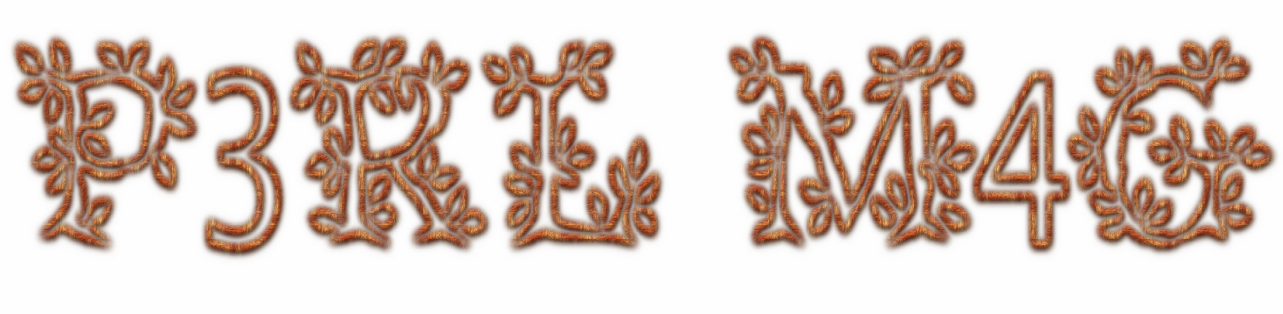
بسم الله الرحمن الرحيم



Perl Data Structure

الوصف:-
ان من اكثر الخواص التي كانت مزعجة في لفة البيرل قبل ان يتم اطلاق الأصدار الخامس من لغة البيرل هي كانت التعامل مع نوع البيانات المعقدة او المعروفة باسم
Complex Data Structures
لانها كانت غير معدومة من الأصدرات القديمة لكن على الرغم من انها لم تكن مدعومة من اللغة فقط المبرمجون الذين لهم باع طويل مع البرمجة كانت لهم القدرة على ان تستخلص البيانات المعقدة ولكن مع تقدم الزمن ومع مرور الزمن تطورت لغة البيرل ونزل الى الأسواق
الأصدار الخامس و الأصدارات المنبثقة منه اصبح بالأمكان استعمال الداتا المعقدة كالمصفوفات

*C0D3

```
for $x (1 .. 10) {
        for $y (1 .. 10) {
            for $z (1 .. 10) {
                $AoA[$x][$y][$z] =
                    $x ** $y + $z;
            }
```

الان اصبح من الممكن ان تتم كتاابة هذه سكربتات وعلى الرغم من هذا السكربت يبدو سهل و بسيط لكن هنالكفي الواقع عدة اسئلة بلطجية تفرض نفسها وهي

كيف نقوم بعملية الطباعة

لماذا لأيمكن ان نقول

*C0D3

```
print $AoA
```

كيف نقوم بعملية الطباعة
كيف نقوم بعمليات تمرير الى الدوال كيف نقوم باسترجاع الدوال او واحدة من هذه الدوال التي قد قمنا بعملية تمريرها
هل يمكن هل يمكن هل يمكن الخ من الأسئلة التي لآتقبل التي تنتهي
الأن يجب ان ترى انه من الممكن ان تكون مرتبك في هكذا نوع من الأسئلة لذا ان يجب ان لأيقوم اي شخص مهما كان متمرس ان يقلل من اهمية اي كود ولو كان سهل بنظره
الأن احنا راح ندخل في المطلوب من هذا الدرس الأوهو ال
ونحن الأن بصدد التركيز على هذه النقاط وسوف نتناول شرحها بعد مدة ان شاء الله

```
(Construtors)
which they are
*arrays of arrays (AoA)
*hashes of Arrays (HoA)
*arrays of hashes (AoH)
*hashes of arrays (HoH)
```

اهم شئ لكي تفهم ال
Data Structures
في لغة البيرل هو عدة اشياء منها هو انه حتى المصوفوفة ذات البعد الثنائي او ما يطلق عليها ب (Two Dimensions Array)
وحتى الهاشات فإن هذه المتغيرات جميعها تبدو من الداخل مجرد مصفوفة ذات بعد واحد وهو بامكانهم ان يحملوا قيمة واحد من نوع

(Scalar)
ولكن هذه القيمة من المعروف ان من الممكن ان تشمل على كل مما يلي

1- number
2- string
3- Reference

Reference

ولكن هذه القيمة التي من نوع سكلر من غير الممكن لها ان تحمل كل من
1- Arrays
2- Hashes

بشكل مختصر يمكن ان ان نقول ان ال
(Reference)
تشبة ال
(Pointers)
وهنا يسأل احد ما هو وجه الشبة بين هذين النوعين والأجابة على هكذا سوال
(reference) انه
(Pointer) مثل
تشير الى شئ وهي تعرف الى ماذا تشير
هذا يعني انه انت عندما تملك شئ وهو يبدو لك على انه مدخل الى مصفوفة الى مصفوفة ثنائية الأبعاد او المصفوفة ذات البعد الثنائي كام يسميها البعض او الى اكثر و/او هاش وفي الحقيقة ان الذي يجري في مثل هذه الحال هو انه ال

(Base Type)
مجرد مصفوفة احادية ذات بعد واحد تحتوي على الرفرنس للخطوة التالية

*C0D3

```
$array[7][12]                    # array of arrays
$array[7]{string}                # array of hashes
$hash{string}[7]                 # hash of arrays
$hash{string}{'another string'}  # hash of hashes
```

الأن نعود الى ما قلنا الى اننا سوف نشرحه فيما بعدوهو ال

*arrays of arrays (AoA)
*hashes of Arrays (HoA)
*arrays of hashes (AoH)
*hashes of arrays (HoH)

والأن سوف نعلم ماهي هذه التراكيب وفي ماذا تستخدم وماهي الفائدة المرجوة منها
ولكن قبل ان نمر على هذا يجب ان نعلم بعض الأشياء عن موضوع يدعى ب (الأخطاء الشائعة)
ونحن الأن سنبدء به بسم الله

Common mistakes

ان اكثر الأخطاء التي يقع بها المبرمجون في لغة البيرل في ال Constructing
اي البناء و التشييد ويسأل احد ماهو هذا ال Constructing
هي التركيب التي ذكرناها في الأعلى
بالتحديد هي هي في حساب اعداد العانصر واخذ الرفرنس الى

نفس المكان مرة بعد مرة بعد مرة وهكذا
والأن سوف ناخذ مثال على الخطأ الذي يحدث في حساب
العناصر

*C0D3

```
for $i (1..10) {
        @array = somefunc($i);
        $AoA[$i] = @array;      وهذا خطأ لايجوز
    }
```

اما ما هو هذا الكود وماذا يعني هو عبارة عن حالة بسيطة من وضع مصفوفة
Array
الى هاش
Hash
و القيام بحساب العناصر فأذا كان هذا هو حقيقة ما تريد فأنة من الممكن ان تتم
اعادة صياغة الكود الأتي كما يلي

*C0D3

```
for $i (1..10) {
        @array = somefunc($i);
        $counts[$i] = scalar @array;
    }
```

وهذا هو الحل الصحيح والأن نعود الى ثاني شئ قمنا بذكره وهو انه اننا نقوم
باخذ الرفرنس الى نفس المكان مرة بعد مرة
يعني مرارا وتكرارا الى نفس المكان في الذاكرة

*C0D3

```
for $i (1..10) {
            @array = somefunc($i);
            $AoA[$i] = \@array;     وهذا ايضا خطأ لايجوز
        }
```

والذي ينظر الى هذا الكود يسأل ما هو الخطاء الموجود فيه الأ يبدو هذا المقطع البرمجي صحيحا؟؟؟

انت في مثل هذه الحالات عند اخذ المصدر الى موقع في الذاكرة و انت لأتريد ان تقع في هذا الخطأ اي ان تأخذ الرفرنس الى نفس المكان في الذاكرة مرة بعد مرة فأنت في مثل هذه الحالة تحتاج الى array of references

