodo IDE (من ميزاته تشغيل الشفرة مباشرة دون استعمال cmd).

محررات متوافقة مع يونيكس (لينوكس): الشهيان Emacs و .vi.

بالنسبة لـ MAC: هناك BBEdit (الذي يعده أغلب المطورون من أحسن أدوات تحرير النصوص على الماك) و Alpha.

5- تثبيت البيرل في النظام

5-1- على لينوكس

كما ذكرنا سابقا، إذا كنت من مستخدمي نظام يونيكس فأنت لا تحتاج إلى تثبيت البيرل؛ لتأكد أكتب :

```
perl -v
```

من المفروض أن تظهر رسالة تفيد برقم النسخة المستعملة (This is perl, v5.x.x) معناه أن البيرل مثبت على النظام في حال وجود مشاكل، مثل (perl: command not found) و كنت متأكد من تثبيته حينها تحقق من المسار /usr/bin أو /usr/local/bin ؛ في حال عدم التمكن من إيجادته تحقق عبر مدير النظام.

للتثبيت تحتاج إلى مصدر البيرل (www.perl.com)، أيضا قد تحتاج إلى مترجم C .

5-2- على ويندوز

مع نظام ويندوز التثبيت أسهل، فقط يكفي تحميل ActivePerl المجاني من الموقع (www.activestate.com)، و التثبيت عادي.

لإضافة البيرل إلى المسار path - مع أنك لن تحتاج لهذه العملية لعد استعمال ActivePerl - بالطريقة السهلة :

[En] Start » Control Panel » System » Advanced » Environment Variables » System variables » Path

[Fr] Démarrer » Panneau de configuration » Système » Avancé » Variables d'environnement » Variables Système » Path

للتأكد أكتب في cmd:

```
perl -v
```

6- قبل البرمجة بالبيرل

ثلاثة نقاط من المهم معرفتها من طرف المتعلم قبل البداية بتعلم هذه اللغة المفيدة.

أولاً: من الممكن عدم استعمال محرر النصوص لكتابة الشفرة ذات السطر الواحد و ذلك باستعمال نافذة الأوامر (cmd) مباشرة. لنأخذ المثال السابق، لإظهار نص الترحيب "Hello ArabTeam2000" يمكن كتابة في سطر الأوامر التالي:

[dos/win]

```
C:\>perl -e "print \"Hello ArabTeam2000\n\";"
```

[unix/linux]

```
perl -e 'print "Hello, World!\n";'
```

طبعا كلمة perl هي الكلمة المفتاحية لتنفيذ الأمر بلغة البيرل

-e هي المعرف بأن على البيرل تنفيذ الأمر التالي كتعليمة "سطر واحد" (أي ليس ملف يحتوي على إجراء متعدد الأسطر).

ثانياً: دائما ما نجد سطرا في بداية ملف بيرل – يدعى شارب الاستفهامية أو shebang- يكون كالتالي:

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

لمستعملي يونيكس هذا السطر هو معيار لمعرفة shell بأن هذا الملف هو ملف بيرل و منه تحديد البرنامج المستعمل لتنفيذه، في هذه الحالة مسار البرنامج الافتراضي هو الذي يلي هذين الحرفين (!#). لذلك صحة المسار مهم جدا لهؤلاء المستعملين. أيضا معظم محررات النصوص في اللينوكس ستستعمل التلوين الخاص بهذه اللغة حتى و لو كان امتداده يختلف عن .pl.

أما بالنسبة لمستعملي الويندوز (و الماك)، ليس من الضروري كتابة هذا السطر لأن # تعبر عن تعليق و بالتالي سيتم تجاهلها. لكن من المهم الاعتياد على هذا السطر لأنه إذا كان الإجراء موجه للويب، فالخادم apache سيحتاج لهذا السطر للتعرف على طبيعة الملف (أيضا لأن إجراءات بيرل على العموم موجهة إلى أغلب أنظمة التشغيل منها اليونيكس).

الأمر الآخر، هذا السطر له أهمية أخرى لا يستغنى عنها الكثير من مستعملي هذه اللغة: و هي توجيهات (options) للمترجم بيرل، مثلا هناك -w الخاص بإظهار التحذيرات (الأخطاء) عند عملية الترجمة.

```
#!/usr/bin/perl -w
```

السطر أعلاه يعتبر في الويندوز (و الماك) كتعليق لكن لا يتم تجاهل التوجيهات (-w). يمكن أيضا استعمال التوجيهات عبر استدعائها:

```
#!/usr/bin/perl -w - إظهار التحذيرات -
```

```
no warnings; # لا نريد إظهار التحذيرات
```

```
use warnings; # إعادة استعمال التحذيرات
```

عند استعمال -W (upper-case) سنجد مترجم البيرل على إظهار التحذيرات حتى ولو كان هناك إخفاء لها داخل التطبيق (قد تنفع كثيرا حين التحقق من سلامة نتائج التطبيق يحتوي على إجراءات مركبة أو متداخلة و لا نريد تضيق الوقت بتتبع إخفاء التحذيرات في كل سطور التطبيق). أما إذا كان هناك استعمال التحذيرات داخل التطبيق و نريد للبيرل أن يترجمه دون إظهارها فما علينا سوى استعمال -X (upper-case) في أول البرنامج مكان -w.

ثالثاً: البيرل حساس لحجم الحروف: يعني perl تختلف عن Perl و تختلف عن PERL، لذلك الانتباه لكتابة أسماء المتغيرات و الدوال.

7- المتغيرات

تعتمد لغة البيرل على الرمز ¹\$ لتعريف المتغيرات و تشتت استخدام المتغيرات وفق تعريفها، بمعنى أن أي استخدام لمتغير لم يتم تهيئته مسبقاً أو مطلقاً يتم تجاهله في عملية الترجمة. (عند استخدام w- ستظهر رسالة تفيد بوجود متغير غير المعروف – استخدام متغير غير مهياً).

التهيئة في البيرل لا تعني أن تعرف المتغير و أن تعطيه قيمة أولية قبل الاستعمال، و إنما المقصود و هو عن استعمال هذا المتغير (خاصة في الطباعة أو عموماً الإخراج) يجب أن تكون له قيمة. الدليل أنك عند القيام بالجمع سيعطي مترجم البيرل للمتغيرات غير المعرفة (المهياة) القيمة الأساسية (0 للأرقام و فراغ للكلمات).

المتغيرات عبارة عن كلمات (من الأفضل أن تكون ذات معنى) لا تبدأ برقم و اختلاف حجم الحروف² مأخوذ بعين الاعتبار. كذلك اسم المتغير يجب أن يحتوي فقط الحروف و الأرقام و علامة الخط على السطر (_) – (الرقم 8 في لوحة المفاتيح).

و لا ننسى أن اسم المتغير يجب أن لا يتجاوز 255 حرف.

أيضاً يمكن (و من المستحسن) تعريف المتغير بالكلمة المحجوزة my، لكن في هذا الدليل عن الأساسيات لن نتطرق إليها – حالياً.

سنجرب جمع متغيرين و إظهار النتيجة على الشاشة كما هو في المثال التالي:

ترقيم الأسطر فقط للشرح، لا تكتب عند التطبيق

```
1 | #!/usr/bin/perl
2 | $a = 10;
3 | $b = 20;
4 | $sum = 0;
5 | print "a+b = $a + $b\n";
6 | $sum = $a + $b;
7 | print "a+b = $sum\n";
8 | print "a+b = sum\n";
9 | print "a+b = $a + $c\n";
```

نلاحظ أن في تعليمة الطباعة الأولى (السطر 5) تم إدراج عملية الجمع مباشرة داخل النص الذي سيتم إظهاره على الشاشة، في حين أن في التعليمة الثانية (السطر 7) تم إدراج ناتج الجمع في النص المُخرج.

في التعليمتين الأخيرتين للطباعة و لكن بتغييرات طفيفة متمثلة في تعليمة السطر 7 تم فيها حذف الرمز \$ الذي سبق المتغير sum، و تعليمة مشابهة للسطر 5 لكن بتغيير b إلى c.

