الفصل السادس
تتبع وتصيد ومنع الأخطاء
(avoiding and handling errors)

إن مصطلح الـdebbug هو من المصطلحات الشائعة والشيقة في عالم البرمجة، هذا المصطلح يشير إلى كيفية إصلاح أخطاء البرنامج وتوقعها قبل حدوثها، هناك أنواع من الأخطاء تحدث بسبب المبرمج وهناك أنواع من الأخطاء تحصل بسبب المستخدم، في العادة يجب أن يكون المبرمج متآلفاً مع مصطلح تتبع الأخطاء وإصلاحها.

قد يكون من أهداف تتبع الأخطاء الحماية بقدر أهمية البرنامج الجاري العمل عليه أو الموقع فكلما كان الموقع مهماً كان وجوب حمايته أكبر.

قد يكون من الأسباب التي تسبب تدميراً للمواقع هو أن صاحب الموقع يغطي كل صغيرة وكبيرة عن برنامجه الذي يركبه في موقعه وقد يكون برنامجه هذا غير محمي بشكل كاف أو يكون مسير بعدة ملفات فيقوم شخص بحذف ملف من الملفات الأساسية بسبب عدم دقة في التراخيص المعطاة مما يؤدي إلى دمار الموقع نهائياً.

وقد يكون صاحب الموقع مهملاً في الحد ذاته فلا يحتفظ بالمعلومات السرية لموقعه مما يسبب مشاكل أكبر من التدمير مثل احتلال الموقع بشكل كامل.

رسائل الخطأ في الـPHP لها طريقتها وتقنيتها الخاصة التي تسير عليها فهي ليست مثل الجافا وليست مثل cgi

فالـ PHP لا تقوم بإرسال الخطأ إلى السيرفر بل تقوم بكتابة رسالة خطأ في مكان الخطأ.

قد يكون هناك أخطاء يصعب تتبعها أو معرفة مكانها في الأصل، وقد يكون هذا بسبب أنك تستخدم الـPHP في صناعة موقع ديناميكي وتشرك معها الجافا سكريبت وتضع علامات التعليق الخاصة التي تقوم بإخفاء الأخطاء في الجافا مما قد يجعلك تشعر بالحيرة وتجن أين مكان الخطأ.

```
<!- -
رساله الخطأ
- - >
```

أنواع الأخطاء

هناك أنواع من الأخطاء منها الإملائية (Syntax Error) ومنها المنطقية ومنها أخطاء تحدث في وقت التنفيذ

ومثال الأخطاء الإملائية:

```
<?
Eco “1”;
//من المفترض أن تكت التالي:
Echo “1”;
?>
```

هذا سيعطيك رسالة خطأ Parse error

ومن الأخطاء الإملائية نسيان الفاصلة المنقوطة (semi-colon) في نهاية الدالة:

```
<?
Echo “hello”
//من المفترض أن تكتب التالي:
Echo “hello”;
?>
```

هنا سوف يعطيك الـPHP رسالة خطأ لكن العجيب أنه لن يعطيك إياها بشكل صحيح فرسالة الخطأ تشير إلى أن السطر الرابع يحتوي على الخطأ بينما الخطأ هو في السطر الثاني.

وهناك خطأ آخر يحصل بسبب نسيان الـbrace (وهي الاقواس):

```
<? Php
for ($loop = 0 ; $loop < 5 ; $loop ++)
{
Echo “”;
?>
```

إذا كنت قد نسيت إغلاق القوس فهذا من الأخطاء الشائعة، والأخطاء الإملائية لايمكن حصرها، إنها أشبه بقواعد اللغة، لكن أكثر الأخطاء الإملائية الشائعة في برامج الـPHP

١ – نسيان الأقواس.

مثال:

```
<?
for ($loop = 0 ; $loop < 5 ; $loop ++)
{
for ($loop1 = 0 ; $loop1 < 10 ; $loop1 ++)
{
for ($loop = 0 ; $loop < 5 ; $loop ++)
{
code ....
}
}
```

في المثال السابق ينقصنا قوس إغلاق التكرار الأخير (}).