وانت في هذا المثال قد فعلت واحدة وليس لجميع العناصر

ولتـلأفي هذا الخلل يجب ان يكون الكود هكذا

*C0D3

```
for $i (1..10) {
            @array = somefunc($i);
            $AoA[$i] = [ @array ];
        }
```

الأقواس المربعة تصع ال الرفرنس لمصفوفة جديدة مع نسخة لما يوجد في ال @array

وهذا هو الذي يجب ان يتم

وهذا الحل الذي قمنا يتقديمه يمكن ان يتم بطريقة اخرى لكن

انا ما احبها لأنها تكون اصعب القرأة و انا اظنها مزعجة نوعا ما لأنه اغلب المبرمجين وحتى ان كانو اهل باع طويل في البرمجة فأنهم على قدر من يقدرون فأنهم يحاولون ان يتركوا الأوقاس وهذا العـلأمات البرمجية التي من الممكن ان تجعل برنامجهم اصعب الى القرأة و غير قابل الى افهم على كل هذا الحل

الثاني

*C0D3

```
for $i (1..10) {
        @array = 0 .. $i;
        @{$AoA[$i]} = @array;
    }
```

الان نأتي على الأسئلة المرطوحة على هذان الكودان هل هما نفس الشئ الأجابة على هذا السؤال هي من المكن نعم ومن الممكن لأ جرب بنفسك وشوف اذا كانا يعطون نفس
القيمة او لأ او اذا كان هنالك فرق وهذا الفرق يمكن اعتباره فرق خبيث
وهو انه انت عندما تقوم بوضع شئ بين اقواس مربعة انت على يقين المعرفة انه دائما رفرنس جديد واذا تحب تتأكد اقرأ الكود الاول المذكور قبل هذا الكود او brand new reference
مع اخذ نسخة من البيانات
شي اخر من الممكن ان يحدث مع هذه الحالة الجديدة

*C0D3

```
@{$AoA[$i]}}
```

الموجدة على الجهة اليسرى من الرفرنس النقيضة للرفرنس الموجودة على الجهة اليسرى من الرفرنس
ان كل هذا يعتمد على

*C0D3

```
$AoA[$i]
```

اذا تم لم تعريفها لكي تبدأ معها او انها حاليا تحتوي على رفرنس
اذا انت كنت مسبقا معرف ال @AoA

مع رفرنس كما يلي

*C0D3

```
$AoA[3] = \@another_array;
```

ثم بعد ذلك المهمة مع انعدام الأتجاه على جانب الجهة
اليسرى سوف يستخدم الرفرنس الموجود لكي تصبح

*C0D3

```
@{$AoA[3]} = @array;
```

Array بالطبع ان هذا سوف يكون له الجانب اكثر تشويقا في التخلص من
@another_array
عندي تعقيب جميل على كلمة مشوق لأنه مرة قريت انه هذه الكلمة عندما يقولها
مبرمج مافانه في هذا يقصد انه هنالك مكيدة او خديعة او غيرها لذا يجب الحذر
دائما تذكر ان تستخدم
hash constructors او array
مع ال
"[]"
او
"{}"
وسوف تكون بخير ان شاء الله
وبشكل مختصر يمكن القول

C0D3*

1.

```
$AoA[$i] = [ @array ];    دائما الأفضل والأسهل لأستعمال
```

*C0D3

2.

```
$AoA[$i] = \@array;                                    طريقة بها الكثير من
الخداع ولأأشجع احد باستعمالها
```

*C0D3

3.

@{ $AoA[$i] } = @array;	اما هذه الطريقة فيعني المكتوب مبين من عنوانه الباب الي يجي منه ريح سده و استريح طريقة صعبة

CAVEAT ON PRECEDENCE

بالكـلأم عن اشياء مثل هذه

@{$AoA[$i]}

اذا صادفك هكذا شئ من الممكن ان يكون صعب لدى بعض المبرمجين لذا لابأس اذا كان هناك بديل لهكذا تركيب يبدو في نظر البعض انه صعب لذا لأبأس بالأستعاضة بتراكيب مثل هذه وتؤدي نفس الغرض

مثل

*C0D3

1.

$aref->[2][2]	هكذا تركيب سهل و بسيط وقابل لفهم اذا سيكون هو الحل بدل من التراكيب الصعبة

*C0D3

2.

$$aref[2][2]	مربك قليـلآ اليس كذلك انا ايضا اظن هكذا لذا سيكون الأول هو المعتمد

اظن انه الأن الفكرة المرجوة من الموضوع اصبحت معروفة

Why we should always " use strict"

طبعا لغة البيرل لديها اكيد بعض الخصائص التي تساعد المبرمج على ان لأ يقع في بعض الشراك ان الطريقة الأفضل لكي تبدأ برنامجك هي

*C0D3

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

اما عن الذين يبرمجون من الصدفة بشكل مباشر فانه يتم استعمال نفس الطريقة بالشكل التالي

*C0D3

```
Perl -t
```

انت اذا استخدمت هذه الطريقة فانك في هذه الحالة انت تجبر كافة المتغيرات التي لديك على ان يتم سبقها بكلمة التملك وهي

my

يعني اذا اخذنا هذا الكود

*C0D3

```
#!/usr/bin/perl  -t
use strict ;
my $aref =[
                ["Ahmed" , "Ali", "Muna"]
                ["Mahmoud" , "Jax","Rosy"]
                ["Mostafa","Jasim","Jack"]
];
print $aref[2][1];
```

الأن اذا جئنا الى عميلة التنفيذ سوف نـلأحظ ان المترجم مباشرة سوف يشير الى خطأ في
at compile time,
يعني انا هنا قمت بجعل خطأ بشكل متعمد اذا اردنا ان نقوم بتفيذ الكود بشكل صحيح فقط علينا ان نعدل التالي

*C0D3

```
#!/usr/bin/perl -t
use strict ;
my $aref =[
                ["Ahmed" , "Ali", "Muna"]
                ["Mahmoud" , "Jax","Rosy"]
                ["Mostafa","Jasim","Jack"]
];
print $aref->[2][2];
```

اذا جئنا الى تنفيذ هذا الكود نـلأحظ انه الترجمة تمت بشكل صحيح و ان ناتج التنفيذ هو

*C0D3

```
Jack
```

والان نـلأحظ انه ان الشئ الوحيد الذي تمت اضافته الى الى الكود لكي يتم تنفيذه بشكل صحيح هو الخظة الاتية فقط حيث كانت

*C0D3

```
Print $aref[2][2];
```

وبعد التصليح قمنا باضافة العملية الأتية فقط

*C0D3

```
Print $aref->[2][2];
```

وشفنا النتجية بعد التصليح ما هي
ويمكن ان نغير جملة الطباعة الى شكل اخر تِدي نفس المهمة اي انها تطبع الناتج

*C0D3

```
Print $$aref[2][2];
```

ومع جملة الطباعة هذه سوف نحصل على نفس النتيجة
الأن هناك من لم يقنع انه اذا تم استعمال عبارة ال

*C0D3

```
Use strict;
```

يعني انا لما قريت البرنامج في بداية برمجتي للبيرل انا لم اكن اتوقع انه هذه الفقرة البرمجية الصغيرة من الممكن ان تغير في البرنامج وتجعله يكون خطأ اذا لم يتم تعريف المتغيرات الى صيغة التملك
my وهي
يعني اذا نفذنا الكود من دون هذه الفقرة البرمجية الناتج سيكون

*C0D3

```
#!/usr/bin/perl -t
use strict ;
my $aref =[
                ["Ahmed" , "Ali", "Muna"]
                ["Mahmoud" , "Jax","Rosy"]
                ["Mostafa","Jasim","Jack"]
];
print $aref->[2][2];
```

