هناك أكثر من طريقة للقيام بذلك

الأساسي في البيرل

ESSENTIAL IN Perl In Arabic

بن العيد
مروان

النسخة: 09.07

الأساسي في البيرل

ESSENTIAL IN

Perl in Arabic

تعلم أساسيات البرمجة بالبيرل
بلغة عربية

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

بن العيد
مروان

27 أكتوبر 2008

أخر تحديث: 16 جويلية 2009

{إِنَّمَا يُوَفَّى الصَّابِرُونَ أَجْرَهُمْ بِغَيْرِ حِسَابٍ}

سورة الزمر الآية ١٠.

هذه الصفحة تُركت فارغة عمداً

الفهرس

مقدمة

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 scripting يعد الحل الأمثل إلا أن batch و ksh لم يكونا الحلين الأمثلين، فكانت هناك حاجة للغة أخرى تكون أكثر سلاسة و أبسط وهو 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 اجتناب أخطاء المنجرّة عن قلة التركيز عبر تحويل الأوامر و التعديلات اليدوية إلى شيفرة بيرل تنفّذ 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 والمكتبة FTP لنقل الملفات. فبينما نحن نحظر أنفسنا للذهاب للفراش يكون الروس قد حَمَّلوا النسخة و شرعوا في تجربة الإصدار الموجود من نظام التشغيل، و في الصباح نجد تقرير الاختبارات 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 القراصنة المحبذة. يعد الكتاب " Breaking into computer networks from the Internet" لـ Roelof Temmingh أبرز داعم لما ذكرت.

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

[1] **Splint**: a tool for statically checking C programs for security vulnerabilities and programming mistakes.

I – المدخل إلى البيرل

I- 1- مقدمة

نظرا لأهمية لغة البرمجة بيرل في تطوير البرمجيات و ما تتسم به من ثراء معرفي نشارككم في هذه الأسطر خبرتنا 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.

I- 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)[I]. 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.

I- 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:[II]

- ✓ 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 للتعمق و التحكم في نظام التشغيل.
- ✓ مبرمج تحتاج إلى لغة نموذجية أكثر تعقيدا و مردودية.
- ✓ تريد استبدال تعبيرات Shell بتعبيرات 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 والتعديل أسرع على المصدر (source).

I- 4- تثبيت البيرل

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 عليها[III] ؛ أما إذا كنت مستعمل ويندوز فيوجد ActivePerl و تتعدد إصداراته وفق نظام التشغيل و المعالج أيضا.

4-1- على لينوكس

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

```
perl -v
```

عند نجاح تنفيذ هذا الأمر نحصل على إصدار البيرل المتوفر في نظام التشغيل، وبطريقة غير مباشرة تم التأكد من تواجده في نظام التشغيل. 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 إيجاده تحقق عبر مدير النظام.

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

4-2- على ويندوز

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

لإضافة البيرل إلى المسار path - مع أنك لن تحتاج لهذه العملية بعد 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
```

أو أكتب في نافذة الأوامر:

```
path
```

أو :

```
PATH
```

الأمر path أو PATH يرجع جملة المسارات المشتركة في نظام التشغيل. وتسجيل مسار تواجد مترجم البيرل يمكن من استدعائه في أي مسار آخر، لكن الحال ليس ذاته بالنسبة للسكريبت الذي نريد ترجمته.

I- 5- قبل البرمجة بالبيرل

1-5 - تحرير الشفرة

للمبرمج الحرية الكاملة في اختيار برنامج كتابة الشفرة رغم وجود برامج خاصة بذلك. هذه الحرية تعود إلى استقلالية كتابة الشفرة عن عملية الترجمة؛ مثال الفيجوال ستوديو هو مجرد محرر للشفرة لكن ارتباطه بالمترجم جعله يتجاوز تلك الوظيفة التقليدية ليبسر عمليات أكثر أهمية كالترجمة و تتبعها خطوة بخطوة أو ما يعرف بإسم Debug.

لذلك فللمطور الحرية الكاملة في اختيار برنامج لتحرير الشفرة شرط أن لا يكون لهذا الأخير تأثير في النص حتى و إن كانت غير مرئية؛ البرنامج WordPad من البرامج التابعة لنظام التشغيل الويندوز بمختلف فروعه. هذا البرنامج يختص بالملفات من نوع معيَّن ولو استخدمناه كمحرر للشفرة فقد تعترضنا أخطاء في الترجمة رغم صحة التعليمات المكتوبة. مثال على هذه الأخطاء:

الشفرة	نتيجة الترجمة
print "hello"; print "Hello"; print "Hello";	C:\>perl c:\test.pl Unrecognized character \xE6 at c:\aa.pl line 2. C:\>perl c:\test.pl panic: utf16_to_utf8: odd bytelen at c:\aa.pl line 1.

لا شك في صحة الشفرة إنما الشك في صياغة النص المحفوظ. تفسير هذه الأخطاء يعود لإضافة البرنامج WordPad لرموز خاصة تساهم في تنظيم النص دون تكون مرئية. اجتنابا لمثل هذه المشاكل يكفي استخدام محرر النصوص المرفق مع نظام التشغيل (الصورة 1.ب) و للترجمة فعليك فتح نافذة الأوامر[1] و كتابة إسم السكريبت من مسار تواجده أو كتابة المسار كاملا كما هو مجسد في الصورة 1.أ.

