Δομή Επανάληψης

Σε όλες τις δομημένες γλώσσες προγραμματισμού (Pascal, Basic, C, Java κ.λ.π.) υπάρχει η δυνατότητα να δημιουργήσουμε **επαναληπτικές δομές** Με τις δομές αυτές μπορούμε να εκτελέσουμε ένα κομμάτι κώδικα πολλές φορές, είτε για ένα συγκεκριμένο αριθμό επαναλήψεων είτε μέχρις ότου ικανοποιηθεί μια συνθήκη. Όπως και στις δομές επιλογής χρειαζόμαστε λογικές παραστάσεις που θα βοηθούν τον υπολογιστή να αποφασίσει από μόνος του πότε θα τερματιστεί μια επαναληπτική δομή.

Οι βασικές εντολές της PHP που χρησιμοποιούμε για το σκοπό αυτό είναι οι παρακάτω:

- **while** – όσο μια συνθήκη είναι αληθής εκτελεί κάποιο κομμάτι κώδικα
- **do...while** - εκτελεί κάποιο κομμάτι κώδικα όσο μια συνθήκη είναι αληθής
- **for** – εκτελεί κάποιο κομμάτι κώδικα για συγκεκριμένο αριθμό επαναλήψεων
- **foreach** - εκτελεί κάποιο κομμάτι κώδικα για κάθε στοιχείο ενός πίνακα

Εντολή While

Η σύνταξη της εντολή είναι:

while (συνθήκη είναι αληθής) {
    Εντολές προς εκτέλεση
}

Η εντολή while είναι η πιο γενική εντολή επανάληψης και μπορεί να επιλύσει όλες τις περιπτώσεις όπου χρειάζεται επανάληψη. Οι υπόλοιπες εντολές επανάληψης έχουν επινοηθεί για να αντικαταστήσουν τη while όταν αυτή γίνεται σύνθετη κατά την επίλυση ειδικών προβλημάτων.

Στα παρακάτω παράδειγμα βλέπουμε πως μπορούμε με τη while να εμφανίζουμε τους αριθμούς 1 έως 100 (αριστερά) και τους περιττούς αριθμούς μεταξύ 1 και 100 (δεξιά).

```php
<?php
$t = 1;
while ($t <= 100) {
      echo $t."<br>";
      $t++;
}
?>
```

```php
<?php
$t = 1;
while ($t <= 100) {
      echo $t."<br>";
      $t += 2;
}
?>
```

Στο παράδειγμα αυτό βλέπουμε ότι ο αριθμός των επαναλήψεων είναι συγκεκριμένος (100) και ότι το κριτήριο για να σταματήσουμε την επανάληψη είναι η $t να «**μετρήσει**» μέχρι το 100. Σε τέτοιες περιπτώσεις λέμε ότι η μεταβλητή παίζει το ρόλο ενός **μετρητή**. Στο παράδειγμα δεξιά ή μόνη διαφορά είναι ότι η $t αυξάνεται κατά 2 σε κάθε επανάληψη. Έτσι περνάει μόνο από τις περιττές τιμές (1,3,5,7….99) και τερματίζει την επανάληψη όταν η $t πάρει τη τιμή 101.

Ακολουθεί ακόμα μια σειρά παραδειγμάτων για

α) τον εμφάνιση 10 τυχαίων αριθμών μεταξύ 1 και 100

β) τον υπολογισμό του αθροίσματος 10 τυχαίων αριθμών μεταξύ 1 και 100

γ) τον υπολογισμό του μέσου όρου 10 τυχαίων αριθμών μεταξύ 1 και 100

```php
<?php
// α) εμφάνιση 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
while ($t <= 10) {
      echo $rand(1,100)."<br>";
      $t++;
}
?>
```

```php
<?php
// β)υπολογισμός αθροίσματος
// 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
$s = 0;
while ($t <= 10) {
      $s += rand[1,100);
      $t ++;
}
echo $s;
?>
```

```php
<?php
// γ) υπολογισμός μέσου όρου
// 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
$s = 0;
while ($t <= 10) {
      $s += rand[1,100);
      $t ++;
}
echo $s/10;
?>
```

Σε όλα τα παραπάνω παραδείγματα ο αριθμός των επαναλήψεων είναι δεδομένος (10) και για τη μέτρησή τους χρησιμοποιούμε τη $t. Αυτή η μεταβλητή παίζει το ρόλο του μετρητή και βοηθά να τερματιστεί η επανάληψη, δηλαδή καθορίζει τη συνθήκη της επανάληψης. Επίσης σε όλα τα παραδείγματα χρησιμοποιούμε τη συνάρτηση **echo** για τη παραγωγή των τυχαίων αριθμών.

Στα παραδείγματα β και γ εμφανίζεται ακόμα μια μεταβλητή η $s. Η μεταβλητή αυτή αρχικοποιείται πριν την επανάληψη σε 0. Αυτό συμβαίνει πάντα όταν θέλουμε μια μεταβλητή να παίξει το ρόλο του **αθροιστή**, δηλαδή να χρησιμοποιηθεί για να προσθέσουμε ένα σύνολο τιμών. Μέσα στην επανάληψη δεν εμφανίζουμε τη τιμή που παράγει η rand, αλλά τη προσθέτουμε στην υπάρχουσα τιμή της $s.