بعد حفظ الشفرة السابقة في ملف test.pl في المسار c:\ (يمكن استخدام أي مسار آخر)، في ما يلي نتيجة الترجمة:

```
C:\perl test.pl
a+b = 10 + 20
a+b = 30
a+b = sum
a+b = 10+
```

أو يمكن التنفيذ بدون الكلمة perl في حال كان الملف الامتداد .pl مخصص لمترجم البيرل – في الويندوز.

```
C:\test.pl
a+b = 10 + 20
a+b = 30
a+b = sum
a+b = 10+
```

عند الترجمة ينبغي تحديد المسار الكامل للملف .pl، إذا لم يكن موجود في المسار الحالي الذي يمكن معرفته بتنفيذ الأمر %cd% echo. لترجمة الشفرة يمكن كتابة اسم الملف مع الامتداد مسبقاً بالتعليمة perl أو بدونها لأن إثر تنصيب البيرل يتم إضافة مكتباته مع قائمة المسارات التي يعتمد عليها نظام التشغيل و يمكن مطالعة باستخدام الأمر %path%. في حين إن إدراج التعليمة perl أمر إجباري إذا كان نظام التشغيل معتمد على يونيكس و فروع.

نعود لما طبعته الشفرة من مخرجات و التي تمتد على أربعة أسطر بنفس عدد تعليمات الطباعة كما أردنا. في السطر الأول من النص المُخرج تم فقط تعويض المتغيرين بقيمتيهما دون القيام بعملية الجمع، و السبب في ذلك أن علامة الجمع كانت بمثابة حرف كباقي

¹ السبب في اختيار هذا الرمز \$ هو أنه يشبه أول حرف في الكلمة Scalar و هو نفس الحرف في تعبير الهاكرز – أي يمكن قراءتها كالتالي \$scalar
² [en] Upper and lower case letter / [fr] Lettre majuscule et minuscule

الحروف المكونة للجملة التي يراد طباعتها على الشاشة؛ في حين أن السطر الثاني تم إظهار النتيجة المنتظرة لأن عملية الجمع تمت خارج تعليمة الطباعة.

الغريب في السطر الثالث تم إرجاع اسم المتغير عوض إبراز قيمته و السبب في ذلك ما ذكر آنفا من ضرورة سيق أسماء المتغيرات بالرمز \$.

في السطر الأخير من المخرجات نتذكر دائما أن المترجم يتجاهل أي استخدام لمتغير لم يتم تهيئته بقيمة أولية (يعطيه قيمة فراغ إذا المراد طباعة نص، و رقما إذا كان داخل عملية حسابية). يمكننا متابعة سطور الشفرة و الحصول على التنبيهات باستخدام الميزة -w المذكورة سابقا، أي:

```
#!/usr/bin/perl -w
$a = 10;
$b = 20;
$sum = 0;
print "a+b = $a + $b\n";
$sum = $a + $b;
print "a+b = $sum\n";
print "a+b = sum\n";
print "a+b = $a + $c\n";
```

سنحصل على الرسالة التالية:

Name "main::c" used only once: possible typo at C:\test.pl line 9.
Use of uninitialized value \$c in concatenation (.) or string at C:\test.pl line 9.

الآن سنرى عملية جمع بين متغيرين، أحدهما ذو قيمة معلومة و الآخر غير مهياً. المثال التالي يوضح العملية (لاحظ غياب ميزة التحذير هنا):

```
#!/usr/bin/perl
$a = 10;
$sum = $b + $b;
print "b*2 = $sum\n";
print "a+b = $a + $b\n";
```

نتيجة الترجمة كالآتي:

```
b*2 = 0
a+b = 10 +
```

ملاحظات:

- إذا أردت إدراج تعليقات في الشفرة فينبغي بدأ جملة التعليق بـ # ، و يتكرر هذا الحرف قبل كل تعليق في حال كان التعليق عبارة عن فقرة متتالية الأسطر.
- دالة طباعة print تماثل printf في السي (C) و نظيرتها في باقي لغات البرمجة حيث يمكن التحكم في شكل النص المُخرج و توزيعه باستخدام الرموز %a ، %t ، %n و غيرها – أنظر الجدول التالي للتعامل مع النصوص.

الجدول: ج 1

الرمز	المعنى
\n	سطر جديد
\r	العودة لبداية السطر الحالي للكتابة فوقه
\b	backspace حذف حرف للوراء
\f	formfeed صفحة جديدة
\a	alarm تصدر صوت
\t	tab أي مسافة جدولة
\e	رمز escape أي 27 أو 033 بالثمانى أو 0x1b بالست-عشري
\0nnn	تعني الرمز المقابل ل nnn حيث nnn رقم بالثمانى مثلا 33 هي escape و 101 تعني حرف A
\xnn	تعني الرمز المقابل ل nn حيث nn رقم بالست-عشري مثلا 1b هي escape و 41 تعني حرف A
\cC	زر الكونترول هنا Ctrl-C
\\	BackSlash
\"	علامة اقتباس مزدوجة (Double quote)
\u	تحول الحرف اللاتيني التالي إلى upper-case أي كبير
\l	تحول الحرف اللاتيني التالي إلى lower-case أي صغير
\U	تحول الحروف اللاتينية التالية إلى upper-case أي كبيرة حتى أول \E
\L	تحول الحروف اللاتينية التالية 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

1-7- الإدخال

عملية الإدخال (أو القراءة من لوحة المفاتيح) تتم عبر التعليمات <STDIN> و يفضل استعمال chomp بحيث تكتب كالتالي:

```
chomp($read = <STDIN>);
```

chomp : مهمتها هي حذف رمز السطر الجديد (/n) بحيث نحصل في الأخير على متغير صاف مثال:

```
#!/usr/bin/perl
print "Enter: ";
$a="";
$a = <STDIN>;
print $a;
print "Enter: ";
chomp($a = <STDIN>);
print $a;
print "Enter: ";
chomp($a=<STDIN>);
print $a;
print "\nAdd \n for new line";
```

هنا لا بد أن نلاحظ أن المتغير يحتوي على علامة سطر جديد
يطبع في سطر جديد
حذف علامة السطر الجديد
لا تحتوي على علامة سطر جديد
يطبع إلى جانب المتغير \$a
حذف علامة السطر الجديد
لا تحتوي على علامة سطر جديد
يجب إضافة علامة سطر جديد
لطباعة هذا السطر في سطر جديد

نتيجة التنفيذ- سنقوم بإدخال "az" كقيمة للمتغير:

```
C:\TestChomp.pl
Enter: az
az
Enter: az
azEnter: az
az
Add \n for new line
```

8- المصفوفات و القوائم

1-8- المصفوفات

تعريف

المصفوفات هي عبارة مجموعة من المتغيرات مخزنة في متغير واحد. للتعبير عن مصفوفة في البيزل نستعمل الرمز @¹، ترتيب محتوياتها يبدأ من الصفر (0) و المجموع من الواحد (1).

ميزة محتويات المصفوفة (و القوائم) أن هذه المحتويات هي متغيرات مستقلة عن بعضها البعض، بحيث يمكن أن تكون إحداها رقمية بينما الأخرى نصية.

حجز (تعريف) مصفوفة

مثال عن حجز مصفوفات بمختلف المحتويات و بصور مختلفة:

```
#!/usr/bin/perl
@10 = (1,2,3,4,5,6,7,8,9,10);
@100 = (1 .. 100);
@thousand = (100 .. 1000);
@alphabet = (a .. z);
print @10;
print @100;
print @thousand;
print @alphabet;
```

أهم ملاحظة هي تسمية المصفوفة : يمكن لاسم المصفوفة أن يبدأ برقم، لكن في هذه الحالة لا يمكن أن يحتوي على حرف. أي @100 صحيحة و مقبولة من البيزل لكن @1_thousands غير مقبولة.

ملاحظة ثانية هي دور علامة الاقتباس المزدوجة (")، كما نلاحظ في الكود فوق قمنا بالطباعة دون استعمال هذه العلامة.. حسنا، ماذا سيحصل لو استعملناها.