```
print "Hello Arabteam2000\n";
```

1.أ. شفرة المثال

```
c:\>test.pl
Hello Arabteam2000
C:\>
```

1.ب. ترجمة المثال

محررات متوافقة مع الويندوز: يمكن استعمال notepad و wordpad، و للمزيد من الميزات هناك nedit، UltraEdit، NotePAD++، Komodo IDE (من ميزاته تشغيل الشفرة مباشرة دون إستعمال cmd)...

محررات متوافقة مع يونيكس (لينوكس): الشهيران Emacs و vi ، أيضا هناك kwrite.

بالنسبة لنظام التشغيل MAC: يمكن استخدام محرر النصوص Alpha أو BBEdit الذي يعتبره أغلب المطورون من أحسن أدوات تحرير النصوص في نظام التشغيل MAC.

[1] cmd: نافذة الأوامر

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

أولا:

من الممكن عدم استعمال محرر النصوص لكتابة الشفرة ذات السطر الواحد و ذلك باستعمال نافذة الأوامر 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, ArabTeam2000\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).

ثانيا:

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 كاتبة 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 - صحيحة
no warnings; # لا نريد إظهار التحذيرات
use warnings; # إعادة استعمال التحذيرات
```

عند استعمال W– (upper-case) سنجبر 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 تضييع 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–.

ثالثا:

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

I- 6- المتغيرات

تعتمد لغة البيرل على الرمز $[1] لتعريف المتغيرات و تشترط استخدام المتغيرات وفق تعريفها، بمعنى أن أي استخدام لمتغير لم يتم تهيئته مسبقا أو مطلقا يتم تجاهله في عملية الترجمة. (عند استخدام w– ستظهر رسالة تفيد بوجود متغير غير مُعرف – استخدام متغير غير مهيأ).

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

المتغيرات عبارة عن كلمات (من الأفضل أن تكون ذات معنى) لا تبدأ برقم و اختلاف حجم الحروف[2] 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.

أيضا يمكن (و من المستحسن) تعريف المتغير بالكلمة المحجوزة my، التي سنتطرق إليها. سنجرب جمع متغيرين وإظهار النتيجة على الشاشة كما هو في المثال التالي ـــ علما أن ترقيم الأسطر فقط للشرح، لا تُكتب 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 تعليمة الطباعة الثانية (السطر 7) تم إدراج ناتج الجمع في النص المُخرج. في التعليميتين الأخيرتين للطباعة ولكن بتغيرات 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 أخر)، في ما يلي نتيجة الترجمة:

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

[1] السبب في اختيار هذا الرمز $ هو أنه يشبه أول حرف في الكلمة Scalar وهو نفس الحرف في تعبير الهاكرز – أي يمكن قراءتها كالتالي $calar

[2] [en] Upper and lower case letter / [fr] Lettre majuscule et minuscule

أو يمكن التنفيذ بدون الكلمة perl في حال كان امتداد الملف 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 echo %cd%. لترجمة الشفرة يمكن كتابة اسم الملف مع الامتداد مسبوقا بالتعليمة perl أو بدونها لأن إثر تنصيب البيرل يتم إضافة مكتباته مع قائمة المسارات التي يعتمدها نظام التشغيل و يمكن مطالعة باستخدام الأمر %path%. في حين إن إدراج التعليمة perl أمر إجباري إذا كان نظام التشغيل معتمد على يونيكس و فروعه.

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 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 ذُكر آنفا من ضرورة سبق 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 مهيأ. 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 توزعه باستخدام الرموز \a، \t، \n و غيرها – أنظر الجدول التالي للتعامل مع النصوص.

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 بالثماني أو 0x1b بالست-عشري
\0nnn	تعني الرمز المقابل ل nnn حيث nnn رقم بالثماني مثلا 33 هي escape و 101 تعني حرف A
\xnn	تعني الرمز المقابل ل nn حيث nn رقم بالست-عشري مثلا 1bهي 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

الجدول 1: التحكم في شكل النص.

6-1- الإدخال

عملية الإدخال (أو القراءة من لوحة المفاتيح) تتم عبر التعليمة <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 $a;                          #  هنا لا بد أن نلاحظ أن المتغير يحتوي على علامة سطر جديد
print "Enter: ";                    # يطبع في سطر جديد
chomp($a = <STDIN>);               # حذف علامة السطر الجديد
print $a;                          # لا تحتوي على علامة سطر جديد
print "Enter: ";                    # يطبع إلى جانب المتغير $a
chomp($a=<STDIN>);                 # حذف علامة السطر الجديد
print $a;                          # لا تحتوي على علامة سطر جديد
print "\nAdd \\n for new line";     #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
```

قد نستخدم متغير دون تعريفه مسبقا سوى سهوا أو خطأ في تعليمات سابقة أو غير ذلك من الأسباب. و للتأكد ما إن كان متغير ما معرفا أم لا، يمكننا استخدام الكلمة المفتاح defined كما هو معرف في المثال التالي:

```
if (defined $VariableName1)
#…
if (not defined $VariableName2)
#…
```

I- 7 - العمليات الحسابية و النصية

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

7 -1- العمليات الرياضية

الجدول التالي يعبر عن العمليات الرياضية في البيرل موضحة بمثال:

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

الجدول: العمليات الرياضية في البيرل.

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 البيرل يعتبر كل ما بين قوسين على أنها إجراء داخلي، عند استعمال الميزة w– ستظهر رسالة تفيد على أن ما بين القوسين بعد دالة الطباعة تترجمه كإجراء، إلا إذا استعملنا الجمع (+) 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 فرضا أن هذا المتغير مهيأ من قبل، 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 البيرل تختلف إن تقدمتا المتغير أو تلته، أي:

```
print $a++ ;  # النتيجة = 0
```

تختلف عن:

```
print ++$a ;  # النتيجة = 1
```

التفسير:

في الأولى، مترجم البيرل يقوم بطباعة قيمة المتغير ثم يزيد إلى القيمة واحد (المتغير سيتم تهيئته بـ 0 في حال فراغه).

في الثانية مترجم البيرل يزيد 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 إلى0
print "\n" ;
print $b ;
```

نتيجة الترجمة كالآتي (السطر الأول فراغ):

```
1
0
1
```

7 -2- العمليات المنطقية

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

المعنى	خاص بالنصوص	خاص بالأرقام
تساوي	Eq	==
غير متساوي	Ne	!=
أصغر	Lt	<
أكبر	Gt	>
أصغر أو يساوي	Le	<=
أكبر أو يساوي	Ge	>=

الجدول: المعاملات

II- 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.