اذا نفذت هذا الكود سوف تحصل على كل هذه الأخطاء لذا احذر وكن متيقظ

*C0D3

```
Global symbol "$aref" requires explicit package name at - line 2.
Global symbol "$aref" requires explicit package name at - line 7.
Execution of - aborted due to compilation errors.
```

وانت لو كنت تنفذ برنامج كبير ستكون في غني عن هكذا اخطاء
الان سوف نرجع الموضوع الأساسي من الدرس هذا الي اكيد اغلب الي بيقروا الكتاب نسيوه هي ال

ARRAYS OF ARRAYS:-

تعريف ال
ARRAYS OF ARRAYS
كما يلي

*C0D3

```
@AoA=(
        [“ahmed”, “Ali”,”Muna”]
        [“Jack”, “Geek”, “Ali]
);
```

هكذا يتم تعريف ال
ARRAYS OF ARRAYS
طباعة عنصر واحد تتم كما يلي

*C0D3

```
Print $AoA[0][0];
```

استدعاء دالة من خـلأل ARRAYS OF ARRAYS
تتم كما يلي
الصيغة العامة

*C0D3

```
for $i ( 1 .. 10 ) {
        $AoA[$i] = [ somefunc($i) ];
    }
```

اضافة عناصر الى سطر موجود في المصوفة تتم كما يلي

*C0D3

```
push @{ $AoA[0] }, "Sami”,”Max;
```

واذا اردنا طباعتهم نطبعهم كما ذكرنا اعلاه
لطباعة
ARRAYS OF ARRAYS
بشكل كامل بالوضع المفهرس نستعمل هذا الكود

*C0D3

```
for $i ( 0 .. $#AoA ) {
        print "\t [ @{$AoA[$i]} ],\n";
    }
```

شرح على هذا الكود نـلأحظ ان الخطوة الأولى من الكود تتم عن طريق استعمال جملة الفور

*C0D3

```
for $i ( 0 .. $#AoA )
```

$#AoA نلاحظ بين الأقواس لجملة الفور وجود ال
اول شي الصفر هو اول عنصر من اول سطر لانه كما نعرف انه المصفوفات تعنون العناصر و الأسطر فيها من الصفر
ثم انه
بشكل عام وال $#AoA
$#
بشكل خاص هي عبارة عن متغير مميز وخاص موجود في المصوفوفات يعني الى اخر عنصر موجود في المصوفة موجود في اخر سطر

Declaration of a HASH OF ARRAYS:-

تعريف ال
HASH OF ARRAYS
بالشكل الأتي

*C0D3

```
%HoA = (
fruit                => ["apple" , "orange"],
animals              => ["cats" , "dogs"],
names                => ["Omar","Mahmoud"],
);
```

تعريفها يتم بهذا الشكل

طباعة عنصر من ال

HASH OF ARRAYS

يتم كما يلي

*C0D3

```
Print  $HoA{fruit}[0];
```

وفي هذه الحالة سوف يطبع تفاح

اضافة عنصر الى قسم موجود في الكود الذي كتبناه في الجزء الأعلى

يتم ذلك كما يلي

*C0D3

```
push @{ $HoA{"fruit”} }, “berry”,”cherry”
```

واذا اردنا ان نطبع جزء من القسم الجديد نتبع الطريقة القديمة المتبعة بدون اي تغيير وستـلأحظ النتائج بعونه تعالى

طباعة ال

HASH OF ARRAYS

بالشكل المفهرس نقوم بكتابة الكود الأتي

*C0D3

```
foreach $perl ( keys %HoA ) {
                    foreach $i ( 0 .. $#{ $HoA{$perl} } ) {
               print " $i = $HoA{$perl}[$i]";
          }
          print "\n";
     }
```

الأن نلاحظ انه قام قام بالطباعة بالشكل المفهرس
ولكن كما لاحظنا انه يأخذ الترتيب من الأسفل الى الأعلى

ARRAYS OF HASHES:-

التعريف العام لها يكون بالشكل التالي

*C0D3

```
@AoH = (
{
apple => "red",
banana => "yellow",
},
{
dog =>"brado",
cat => "kitty",
},
{
windows => "lame",
linux =>"free",
}
);
```

هكذا يكون التعريف العام لها

اما عن كيفية وطريقة قرأة عنصر منها تتم كما في الشكل التالي

*C0D3

```
print $AoH[0]{apple};
```

واذا قمنا بتنفيذ الكود هذا سوف نلاحظ انه سوف يقوم بطباعة ما يقابل ال

apple

وهو لون التفاح و الذي قمنا بتخصيصه على انه احمر

اما عن طباعتها بالشكل المفهرس فهي تتم كما يلي

*C0D3

```
for $i ( 0 .. $#AoH ) {
        print "$i is { ";
        for $role ( keys %{ $AoH[$i] } ) {
            print "$role=$AoH[$i]{$role} ";
        }
        print "}\n";
    }
```

حيث اذا ما قمنا بتنفيذ هذا الكود سيكون ناتج هذا التنفيذ هو

*C0D3

```
0 is { banana=yellow apple=red }
1 is { cat=kitty dog=brado }
2 is { windows=lame linux=free }
```

الان نقوم بالأنتقال الى الجزء الأخير من هذه التركيب المعقدة وهي ال

HASHES OF HASHE:-

التعريف العام لها يكون بالشكل التالي

*C0D3

```
%HoH = (
 fruit => {
 apple => "red",
 banana => "yellow",
},
animals => {
dog =>"brado",
cat => "kitty",
},
system =>{
linux => "free",
windows => "retard" ,
},
);
```

هكذا يكون شكل التعريف العام لها

اما عن طريقة طباعة عنصر موجود فيها ذلك يتم من خلال الكود الاتي

*C0D3

```
print $HoH{system}{linux};
```

اذا قمنا بتفيذ هذا الكود سنـلأحظ انه سيقوم بطباعة الموافق لكملة الينكس وهي الحرية اي فري

الأن انتهي هذا الدرس ان شاء الله تم فهم هذا الدرس بشكل جيد وكان سهل ووفقت في تقديم المعلومة الى الجميع

Regular Expressions

التعابير المتظمة او التعابير القياسية هكذا تسمى في لغتنا العربية كثر الحديث عن هذا الموضوع وعن كونه موضوع صعب و معقد وغير قابل للفهم ما ادري ليش و لأاعرف شنو السبب بس لكن ان شاء الله سنحاول الأن ان ندرس او نتعرف على ماهي هيأة هذا الموضوع وفي مذا نستخدم وماهي الفائدة من هكذا موضوع الخ ..الخ ..الخ

التعابير القياسية او المنتظمة هكذا اطلقنا عليها هذه التعابير تسمى اختصارا ب (regexp)

String هذا التعابير المنتظمة ببساطة هي عبارة عن سلسلة نصية او اويمكن القول انها عبارة عن سلسلة نصية من الرموز او هكذا يطلق عليها في الأنكليزية

String of characters

التعابير المنتظمة هي عبارة عن كلمة تطابق اي سلسلة نصية تحتوي على تلك الكلمة

الأن سوف نأخذ ابسط مثال لمعرفة مفهوم التعابير القياسية

*C0D3

```
“hello world” =~ /hello/;          تطابق
```

الأن نأتي على شرح هذا الكود البسيط جدا كما يلي

"hello” كلمة

هي التعبير المنتظم

الأقواس المغلقة هي وظيفتها تخبر البيرل ان تبحث عن تطابق

يعني في حالتنا هذه الأقواس المغلقة تخبر البيرل ان تبحث عن

*C0D3

```
/hello/
```

اما عن العملية

*C0D3

```
=~
```

فهي شريكة للسلسة النصية في عملية التطابق وتنتج الأجابات الصحية اذا كان التطابق موجود وتنتج الحالأت الخاطئة اذا كان التطابق غير موجود
طبعا هناك طريقتين للتطابق الطريقة الاولى هي المستعملة دائما لكني ما احبها لأنها طريقة تقليدية اما الطريقة الأخرى هي الطريقة الغير الدارجة و الغير المستعلمة من الكثر و التي انا احبها
اليكم الطريقتان وانتم احكموا اي من هذه الطريقتان هي الأنفع لك و التي تجعل البرمجة لديك افضل
الطريقة الأولى