٢ - نسيان الفاصلة المنقوطة.

مثال:

```
<?
Echo 10
<?
```

٣ - خطأ إملائي في اسم **function**.

مثال:

```
<?
Htmlspecialchar($I);
?>
```

سيعطيك رسالة خطأ:

```
Fatal error: call to Undefined function: htmlspecialchar().
```

وتصحيحها أن تكون:

```
<?
Htmlspecialchars($I);
?>
```

٤ - نسيان إغلاق النص.

مثال:

```
<?
Echo “arabbuilder;
?>
```

نسي الـ(") في نهاية الكلمة. وسيعطيك Parse error

الأخطاء المنطقية [Logical Errors]

إن الأخطاء المنطقية هي الأكثر صعوبة في التتبع فقد تجد برنامجك يعمل بشكل صحيح وبكل سلامة ولكنه عند نقطة ما لا يتم تنفيذها كما تريد أنت، لنضرب مثلاً على خطأ منطقي بسيط جداً، لنفرض أنك قمت بعمل نموذج مكون من مربع نص وزر، عند ضغطك لهذا الزر فأنت تريد أن تتم كتابة كلمة كبير إذا كان الرقم أكبر من ٣٠ وكلمة صغير إذا كان الرقم أصغر من ٣٠ لنقم بكتابة الكود للمثال الأول:

```
<?
" ;أدخل عمرك:echo "
echo '<br>
<form method = "post" action = "age.php">
<input type= "text" name = "age">
<br>
" >؟هل أنا كبير أم صغير<input type= submit value = "
</form>' ;
?>
```

في ملف age.php اكتب الكود التالي:

```
<?
";أنت صغيرIf ($age<30) echo "
";أنت كبيرIf ($age>30) echo "
?>
```

سيعمل السيكريبت بشكل صحيح.. ولكن ربما تخطئ أنت في كتابة العلامات المنطقية (التي باللون الأحمر) فتأتي النتائج بشكل خاطئ.

ومن الأخطاء المنطقية الأخطاء التي تقع في وقت التشغيل (Run times error) والتي قد تقوم بإيقاف برنامجك بشكل كامل

مثال:

```
<?
$t=0;
$r=1;
$f=$r/$t;
?>
```

وعندها سينتج لك الرسالة التالية

Warning: Division by zero in (path) on line (line number)

هناك نوع آخر من الأخطاء المنطقية (unexpected) وهو لا يقوم بإيقاف البرنامج نهائياً بل يقوم بإخراج رسالة الخطأ في مكان الخطأ أو قد يقوم بتنفيذ البرنامج وإخراج البيانات بشكل غير صحيح أو قد لا يقوم بإخراج بيانات
وهو المثال الأول الذي ذكرناه سابقاً (تقييم العمر).

أخطاء التكرارات:

قد يكون لديك أيضاً تكرار فيه خطأ ولا يقوم بالتوقف نهائياً مثل هذا التكرار:

```
$c=1;
$t=true;
while ($t=true)
{
$c++;
}
```

لم نقم بعمل شي يوقف التكرار مثل أن تضع شرط يختبر قيمة المتغير ($c) ثم يقوم بإيقافه عند تعديه رقم معين وعلى ذلك فإن التكرار سيستمر بشكل غير متوقف ولن يعمل البرنامج.

عدم إرجاع قيمه من function:

مثال:

```
<?
Function ($d)
{
$d =$d+$d;
}
```

الخطأ هنا أننا لم نستخدم الـreturn لكي ننهي الدالة أو قد تكون الدالة تحتوي على أكثر من قيمة وننسى أن نقوم بتحديد القيمة النهائية للدالة.