II-1- الجمل الشرطية

الشروط if .. else و elseif

فيما يلي مثال لمقارنة بين نصين لاتينيين باستخدام كافة علامات المقارنة، و الغاية منه أولا تجربة هذه المعاملات و ثانيا 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 كالآتي:

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

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

المقارنة بين النصوص ترتكز على الترتيب للحروف و الحرف الأسبق من غيره يعد الأصغر، أما داخل النص في حد ذاته فهناك ترتيب تفاضلي لمدى أهمية الحروف أيضا حسب ترتيبه في الجملة. لذلك فإن الحرف الأول هو الأكبر. لو اِطَّلعنا على النصين المعنيين فسنلاحظ أن النص الأول يبدأ بالحرف A في حين الثاني يبدأ بالحرف B والأخير هو أكبر، و رغم عكس الحالة و وضع حرف ثاني في النص الأول أكبر من الحرف الثاني في النص الثاني فإن النتيجة لم تتغير نظرا 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 البيرل، عند المقارنة بين نصين يجب استعمال معاملات الخاصة بها، و العكس بالنسبة للأرقام لأنها ستحوّل إلى أرقام، مثلا:

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

مثلا:

```
#!/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 البيرل سيرتب المقارنات رقميا أو أبجديا.

مثال، لنقوم بالمقارنة التالية هل 99 أصغر من 100.. نحن نعلم النتيجة، لكن لنرى كيف نقدمها لمترجم البيرل 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 كالآتي:

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

التفسير: عدديا 99 أصغر من 100 (رقميا)

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

الشرط unless

كما نستطيع استعمال unless التي تعبر عن عكس if، إنها من الوسائل التي يوفرها البيرل 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 : إطبع "shame on you!" مهما كانت قيمة المتغير what إلا في حالة إذا كانت القيمة هي "freedom". الدالة uc هي محول النص في المتغير إلى حروف كبيرة (upper-case).

أيضا، توجد طريقة أخرى يوفرها البيرل 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 البيرل 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";
```

II-2- الحلقات

في البيرل نجد نوعين من الحلقات:

- الحلقات التكرارية و تتمثل في for و fereach.
- 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 سهلا.

أبسط مثال لكتابة حلقة تكرارية for نكتبه توضيحيا حيث للحلقتين 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 للحلقتين 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 للحلقة 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
```

ما يميزها أنها تعامل المتغير الخاص بها كمتغير "داخلي خاص بها" حتى و لو كان هناك متغير يحمل نفس الاسم خارجها – و الفائدة الجلية لهذه الحلقة تظهر عند التعامل مع المصفوفات و القوائم (خاصة المتمددة – 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 البيرل 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 تضيع القيمة السابقة و تعود var لما كانت عليه أي 42. حتى ولو تم التعديل على المتغير بعد عملية الطباعة مباشرة، لن تتغير قيمته، مثال

```
#!/usr/bin/perl -w
$var = 42;
print "Before: $var \n";
foreach $var (1..5) {
        print "Inside: $var \n";
        $var = 100;
}
print "After: $var \n";
```

نتيجة الترجمة:

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

لكن قبل الطباعة سيتأثر قيمة المتغير داخل الحلقة، لكن لا يتأثر المتغير العام:

```
#!/usr/bin/perl -w
$var = 42;
print "Before: $var \n";
foreach $var (1..5) {
        $var = 100;
        print "Inside: $var \n";
}
print "After: $var \n";
```

نتيجة الترجمة:

```
Before: 42
Inside: 100
Inside: 100
Inside: 100
Inside: 100
Inside: 100
After: 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
```

السبب لأنها متغير "عام" لكل البرنامج و ليس "خاص" – (أنظر الإجراء my).

الحلقة الشرطية while

نسمي هذه الحلقة بالشرطية لأنه لا تتعامل مع المتغيرات لكي تتابع عملها، و إنما تتعامل مع الشروط (المنطقية والرياضية)، 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 حرية 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 أن 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.
```

III- بعض الدوال

III- 1- الإجراء my

my إجراء يحجز قائمة للاستعمال المحلي لفائدة القطعة المتواجد ضمنها. يمكن استخدامها لحجز متغير واحد، أو قائمة وفي هذه الحالة يجب أن تكون هذه القائمة بين قوسين.

تظهر فائدتها الجلية في الإجراءات الداخلية للبرنامج، و أيضا عند كتابة الوحدات.