Σε όλα τα παραδείγματα που είδαμε μέχρι τώρα ο αριθμός των επαναλήψεων ήταν γνωστός και γι' αυτό το λόγο χρησιμοποιήσαμε μια μεταβλητή ως μετρητή ($t). Τέτοιες περιπτώσεις μπορούν να αντιμετωπιστούν, όπως θα δούμε και παρακάτω, και με την εντολή for. Ας δούμε όμως μια περίπτωση όπου το πλήθος των επαναλήψεων δεν είναι γνωστό.

Έστω ότι θέλουμε να παράγουμε και να εμφανίζουμε τυχαίους αριθμούς από το διάστημα -10 έως 10 και να σταματήσουμε όταν βγει ο αριθμός 0. Έστω επίσης ότι θέλουμε να εμφανιστεί και το 0 που είναι ουσιαστικά και ο τελευταίο αριθμός που θα παραχθεί.

```php
<?php
$t = 1;
while ($t != 0) {
      $t = rand(-10,10);
      echo $t."<br>";
}
?>
```

```
Η αρχική τιμή 1 στην $t πριν
την επανάληψη βοηθά να μπούμε
τη πρώτη φορά στην επανάληψη.
Στη συνέχεια παράγονται τιμές
από την rand για να συνεχιστεί
η διαδικασία
```

Η αρχικοποίηση της $t σε 1 στο είναι περισσότερο ένα τέχνασμα. Αν θέλαμε να την αποφύγουμε χωρίς να κάνουμε το κώδικα πιο περίπλοκο θα μπορούσαμε να χρησιμοποιήσουμε την εντολή **do…while** που ακολουθεί.

Εντολή do…while

Η σύνταξη της εντολή είναι:

do {
Εντολές προς εκτέλεση
} **while** (συνθήκη είναι αληθής);

Η εντολή **do…while** είναι και αυτή μια γενική εντολή επανάληψης και μπορεί να επιλύσει όλες τις περιπτώσεις όπου χρειάζεται επανάληψη. Η διαφορά της από την απλή **while** είναι ότι ανάλογα με το πρόβλημα που επιλύεται ο κώδικας μπορεί να είναι πιο απλός όταν λύνεται με τη μια από αυτές και πιο πολύπλοκος με την άλλη. Έτσι μπορούμε να επιλέξουμε ανάμεσα από τις δυο για να απλοποιήσουμε το κώδικα.

Η βασική λειτουργική διαφορά τους είναι ότι η **while** έχει τη συνθήκη τερματισμού πριν από την ομάδα εντολών της επανάληψης ενώ η **do while** μετά. Αυτό έχει σαν αποτέλεσμα η **do while** να εκτελείται οπωσδήποτε μια τουλάχιστον φορά πριν γίνει ο έλεγχος της συνθήκης και αποφασιστεί η συνέχιση ή ο τερματισμός.

Στα απλά παραδείγματα που είδαμε πριν με συγκεκριμένο αριθμό επαναλήψεων, δεν έχει σημασία ποια από τις δυο θα χρησιμοποιηθεί. Και οι δυο παράγουν το ίδιο κώδικα σε έκταση και αποδοτικότητα.

Ας δούμε τα παραδείγματα με τα αθροίσματα λυμένα με τη **do**…**while**.

```php
<?php
// α) εμφάνιση 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
do {
      echo $rand(1,100)."<br>";
      $t++;
} while ($t <= 10);
?>
```

```php
<?php
// β)υπολογισμός αθροίσματος
// 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
$s = 0;
do {
      $s += rand[1,100);
      $t ++;
} while ($t <= 10);
echo $s;
?>
```

```php
<?php
// γ) υπολογισμός μέσου όρου
// 10 τυχαίων αριθμών
// μεταξύ 1 και 100
$t = 1;
$s = 0;
do {
      $s += rand[1,100);
      $t ++;
} while ($t <= 10);
echo $s/10;
?>
```

Ας δούμε όμως και το παράδειγμα με τη παραγωγή τυχαίων αριθμών μέχρι να παραχθεί το 0. Αν το λύσουμε με τη βοήθεια της do…while τότε θα μπορέσουμε να

αποφύγουμε την αυθαίρετη αρχικοποίηση του $t σε 1, αφού ο έλεγχος της $t θα γίνει στο τέλος, ύστερα δηλαδή από τη παραγωγή της πρώτης τιμής με τη rand. Μπορεί εδώ το όφελος να φαίνεται μικρό μιας και γλιτώνουμε μια μόνο εντολή, αλλά αυτό που μας ενδιαφέρει είναι και η καθαρότητα του κώδικα που τον κάνει πιο ευανάγνωστο και κατανοητό, αποτρέποντας λογικά λάθη κατά τη τροποποίηση και διόρθωσή του.

```
<?php
do {
      $t = rand(-10,10);
      echo $t."<br>";
} while ($t != 0);
?>
```

```
Η αρχικοποίηση της $t σε 1 δεν
είναι πλέον απαραίτητη. Η $t
θα πάρει τιμή από τη rand πριν
γίνει ο πρώτος έλεγχος τιμής.
```