بدون استعمال علامة الاقتباس المزدوج سيقوم البيزل بطباعة المحتوى تباعا و في النهاية ستظهر كسطر واحد لا نفرق بين عناصر هذه المصفوفة؛ أما في حال استعمالها سيقوم البيزل بطباعة عناصر المصفوفة تباعا مع فصلها عن بعضها بفراغ.

مثال:

```
#!/usr/bin/perl
@10 = (1,2,3,4,5,6,7,8,9,10);
@alphabet = (a .. z);
print @10;
print "\n";
print @alphabet;
print "\n\n*****with double quote*****\n\n";
print "@10";
print "\n";
print "@alphabet";
```

النتيجة:

```
12345678910
abcdefghijklmnopqrstuvwxy

*****with double quote*****

1 2 3 4 5 6 7 8 9 10
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

يمكن استبدال الأقواس بالتعليمة qw (Quoted Word) -النصوص المعلّمة- و دورها تجاهل الفراغات و علامات السطر الجديد، مع أخذ ما بينهما كعناصر بحيث سنحصل على تقسيم الجملة أو الفقرة إلى عناصر متفرقة. مثل:

```
@week = ("Sat", "Sun", "Mon", "Tue", "Wed", "Thu", "fri");
```

¹السبب في اختيار هذا الرمز @ هو أنه يشبه أول حرف في الكلمة Array وهو نفس الحرف في تعبير الهاكرز – أي يمكن قراءتها كالتالي @rray

تكافئها:

```
@week = qw(Sat Sun Mon Tue
            Wed
            Thu          fri);
```

و التعليمة qw تسمح باستبدال الأقواس بأي حرف آخر، مثال:

```
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
@week = qw! Sat Sun Mon Tue Wed Thu Fri !;
@week = qw? Sat Sun Mon Tue Wed Thu Fri ?;
@week = qw# Sat Sun Mon Tue Wed Thu Fri #;
@week = qw[ Sat Sun Mon Tue Wed Thu Fri ];
@week = qw+ Sat Sun Mon Tue Wed Thu Fri +;
@week = qw< Sat Sun Mon Tue Wed Thu Fri >;
```

الفرق بين الأقواس و التعليمة qw هو أننا لا يمكن استعمال رموز الطباعة (مثل \n) للتعبير عن سطر جديد لأن qw سيعتبرها حروف نصية عادية متضمنة في العنصر.

ل للوصول إلى العناصر و استعمالها نستعمل العارضتان [..] كالتالي:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
@alphabet = (A .. Z);
print "The sixth day is ". @week[5];
print "\n";
print "The sixth day from the bottom is ". @week[-6];
print "\n";
print "The 19th letter in alphabet is ". @alphabet[18];
```

لاحظ العدد السالب حيث 1- هو الأخير و الذي يسبقه -2، النتيجة:

```
The sixth day is Thu
The sixth day from the bottom is Sun
The 19th letter in alphabet is S
```

منه أيضا تستعمل لملء المصفوفة، مثل:

```
#!/usr/bin/perl
@week[0] = Sat;
@week[1] = Sun;
@week[2] = Mon;
@week[3] = Tue;
@week[4] = Wed;
@week[5] = Thu;
@week[6] = Fri;
print "The sixth day is ". @week[5];
```

استخراج بعض العناصر كمصفوفة من المصفوفة الأم

يمكننا استخراج بعض العناصر محددين من المصفوفة و تخزينهم في مصفوفة أخرى كالتالي:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
@weekend = @week[5,6];
print "Days of weekend are @weekend \n";
@SameWeekend = @week[-2,-1];
print "Days sleeping until 10:am are @SameWeekend \n";
@workdays = @week[0 .. 4];
print "Workdays are @workdays \n";
```

لاحظ العدد السالب حيث -1 هو الأخير و الذي يسبقه -2، النتيجة:

```
Days of weekend are Thu Fri
Days sleeping until 10:am are Thu Fri
Workdays are Sat Sun Mon Tue Wed
```

حجم (طول) المصفوفة

يمكننا معرفة طول المصفوفة بطريقتين إحداهما عن طريق متغير و الثانية عن طريق تعليمة بيرل.

عن طريق متغير سواء كان ثانوي أو استدعاء المصفوفة بدون عوارض لكن خارج النص المراد طباعته:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
$numdays = @week;
print "Number of days in a week is $numdays \n";
print "Number of days in a week is ". @week;
```

النتيجة:

```
Number of days in a week is 7
Number of days in a week is 7
```

أو يمكن استعمال التعليمة scalar التي تتمثل مهمتها في إرجاع عدد العناصر في المصفوفة أي معاملة المصفوفة كمتغير منه عدد العناصر، مثال:

```
#!/usr/bin/perl
@week1 = qw{ Sat Sun Mon };
@week2 = qw{ Tue Wed Thu Fri };
@days = (@week1,@week2);
$numdays = (scalar(@week1), scalar(@week2));
print "Days of week are: @days\n";
print "Number of days in week1 is @numdays[0]\n";
print "Number of days in week2 is @numdays[1]";
```

النتيجة:

```
Days of week are: Sat Sun Mon Tue Wed Thu Fri
Number of days in week1 is 3
Number of days in week2 is 4
```

استخراج أقصى عنصر في المصفوفة و ترتيبه

يمكن معرفة ترتيب أقصى عنصر في مصفوفة (الأخير) عبر استعمال التعبير التالي \$#array بحيث array هو اسم المصفوفة. من بين فوائده الجلية تظهر في الحلقات، لكن نأخذ المثال البسيط التالي:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
print "Max days in a week is $#week+1\n";
```

النتيجة:

```
Max days in a week is 6+1
```

لا ننسى أن البداية هي الصفر (0).

إضافة و حذف العناصر من المصفوفة

يمكننا من خلال إجراءات في البيزل أن ضيف أو نحذف عناصر من المصفوفة، و الإمكانية من الجهتين:

أ- العمليات على نهاية المصفوفة `push` و `pop`

pop: تحذف عنصرا من آخر المصفوفة، و ترجع العنصر المحذوف، مثال:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
$friday = pop(@week);
print "@week\n";
pop @week;
print "@week\n";
print $friday;
```

النتيجة:

```
Sat Sun Mon Tue Wed Thu
Sat Sun Mon Tue Wed
Fri
```

في حال كانت المصفوفة فارغة، فهذه العملية لن تحذف شيئا و ترجع `undef`

push: تضيف عنصرا في آخر المصفوفة، مثال:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed };
push @week, Thu;
push(@week, Fri);
print "@week\n";

@ten = ();
push @ten, 1 .. 10;
print "@ten\n";

push @mix, @ten, qw/11 ahmed 1.25 arab/;
print "@mix\n";
```

النتيجة:

```
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11 ahmed 1.25 arab
```

ب- العمليات على بداية المصفوفة `shift` و `unshift`

Shift: حذف أول عنصر في المصفوفة، و ترجع العنصر المحذوف، مثال:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
$saturday = shift(@week);
print "@week\n";
shift @week;
print "@week\n";
print $saturday;
```

النتيجة:

```
Sun Mon Tue Wed Thu Fri
Mon Tue Wed Thu Fri
Sat
```

في حال كانت المصفوفة فارغة، فهذه العملية لن تحذف شيئا و ترجع `undef`

unshift: تضيف عنصرا في أول المصفوفة، مثال:

```
#!/usr/bin/perl
@week = qw{ Mon Tue Wed Thu Fri };
unshift @week, Sun;
unshift(@week, Sat);
print "@week\n";

@ten = ();
unshift @ten, 1 .. 10;
print "@ten\n";

unshift @ten, qw/11 ahmed 1.25 arab/;
print "@ten\n";
```

النتيجة:

```
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4 5 6 7 8 9 10
11 ahmed 1.25 arab 1 2 3 4 5 6 7 8 9 10
```

استبدال، إضافة وحذف عناصر من المصفوفة

يمكننا استبدال، إضافة وحذف عناصر في مصفوفة بعملية واحدة في سطر واحد، وذلك باستعمال `splice()`.
`splice(@array, first-element, sequential_length, new elements)`
 (العناصر الجديدة , الطول-تتابعيا , العنصر-الأول , المصفوفة)