مثال:

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

نتيجة الترجمة:

```
42
12345
6
```

III- المصفوفات و القوائم

III- 1- المصفوفات

المصفوفات هي عبارة مجموعة من المتغيرات مخزنة في متغير واحد. للتعبير عن مصفوفة في البيرل 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 البيرل لكن @1_thounsand 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 البيرل بطباعة المحتوى تباعا و في النهاية ستظهر كسطر واحد لا نفرق بين عناصر هذه المصفوفة؛ أما في حال استعمالها سيقوم البيرل 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";
```

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

النتيجة:

```
12345678910
abcdefghijklmnopqrstuvwxyz

*****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 وعلامات 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");
```

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. 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 البيرل أن ضيف أو نحذف عناصر من المصفوفة، و الإمكانية من الجهتين:

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

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";
```

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";
```

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 و حذف 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)

splice(المصفوفة, العنصر-الأول, الطول-تتابعيا, العناصر الجديدة)

"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 (8،7،6) 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";      # 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";
```

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";
```

النتيجة:

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

III- 2- 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 على الترتيب،

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";
```

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
```

III-3- ملاحظات

يجب الانتباه عند التعامل مع القوائم - توسيطها بين علامتي الاقتباس المزدوجة - و المصفوفات - توسيطها بين الأقواس-

.

مثال يوضح الفرق:

```
1| #!/usr/bin/perl
2|
3| @MyArray = (1,2,3,4,5);          # مصفوفة ذات خمسة (5) عناصر
4| ($first) = @MyArray;             # قائمة ذات عنصر واحد و المصفوفة كعدد العناصر
5| ($elements1) = "@MyArray";       # قائمة ذات عنصر واحد
6|                                  # و المصفوفة كسطر واحد يفصل بين عناصره فراغ
7| $count = @MyArray;               # المصفوفة كعدد العناصر
8| $elements2 = "@MyArray";         # المصفوفة كسطر واحد يفصل بين عناصره فراغ
9|
10| print $first."\n";
11| print $elements1."\n";
12| print $count."\n";
13| print $elements2;
```

النتيجة:

```
1
1 2 3 4 5
5
1 2 3 4 5
```

التفسير:

أولا لدينا مصفوفة تتكون من خمسة (05) عناصر مرقمة من 1 إلى 5.

في السطر الرابع (4) لدينا قائمة مكونة من عنصر (متغير) واحد، بالتالي سيتم أخذ قيم العناصر من المصفوفة على الترتيب و بما أنه لدينا عنصر واحد في القائمة (الذي هو المتغير $first) فإنه سيأخذ أول عنصر، و البقية سيتم تجاهلها لعدم وجود مكان تخزن فيه.

في السطر الخامس (5) لدينا قائمة مكونة من متغير واحد، و أيضا لدينا مصفوفة في طرفيها علامتي اقتباس مزدوج (")، بالتالي سيتم أخذ قيم عناصر المصفوفة - على الترتيب يفصل بينها فراغ - كقيمة واحدة؛ و بما أنه لدينا عنصر واحد في القائمة (الذي هو المتغير $first) فإنه سيأخذ هذا العنصر الناتج من المصفوفة - نفس عملية طباعة المصفوفات التي ذكرناها في دور علامة الاقتباس المزدوج (") في فقرة **حجز مصفوفة** في بداية هذه الفقرة.

في السطر السابع (7)، لدينا متغير سيأخذ **"عدد"** عناصر المصفوفة (لا توجد أقواس تضم المتغير، ولا علامة اقتباس مزدوج في طرفي المصفوفة).

في السطر الثامن (8)، لدينا متغير سيأخذ عناصر المصفوفة كقيمة واحدة تضم قيم كل العناصر يفصل بينها فراغ (لا توجد أقواس تضم المتغير، وتوجد علامة اقتباس مزدوج في طرفي المصفوفة).

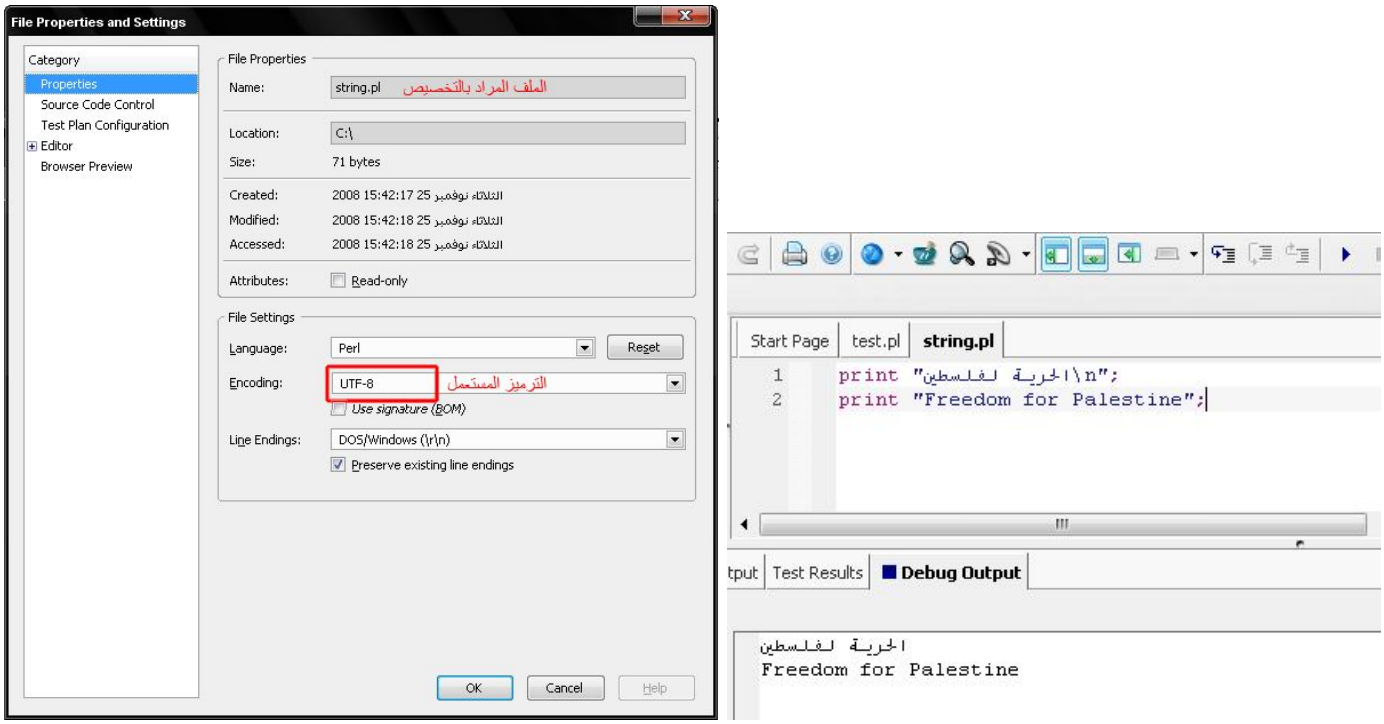
إذن، يبقى الاختيار للمبرمج حسب ما يريد كنتيجة أو معاملة المعطيات.

IV – معالجة الكلمات و النصوص

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

قبل المباشرة في هذا الفصل من المفيد معرفة أن من أبرز نقاط قوة لغة البيرل هي في معالجة الكلمات والنصوص، ومنذ النسخة 5.6 للبيرل أصبح لديه القدرة على التعامل مع ملفات Unicode التي تهمنا 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 ☺. في البيرل 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 الوحيدين سيأخذ كما هو دون اعتبار لقيمة المتغيرات.