*C0D3

```
if ("hello world " =~ /world/) {
print "there is match";
}
print "\n";
```

شرح الطريقة الأولى اولا استعملنا جملة الشرط و دخلنا الستريك و التبير المنتظم ثم وضعناهم ضمن كتلة برمجية داخل جملة الشرط ثم وضعنا جملة الطباعة التي تخبرنا بالنتجية حيث اذا كان هناك تطابق فانه سوف تقول لنا هناك تطابق و الأ فانه جملة الطباعة سوف لن تخرج شي
ثم اغلقنا الكتلة البرمجية ثم قمان بطباعة الكود المسئول عن بداية سطر جديد عن التنفيذ الطريقة الاولى انتهت
الطريقة الثانية

*C0D3

```
$a="hello world";
$b="hello ";
if ($a=~/$b/){
print "there is a match";
}
print "\n";
```

الأن نأتي على شرح الطريقة الثانية
هذه الطريقة المحببة الي لأ ادري ليش بس اظن انها اسهل و احسن من الطريقة الأولى
على كل انه احنا الستريك و التعبير المنتظم وضعنا كل منهم الى متغير ثم قمنا بنفس العمليات التي قمنا بها في الطريقة الأولى يعني الطريقة ما تحتاج الى اي شرح
طيب الأن لنفرض انه الكلمات التي قمنا بادخالها هي كلمات غير متطابقة ماذا نفعل لكي نعلم ان هذه الكلمات غير متطابقة
راح نعمل كما يلي في هذا الكود

*C0D3

```
$a="hello world";
$b="hello";
if ($a=~/$b/){
print "there is a match";
}
else {
print "there is a mistake";
}
print "\n";
```

يعني ملخص عن هذا البرنامج انه اذا كان يوجد تطابق اطبع العبارة التي تدل على وجود التطابق اما اذا لك يكن هنالك اي تطابق فانه في هذه الحالة قمنا باستعامل ال

*C0D3

```
else
```

التي نعمل الى استعمالها كخيار بديل لكي نعلم انه لأيوجد تطابق و الأن سوف نرى كيف

*C0D3

```
$a="hello world";
$b="Hello";
if ($a=~/$b/){
print "there is a match";
}
else {
print "no match";
}
print "\n";
```

الان استعملنا هذه الطريقة ولأحظنا انه عند عدم وجود تطابق كيف تعمل الى else
بالمناسبة انه هنا في هذه الحالة يسأل احد لماذا لم يتم التطابق لانه في هذه الحالة لأحظنا انه احرف الاول من كلمة هلو هو حرف كبير و لغة البيرل حساسة من ناحية الحروف اس انه الحروف الكبيرة و الصغيرة تشكل فرق لذا يجب الأنتباه و الحذر
الأن بعد ان عرفنا ما هي المبادءئ الأساسية في التعابير المنظمة الأن سوف نأخذ عملية او معامل من شأنه
ان يعمل على عكس المعامل

*C0D3

```
=~
```

وهو المعامل

*C0D3

```
!~
```

الذي يعمل عكس المعامل الموجود في الأعلى منه
يعني سوف نرى ما هو فعله البرمجي من خـلأل هذا الكود

*C0D3

```
$a="hello world";
$b="Hello";
if ($a!~/$b/) {
print "there is a match";
}
else {
print "no match";
}
print "\n";
```

حيث اذا قمنا بتنفيذ هذا الكود سوف نـلأحظ انه سوف يقوم
بطباعة الجملة التي تدل على وجود تطابق في البرنامج الذي قمنا بكتابته
اذن علمنا انه فكرة التعابير القياسية تدور حول التطابق الأن سوف ناخذ عدد من
الأمثلة لكي نعرف ماهي حالة التطابق فيها و اذا كان موجدود او لأ
الحالة الأولى

*C0D3

```
"Hello World" =~ /world/
```

هنا نلاحظ عدم وجود تطابق نظرا لحالة احرف لانه في هانك حرف كبير و سغير وكما ذكرنا ان لغة البيرل حساسة من ناحية الحروف

الحالة الثانية

*C0D3

```
 "Hello World" =~ /o W/
```

هنا نلاحظ وجود تطابق الفراغ هنا يعامل على انه حرف ورمز عادي حاله حال الحروف العادية

الحالة الثالثة

*C0D3

```
$a="hello world";
$b="world ";
if ($a=~/$b/){
print "match";
}
```

هنا نـلأحظ عدم جود تطابق في هذه الحالة السبب في عدم وجود التطابق في هذه الحالة هو يمكن في ما يلي

*C0D3

```
$a="hello world";
```

هنا تلاحظ عدم وجود فراغ في نهاية الكلمة حيث نـلأحظ وجود مباشرة عـلأمة الأقتباس بعد انتهاء الكلمة

اما في المتغير بي

نـلأحظ

*C0D3

```
$b="world ";
```

هنا نلاحظ وجود فراغ بعد النهاية يعني يوجد فراغ في نهاية الكلمة لي بمعني ادق انه يوجد بعد الحرف الاخير فراغ

لذا هنا نـلأحظ انه لأيوجد تطابق

في التعابير القياسية يمكن ايضا مقارنة الأحرف وتكون النتيجة تدل على وجود تطابق اذا كان الشئ الذي نبجث عنه موجود كما في الكود الأتي

*C0D3

```
$a="hello world";
$b="o";
if ($a=~/$b/){
print "there is a match";
}
```

اذا طبقنا هذا الكود نـلأحظ سوف نجصل على الجملة التي تدل على وجود

تطابق هنا طبقنا الحالة التي نبحث على حرف واحد موجود في كلمة الأن

سوف نبحث على حرف موجود في كلمة وحرفان موجودان في الكلمة الأخرى
لكي نرى ماذا سوف نحصل من عملية البرمجة هذه هل سنجد تطابق امألأ

*C0D3

```
$a="hello world";
$b="o wo";
if($a=~/$b/){
print "there is a match";
}
```

اذن نـلأحظ عند تطبيق هذا الكود نلاحظ الحصول على الجملة التي تؤكد لنا انه
يوجد تطابق في البحث و الشئ الذي نريد ان نبحث عنه
في التعابير القياسية اذا كنا نبحث عن حرف موجود في جملة
وكان الحرف موجود في الكلمتان الموجودتان في الجلمة وحصل وجود تطابق
فان التطابق يتم اعتمادا على اول حرف موجود في الجملة يعني كما يلي في هذا
الكود

*C0D3

```
$a="hello world";
$b="o";
if($a=~/$b/){
print "there is a match";
}
```

الان نلاحظ انه الجملة التي تدل على التطابق سوف تطبع اما عن ميكانيكية
التطابق هي كما يلي
ان حرف ال

o

موجود في كلمة من كلمة هلو وكلمة العالم في هذه الحالة ان ميكانيكية التطابق
تم من خلال اعتماد اول حالة تطابق يتم العثور عليها وهنا اول حالة وجود
للحرف هي كلمة هلو حتى و ان كان هذا الحرف غير موجود في كلمة العالم
فأن جلمة التطابق سوف تطبع لأنه موجود في الكلة الأولى
الأن سوف نطبق على الجملة يعني كما يلي في هذا الكود

*C0D3

```
$a="the rat in the hat";
$b="at";
if ($a=~/$b/){
print "there is a match";
}
```

الأن عندما ننفذ هذا الكود ايضا سوف نـلأحظ طباعة الجملة التي تدل على وجود التطابق
وعن ميكانيكية البحث نفس الطريقة ان هذان الحرفان موجودان في الكلمة الثانية في **رات** لذا تم اعتماد التطابق وحتى وان كلمة **الهات** غير موجودة فأن حالة التطابق قد اعتمدت