"العنصر-الأول"، "الطول-تتابعيا" و "العناصر-الجديدة" غير إجباري كتابتهم في حال أردنا عدم تحديد ما نريد، مثال:

```
#!/usr/bin/perl
@numbers = (1 .. 10);
splice(@numbers, 5, 3, -3..-1);
# استبدال العناصر الثلاثة (6,7,8) بعد العنصر 5 بالأعداد السالبة

print "@numbers\n";

splice(@numbers, 5, 5); # حذف العناصر الخمسة بعد العنصر الخامس
print "@numbers\n";

splice(@numbers);
print "@numbers\n"; # المصفوفة فارغة
```

النتيجة (السطر الأخير فراغ):

```
1 2 3 4 5 -3 -2 -1 9 10
1 2 3 4 5
```

ترتيب تصاعدي لعناصر مصفوفة

يمكن ترتيب عناصر مصفوفة تصاعديا باستعمال الإجراء `sort()`، و الذي مهمته ترتيب العناصر نصيا و ليس رقميا. مثال:

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
@week = sort(@week);
print "@week\n";

@num = (1..5,95..110);
@num = sort(@num);
print "@num\n";
```

النتيجة :

```
Fri Mon Sat Sun Thu Tue Wed
1 100 101 102 103 104 105 106 107 108 109 110 2 3 4 5 95 96 97 98 99
```

عكس ترتيب عناصر مصفوفة

لعكس ترتيب عناصر مصفوفة نستعمل الإجراء reverse()، الذي يرجع مرآة المصفوفة، مثال

```
#!/usr/bin/perl
@week = qw{ Sat Sun Mon Tue Wed Thu Fri };
print "@week\n";
@week = reverse(@week);
print "@week\n";
```

النتيجة :

```
Sat Sun Mon Tue Wed Thu Fri
Fri Thu Wed Tue Mon Sun Sat
```

2-8- القوائم

القوائم لا تختلف عن المصفوفات إلا أنها تعتبر حالة خاصة منها بحيث لا تحتاج لتعريف و إنما التعامل مباشرة معها. مثال:

```
#!/usr/bin/perl
print (1..10);
print "\n";

($FLCLetter, $LUCLetter) = (a, Z);
print ($FLCLetter, $LUCLetter);
```

النتيجة :

```
Fri Mon Sat Sun Thu Tue Wed
1 100 101 102 103 104 105 106 107 108 109 110 2 3 4 5 95 96 97 98 99
```

من بعض فوائدها الرائعة هي عكس (تبادل) قيمة متغيرات مختلفة- أو إعطاء عدة متغيرات قيم تراتيبيا، مثال

```
#!/usr/bin/perl
$JanShort = "January";
$JanMonthLong = "Jan";
print "First month in short is $JanShort\n";
print "First Month in long is $JanLong\n";

print "\nOOPS, wrong\n\n";

($JanShort, $JanLong) = ($JanLong, $JanShort);
print "First month in short is $JanShort\n";
print "First Month in long is $JanLong\n";
```

النتيجة :

```
First month in short is January
First Month in long is Jan

OOPS, wrong

First month in short is Jan
First Month in long is January
```

9- العمليات الحسابية و النصية

في أي لغة برمجة لا نستطيع الاستغناء عن العمليات الحسابية و النصية.

9-1- العمليات الرياضية

الجدول التالي يعبر عن العمليات الرياضية في البيزل موضحة بمثال:

الجدول: ج 2

العملية	الوصف	المثال	النتيجة
+	الجمع	$3 + 4$	7
-	الطرح (رقمين)	$4 - 2$	2
	السالب (رقم واحد)	-5	-5
*	الضرب	$5 * 5$	25
/	القسمة للأعداد ذات الفاصلة	$15 / 4$	3.75
%	باقي القسمة	$15 \% 4$	3
**	الأس	4^{**5} مكافئ لـ (4^5)	1024
++	الزيادة بواحد	$a=10 ; a++ ;$	$a=11$
--	النقصان بواحد	$a=10 ; a-- ;$	$a=9$
+=	الزيادة	$a=10 ; a+=5 ;$	$a=15$
-=	النقصان	$a=10 ; a-=5 ;$	$a=5$
=	الضرب في نفس العدد	$a=10 ; a=5 ;$	$a=50$
/=	القسمة من نفس العدد	$a=10 ; a/=3 ;$	$a=3.33$
%=	باقي القسمة من نفس العدد	$a=10 ; a\%=3 ;$	$a=1$
=	الأس من نفس العدد	$a=10 ; a=3 ;$	$a=1000$

للتوضيح :

```
$a = 10 ;
$a += 5 ;
```

مكافئة للتالي :

```
$a = 10 ;
$a = $a+5 ;
```

ملاحظة مهمة: عند القيام بالعمليات الرياضية (مثل الجمع) في الطباعة مباشرة يجب الانتباه إلى الأقواس، مثال توضيحي:

```
$a = (1+2)+3 ;
print $a ;
print "\n" ;
print (1+2)+3 ;
```

نتيجة الترجمة:

```
6
3
```

لماذا؟ .. بكل بساطة لأن البيزل يعتبر كل ما بين قوسين على أنها إجراء، عند استعمال الميزة w- ستظهر رسالة تفيد على أن ما بين القوسين بعد دالة الطباعة تترجمه كإجراء، إلا إذا استعملنا الزائد (+) قبل القوس الأول

```
print (...) interpreted as function at C:\test.pl line 4.
Useless use of addition (+) in void context at C:\test.pl line 4.
```

للحصول على ما نريد نستعمل إحدى الطرق التالية:

```
$a = (1+2)+3;
print $a ;
print "\n" ;
print $b = (1+2)+3;    # -w أو تعطيل التحذيرات
print "\n" ;
print +(1+2)+3;
print "\n" ;
print ((1+2)+3);
```

نتيجة الترجمة:

```
6
6
6
6
```

الزيادة و النقصان

الزيادة (++) و النقصان (--)، اللتان تستعملان مع الأعداد بحيث تكون الزيادة/النقصان بواحد (1)، في البيزل تختلف إن تقدمتا المتغير أو تلتته، أي:

```
print $a++ ; # النتيجة : 0
```

تختلف عن:

```
print ++$a ; # النتيجة : 1
```

التفسير: في الأولى مترجم البيزل يقوم بطباعة قيمة المتغير ثم يزداد إلى القيمة واحد (المتغير سيتم تهيئته ب 0 في حال فراغه). في الثانية مترجم البيزل يزداد واحد إلى قيمة المتغير ثم يطبع.

مثال توضيحي:

```
print $a ;
print "\n" ;
print ++$a ; # قبل العملية الرياضية سيحول المتغير الفارغ إلى 0
print "\n" ;
print $b++ ; # قبل العملية الرياضية سيحول المتغير الفارغ إلى 0
print "\n" ;
print $b ;
```

نتيجة الترجمة كالآتي (السطر الأول فراغ):

```
1
0
1
```

2-9- العمليات المنطقية

نذكر مجموعة المعاملات (Operators) التي يمكن استخدامها في عمليات المقارنة بين المتغيرات.

الجدول: ج 3

المعنى	خاص بالنصوص	خاص بالأرقام
تساوي	eq	==
غير متساوي	ne	!=
أصغر	lt	<
أكبر	gt	>
أصغر أو يساوي	le	<=
أكبر أو يساوي	ge	>=

10- الجمل الشرطية و الحلقات

لا شك أن الحلقات و الجمل الشرطية تمثل العمود الفقري لأي لغة برمجة بما فيها البيرل، حيث تمكن المطور من معالجة كافة الاستثناءات التي تحدث و تجنب التكرار الآلي للتعليمات.

10-1- الجمل الشرطية

الشروط if .. else و elseif

فيما يلي مثال لمقارنة بين نصين لاتينيين باستخدام كافة علامات المقارنة، و الغاية منه أولاً تجربة هذه المعاملات و ثانياً فهم أساس المقارنة بين النصوص.