```
#!/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
```

IV- 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- 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";       # طباعة النص – نتيجة الجمع بين نصين
print $string1.$string2."\n";     # طباعة النص – نتيجة الجمع بين نصين
print $string1.=$string2."\n";    # طباعة النص – نتيجة الجمع بين نصين
print
   $string3=$string1.$string2."\n"; #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- تقسيم الكلمات و النصوص

سنكتفي في هذا الجزء بإبراز كيفية استخدام الدالة split. تعتمد هذه الدالة أساسا على مدخلين و هما النص المدخل والنص الذي سيعتمد عليه في التقسيم. أما مخرجاتها فهي مصفوفة متكونة 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 البيرل القائمة على المعنى المشهور عن لغة البيرل 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) افتراضيا، بمعنى هنا الدالة split لا يوجد متغير لتحفظ فيه النتيجة لكنها ستخزن المعلومات المستخرجة في $_.

طبعا استعمال الحلقة foreach أو for، كلا الحلقتين تعطيان نفس النتيجة في هذه الحالة.

في حال استعمال الميزة w– سنرى تنبيها 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 مسبوقان بفراغ):

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

3- 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 ننتقل 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"){}
```

IV- 2- تكرار الكلمات و النصوص

من العمليات المفيدة في البيرل 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
```

IV- 3- العمليات على حجم الحروف

بما أن البيرل في الأساس لغة تقارير فمن غير الممكن أن لا تعنى بظهور الأحرف اللاتينية بحجميها (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 uc و lc اللتان تمكنان من تحويل جميع حروف النص، أما إذا أردنا تغيير حجم الحرف الأول من النص فنستعمل ucfirst و lcfirst.

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 "جـ 1"، مثال:

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

نتيجة الترجمة:

```
TUTORIAL
tutorial
Tutorial
tUTORIAL
```

IV- 4- تحويل النصوص إلى أرقام

عند االبرمجة 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 فأن البيرل سيأخذ أحسن النتائج (يلغي 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;
```

IV- 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
```

V- الملفات و المجلدات

نواصل رحلتنا في تعلم البيرل و ما تخفيه هذه اللغة من ثراء، لنتطلع على أسس التعامل مع الملفات و المجلدات من حذف ونسخ و تعديل و غير ذلك من ما قد يحتاجه المطور.

IV- 1- الملفات

قد نحتاج لفتح ملف ما للقراءة منه، الكتابة فيه أو تحديث محتوياته. كباقي لغات البرمجة توفر لغة البرمجة البيرل هذه الصلاحية بالإضافة إلى إمكانية تتبع الاستثناءات كغياب الملف الذي نود تحديث نصه في المسار المحدد، أو فقدان الصلاحيات الكافية. نبدأ تعلم التعامل مع الملفات عبر تجربة التعليمتان التاليتان:

```
open(TEST_FILE, "TestFile1.txt");
open(TEST_FILE, ">TestFile.txt");
```

تكمن الاختلافات بين التعليمتان في اسم الملف و إدراج الرمز ">". عند تنفيذ هاتان التعليمتان يتم إنشاء ملف واحد TestFile.txt عوض عن ملفين. من هنا نفهم أن الرمز المضاف للتعليمة الثانية يمكن من إنشاء الملف. يستخدم هذا الأخير للكتابة في ملف و إنشائه في حال عدم تواجده كما هو مجسد في الجدول التالي:

الرمز	الوضـــــيفة
>	الكتابة
>>	التحديث
<	القراء

المسار التلقائي للملفات هو مسار استدعاء السكريبت. لذلك يجب التأكد أولا من تواجد الملف المعني في مسار استدعاء السكريبت أو تحديد المسار الكامل. لمزيد فهم هذه النقاط سنطور سكريبت يقرأ لائحة من المسارات من ملف و ينشأ ملف في كل منها و ينسخ فيه مساره. قبل تجربة الشفرة ينبغي إنشاء الملف و نسخ لائحة المسارات فيه.