ليس كل الرموز التي تستخدم في عمليات التعابير القياسية حيث هنالك بعض الرموز التي يطلق عليها اسم ال
Meta Characters
وهم

*C0D3

1.

```
{}
```

*C0D3

2.

```
[]
```

*C0D3

3.

```
()
```

*C0D3

4.

```
^
```

*C0D3
5.

```
$
```

*C0D3
6.

```
.
```

*C0D3
7.

```
|
```

*C0D3
8.

```
*
```

*C0D3
9.

```
+
```

*C0D3
10.

```
?
```

*C0D3
11.

```
\
```

هم هولأء الأحدى عشر رمز

اذا كانت هذه الرموز ضمن صيغها العادية فإن جملة الطباعة لن تنفذ اما عن
عن جملة صيغتها العادية ساشرحها لاحقا
الأن سوف نـلأحظ هذا الكود ونرى ماذا سيتم

*C0D3

```
$a="2+2=4";
$b="2+2";
if ($a=~/$b/){
print "there is match";
}
else {
print  "no match";
}
print "\n";
```

الأن اذا تم تنفيذ هذا الكود فأن ناتج التنفيذ لهذا الكود هو العبارة التي تدل على عدم جود تطابق زين السبب لعدم وجود التطابق هو انه علمية الزائد او الجمع هي عبارة عن

Meta character

الأن ارجع واقول انه قسم من التعابير القياسية تم فهم عملها وكما علمنا الفكرة الأساسية منها هي البحث عن شئ حيثما وجد تطابق في تلك اللحظة نكون قد بلغنا مردانا لكن لنفرض انه لدينا جملة مكونة من كلمتين واحنا كنا نريد ان نبحث عن جزء محدد منها اي اذا كنا نريد ان نختبر هذه الجملة ليش بشكل عشوائي ولكن عن عن طريق شكل محدد

اذن في هذه الحالة اذا كنت تريد ان تعمل شئ مثل هذا فأنت سوف تلقى مرادك في التعابيرالقياسية هذه تتم من خلال ما يلي

وعن طريق رمزين هما

*C0D3

1.

```
^
```

*C0D3

2.

```
$
```

زين الأن ما هي هذه العلامات ؟؟

ما هي فائدتها ؟؟

في ماذا تستخدم هذه ؟؟
الأن سوف نتعرف على الرمز الأول اولاً هذه الرموز غالبا يطلق عليها برمجيا ب
(Anchor)
التي تعني بلغتنا العربية المرساة او المعتمد
على كل المعتمد الأول يستخدم اذا كنت تبحث عن شئ محدد وهذا الشئ المحدد الذي كنت تبحث عنه في بداية الجملة يعني كما يلي في هذا الكود

*C0D3

```
$a="hello world";
$b="^hello";
if ($a=~/$b/){
print "there is a match";
}
```

الأن اذا نفذنا هذا الكود نلاحظ انه اننا سنحصل على الكلمة التي تدل على وجود التطابق لأننا كنا نريد ان نبحث عن كلمة في البداية و استخدمنا المعتمد الخاص لهذا الغرض
المعتمد الثاني
اذا كنت تريد ان تبحث عن شئ موجود في النهاية الجملة التي تريد ان تطبق البحث فيها فأن هذا المعتمد يولد لك ما تريد من البحث الخاص في النهاية كما في هذا الكود

*C0D3

```
if ("hello world" =~ /world$/){
print "there is a match";
}
```

اولاً هنا استخمدت الطريقة الأولى لانه الطريقة الثانية راح تطلع معك اخطاء لذا تجنبها على كل احنا الأن بحثنا في اخير الجملة
عن الكلمة التي نريدها
وحصلنا على مرادنا

الأن سنأتي الى موضوع جديد هو

Using character classes

في هذا الموضوع يمكن القول انه لديك مجموعة من الأختيارات
او الأحتمالأت بدلاً من واحد هذه اي
character classes
تكون معنونة ب
[ ]
حيث هذه تكون العنوان الدال لها
الأن سوف ناخذ مثال عادي على ما اخذنا في الشرح الماضي لكي نرى الفرق

*C0D3

```
$a="cat";
$b="cat";
if ($a=~/$b/){
print "there is a match ";
}
```

طبعا هنا وضعنا حالة عادية من التعابير القاسية وانا كنت متعمد في وضعها
لكي نرى الفرق بينها وبين الشئ الذي نحن الأن بصدد دراسته

*C0D3

```
$a="cat";
$b="[bcr]at";
if ($a=~/$b/){
print "there is a match”;
}
```

زين الأن اذا نفذنا هذا الكود راح نحصل على التطابق ونعلم ان الكود صحيح
والأحرف الموجدة خارج اقواس ال
هذه عبارة عن احرف احنا لما كتبنا الكود وضعنا كلمة

character classes
rat او كلمة bat
فأن النتيجة التي سنحصل عليها هي العبارة الرائعة التي تخبرنا بانه كودنا صحيح وانه برنامجنا هو صحيح
الان سنأتي الى نقطة هي صغيرة جدا لكن هي جدا جدا مهمة احنا اخذنا في التعابير القاسية انه اذا كانت الكلمات مختلفة من ناحية التسيق اي انه قسم منه كبير و الأخر صغير وقمنا بعملية المطابقة فإن الناتج هو خطأ لأنه لغة البيرل هي لغة حساسة من ناحية حالة الحروف فيما اذا كانت كبيرة و صغيرة
الأن نحن بصدد تعلم طريقة ستعمل على الغاء هذه العقبة من امامنا وهي تكن كما يلي في هذا الكود

*C0D3

```
$a="hello world";
$b="HELLO";
if ($a=~/$b/i){
print "there is a match ";
}
print "\n";
```

الأن اذا قمنا بطابعة هذا الكود سوف نحصل على الجملة التي تخبرنا بان التطابق موجود لكن ما الذي تغير على كل هو المعامل
i
الذي قمنا لوضعه قبل نهاية سطر جملة الشرط هذا المعامل يخبر لغة البيرل انه لأداعي لاعتبار حالة الأحرف في عملية التطابق لذا فأننا نـلأحظ انه راح نحصل على التطابق
وهذا المعامل حرفيا يعني
match case-insensitive.
ايضا ال
character classes
لها رموز عادية ولها رموز خاصة او يطلق عليها رموز مميزة

هذه الرموز المميزة الموجودة في ال
character classes
هي

*C0D3
1.

```
-
```

*C0D3
2.

```
]
```

*C0D3
3.

```
\
```

*C0D3
4.

```
^
```

*C0D3
5.

```
$
```

هذه هي الرموز المميرة الموجدة في ال
character classes
الان سوف ناخذ المتغير المميز الأول وهو
/
انا هنا في اخذ المتغيرات المميزة في
character classes
لم اسير على التسلسل الذي وضعته في الجداول التي في الأعلى لذا ارجو الأنتباه

*C0D3

```
$a="cdef";
$b="[\]c]def";
if ($a=~/$b/){
print "there is match";
}
```

الأن شوية مـلأحظات على هذا الكود الأن الكلمات التي ممكن ان تكون عملية المطابقة هي
الكملة التي قمت باستخدامها في الكود و الكلمة الأخرى هي

*C0D3

```
$a="cdef";
$b="[\]c]def";
if ($a=~/$b/){
print "match";
}
```

ومن الممكن ايضا ان نستخدم كلمة

]def

و الناتج من هذه المقارنة تطابق
الأن سو ف نتناول حالة اخرى ولكن على نفس الرمز المميز
لنفرض لديك هذا المتغير

*C0D3

```
$x="bcr";
```

سنقوم بعدد من علميات المطابقة و لكن بطرق مختلفة لكي نرى ماهو الفرق بين حالة و اخرى

*C0D3

```
$x="bcr";
$a="cat";
$b="[$x]at";
if ($a=~/$b/){
print "match";
}
print "\n";
```