```
$var1= "AZX" ;
$var2= "BAX" ;
if ($var1 ne $var2){
    print "var1 isn't equal to var2.\n" ;
}
if ($var1 eq $var2){
    print "var1 is equal to var2.\n" ;
}
if ($var1 ge $var2){
    print "var1 is greater or equal to var2.\n" ;
}
if ($var1 le $var2){
    print "var1 is less or equal to var2.\n" ;
}
```

نتيجة الترجمة كالاتي:

```
var1 isn't equal to var2 .
var1 is less or equal to var2 .
```

جلي أن المتغيران غير متعادلان و لكن لماذا المتغير الثاني أكبر من الأول ؟

المقارنة بين النصوص ترتكز على الترتيب للحروف و الحرف الأسبق من غيره يعد الأكبر، أما داخل النص في حد ذاته فهناك ترتيب تفضلي لمدى أهمية الحروف أيضاً حسب ترتيبه في الجملة. لذلك فإن الحرف الأول هو الأكبر. لو اطلعنا على النصين المعنيين فسنلاحظ أن النص الأول يبدأ بالحرف A في حين الثاني يبدأ بالحرف B و هو أكبر، و رغم عكس الحالة و وضع حرف ثاني في النص الأول أكبر من الثاني فإن النتيجة لم تتغير نظراً لأهمية ترتيب الحروف. بالإمكان الآن إعادة التجربة باستخدام الرموز الرياضية للمقارنة و ستكون النتيجة ذاتها.

تُستخدم else و elseif في الجمل الشرطية و يبقى استخدامها مماثل لبقية لغات البرمجة.

```
$var1= "AZX" ;
$var2= "BAX" ;
if ($var1 eq $var2){
    print "var1 is equal to var2.\n" ;
} else {
    print "var1 isn't equal to var2.\n" ;
}
```

أو

```
$var1= "AZX" ;
$var2= "BAX" ;
if ($var1 ne $var2){
    print "var1 isn't equal to var2.\n" ;
} elseif ($var1 ge $var2){
    print "var1 is greater or equal to var2.\n" ;
} else {
    print "var1 is less or equal to var2.\n" ;
}
```

ملاحظة: في البيزل، عند المقارنة بين نصين يجب استعمال معاملات الخاصة بها، و العكس بالنسبة للأرقام لأنها ستحوّل إلى أرقام، مثلاً:

```
5 < 8;                # True
"5" < 8;              # True - ستحوّل إلى عدد
5 < 2+6;              # True - العمليات أولاً ثم المقارنة
5 < 8 > 10;           # Error - لا يمكن القيام بالمقارنات المركبة
"Arab" == "Arab";    # Error - يجب استعمال الشروط الخاصة بالنصوص
"Arab" eq "Arab";    # True
"Arab" eq "arab";    # False
"Arab" eq "Arab ";   # False - الفراغ محسوب في المقارنة
```

مثلاً:

```
#!/usr/bin/perl -w
$a = "False";
print "a = $a\n";
if ("Arab" == "Team"){
    $a = "True";
}
print "a = $a\n";
```

بعد التنفيذ

```
a = False
Argument "Team" isn't numeric in numeric eq (==) at C:\test.pl line 4.
Argument "Arab" isn't numeric in numeric eq (==) at C:\ test.pl line 4.
a = True
```

نلاحظ أن الشرط ينفذ مهما كانت الكلمتان في طرفي المقارنة. لذلك استعمال المعاملات الخاصة بكل نوع (نصي أو رقمي) واجب.

لماذا تم الفصل بين المقارنات النصية و الرقمية؟

بكل بساطة لأن البيزل سيرتب المقارنات رقمياً أو أبجدياً.

مثال، لنقوم بالمقارنة التالية هل 99 أصغر من 100.. نحن نعلم النتيجة، لكن لنرى كيف نقدمها لمترجم البيزل و كيف يتعامل معها:

```
#!/usr/bin/perl -w
print "Test if 99 is less than 100\n";
print "\tNumerically: ";
if (99 < 100){
    print "True";
} else {
    print "False" ;
}
print "\n\tAlphabetically ";
if (99 lt 100){
    print "True";
} else {
    print "False" ;
}
```

نتيجة الترجمة كالاتي:

Test if 99 is less than 100
Numerically: True
Alphabetically: False

التفسير: عدديا 99 أصغر من 100

أبجديا 100 أصغر من 99، لأن 1 في أقصى اليسار يأتي قبل 9.

الشرط unless

كما نستطيع استعمال unless التي تعبر عن عكس if، إنها من الوسائل التي يوفرها البيزل للتعامل مع الجمل الشرطية بأكثر من وسيلة التي تشمل أكثر من طريقة تفكير منه أكثر طريقة للحل.

```
#!/usr/bin/perl -w
print "What did you want for Palestine? [Freedom/No Freedom]";
chomp($what = <STDIN>);
unless (uc($what) eq "FREEDOM") {
    print "shame on you!\n";
}
```

في هذه الجملة الشرطية : إطبّع "shame on you!" مهما كانت قيمة المتغير what إلا في حالة إذا كانت القيمة هي "freedom".

uc هي محول النص في المتغير إلى حروف كبيرة (upper-case).

أيضا، توجد طريقة أخرى يوفرها البيزل و تقلل عدد الأسطر مما يكسب المبرمج السرعة في تحرير الشفرة

إذا استعملنا if و else سنحتاج إلى خمسة أسطر ذات الإخراج الحسن و المقروء، أو إحدى الطرق التي يوفرها البيزل هي:

المعاملات && و ||

هذان المعاملان يفيدان في اختبار المقارنة و التنفيذ حسب نتيجتها. المعامل && ينفذ الطرف التالي له إذا كان الطرف السابق لهذا المعامل صحيح، أما || على العكس ينفذ في حالة إذا كان الطرف السابق له خاطئ.

```
#!/usr/bin/perl -w
10 == 25-15 && print "Test is true";
10 == 25-15 || print "Test is false";
```

نتيجة الترجمة

Test is true

و في حالة أردنا التحقق و التنفيذ حسب النتيجة نستعمل الصيغة التالية:

Test ? True_thing : False_thing;

```
#!/usr/bin/perl -w
10 == 25-15 ? print "Test is true" : print "Test is false";
```

2-10 الحلقات

في البيزل نجد نوعين من الحلقات:

- الحلقات التكرارية و تتمثل في for و foreach.
- الحلقات المشروطة (أو الشرطية) و تتمثل في while و until.

الحلقة التكرارية for

هذه الحلقة لا تختلف عن الصيغة الموجودة في لغة البرمجة C، لذلك – لمن لهم دراية بها – سيكون استيعابها سهلاً. أبسط مثال لكتابة حلقة تكرارية for نكتبه توضيحياً حيث للحقتين التاليتين نفس النتيجة:

```
#!/usr/bin/perl -w
for $i (1 , 2 , 3 , 4 , 5) {
    print $i ;
}
print "\n" ;
for $i (1 .. 5) {
    print $ i ;
}
```

نتيجة الترجمة:

```
12345
12345
```

و هو ما يثبت أن للحقتين نفس النتيجة

لكن يمكن الكتابة بالصيغة المعروفة لأغلب المبرمجين:

```
#!/usr/bin/perl -w
for ($i = 1; $i < 6; $i++) {
    print $i;
}
print "\n" ;
```

نتيجة الترجمة:

```
12345
```

الحلقة التالية غير منتهية:

```
#!/usr/bin/perl -w
for (;;) {
    print "not finish yet\n";
}
```

معلومة : للخروج من برنامج قيد التنفيذ أو حلقة غير منتهية (أثناء التنفيذ) نضغط على **ctrl-c**

الحلقة التكرارية foreach

قد تعتقد أن هذه الحلقة مشابهة للـ for، لكن في الحقيقة لديها بعض ما يميزها عنها. قبل ذلك لنمر على مثال توضيحي لفهم آلية هذه الحلقة. المثال التالي مشابه للحلقة الثانية في المثال السابق، و التغيير الطفيف هو في كلمة for حيث أصبحت foreach.