الخلط في المعاملات الحسابية والمنطقية:

مثال:

```
If ($y=10) echo 12 ;
```

والمفترض أن تكون:

```
If ($y= =10) echo 12 ;
```

أفكار جيدة لتفادي الأخطاء:

التعليقات:

إن من الأفكار الجيدة للتقليل من الأماكن التي تبحث فيها عن الخطأ هو وضع تعليقات لوصف وظيفة دالة معينة.

مثال:

```
<?
//هذه الكود يقوم بطباعة كلمة أحمد
Echo " أحمد " ;
?>
```

الدوال

وأيضاً من الأفكار الجيدة أن تقوم بتقسيم وظائف البرنامج على دوال بحيث أن لكل دالة وظيفتها المعينة:

```
<?
/*
+- ----------------------------------------------------------+
|              هذه الدالة تقوم بقسمة العدد على ٢              |
+- ----------------------------------------------------------+
* /
function ($U)
{
$U=$U/2;
return $U ;
}
?>
```

Regular Expressions

هذه التقنية تساعدك على تفادي الأخطاء في صفحتك عند حدوثها مثل أن يقوم مستخدم ما بكتابة بريد إلكتروني غير صحيح (مثال: a@y@.k.d) هذا البريد غير صحيح ولأجل أن تقوم بمنع حصول أي خطأ مثل ذلك وتقييد العبارات التي يدخلها المستخدم فإنك تقوم باستخدام الـ RE/(Regular Expressions) إنك بالأصح تجعل قواعد للكلمات التي يدخلها المستخدم فمثلاً تجعل المستخدم لا يدخل سوى أرقام أو حروف فقط أو شكل معين من الكلمات، تقوم أولاً بإنشاء نمط للكلمة التي تريد المستخدم أن يقوم بإدخالها.

النمط [pattern]

ماهو النمط ؟ ما رأيك إذا كتب المستخدم جملة في مربع نص تحتوي على عدة كلمات وتريد أن تتأكد من وجود كلمة معينة وسط هذه الجملة، على حسب ما أخذناه من معلومات عن المصفوفات سابقاً نستطيع فعل ذلك كالتالي:

```
<?
";، five، four، three، two،$words="one
$ty);،$ty =explode ("“
foreach ($ty as $w) {
 “six”) echo “found string ‘two’”; if ($w = =
}
?>
```

لقد كان المتغير $words يحتوي على جملة تتكون من عدة كلمات وعندما أردنا فحصه قمنا باستخلاصه في مصفوفة ثم بعد ذلك قمنا بفحص المصفوفة باستخدام التكرار foreach، ومع ذلك فإن الذي فعلناه بهذا الاستخدام غير عملي بتاتاً وهنا تبرز قوة Regular Expressions لاحظ الآن كيف نستخرجه بواسطة الـ Regular Expressions:

```
<?
";، five، four، three، two،$words="one
 ' one' " ;  لقد وجدت العدد$words)) echo "،if (ereg("one
?>
```

في هذا المثال قمنا باستخدام الدالة (ereg) ووضعنا في خانتها الأولى النمط (pattern) الذي نريد أن نتأكد من وجوده (أوالكلمة المراد البحث عنها) ووضعنا في الخانة الثانية المتغير الذي سيتم البحث فيه عن الكلمة أو النمط.

تقوم الدالة ereg بإعطاء القيمة true إذا تم العثور على الكلمة.

في الواقع هناك استخدامات أكثر فعالية للأنماط.

يمكننا مثلاً تخزين الكلمة إذا تم وجودها في مصفوفة خاصة كالتالي:

```
<?
";، five، four، one، two،$words="one
$rok))  ;،$words،if (ereg("one
echo $rok[0];
echo $rok[1];
?>
```

نقوم بوضع اسم المصفوفة حيث نريد تخزين البيانات في الخانة الثالثة.. لاحظ مع أنه يوجد كلمتين في الجملة توافق النمط إلا أنه أعطانا كلمة واحدة فقط إذ إن وظيفته أن يتأكد من وجود النمط في الجملة فقط فإذا تأكد من وجوده مرة واحدة استكفى واعتبر الموضوع قد انتهى.