```
c:\Perl\
d:\
c:\Windows\

c:\
c:\windows\System32\
```

تعمدت ترك سطر فارغ في لائحة المسارات و سأعود لذلك لاحقا عند شرح شفرة السكريبت.

```
#!/usr/bin/perl -w
$index=0;

open(PATHS_FILE, "< c:\\Paths.txt");
while (<PATHS_FILE>)
{ $path=$_;
  chomp($path);
  $FileFullPath = $path.$index.".txt";
  open(IN_FILE, ">$FileFullPath");
  print (IN_FILE $FileFullPath);
  close($FileFullPath);
  $index++;
}
```

قبل الشروع في شرح الشفرة و مختلف تعليماتها، أذكر بضرورة التركيز و محاولة فهمها فهما جيدا لكونها حوصلة للكثير مما تم ذكره سابقا.

- في بداية الشفرة عرّفت المتغير index على أنه خاص لأنه لن يكون مشترك بين عدة دوال و إلا كنت أضفت له الكلمة المفتاح my. سيشكل هذا المتغير أسماء الملفات التي سيتم إنشائها في مختلف المسارات.
- عند فتح ملف ما لابد من استخدام مؤشر له و هو ما يعلل استخدام المتغيرPATHS_FILE الذي سنستخدمه لاحقا للقراءة من الملف.
- سبق و تحدثنا عن الحلقات بمختلف أنواعها، و في هذا المثال تجسيد لكيفية استخدامها للقراءة من ملف، سطرا سطرا حتى بلوغ آخره. المتغير $_ محجوز في البيرل و يحتوي آخر نتيجة متحصل عليه، و في مثالنا هذا النتيجة هي قيمة السطر المحمّل من الملف النصي.
- المسارات مكتوبة في الملف في شكل سطور مستقلة و هو ما يعني تواجد الرمز"\n" في نهاية كل سطر. لذلك استخدمت الدالة ompch لحذف ذلك الرمز و إرجاع نص المسار فقط.
- إثر الحصول على نص المسار من الملف، نحتاج لإضافة اسم الملف و امتداده. باعتماد ما ورد سابقا في فقرة تركيب النصوص تمكنت من إضافة إسم الملف، و هو قيمة المتغير index مع الامتداد"txt.".
- بنفس طريقة إنشاء الملف TestFile.txt، نقوم بإنشاء هذه اللائحة من الملفات و لكن في هذه المرة تم تحديد المسار الكامل للملف عبر المتغير FileFullPath الذي يتم بناء قيمته في كل مرة.
- نحتاج الآن لكتابة المسار الكامل للملف داخله، هذه العملية ممكنة باستخدام الدالة print التي اعتمدناها سابقا لإظهار النص في الشاشة. لكن هذه المرة سنحول و جهة النص إلى الملف الذي تم إنشائه ثم نغلقه باستدعاء الدالة close.
- نختم الشرح بالتذكير بالعمليات الرياضية التي تم الإشارة لها سابقا، حيث استخدمنا عملية الزيادة الذاتية للمتغير index في نهاية كل حلقة. بهذه الكيفية نتمكن من إنشاء ملفات بأسماء مختلفة.

قبل المرور إلى ما يلي بودي لو تبحثوا عن الملف 3.txt و تعليل مسار تواجده.

بعد معرفة كيفية الكتابة و القراءة في الملفات ننتقل إلى الجزء الموالي للإطلاع على كيفية حذف، نسخ و تعديل الملفات. لكن قبل الشروع في ذلك أذكر أن لغة البرمجة البيرل تشبه باقي لغات البرمجة من حيث إمكانية الإعتماد على مكتبات خارجية يتم استدعائها عبر الكلمة المفتاح use و التي يقابلها في لغة السي شارب using و include في لغتي السي

والسي++. لذلك ينبغي التأكد أن المكتبة المستدعاة متواجدة في المسار Perl/lib أو Perl/site/lib، وفق نوع التطبيق. في حال عدم تواجد المكتبة فسنصطدم برسالة خطاء شبيهة بالرسالة التالية:

```
Can't locate Net/Telnet.pm in @INC (@INC contains:
c:/Perl/lib c:/Perl/site/lib.) at Copier.pl line 3.
BEGIN failed--compilation aborted at Copier.pl line 3.
```

غالبا ما تكون المكتبات عبارة عن ملفات من نوع pm، يمكن قراءة شفرتها و تعديلها. سنعود لاحقا إنشاء الله لموضوع المكتبات بمزيد من التفاصيل و قبل ذلك نعود للتعامل مع الملفات. لنسخ الملفات و نقلها نحتاج لإدراج المكتبة، تحتوي هذه المكتبة العديد من الوظائف و الدوال أبرزها الدالتان copy و move.

```
#!/usr/bin/perl -w
use File::Copy;