```
#!/usr/bin/perl -w
foreach $i (1 .. 5) {
    print $i ;
}
```

نتيجة الترجمة:

```
12345
```

الفائدة الجلية لهذه الحلقة تظهر عند التعامل مع المصفوفات و القوائم (خاصة الممتدة – مجهولة الطول)، أو عند التعامل مع ملف و تريد التعامل مع كل سطر... إلخ.

الآن سنرى مثال ينبهنا لأمر قد يجنبنا الحصول على نتائج خاطئة أو استعماله لتجنب حجز متغيرين في حين يمكن القيام بالعملية بمتغير واحد (غير مستحبة):

```
#!/usr/bin/perl -w
$var = 42;
print "Before: $var \n";
foreach $var (1..5) {
    print "Inside: $var \n";
}
print "After: $var \n";
```

قد يتبادر إلى الذهن أن الطباعة الأخيرة سيكون فيها الرقم 5 .. لا

نتيجة الترجمة:

```
Before: 42
Inside: 1
Inside: 2
Inside: 3
Inside: 4
Inside: 5
After: 42
```

الملاحظة هنا هي عودة القيمة إلى ما كانت عليه قبل الدخول في الحلقة أي الرقم 42. لماذا؟

في البيزل كل كتلة ستنفذ بمعزل عن الشفرة الكلية (إلا في حال قصدنا التداخل) و المتغيرات هي داخلية ، بمعنى:

قبل الدخول في الحلقة كانت قيمة var تساوي 42، بعد الدخول سيتم التعامل مع هذا المتغير بالقيم الداخلية (1، 2، 3، 4، 5) و في الأخير بعد الخروج لن تضعيق القيمة السابقة و تعود var لما كانت عليه أي 42. هذه المعلومة صحيحة مع الحلقة for للمثال التالي:

```
#!/usr/bin/perl -w
$i = 42;
print "$i\n";
for $i (1 , 2 , 3 , 4 , 5) {
    print $i ;
}
print "\n$i\n";
for $i (1 .. 5) {
    print $ i ;
}
print "\n$i";
```

نتيجة الترجمة:

```
42
12345
42
12345
42
```

لكنها غير صحيحة مع المثال التالي:

```
#!/usr/bin/perl -w
$i = 42;
print "$i\n";
for ($i = 1; $i < 6; $i++) {
    print $i;
}
print "\n$i";
```

نتيجة الترجمة:

```
42
12345
6
```

الحلقة الشرطية while

نسمي هذه الحلقة بالشرطية لأنه لا تتعامل مع المتغيرات لكي تتابع عملها، وإنما تتعامل مع الشروط (المنطقية و الرياضية)، لذلك هي أحسن (أبسط) لمعرفة آخر القيم في المتغيرات بعد انتهاء الحلقة. مثال:

```
#!/usr/bin/perl -w
$i=1;
while($i<=5) {
    print $i;
    $i++;
};
```

نتيجة الترجمة:

```
12345
```

مثال آخر - الحلقة لن تنتهي حتى يكتب المستعمل كلمة حرة بالانجليزية:

```
#!/usr/bin/perl -w
while(uc($what) ne "FREEDOM") {
    print "Freedom for Palestine.";
    chomp($what = <STDIN>);
}
```

الحلقة الشرطية until

هذه الحلقة هي عكس الحلقة while من ناحية التعامل مع الشرط، نفس المثال السابق يعطي نفس النتيجة لكن بشرط عكسي:

```
#!/usr/bin/perl -w
until(uc($what) eq "FREEDOM") {
    print "Freedom for Palestine.";
    chomp($what = <STDIN>);
}
```

الحلقة الشرطية do..while و do..until

أحيانا كثيرة نحتاج إلى تنفيذ محتوى الحلقة مرة واحدة على الأقل، و هذا ما لا توفره لنا الحلقتين الشرطيتين while و until. لاستيعاب الفائدة من هذه الحلقة نعود للمثال السابق: فرضا المتغير كان يحمل القيمة الصحيحة لعدم تنفيذ الحلقة: في هذه الحالة لن نرى جملة "الحرية لفلسطين"، باستعمال هذا النوع (do) سنكون على يقين أن هذه الجملة ستظهر على الشاشة - على الأقل مرة واحدة - ثم يأتي التحقق من الشرط.

```
#!/usr/bin/perl -w
do{
    print "Freedom for Palestine.\n";
    chomp($what = <STDIN>);
} while(uc($what) ne "FREEDOM");
```

نتيجة الترجمة:

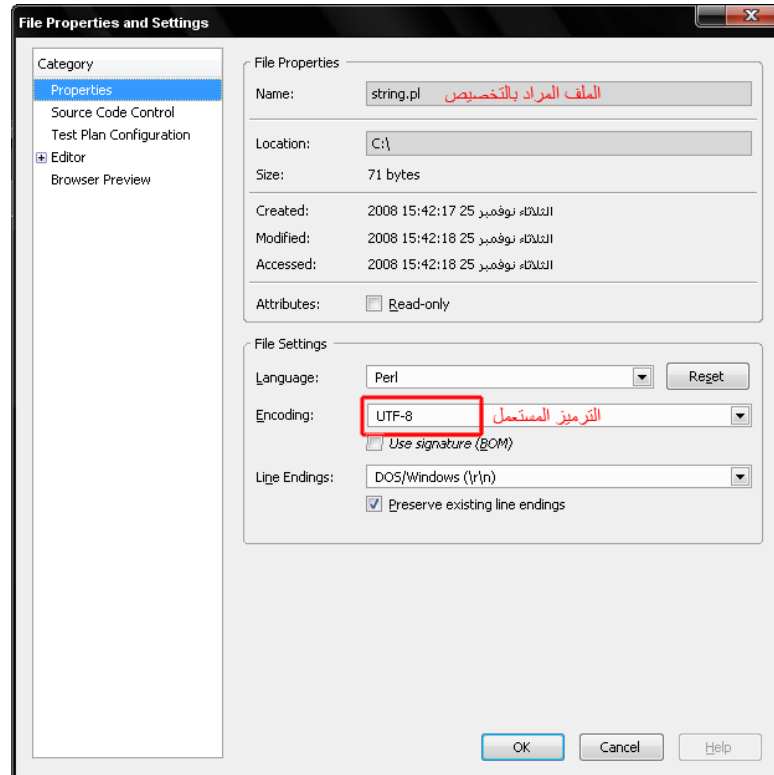
```
Freedom for Palestine.
```

II – معالجة الكلمات والنصوص

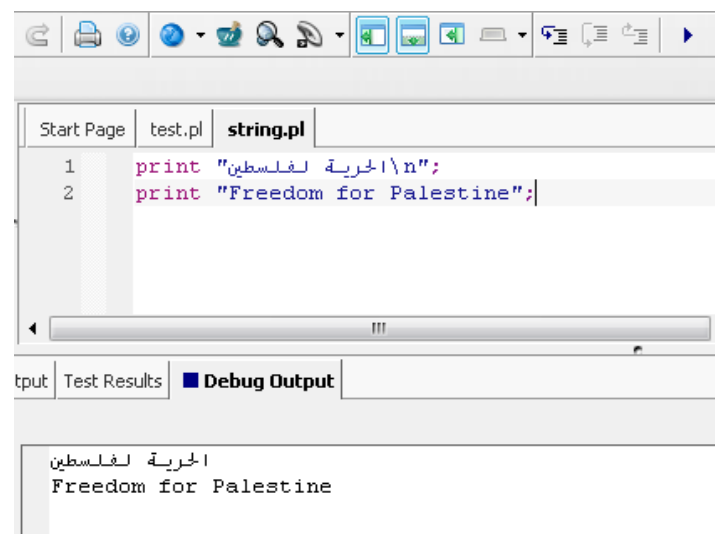
في هذا الفصل سنهتم بمعالجة الكلمات والنصوص و ما يتعلق بها من ربط، و تقسيم و بحث داخلها و غير ذلك من المهام التي يمكن للمبرمج احتياجها.

قبل المباشرة في هذا الفصل من المفيد معرفة أن من أبرز نقاط قوة لغة البيرل هي في معالجة الكلمات و النصوص، و منذ النسخة 5.6 للبيرل أصبح لديه القدرة على التعامل مع ملفات unicode التي تهمننا منها هنا اللغة العربية، و مع أن shell لا يُظهر الحروف الشرقية (لأنه يعتمد على الحروف ASCII) هذا لا يعني أن مترجم البيرل غير قادر على إخراج الحروف بهذه الصيغة، لذلك قبل إخراج الكلمات unicode يجب مراعاة محل الإخراج، مثلا متصفح الأنترنت أو أي برنامج قادر على قراءة الخط الشرقي.

مثلا عند استعمال komodo IDE ستكون طباعة الكلمات سليمة فقط بعد تغيير إعدادات الملف المستعمل بحيث يصبح الترميز حسب الخط المراد من خلال Edit << Current file settings..



مثال عن طباعة كلمة "الحرية لفلسطين"



أيضا هذا ما يبعث الارتياح لعرض التقارير لمختلف المجالات (الانترنت، ملفات نصية،...) أيضا كتابة التعليقات بالعربية ☺

معلومة: في البيزل يمكن استعمال الاقتباس المزدوج (") أو الوحيد (')، لكن هناك فرق جوهري بين استعمالهما. الاقتباس المزدوج (") : في حالة كان في طرفي النص (الجملة) فإن المتغيرات التي تتخلله ستظهر قيمتها و ليس اسمها. الاقتباس الوحيد (') : عكس المزدوج، أي ما بين الاقتباسين الوحيديين سيأخذ كما هو دون اعتبار لقيمة المتغيرات. مثال:

```
#!/usr/bin/perl -w
$what = "Freedom for Palestine.";
print "All Muslim world say : $what\n";
print 'All Muslim world say : $what';
```

نتيجة الترجمة:

```
All Muslim world say : Freedom for Palestine.
All Muslim world say : $what
```

1- العمليات الأساسية

المقصود بمعالجة الكلمات ذكر العمليات الأساسية التي قد يحتاجها المطور من جمع للكلمات، تقسيمها و البحث فيها. هذه المعالجة هي ذات طابع تطبيقي عملي أكثر من كونه نظري تحليلي.

1-1- جمع الكلمات

تتعد الطرق لجمع النصوص و الكلمات و يبقى أبرزها عبر استخدام الدالة join. بالإضافة إلى هذه الأخيرة يمكن استخدام المعامل "." (النقطة) أو بكل بساطة الجمع الضمني داخل نص آخر. المثال التالي يوضح كيفية استخدام الطرق الثلاثة و يبقى للمبرمج استخدام الطريقة الأنسب له و لما يريده كنتيجة.

```
#!/usr/bin/perl -w
$string1="This is ";
$string2="a tutorial";
$string3="$string1$string2\n";
print $string3;           # طباعة النص - نتيجة الجمع بين نصين

$string1="This is ";
$string2="a tutorial";
$string3=$string1.$string2;
print "$string3\n";       # طباعة النص - نتيجة الجمع بين نصين

$string3="";
$string3.= $string1;      # <=> $String3 = $String3 . $string1;
$string3.= $string2;
print $string3;           # طباعة النص - نتيجة الجمع بين نصين

$string3="";
$string3= join($string3, $string1 , $string2);
print "\n$string3";       # طباعة النص - نتيجة الجمع بين نصين
```

نتيجة ترجمة الشفرة كانت نفس النص ثلاث مرات. النص المُخرج في كل مرة هو جمع (تسلسل) النصين المختارين مسبقا و لكن الربط بينهما كان بطرق مختلفة.

```
This is a tutorial
This is a tutorial
This is a tutorial
```

ملاحظة: عند الطباعة يمكن الربط مباشرة بدون المرور بمتغير وسيط، لكن يجب الانتباه إلى علامات الاقتباس المزدوجة (").

```
#!/usr/bin/perl -w
$string1="This is ";
$string2="a tutorial";
print "$string1$string2\n";
print $string1.$string2."\n";
print $string1.$string2."\n";
print $string3=$string1.$string2."\n";
```

طباعة النص - نتيجة الجمع بين نصين
 طباعة النص - نتيجة الجمع بين نصين
 طباعة النص - نتيجة الجمع بين نصين
 نتيجة الجمع بين نصين في سطرين

في السطر السادس (6) ستتغير قيمة string1 من This is إلى This is a tutorial بالإضافة إلى علامة الرجوع إلى السطر. منه الترجمة ستكون:

```
This is a tutorial
This is a tutorial
This is a tutorial
This is a tutorial
a tutorial
```

كما نحتاج لربط الكلمات و النصوص نحتاج أيضا إلى تقسيمها، و هو ما سيكون هدف الفقرة التالية من هذا الفصل.

2-1- تقسيم الكلمات و النصوص

سنكتفي في هذا الجزء بإبراز كيفية استخدام الدالة split. تعتمد هذه الدالة أساسا على مدخلين و هما النص المدخل و النص الذي سيعتمد عليه في التقسيم. أما مخرجاتها فهي مصفوفة مكونة من أجزاء النص المقسم.

يمكن أن تجدها معبر عنها بـ "تحويل النصوص إلى قائمة" (Converting Strings into Lists and Hashes).

```
#!/usr/bin/perl -w
$InputString="This is a tutorial";
$Separator=" ";
$count=0;
@Array = split($Separator, $InputString);
$count = @Array;
for($k=0; $k<$count; $k++){
    print "$k) @Array[$k]\n";
}
```

ملاحظة: يمكن الكتابة أيضا (الفصل بالأقواس أحسن قراءة و أسهل صيانة - في المستقبل):

```
@Array = split $Separator, $InputString ;
```

اعتمدنا في عملية التقسيم على الفراغات بين الكلمات إن وجدت. إثر تقسيم و خزن أجزاء النص المقسم إلى كلمات في مصفوفة، تم استخدام دالة الطباعة لإظهار محتوياتها فكانت النتيجة كالتالي:

```
0) This
1) is
2) a
3) tutorial
```

الآن سنرى طريقة أخرى لطباعة ما نريده من كلمات دون استعمال مصفوفة (مما يعني حجز ذاكرة أقل، عدد سطور الشفرة أقل - < تنفيذ أسرع) و هذا دليل آخر على فكرة البيزل القائمة على المعنى المشهور عن لغة البيزل There's more than one way to do it هناك أكثر من طريقة للقيام بذلك أو الاختصار TMTOWTDI.

```
$InputString="This is a tutorial";
$Separator=" ";
foreach (split($Separator, $InputString)) {
    print $count++." ) $_\n";
}
```

الآن من 08 سطور إلى 05 سطور، و بدون مصفوفة، بدون المتغير k، أيضا بدون أن تهيئة لـ count.

"المتغير الخاص" \$_ هو متغير خاص (محجوز) بالبيرل يفيد باحتوائه على النتيجة الحالية (مثل داخل الحلقات) افتراضياً، بمعنى هنا الدالة split لا يوجد متغير لتخزين المعلومات المستخرجة في \$_. طبعاً استعمال الحلقة foreach أو for، كلا الحلقتين تعطيان نفس النتيجة في هذه الحالة. في حال استعمال الميزة w- سنرى تنبيهاً حول المتغير count (يترجم على أنه ربما "خطأ مطبعي") نتيجة الترجمة:

- 0) This
- 1) is
- 2) a
- 3) tutorial

أذكر أن في عملية التقسيم يتم حذف الحرف أو النص المستخدم في عملية التقسيم. مثال على ذلك تبرزه نتيجة ترجمة الشفرة السابقة باستخدام الحرف s عوض عن فراغ (السطران 1 و 2 مسبقاً بفراغ):

- 0) Thi
- 1) i
- 2) a tutorial

3-1- البحث داخل النص

قد يحتاج المبرمج للبحث داخل نص معين ما إن كان يتطابق مع مواصفات معينة أو لا. للاستجابة لهذه الحاجة يتوفر البيرل على عدة دوال و سبل للبحث أبرزها استخدام الدالة index. هذه الدالة ترجع 1- إذا لم يوجد النص المعني في نص البحث و غير ذلك إذا وُجد (أي ترتيبه في نص البحث - البداية من 0 و النتيجة تعبر عن أول حرف من النص المعني).