ماذا لو أردنا أن نتأكد من عدة كلمات، عند ذلك فإننا نفعل التالي:

```
<?
";، five، four، one، two،$words="one
$rok))  echo $rok[0];،$words،if (ereg("one
$rok))  echo $rok[0];،$words،if (ereg("two
?>
```

وأريد أن أنبهك أن الـereg يقوم بإنشاء المصفوفة من جديد عند كل استعمال له فخذ حذرك من هذه النقطة

أيضاً فإن الـereg حساس لحالة الأحرف لاحظ هذا المثال:

```
<?
";، five، four، vcx، two،$words="one
$rok))  echo $rok[0];،$words،if (ereg("One"
?>
```

لن يقوم بإخراج أي شي فقط لأن حرف الـO مختلف.

أيضاً يمكنك البحث عن كلمة يسبقها فراغ مثلاً كالتالي:

```
<?
";، five، four، vcxone، two،$words="one
$rok))  echo $rok[0];،$words،if (ereg("one"
?>
```

مثال آخر:

```
<?
";، five، four، vcxone، two،$words="oned
$rok))  echo $rok[0];،$words،if (ereg("one"
?>
```

لاحظ في هذين المثالين أنه مع أن كلمة one غير موجودة بمفردها إنما موجودة كجزء من vcxone وoned ورغم ذلك فإن الدالة لم تأخذ اعتباراً لذلك بينما لو كتبنا كالتالي:

```
<?
";، five، four، vcxone، two،$words="oned
$rok))  echo $rok[0];،$words،if (ereg(" one"
?>
```

فإنه سيبحث عن الكلمة مفصولة عن أي حرف ولن يجد كلمة كذلك فلن يقوم بكتابة أي شي.

يمكننا أن نفحص قيمة موجودة في متغير كالتالي:

```
<?
$reu = "one";
";، five، four، vcxone، two،$words="one
$rok))  echo $rok[0];،$words،if (ereg($reu
?>
```

هل لاحظت أننا فحصنا قيمة المتغير $rue بواسطة ereg مع $word ولم يتطلب منا ذلك أي شيء إضافي غير اسم المتغير المراد البحث عن قيمته في الجملة.

يمكننا بالـ Regular Expression استعمال بعض الأحرف بشكل خاص التي لها استعمالها الخاص بواسطة الـ Regular Expressions

الأحرف الخاصة في الـ Regular Expression هي كالتالي:

```
. * ? + [ ] () { } ^ $ | \
```

هذه الأحرف لها معناها الخاص في الـ Regular Expression

فقديماً مثلاً كنا نقول إنه لا يمكننا أن نستخدم علامتي تنصيص متداخلة من نفس النوع كالتالي:

```
<?
$r=”u\””;
?>
```

ولكي يتجاهل الـPHP هذا المعنى فإننا نقوم بوضع (\) قبل علامة التنصيص.

أيضاً مع الـ ereg فإن لل(.) قداستها ولكي يتم تجاهلها فإننا نستخدم الـ(\)

تقوم الـ(.) بأخذ مكان حرف أو فراغ فمثلاً لاحظ المثال التالي:

```
<?
$P="I love yamen";
$R)) echo $R[0];،$P،if (ereg ("love...."
?>
```

هل لاحظت الناتج ؟؟

ولكي يتم تجاهل قداسة الـ(.) في الـRegular Expressions نقوم بوضع (\) قبلها.

مثال:

```
<?
$P="I love yamen";
$R)) echo $R[0];،$P،if (ereg ("love\.\.\.\."
?>
```

في هذا المثال لن تتم طباعة أي شيء لأنه لا يوجد أي كلمة تطابق (love....) لأن الـ(.) فقدت قداستها وبدأ التدقيق في الكلمة حرفاً حرفاً.

صناعة فئة حروف [xyz]

أقصد بذلك أنني أحدد نطاق معين من الكلمة من الممكن أن يكون في هذا النطاق أي حرف من الفئة التي أقوم بتحديدها أو الحروف التي أقوم بتحديدها.