الأن استعملنا خطوة متقدمة شوية عمليات تبديل من خلال احــلأل متغير و اعطاء هذا المتغير قيمة و ادخال متغير ثاني يحتوي على قيمة اخرى ثم ادخال متغير ثالث يحتوي على قيمة المتغير الثالث واعطاء هذا المتغير الثالث قيمة المتغير الأول لا عن طريق ادخال قيمة المتغير الثالث بس عن طريق ادخال المتغير ذاته في عملية المقارنة ثم اجراء العلمية وننفذ البرنامج ونحصل على النتيجة التي تخبرنا بانه العملية صحيحة وبشكل جيد قد تمت
الحالة الثانية لكن من ضمن نفس المتغير المميز ولكن من منظور اخر

*C0D3

```
$x="bcr";
$a="cat";
$b="[\$x]at";
if ($a=~/$b/){
print "match";
}
else {
print "no match";
}
print "\n";
```

الأن لو ناخذ نظرة عامة عن هذا المثال لكي نرى لماذا عندما نقوم بعملية المقارنة ومن ثم التنفيذ لهذا الكود راح نحصل على العبارة التي تخبرنا انه لا يوجد تطابق لماذا لأيوجد هذا التطابق و ماهو المختلف في هذا البرنامج عن

سابقه ؟؟؟

\ الأختـلأف الموجود عن البرنامج الموجود في السابق هو عدم وجود ال
اما في هذا البرنامج فهي موجودة في هذا المكان

*C0D3

```
$b="[\$x]at";
```

هنا نقطة الأختـلأف كما ذكرنا في الأعلى قبل عدة صفحات هو هذه العلامة
تعامل المتغير على انه متغير عادي اي انها تلغي الخواص المميزة الموجودة في
هذا المتغير ومن خـلأل هذا سيصبح مجرد حرفين عاديين فقط
الحالة الأخرى لنفس المتغير المميز لكن ايضا من منظور اخر

*C0D3

```
$x="bcr";
$a="cat";
$b="[\\$x]at";
if($a=~/$b/){
print "there is a match ";
}
print "\n";
```

الأن الحالة قد تغيرت لانه الان نحن في هذا الكود قد وضعنا السطر الأتي و هو
الذي غير البرنامج و جعل عملية المطابقة قد تمت وغيرت الموازيين
وهذا السطر هو

*C0D3

```
$b="[\\$x]at";
```

الأن ننتقل الى المتغير المميز وهو
(-)
وهو يعمل كمجرد مدى بين حدين فقط لا اكثر ولأاقل عمل هذا الرمز المميز
راح يتضح من هذا الكود

*C0D3

```
$a="item3";
$b="item[0-8]";
if ($a=~/$b/){
print "there is a match";
}
print "\n";
```

الأن ماهو الواضح من عمل هذا الرمز المميز؟؟
كما نـلأحظ الان نحن وضعنا في السطر الأول متغير قيمته
item3
وضعنا في السطر الثاني
متغير قيمته هي
item[0-9]
ولأحظنا انه اذ نفذنا البرنامج راح نحصل على عبارة التطابق
اذن المتغير الثاني هو عبارة متغير يشمل كل من عبارة

```
Item1
item2
item3
item4
item5
item6
item7
item8
```

الأن عندي ملحوظة احب ان انوه اليها هو انه لوعملت عملية مقارنة بهذا الشكل

*C0D3

```
$a="item9";
$b="item[0-8]";
if ($a=~/$b/){
print "match";
}
```

وقمت بتنفيذ البرنامج فإنك في هذه الحالة سوف لن تحصل على نتيجة المطابقة وهذا شي طبيعي لانه نحن حددنا الى حد رقم ثمانية ونحن في البرنامج دخلنا الرقم 9 وهو رقم بصور ة الحالة غير موجود لذا لن نحصل على نتيجة الحل ولن تحصل مطابقة

لكن هناك مفارقة غريبة هو انك لو كتبت كود بهذا الشكل راح تحصل على نتيجة مطابقة

*C0D3

```
$a="item8456";
$b="item[0-8]";
if ($a=~/$b/){
print "there is a match";
}
```

الأن اذا ننفذ هذا الكود راح نحصل على التطابق لأنه لغة البيرل في هذا المتغير المميز تقرأ فقط اول رقم لذا الى هذه النقطة حبيت فقط ان انوه

اذا هذا المتغير المميز – جاء في البداية او في النهاية فأنه سيعامل على انه رمز عادي وسيفقد خواصه المميزة

الان سنأخذ المتغير المميز الجديد

وهو

(^)

هذا المتغير المميز يكون في بداية ال Characters class

حيث والذي يعرف ب

(negated character class)

حيث يطابق الرموز عذا الرموز الموجودة داخل الأقواس كما يلي كما خـلأل هذا الكود الذي يشرح عمله

*C0D3

```
$a="bat";
$b="[^a]at";
if ($a=~/$b/){
print "match";
}
print "\n";
```

هذا الكود الكود اذا نفذ راح يعطي اشارة على انه التطابق موجود اما عن الية عمله فهي تتم كما يلي

الكلمات التي تطابق هنا هي ليست هذه

at or aat

a لكن الكلمات التي تطابق هنا هي جميع الأحرف الموجودة فوق الحرف

ولهذا عندما استخدمنا كلمة بات لأحظنا وجود التطابق

ويعني انت الأن تقدر ان تدخل جميع الأحرف الموجودة الى غاية الحرف

z

اما في هذه الحالة ستكون عملية التطابق ستم كما يلي

*C0D3

```
$a="spawn";
$b="[^0-9]";
if ($a=~/$b/){
print "match";
}
print "\n";
```

هنا عملية التطابق تتم حتى لو ادخلنا اي قيمة حرفية لأنه هنا التطابق لأيتم مع اي قيمة رقمية من الصفر الى التسعة كما هو موضح في الأعلى

حالة اخرى من هذا المتغير المميز لكن من منظور اخر

*C0D3

```
$a="^at";                    # the other we could use in the matching is “aat”
$b="[a^]at";
if ($a=~/$b/){
print "there is a match";
}
print "\n";
```

هنا الحالة لهذا المتغير المميز قد تغيرت لأنه هنا قد فقد
خواصه المميزة لأنه لم يكن في البداية بل جاء في النهاية

Grouping things and hierarchical matching

ان رموز التجميع في البيرل وهي تعرف بهذا الرمز
()
تسمح لجزء من التعابير القياسية لكي تعامل على اساس انها
وحدة مستقلة اجزاء من التعابير القياسية توضع بين علامات الحصر على سبيل
المثال كان لدينا التعبير القياسي الأتي

*C0D3

```
spawn(perl|programmer)
```

هنا المطابقة تتم اذا عملنا المقارنة بين

*C0D3

```
spawnperl
spawnprogrammer
```

هنا كل من هذين الكلمتين اذا عملنا المقارنة بهما فإن المقارنة سوف تتم

*C0D3

```
$a="spawnprogrammer";
$b="spawn(hacker|programmer)";
if ($a=~/$b/){
print "match";
}
print "\n
```

اما عن الكلمة الثانية

*C0D3

```
$a="spawnprogrammer";
$b="spawn(hacker|programmer)";
if ($a=~/$b/){
print "match";
}
print "\n
```

وكل من هذين البرنامجين اذا قمنا بتنفيذهم فإن الناتج هو المقارنة الصحيحة
الأن نحن بصدد اخذ رمزين مهمين في عمليات المقارنة وهما
رمزان يدخلان في عمليات المقارنة وهما

*C0D3

1.

```
s
```

*C0D3

2.

```
m
```

الرمز الأول هو اختصار لكملة الأنكليزية وهي
(single)
والتي تعني مفرد والذي يدخل في جعل علمية المقارنة حتى واذا كانت السلسلة النصية مكونة من اسطر يعاملها على انها
سطر واحد ويهمل الرموز التي من شأنها جعل السلسلة مكونة من عدة اسطر