copy("C:\\Paths.txt","D:\\Paths.txt");
move("D:\\Paths.txt","C:\\Paths.txt");
copy("C:\\Paths.txt",\*STDOUT);
unlink("C:\\Paths.txt");
```

في المثال أعلاه نسخت الملف من المسار C:\Paths.txt إلى D:\Paths.txt ثم نقلته إلى مكانه الأصلي ثم أظهرت نصه على الشاشة ثم حذفته. ربما هذه السلسلة من العمليات عقيمة و بدون معنى و لكن بما أن الغاية تعليمية فيمكن تقبل مثل هذه الخوارزميات.

- تمكن الدالة copy من نسخ ملف من مسار إلى مسار آخر مع التنبيه لضرورة حسن كتابة المسار كما هو في المثال لأن الرمز "\" محجوز في البيرل و إذا أردنا إدراجه داخل نص فعلينا كتابته مرتين متتاليتين. الدالة copy قادرة أيضا على إظهار نص ملف ما على الشاشة. قد تمكننا هذه الوظيفة من اجتناب كتابة خوارزمية كاملة.
- أخيرا إذا أردت نقل ملف من مسار إلى مسار آخر فأنصحك بالدالة move، فهي قادرة على ذلك طالما لم يحدث خلل ما. أعني بالخلل إمكانية تعطل عملية نقل الملف لعدة أسباب كفتح الملف أثناء نقله أو أن الملف مصاب... في هذه الحالة نحصل على نسخة جديدة غير كاملة من الملف و نفقد النسخة الأصلية، بما أن عملية النقل بمثابة نسخ الملف ثم حذف النسخة الأصلية.
- أخيرا لحذف ملف ما يمكن استخدام الدالة unlink. هذه الأخيرة لا تنتمي للمكتبة File::Copy لذلك لست مجبرا على إدراج المكتبة عند الحاجة لحذف ملف.

لتسهيل المهمة على المبرمجين الذين هم من محبي الاختصارات، أذكر بإمكانية استخدام الكنية[1] cp عوضا عن اسم الدالة copy شرط تعريف الكنية مسبقا:

```
#!/usr/bin/perl -w
use File::Copy "cp";
cp("C:\\Paths.txt","D:\\Paths.txt");
```

أجد نفسي مجددا مجبرا على سبق الأحداث، لألمّح لإمكانية استخدام الدوال في البيرل، و يكون ذلك عبر استخدام الكلمة المفتاح sub. بطبيعة الحال يمكن تمرير مدخلات لهذه الدوال و قراءة مخرجاتها إن وجدت.

[1] الكنية : Alias

الغاية من هذا التقديم السريع هو حاجتنا لاستخدام دالة في البحث عن الملفات، هذه الدالة تستدعى إذا ما وجد الملف أو الملفات المعنية، كما هو في المثال التالي حيث إذا وجد الملف المطلوب تظهر لنا رسالة على الشاشة معللة نجاح عملية البحث.

```
#!/usr/bin/perl -w
use File::Find;

sub wantedFile
 { print "$File::Find::name was found.";   }

find(\&wantedFile, "c:\\MyTextFile.txt");
```

أختم الحديث عن الملفات بالمثال التالي حيث المقارنة بين ملفين. ترجع الدالة Compare القيمة 0 إذا تماثل الملفان، 1 إذا اختلفا و -1 إذا طرأ خلل ما في عملية المقارنة كما هو مجسد في الاستدعاء الأول للدالة نظرا لغياب المسار /etc/hosts في نظام التشغيل ويندوز.

```
#!/usr/bin/perl -w
use File::Compare;

print compare("\etc\hosts","c:\\wepkeys.txt");
if (compare("c:\\File1.txt","c:\\File2.txt") == 0)
{   print "\nThey are equals\n";   }
```

IV- 2 - المجلدات

بعد معرفة كيفية الكتابة و القراءة في الملفات ننتقل إلى الجزء الموالي للإطلاع على كيفية حذف، نسخ و تعديل الملفات.

نبدأ بالمثال الأول حيث نتعلم كيفية فتح مجلد، قراءة محتوياته ثم غلقه. و بعد ذلك نطبع على الشاشة ما إن كان العنصر المقروء ملف أو مجلد.

```
opendir DIR, ".";
@contents = readdir DIR;
closedir DIR;

foreach $item ( @contents )
{
  if ( -d $item )
    { print "$item is a directory!\n";}
  else
    { print "$item is a file\n";}
}
```

سلسة تمارين

1. أكتب سكريبت يبحث عن الكلمة Rejected أو Crashed في ملف نصي Start_processes.log و ينسخ الأسطر التي تحتوي هذه الكلمات في ملف منفصل Start_Errorssproces.log.

2. أكتب سكريبت يقرأ ملف cpp. و يرجع عدد تعليماته و عدد الأسطر الفارغة.

3. أكتب سكريبت ينظم مخرجات التعليمة dir المحفوظة في ملف dir.log و ينسخها في ملفين منفصلين أحدهما خاص بالملفات Dir_Files.txt و الثاني بالمجلدات txt. Dir_irsD.

4. مطلوب سكريبت يتم إستدعائه يوميا لتنظيم ملفات يخرجها برنامج ما في المسار C:\MyProgram و يناهز عددها السبعة آلاف ملف. يمكن التميز بينها عبر إمتداداتها rpt، LFB و log. المطلوب حذف الملفات rpt و نسخ الملفات log في المسار D:\Applis\logs. أما الملفات LFB، فينبغي نقلها للمسار D:\Applis\LFB بعد تعويض الجملة RAM_MAX=524 بالجملة RAM_MAX=1024 في كل منها.

1

```
#!/usr/bin/perl -w
$LogFile = "c:\\processesLog.log";
open(LogFile, "< $LogFile") or die "Failed to open
$LogFile";

while (!eof(LogFile))
{
   $line = <LogFile>;
    if ($line=~"Crashed|Rejected")
  {     print $line; }
}
```

2

```
#!/usr/bin/perl -w
if($#ARGV<0)
{ die("No cpp soure file was entred.");}

$CppSourceFile = $ARGV[0];
open(Source_File, "< $CppSourceFile");
$EmptyLines=1;
$TotalLines=1;
while (<Source_File>)
{ if(length($_)>1)
 {$EmptyLines++;}
  $TotalLines++;
}
print "\n\tTotal number of lines=$TotalLines\n\tTotal
number of empty lines=$EmptyLines"
```

3