مثال:

```
<?
$y=”how are you ? “ ;
 $y)) echo “true”;،if (ereg(“h[oe]”
?>
```

هنا قام الـregular expression بالبحث عن أي كلمة تبدأ بالحرف h ومن ثم يتبعها أحد الحرفين o أو e

مثال هذه الكلمات:

Hey – He – Hew - Homer

ولكنها لاتطابق:

Hty – Hnt - Hlay

أتمنى أن تكون فهمت ما أرمي إليه.

يمكننا أيضا أن نقوم بإخبار الـregular expression بأن لا يقوم باختيار كلمات تحتوي على حروف معينة وذلك فقط بإضافة ^

```
<?
$y=”how are you ? “ ;
 $y)) echo “true”;،if (ereg(“h[^oe]”
?>
```

نقوم هنا بإخبار الـre بأن يقوم بفحص الجملة فإذا وجد أي كلمة تبدأ بـh ولا تحتوي على o أو e فإنه يقوم بإعطاء true وإذا لم يجد يقوم باعطاء false

وهذا الكلام يطابق الكلمات التالية:

Hay - Hana - Hkg

ولا يوافق هذه الكلمات:

Home – Hore - Here

يمكننا استعمال اختصارات لبعض الأمور فمثلاً إذا كنا نريد كلمة لا تحتوي على أي رقم كنا سنكتب كالتالي:

[^123456789]

يمكننا أن نستعمل اختصار لهذا الموضوع كالتالي:

[^0-9]

وحتي إذا أردنا أن يتأكد من وجود رقم من واحد إلى تسعة فقط علينا مسح الـ^

[0-9]

وأيضاً الحروف الصغيرة من a إلى z

[a-z]

وإذا أردنا التأكد من عدم وجودها

[^a-z]

نفس القصة مع الحروف الكبيرة.

هناك اختصارات أخرى لهذا الموضوع كالتالي:

الاختصار	المطابق له	معناه ووظيفته
\d	[0-9]	أي رقم من ٠ إلى ٩
\D	[^0-9]	ممنوع الأرقام من ٠ إلى ٩
\w	[0-9A-Za-z_]	أي رقم من ١-٩ أو حروف A-Z أو أحرف صغيره أو_
\W	[^0-9A-Za-z_]	عكس السابق
\s	[\t\n\r]	يقبل مسافة أو سطر جديد أو علامة جدولة (tab)
\S	[^\t\n\r]	عكس السابق

تحديد مكان الكلمة:

يمكننا أن نقوم بتحديد مكان الكلمة، أقصد بذلك أنه يمكنك تحديد مكان الكلمة إذا كانت في بداية أو نهاية النص ونستخد لهذا الأمر العلامتين (^) لتحديد المكان لبداية الجملة و ($) لنهاية الجمل.

مثال:

```
<?
$y="how are you ? " ;
$y)) echo "true";،if (ereg("^h"
?>
```

هنا سيقوم الـphp بالبحث عن H في الجملة فإذا وجد الجملة تبدأ بحرف h كانت قيمة الـereg تساوي true وإذا لم يجد كانت قيمة الـereg تساوي false .

```
<?
$y="how gone?" ;
$y)) echo "true";،if (ereg("^g"
?>
```

في هذا المثال ستكون قيمة الـereg خطأ لأن العبارة لا تبدأ بحرف g

يمكننا فعل العكس بواسطة العلامة ($) التي عملها عكس (^) فهي تفحص إذا كان الحرف المراد فحصه موجود في نهاية الجملة.

مثال:

```
<?
$y="how g" ;
$y)) echo "true";،if (ereg("g$"
?>
```

يمكننا أيضا اختيار إذا ما كان واحد من النمطين صحيحاً بواسطة العلامة (|)

```
<?
$y="how g" ;
$y)) echo "true";،if (ereg("^y | g$"
?>
```

في هذا المثال سيقوم الـPHP بفحص الجملة فإذا وافقت أحد النمطين كانت قيمة الـereg عند ذلك true.