اما الرمز الثاني فهو اختصار للكلمة الأنكليزية
(Multiple)
والتي تجعل السلسلة النصية مكونة عدة السطر اذا كانت تحتوي على الرمز
المسئول عن عملية الطباعة في سطر جديد
على عكس الرمز الموجود في الأعلى منها اي واحج يعمل على عكس الأخر
الأن ناخذ امثلة على هذه الرموز
وانا الأن سوف استخدم المثال الرسمي لهذه الحالة حتى تصل الفكرة اسرع

*C0D3

```
$x = "There once was a girl\nWho programmed in Perl\n";
```

ترجمة الكود
انه هناك فتاة برمجت في لغة البيرل مرة واحدة
الأن الأمثلة

*C0D3

```
$x =~ /^Who/;
```

الشرح على الكود الحالة هنا غير تطابق لانه الرمز الموجود بين الأوقواس
المغلقة هو رمز يستخدم من اجل المطابقة الموجودة في بداية السلسلة النصية اما
who هنا فكلمة
ليست في بداية السلسلة النصية
الحالة الثانية

*C0D3

```
$x =~ /^Who/s;
```

شرح الكود هنا ايضا الحالة تدل على انه لأيوجد تطابق لأنه الوسم المستخدم
who يعمل على المقارنة في بداية السلسلة النصية وكلمة
ليست في البداية اضف الى هذا انه نحن استخدمنا الوسم
s
الذي يعامل السلسة النصية على انها سطر واحد

*C0D3

```
$x =~ /^Who/m;
```

هنا الحالة اذا نفذت فانه تدل على التطابق لأنه استعلمنا الرمز الذي يستعمل للمقارنة في البداية و استعلمنا الرمز

m

الذي يعمل على معاملة السلسة النصية على انها اسطر متعددة

الأن سنأخذ امثلة اكثر لكن من شكل اخر على الرمز المميز الأخر

*C0D3

```
$x =~ /girl.Who/;
```

هنا اذا نفذنا هذا الكود راح نلاحظ انه عملية التطابق سوف لن تتم لأنه كما هو لأحظ بعد كلمة (كيرل) هناك علامة

(.)

بينما في المتغير الاول الذي وضعناه لم تكون هناك هذه النقطة بل كان

(“\n”)

وهذا الرمزان غير متساويان في اجراء عملية المقارنة لذا فإن المقارنة سوف تفشل

*C0D3

```
$x =~ /girl.Who/s;
```

شرح هذا الكود بما انه نحن استخدمنا الرمز الذي يعمل على معاملة السلسلة النصية على انها سطر واحد متصل فإنه اذا تمت عملية المقارنة فأن المقارنة هنا ناجحة لأنه هنا عـلأمة ال (.)

و ال

(“\n”)

متساويان

مثال اخر

*C0D3

```
$x =~ /girl.Who/m;
```

هنا اذا نفذنا البرنامج فإن عملية المطابقة سوف لن تنفذ لأنه نحن هنا استخدمنا

الحرف الذي يعمل على معاملة السلسلة النصية على انها مجموعة من الأحرف لذا فإنه في مثل هذه الحالة يعبر ال (.) النقطة غير مساوية لل ("n")

في عملية المقارنة

الان ننتقل الى موضوع جديد في عمليات

ان عملية المعرفة فيما اذا كان البديل هو مطابق ام لأ و الأنتقال الى البديل الأخر اذا كان لايوجد تطابق فان هذه العملية تدعى بعملية ال (Back Tracking)

و التي تعني الأرتداد من حيث القدوم في اللغة العربية ان هذا الأسم اي الباك تراك قد جاء من فكرة انه مطابقة تعبير قياسي هو اشبه بالمشي في الغابات اذا كانت المطابقة صحيحة وكما هو مطلوب منها فإن الغرض من المطابقة قد تم وهذا اشبه بالوصول الى الغاية في الغابة . هناك عدة رووس طرق لكي يتم التجربة بها من خـلأل المطابقة وكلها تستخدم من خـلأل التسلسل و الأتجاه من اليسار اليمين ومن كل طريق هنالك عدة مجالأت لكي يتم التجربة بها عندما تمشي في طريق و تصل الى نهاية مسدودة او كما يقال انها d34d 3nd or dead end

فانك في هذه الحالة انت لم تصل الى مبتغاك وهكذا فإن عملية المطابقة لم تتم و تاخذ في تغيير خياراتك اما اذا وصلت الى مبتغاك فإنك في هذه الحالة سوف تقف مباشرة و سوف لن تحاول او تفكر في ان تجرب طريق او احتمال اخر لانك في هذا الوقت انت في المكان الذي تريد ان تكون فيه و الأن ساعطيكم مثال عن كيفية تتم عملية المطابقة خطوة بخطوة بالتفصيل الممل المزعج

*C0D3

```
"abcde" =~ /(abd|abc)(df|d|de)/;
```

مطابقة اول حرف في السلسلة النصية وهو الحرف (a)

تجربة اول بديل موجود في المجموعة الاولى وهي المجموعة

(abd)
(a) نـلأحظ وجود تطابق في اول حرف هو
ثم وجود تطابق الحرف الذي بعده وهو الحرف بي

الأن يوجد تطابف في الحرف الأول وهو ال
(a)
ثم بعده ايضا حالة تطابق في الحرف(بي) الى الأن المسيرة
جيدة جدا
اما الأن الحرف الموجود في السلسلة النصية هو الحرف سي
وهو لأيطابق الحرف الموجود في التعبير القياسي وهو الحرف
D
اذن هنا توجد هنا توجد نهاية مسدودة اي
(DEAD END)
لذا الأن سيكون لدينا باك تراك مع رمزين هما
(ab)
abc والتقاط بديل ثاني في المجموعة الأولى وهي ال
والأن يوجد تطابق
ثم (a)
ثم (b)
(c)
اذن الأن نحن نمشي بشكل جيد
الأن ننتقل الى المجموعة الثانية و اختبار البديل الأول وهو
(df)
الأن الحرف الرابع من السلسة النصية يطابق الحرف الأول من
المجوعة الثانية في البديل الأول
اما الأن الحرف الموجود في التعبير القياسي وهو الحرف
(f)
لأيطابق الحرف الموجود في السلسلة النصية وهو الحرف
(e)
لذا هنا ايضا توجد نهاية مسدودة لذا الأن باك تراك مع رمز واحد
واختيار البديل الثاني الموجود في المجموعة الثانية
(d) وهو الحرف

اذن الأن الحرف دي الموجود في التعبير القياسي يطابق الحرف الموجود في السلسلة النصية في المجموعة الثانية في البديل الثاني

الأن المطابقة قد تمت لكن الأن هنالك شي احب ان اذكره هو انه احد قد يسأل لماذا قد تم اهمال البديل الثالث الموجود في المجموعة الثانية وهو البديل

(de)

لأنه نحن الأن المطابقة تمت وهذا هو الغرض منها لكن لأيوجد ضرر من الأكمال و كذلك لأيوجد ضرر من المتابعة مادام الغرض الأساسي قد تم وهو المطابقة

Matching this or that

احيانا و في بعض الأحيان نرغب للتعبير القياسي الذي نكتبه ان ان يقارن بين مجموعة من الرموز او مجموعة من السلأسل النصية طبعا هذا يتم من خلال رمز هو رمز يدعى برمز الأحتياط او رمز المناوبة وهو

(|)

لكي نطابق مثل

cat او dog

فأنه من اجل هذا نكون تعبير قياسي وهو

cat|dog

وهكذا فأن لغة البيرل سوف تطابق اول احتمال محتمل من اجل ان يتم تكوين التطابق ويكون الكود ناجح

الأن ناخذ مثال لكي نوضح الحالة

*C0D3

```
$a="cats and dogs";
$b="cats|dogs|bird";
if ($a=~/$b/){
print "match";
}
print "\n";
```