```
#!/usr/bin/perl -w
open(Dir_Log, "< c:\\dir.log");
open(Dir_Files, ">c:\\Dir_Files.txt");
open(Dir_Dirs, ">c:\\Dir_Dirs.txt");
while (<Dir_Log>)
{ if(index($_,"<DIR>")==-1)
 {print (Dir_Files $_);}
  else
  {  print (Dir_Dirs $_);}

}
```

4

```
#!/usr/bin/perl -w
use File::Copy;
$WorkingPath ="c:\\MyProgram\\";
$LFBFilesPath ="D:\\Applis\\LFB\\";
$LogsFilesPath ="D:\\Applis\\logs\\";

opendir DIR, "$WorkingPath";
@contents = readdir DIR;
closedir DIR;

foreach $item ( @contents )
{
  if ( -d $item )
   {#Nothing to do
   }
  else
    {
 if($item=~".rpt|.RPT")
  {unlink($WorkingPath.$item);}
 elsif($item=~".log|.LOG")
       {move($WorkingPath.$item,$LogsFilesPath.$item);}
 elsif($item=~".lfb|.LFB")
  { ######
   open(LFBFile, "< $WorkingPath$item") or
                                die "Failed to open
$WorkingPath$item";
         open(tmpLFBFile,"> $WorkingPath$item.tmp") or
                                die "Failed to open
$WorkingPath$item";
          while (!eof(LFBFile))
           {
             $line = <LFBFile>;
             if ($line=~"MAX_RAM=524")
          {
#$line=replace("MAX_RAM=524","MAX_RAM=1024");
                  $line=~
s/"MAX_RAM=524"/"MAX_RAM=1024"/;
            print $line;
          }
          print (tmpLFBFile $line);
           }

     close(tmpLFBFile);
     close(LFBFile);

     move($WorkingPath.$item.".tmp",
$WorkingPath.$item);
     move($WorkingPath.$item, $LFBFilesPath.$item);
        }
   }
}
```

VI- 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 وحيدين 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`;
```

ما سبق ذكره عبارة عن عملية "تحيل" في البرمجة. حيث نرسل التعليمات التي نريد تنفيذها في شكل نص مخرج من دالة الطباعة. و لكن ماذا لو أردنا تنفيذ سكريبت آخر خاصة إن كن مبرمج بلغة غير البيرل مثل batch, ksh أو sh غير ذلك .الحل يكمن في استخدام الدالة system. هذه الدالة تقوم بتنفيذ الأوامر الممررة لها في شكل نص كما هو مجسد في المثال التالي:

```
#!/usr/bin/perl -w
system("dir c:");
system("start dir c:");
system("call notepad.exe");
```

كما هو مجسد في المثال السابق, تم استدعاء الدالة system ثلاث مرات. في المرة الأولى لطباعة محتويات المسار الرئيسي للحاسوب C:\. تماثل وضيفة الأمر الثاني سابقتها لكن هذه المرة الطباعة في واجهة cmd جديدة. أخيرا استدعاء الدالة system لتنفيذ ملف تشغيل notepad.exe.

عادة نتعلم البرمجة بالطرق الرعوانية و مع الوقت نفهم ما كنا نلصق من أكود بطريقة تلقائية. اتبعت هذا المبدأ في التقديم للدالة system حيث ذكرت أن استدعاء الدالة ممرين لها نص تقوم بتنفيذه آليا، دون ذكر أن هذه الدالة تقبل أكثر من متغير و ترجع أيضا قيمة. هذه الأخيرة تمكننا من التأكد من نجاح عملية الاستدعاء أو فشلها كما هو في المثال التالي:

```
#!/usr/bin/perl -w
system("color b1");
$cmd = 'caller.cmd';
@cmd_args = qw/ arg1 arg2 1 2 3 /;
$exit_code = system ( $cmd, @cmd_args );
print 'exit code from ', $cmd, ' is ', $exit_code, $/;
```

محتوى الملف caller.cmd

```
@echo off
cls
echo %1%
echo %2%
echo %3%
echo %4%
```

ينقسم المثال إلى جزأين: الجزء الأول بمثابة سكريبت سنقوم باستدعائه في شفرة بيرل. هذا السكريبت يقوم بطباعة المتغيرات الممررة له على الشاشة. يتمثل الجزء الثاني و الأهم في سكريبت بيرل. يبدأ هذا الأخير باستدعاء للدالة system لتغير لون الكتابة و خلفية الواجهة أما الاستدعاء الثاني للدالة فيتمثل في تمرير اسم السكريبت و مصفوفة من المتغيرات. تخزن نتيجة هذا الاستدعاء في المتغير exit_code الذي ستتم طباعة نصه لاحقا.

الخاتمة

نختم هذا الفصل بالتذكير أن لغة البيرل ثرية جدا و لا تكفي الصفحات و الصفحات لكشف خباياها و مميزاتها. لذلك فإن هذا الكتاب بمثابة مقدمة لتعلم البيرل أو بالمعنى الأصح هذا الكتاب يحتوي كل ما قد يحتاجه المطور لتعلم البيرل, لسبب واحد هو أن من لا يكتفي بنصف الإجابة لن يستطيع أن يكون مبرمج.

[I] http://www.linuxjournal.com/article/3394
[II] http://www.cio.com/article/175450/You_Used_Perl_to_Write_WHAT_
Teach Yourself Perl in 21 Days, Second Edition (2002) [page 09]; by Laura Lemay
[III] Automating Windows with Perl (1999) [page 05]; by Scott McMahan