يمكننا أيضاً تحديد إذا ما كان حرف أو جملة متكررة بعدد من المرات أو مرة واحدة باستخدام أحد هذه الثلاثة رموز (❖، +، ?)

تقوم علامه الضرب بالتحقق من أن الحرف الذي يسبقها مكرر مرة أو أكثر أو غير موجود بتاتاً

مثال:

```
Bea*t
```

وتوافق:

Bet
Beat
Beaat

تقوم علامة الجمع (+) بالتأكد من وجود عنصر مرة أو أكثر:

```
Bea+t
```

وتوافق:

Beat
Beaat
Beaaaaat

أما علامة الاستفهام فتقوم بالتأكد من وجود عنصر مرة واحدة أو عدم وجوده بتاتاً:

```
Bea?t
```

وتوافق:

Bet
Beat

وتأكد دائماً أن هذه الثلاث علامات مسبوقة بحرف.

وعند إرادتك مثلاً التأكد من سبق حرفين أو ثلاث بشكل تحديدي يمكنك استخدام القوسين

مثال:

```
(wo)?man
```

ويوافق:

man
woman

يمكننا التأكد من تكرر حرف بشكل معين من المرات أو أكبر من عدد معين من المرات أو أصغر من عدد معين من المرات باستخدام القوسين {x ، y}

فمثلاً لو أردنا أن نتأكد من أن حرف (d) مكرر مرتين إلى أربع مرات:

```
d{2،4}
```

أما إذا أردنا أن نتأكد من أنه مكرر أكثر من مرتين إلى عدد غير محدود من المرات:

```
d{2،}
```

أما إذا أردناه أن يتكرر ٤ مرات على الأكثر:

```
4},d{
```

أو إذا أردناه أن يتكرر بعدد محدود من المرات:

```
d{8}
```

أخيراً نريد أن نلفت النظر إلى الاختصار (\b) الذي معناه أي شيء ولكن ليس حرفاً (الحروف التي بين \w وبين \W تقريباً)

ملخص ما أخذناه من القواعد تجدونه في الجدول التالي:

القاعدة	المعنى
[abc]	أي حرف كان a أو b أو c
[^abc]	أي حرف غير a و b و c
[a-z]	كل الحروف من a إلى z
\d\D	\d للأرقام و \D لغير الأرقام
\w\W	\w للحروف جميعها و\W لغير الحروف
\s\S	\s للفراغ (space) و\S لغير الفراغ (no space)
\b	الحروف التي بين \w و\W
.	أي حرف
(abc)	تقوم باعتبار abc كمجموعة..
؟	حرف أو مجموعة حروف مكررة مره أو غير مكررة نهائياً
+	حرف أو مجموعة حروف تتكرر مرة أو أكثر
*	حرف أو مجموعة حروف تتكرر مرة أو

	أكثر أو قد لا تتكرر نهائياً
{x،y}	تكرير بعدد معين من المرات..
{،y}	تكرير بحد أقصى من المرات..
{x،}	تكرير بحد أدنى من المرات...
{x}	تكرير بعدد معين من المرات
^	في بداية النص
$	في نهاية النص

تعبير للتأكد من إيميل

^[_a-zA-Z0-9-]+(\.[_A-Za-z0-9-]+)*@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$

شرح التعبير

الرمز	الشرح
^	يجب أن يبدأ النص
[_A-Za-z0-9]	أي حرف من a-z كبيراً كان أو صغيراً أو _أو أرقام
+	وقد يكون هذا الحرف متكرراً أكثر من مره
(\.[_A-Za-z0-9-]+)	بالاضافه إلى أنه قد يتبع النقطة وحروف وأرقام
*	وقد لا يتبعه أو قد يتبعه ويتكرر أكثر من مرة
@	وبعد ذلك يكون لديه حرف ال@
[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$	وأيضاً نفس القواعد في النهاية