هنا نـلأحظ عند تنفيذ الكود يكون التطابق موجود و ان اول كلمة في السلسة النصية هي القطة و كذلك الحال مع التعبير القياسي لذا تمت المطابقة

اما اذا غيرنا التسلسل في التعبير القياسي فأنه ايضا هذا لأيؤثر على عملية المقارنة لذا سنلاحظ هذا في هذا البرنامج

*C0D3

```
$a="cats and dogs";
$b="dogs|cats|birs";
if ($a=~/$b/){
print "match";
}
print "\n";
```

على الرغم من التسلسل في التعبير القياسي قد تغير الأ انه وايضا على الرغم من كلمة الكلب هي ذات التسلسل الأول في التعبير القياسي الأ انه كلمة القطة لأ تزال تحظى بالأسبقية في عملية المطابقة

اما اذا كانت لدينا السلسة النصية هي كلمة

cats

وكان التعبير القياسي هو بهذا الشكل كما في الكود فان عملية المقارنة ايضا سوف تتم

*C0D3

```
$a="cats";
$b="c|ca|cat|cats";
if ($a=~/$b/){
print "match";
}
print "\n";
```

الأن عملية المطابقة تمت من خلال المقارنة لان اول حرف في التعبير القياسي هو حرف السي وهكذا تمت المقارنة بنجاح

Matching repetitions

طبعا هذا الموضوع يعني مطابقة التكرارات وهو موضوع مش كثير الأستخدام في لغة البيرل ولكن سوف اشرح تعريفه العام وما اعرفه عنه
اولأ رموز تكرار المطابقات هي

*C0D3

1.

```
?
```

*C0D3

2.

```
*
```

*C0D3

3.

```
+
```

*C0D3

4.

```
{}
```

امثلة على هذه الرموز الخاصة بتكرار المتطابقات

*C0D3

1.

"a?"
هذه تعني مطابقة ال a
1
او صفر من المرات

*C0D3

2.

"a*"
هذه تعني مطابقة ال a
صفر من المرات وهذا يعني اي عدد من المرات

*C0D3

3.

"a+"
هذه تعني مطابقة ال a
من المرات يعني على الاقل مرة واحدة 1

*C0D3

4.

"a{n,m}"
هذه تعني مطابقة ال a عدد
n
من المرات بس شريطة ان لاتكون اكثر من m

*C0D3

5.

"a{n,}"
هذه تعني مطابقة ال n
من المرات n على الاقل عدد

*C0D3

6.

```
"a{n}"
عدد a هذه تعني مطابقة ال
n يعني لا اكثر ولا اقل n من المرات بس شرط ان يكون بقدر عدد ال
```

الأن ننتقل الى موضوع جديد وجميل ايضا في التعابير القياسية
وهو

Search & Replace

التعابير القياسية في لغة البيرل تلعب دورا مهما وكبيرا في عملية البحث و الأستبدال حيث انه هذه العملية ان هذه العملية تتم من خلال دالة هي دالة

*C0D3

```
S///
```

اما الشكل العام لها فهو

*C0D3

```
S/regexp/modifiers/
```

اما عن التعريف العام لهذه الدالة هو
التصنيف :-باترونات
القيم المعادة:-عددية
التعريف العام
طبعا قبل ما ان اقدم اي شي عن هذه العميلة قسم من المبرمجين يعتبرون هذه الدالة وقسم من هولأء المبرمجين يعتبروها عملية وكل فريق له ادلته و براهينه التي ثبيت انها دالة للفريق الذي يدعى انها دالة و الفريق الأخر له الأدلة التي تدل على انها عملية اما انا من جهتي فانا اعتبرها دالة لسبب بسيط انه هذه الدلة لها قيم معادة و تصنيف برمجي حالها حال كل الدوال و الله اعلم على عمل هذه الدالة تعمل على البحث في السلسة النصية الموجودة في التعبير القياسي اعتمادا على اساليب التعابير القياسية وهذه الفقرة سوف اوضحها بعد قليل ثم تعمل على تبديل الجملة الموجدة في التعبير القياسي بالكلمة الموجودة في الدالة التي

عطيناها
اساليب البحث لهذه الدالة ذكرنا انها تعتمد على اسلوب التعابير القياسية و هذه الخيارات التي تعتمد عليها هي

*C0D3

1.

i
اهمال حالة الحروف و الغاء الحالة الحساسة للاحرف

*C0D3

2.

m
اعتبر التعبير القياسي مكون من multiple lines

*C0D3

3.

s
اعتبار التعبير القيا سي مكون من سطر واحد

*C0D3

4.

g
global

*C0D3

5.

e
تقييم الطرف الايمن على انه تعبير قياسي

هذه الخيارات التي تتقيد بها الدالة اثناء العمل
اما الأن فسوف ناخذ مثال برمجس على كيفية عمل ابحث و التبديل

*C0D3

```
$a="spawn geek";
$a=~s/geek/programmer/;
if ($a=~ s/(spawn.*programmer)$/$1 for perl!/){
print $a;
print "\n";
}
```

الأن نأتي نشرح الكود هذا خطوة بخطوة

*C0D3

```
$a=”spawngeek”;
```

متغير قيمته سلسلة حرفية بين علامات الاقتباس الى الأن لأيوجد شي غريب

*C0D3

```
$a=~s/geek/programmer/;
```

هذه الخطوة من الكود البرمجي هي

s/geek/programmer

الحرف الأول وهو ال اس هو مستحوى من اسم الدالة

geek اما الكلمة الثانية وهي

فهي الكلمة الموجودة في التعبير القياسي اما الكلمة الثانية الموجودة بعدها وهي مبرمج فهي هذه الكلمة التي نريد ان تتم بها عملية الأستبدال اي انه تلغى كلمة geek

ويكون مكانها كلمة مبرمج عند التنفيذ

*C0D3

```
if ($a=~ s/(spawn.*programmer)$/$1 for perl!/){
```

هنا في هذه الخظة هي عبارة عن جلمة الشرط تحتوي بداخلها المتغير ثم بين الأقواس اول كلمة موجودة في التعبير

القياسي ثم بعد ذلك التي تضاف

*C0D3

```
$/$1
```

اما عن هذه في التحديد فلم اعرف ماهو عملها لذا لن استطيع ان اشرحها
اما الأن عند تنفيذ البرنامج فإن الناتج الذي سو ف نأخذه من تنفيذ البرنامج هذا هو

*C0D3

```
spawn programmer for perl!
```

Power Tools

طبعا الأن لدينا المزيد من ال
Escape Sequences
التي نحتاج اليها في التعابير المنتظمة وهي

*C0D3

1.

```
\U
```

*C0D3

2.

```
\L
```

الأن مباشر ة سوف ناخذ مثالأن على على هذا الرمزان لكي نعرف ماهي اهمية استخدام كل واحد منهما

*C0D3

```
$a="\L SPAWN";
print $a;
print "\n";
```

اذا نفذنا هيا الكود سوف نحصل على الكلمة الموجودة في المتغير و لكن في حالة الحروف الصغيرة اذن ال
\\L
Lower case word تعني في الأنكليزي

اي طباعة الكلمات في الحرو ف ذات الشكل الصغير

*C0D3

```
$a="\U spawn";
print $a;
print "\n";
```

تعمل على عكس عمل التي فوقها تماما

*N0T3

هذه المجلة وضعت تحت رخصة ال
(GPL)
اي انه من الممكن التعديل فيها و النشر والأقتباس دون اذن من الناشر
لكن شريطة ان يتم ذكر صاحب الجهد

Now Th3 magazine is ov3r but i wanna send my thanks to the

WwW.SoQoR.NeT
WwW.SeCuRiTyGuRuS.nEt

Sp3c!4l gr33tz to:-
HACKERS PAL
And The
SecUritY k1ck3rs T34M
Admins & m3mb3rs

M4ilm3:-Mahmoud_najafy@hotmail.com

Wr0t3 By:-M_SPAWN

تمت بحمد الله