```
#!/usr/bin/perl -w
$InputString="This is a tutorial";
if (index($InputString,"H")==-1) {
    print "Not Found";
}else{
    print "Found";
}
```

في الشفرة السابقة بحث ما إن كان الحرف H (انتبه إلى أن H يختلف عن h) من بين الحروف المكونة للنص المعطى أم لا مع طباعة نص تأكيد بعدم وجوده أو العكس.

هذه الدالة تبدأ البحث من أول حرف إلى آخر حرف من النص، لتحديد البداية يمكن إدراج الخاصية الاختيارية "المكان" (position)، للتوضيح نأخذ مثال للبحث عن الحرف s في النص السابق، لكن هذه المرة سنستخرج مواضع هذا الحرف في كل النص:

```
#!/usr/bin/perl -w
$InputString="This is a tutorial";
$MyIndex = -1;
do{
    $MyIndex = index($InputString,"s",++$MyIndex);
    if ($MyIndex != -1){
        print "$MyIndex\n";
    }
} until($MyIndex == -1);
print "Done.";
```

أو بكل اختصار (لا حظ أن داخل الحلقة do العملية عبارة عن سطر واحد):

```
#!/usr/bin/perl -w
$InputString="This is a tutorial";
$MyIndex = -1;
do{
    ($MyIndex=index($InputString,"s",++$MyIndex)) != -1
    && print "$MyIndex\n";
} until($MyIndex == -1);
print "Done.";
```

نتيجة الترجمة في كلا المثالين:

```
3
6
Done.
```

يقابل استخدام الدالة index المعامل ~ (حيث ~ = تتحقق من الوجود، ~! تتحقق من العدم) فيصبح المثال السابق كما يلي:

```
#!/usr/bin/perl -w
$InputString="This is a tutorial";
$String1="H";
$String2="tu";
if($InputString !~ $String1){
    print "\"$String1\" was not found in \"$InputString\".\n";
}
if ($InputString =~ $String2){
    print "\"$String2\" was found in \"$InputString\".";
}
```

كالعادة ننقل إلى نتيجة الترجمة و نلاحظ في هذا المثال أنه تم استخدام ثلاثة متغيرات، سنقوم بالتأكد من وجود قيمتان في قيمة المتغير الثالث.

```
"H" was not found in "This is a tutorial".
"tu" was found in "This is a tutorial".
```

بإمكان المبرمج البحث عن أكثر من نص في نص آخر، و للقيام بذلك يكفي استخدام المعامل "|". مثال على ذلك:

```
if ($InputString =~ "$String2|$String1") {}
```

2- تكرار الكلمات و النصوص

من العمليات المفيدة في البيزل هي سهولة تكرار الكلمات أو النصوص دون عناء، تظهر فائدتها الجلية عند التعامل مع المساحات المجهولة الأبعاد (مثل أبعاد الشاشة).

المعامل المخصص لهذه العملية هو x (و ليس X و ليس *)، مثال (لاحظ الفراغ بين رقم التكرار و نقطة الربط):

```
#!/usr/bin/perl -w
$string="Tutorial";
print $string x 3 . "\n";
print "me" x 4 . "\n";
print 50 x 10;
```

نتيجة الترجمة

```
TutorialTutorialTutorial
memememe
50505050505050505050
```

3- العمليات على حجم الحروف

بما أن البيزل في الأساس لغة تقارير فمن غير الممكن أن لا تعنى بظهور الأحرف اللاتينية بحجمها (upper & lower). و لأهميتهما في عملية تنسيق النص للإخراج أو لتوحيد صيغة النص المدخل سنلقي نظرة على مختلف العمليات المخصصة له.

على الأقل نجد أربعة إجراءات تمكننا من تغيير أحجام الحروف (الكلمات أو النصوص): لدينا lc و lcfirst اللتان تمكنان من تحويل جميع حروف النص، أما إذا أردنا تغيير حجم الحرف الأول من النص فنستعمل lcfirst و ucfirst.

مثال:

```
$string1="tutorial";
$string2="TUTORIAL";

print uc($string1)."\n";
print uc("tutorial")."\n";

print lc($string2)."\n";
print lc("TUTORIAL ")."\n";

print ucfirst($string1)."\n";
print ucfirst("tutorial")."\n";

print lcfirst($string2)."\n";
print lcfirst("TUTORIAL ");
```

نتيجة الترجمة:

```
TUTORIAL
TUTORIAL
tutorial
tutorial
Tutorial
Tutorial
tUTORIAL
tUTORIAL
```

أيضا يمكن استعمال الرموز في الجدول "ج 1"، مثال:

```
print "\Ututorial\E\n";
print "\LTUTORIAL\E\n";
print "\ututorial\n";
print "\lTUTORIAL";
```

نتيجة الترجمة:

```
TUTORIAL
tutorial
Tutorial
tUTORIAL
```

4- تحويل النصوص إلى أرقام

عند البرمجة بالبيرل فإنك لن تهتم كثيرا للتحويل من الأرقام إلى النصوص أو العكس، لأنه يهتم بهذا الجانب، و هذا ما يسهل علينا البرمجة خاصة عند الطباعة. في حال كان المتغير يحمل مزيج من الأرقام و النصوص فإن البيرل سيأخذ أحسن النتائج (بلغى من أول كل الحروف التي تلي الرقم المستخرج) مع إظهار رسالة تفيد بأن المتغير المراد تحويله إلى نص (أو العكس) ليس رقما صافيا.

مثال:

```
#!/usr/bin/perl -w
$string="125ab5";
print $string + 1 . "\n";
print "125ab5" + 10;
```

نتيجة الترجمة:

```
126
135
```

في حال وجدت صعوبات يمكن استعمال الإجراء `int` :

```
#!/usr/bin/perl -w
$string="125ab5";
print int($string) + 1 . "\n";
print int("125ab5") + 10;
```

5- عكس الكلمات و النصوص (المرآة)

من الإجراءات المفيدة جدا في لغة البيرل هي عكس الكلمات و النصوص، بعبارة أخرى عملية المرآة للنصوص. المثال:

```
#!/usr/bin/perl -w
$string="em daer nac uoy won";
print $string = reverse $string;
```

نتيجة الترجمة:

```
now you can read me
```

6- تنفيذ أوامر عبر shell

يمكن عبر البيرل تنفيذ أوامر عبر shell و هذا من الأمور التي سهلت على مدراء الأنظمة اختصار الوقت بتفادي الأوامر الروتينية اليومية.

لتنفيذ أمر عبر shell نستعمل دالة الطباعة، ثم نكتب الأمر بين اقتباسين وحيدتين معكوسين (``)، مثال:

[Dos/win]

```
print `dir c:`;
```

[Unix/linux]

```
print `ls /home`;
```

أحيانا مع نسخ لينوكس لا تظهر النتائج إلا بعد إرسال الأمر `dir`، كالتالي:

[Unix/linux]

```
print `ls /home` . "\n" . `dir`;
```

الخاتمة

نختم هذا الفصل بالتذكير أن لغة البيرل ثرية جدا و لا تكفي الصفحات و الصفحات لكشف خباياها و مميزاتها، و تذكر أننا ركزنا على الأساسيات و سردنا النقاط بشكل عملي أكثر منه نظري.

يتبع ..

ⁱ <http://www.linuxjournal.com/article/3394>

ⁱⁱ [http://www.cio.com/article/175450/You_Used_Perl_to_Write_WHAT_Teach_Yourself_Pperl_in_21_Days_Second_Edition_\(2002\)_page_09](http://www.cio.com/article/175450/You_Used_Perl_to_Write_WHAT_Teach_Yourself_Pperl_in_21_Days_Second_Edition_(2002)_page_09); by Laura Lemay

ⁱⁱⁱ Automating Windows with Perl (1999) [page 05]; by Scott McMahan