مثال:

```
<?
$t)،Function mailcheck($mail
{
$T="^[_a-zA-Z0-9-]+(\.[_A-Za-z0-9-]+)*@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$";
$mail))،If (EREG($T
{
$r="the mail is true";
echo $r;
}
else
{
$r="the mail is not true";
echo $r;
}
return ;
}
$t);،mailcheck("alfareees@hotmail.com"
?>
```

eregi()

الفرق بين هذه الدالة والدالة ereg أنها غير حساسة لحالة الأحرف كبيرة أو صغيرة أي أنه يمكننا كتابة المثال السابق كالتالي:

```
<?
$t)،Function mailcheck($mail
{
$T="^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*$";
$mail))،If (EREG($T
{
$r="the mail is true";
echo $r;
}
else
{
$r="the mail is not true";
echo $r;
}
return ;
```

```
}
$t);،mailcheck("alfareees@hotmail.com"
?>
```

ereg_replace()

ماذا لو أردت تحرير عبارة ما من أحرف معينة وقد تكون متكررة في جملة أو غير ذلك.

لنفرض أن لدينا العبارة التالية:

```
Mohmed love his game .....
```

ونريد أن نتخلص من النقاط التي في نهاية العبارة

أو لدينا مثلاً هذا المسار:

```
C:\windows\desktop
```

ونريد أن نستبدل العلامة (\) بـ (/)

كل ذلك ممكن بواسطة الدالة ereg_replace وقواعد الـ rgular expression التي أخذناها سابقاً

البنية التي نستخدمها للدالة كالتالي:

```
var);،string،Ereg_replace(reg
```

نضع في مكان reg القاعدة لـ regular expression ونضع مكان الـstring الحرف الجديد ونضع بدلاً من الـvar المتغير الذي نريد استخلاص الحروف منه.

مثال:

```
<?
$path =" C:\windows\desktop";
$tell= "Mohmed love his game.....";
$path);،"/"،$newpath= Ereg_replace("[\.]"
$tell);،""،$newtell= Ereg_replace("\."
echo $newpath;
echo "<br><br>";
echo $newtell;
?>
```

أساليب أخرى لتتبع الأخطاء

استخدام عباره echo

هو من أقدم الأساليب وكان يستخدم مثلاً في فحص بعض متغيرات نموذج، فمثلاً أنت لديك نموذج يقوم بإرسال معلومات إلى النموذج وقد تستخدم في اختبار الأخطاء المنطقية التي يستصعب متابعتها في الكود.

مثال:

```
<?
Echo “this is: $name”;
Echo “<br>”;
Echo “this is: $Email”;
//كود يقوم بمعالجة معلومات المتغيرين
//طباعة المتغيرين بعد أداء عملية المعالجة ورؤية النتائج
Echo “this is after: $name”;
Echo “<br>”;
Echo “this is after: $Email”;
?>
```

فحص كود الـ html

قد تستخدم كود جافا سكريبت ويتم إخفاء الأخطاء وسط علامات التعليقات فعليك حينئذ فحص كود الـhtml لرؤية إن كان هناك بعض الأخطاء المخفية أم لا.

تجاهل الأخطاء

لنفترض أنك تعلم أن الدالة التي صنعتها بها أخطاء ولكنك تريد تجاهل هذه الأخطاء فكل ما عليك أن تقوم به وضع @ أمام الدالة لكي يتم تجاهل الخطأ عند حدوثه.

مثلاً نحن نعلم أن القسمة على الصفر من الأشياء الغير مقبولة في الـPHP وأنت صنعت دالة تقوم بالقسمة على صفر ولن يتم تنفيذها لأنها بالأصل خطأ ولكنك تريد أن يقوم PHP بتجاهلها فكل ما عليك أن تفعله هو وضع @ أمام الدالة.

مثال:

```
<?
function amail ($y)
{
$y=$y/0;
return $y;
}
$s= @amail(44);
echo $s;
?>
```