Εντολή for

Η εντολή επανάληψης **for** χρησιμοποιείται <u>όταν ξέρουμε, ή μπορούμε με κάποιο τρόπο να εκφράσουμε, τον αριθμό των επαναλήψεων</u>. Όπως έχουμε πει αυτό μπορεί να γίνει με τη **while** αλλά έχουμε επινοήσει την **for** για να απλοποιήσουμε τη γραφή αυτής της εντολής, μιας και χρησιμοποιείται πολύ συχνά. Η for είναι στη PHP μια πολύ δυνατή εντολή και μπορεί να καλύψει και άλλες περιπτώσεις, αλλά εμείς θα σταθούμε μόνο στη βασική της λειτουργικότητα.

Όπως φαίνεται από τη σύνταξή της η for είναι υπεύθυνη να χειριστεί έναν μετρητή που θα βοηθήσει στη μέτρηση των επαναλήψεων και στο τερματισμό της όλης διαδικασίας. Έτσι παρατηρούμε ότι ενσωματώνει σε μια σειρά όλες τις λειτουργίες που με την while εμφανιζόντουσαν σκόρπιες μέσα στο κώδικα. Αυτές οι λειτουργίες είναι:

1) Αρχικοποίηση του μετρητή (στη while γινόταν πριν την επανάληψη)
2) Έλεγχος του μετρητή (η συνθήκη της while)
3) Τροποποίηση του μετρητή (στη while γινόταν μέσα στο σώμα των εντολών)

Έτσι η σύνταξη της εντολή είναι:

for (αρχικοποίηση μετρητή; έλεγχος μετρητή; Μεταβολή μετρητή) {
    Εντολές προς εκτέλεση
}

Ας δούμε ένα απλό παράδειγμα γραμμένο με την while και τη for σε αντιπαραβολή. Μπορεί για άλλη μια φορά το όφελος από τη χρήση της for να φαίνεται μικρό, αλλά σε πολύπλοκο κώδικα η for μαζεύει όλα τα κομμάτια κώδικα σχετικά με τον μετρητή, σε ένα σημείο και αποτρέπει λάθη από αβλεψίες. Ο κώδικας έτσι γίνεται όχι μόνο μικρότερος αλλά πιο κατανοητός και εύκολος στη διόρθωση.

```
<?php
// εμφάνιση 10 τυχαίων αριθμών
// μεταξύ 1 και 100
// με τη χρήση της while
$t = 1;
while ($t <= 10) {
      echo $rand(1,100)."<br>";
      $t++;
}
?>
```

```
<?php
// εμφάνιση 10 τυχαίων αριθμών
// μεταξύ 1 και 100
// με τη χρήση της for
for ($t = 1;$t <= 10;$t++) {
      echo rand(1,100)."<br>";
}
?>
```

Ας

Ας δούμε και μερικά παραδείγματα ακόμα για τη δημιουργία δομών HTML με τη χρήση της for. Τονίζουμε ότι ο μετρητής της for Μπορεί να χρησιμοποιηθεί εντός της επανάληψης σαν οποιαδήποτε μεταβλητή και δεν χρησιμοποιείται αποκλειστικά για το χειρισμό της επανάληψης.

Ακολουθεί ένα παράδειγμα για τη δημιουργία ενός HTML πίνακα όπου εμφανίζονται οι αριθμοί 1 έως 10, και δίπλα στον κάθε ένα το η δεύτερη και η τρίτη δύναμή του, όπως φαίνεται παρακάτω. Στο παράδειγμα δε δίνουμε βάση σε όλο τον HTML κώδικα και στη μορφοποίηση CSS που θα μπορούσαμε να προσθέσουμε.

1	1	1
2	4	8
3	9	27
4	16	64
κ.ο.κ.		

```
<!-- εμφάνιση αριθμών 1 έως 10
 με τις δεύτερες και τρίτες δυνάμεις τους
 μέσα σε HTML πίνακα ->
<table>
<?php
for ($t = 1;$t <= 10;$t++) {
      echo "<tr>";
      echo "<td>" . $t . "</td>";
      echo "<td>" . $t**2 . "</td>";
      echo "<td>" . $t**3 . "</td>";
      echo "</tr>";
}
?>
</table>


<table>
<?php for ($t = 1;$t <= 10;$t++) { ?>
      <tr>
      <td><?php echo $t; ?></td>
      <td><?php echo $t**2; ?></td>
      <td><?php echo $t**3; ?></td>
      </tr>
<?php } ?>
</table>
```

Στο παραπάνω παράδειγμα βλέπουμε δυο τρόπου γραφής του κώδικα που θα έχουν το ίδιο αποτέλεσμα ως τελική **HTML**. Στο πρώτο παράδειγμα έχουμε ένα μεγάλο **script** (από **<?php** έως **?>**) όπου μέσα δημιουργούμε όποιο καινούργιο **HTML tag** θέλουμε

με την echo. Επειδή βρισκόμαστε εντός του **script**, η **HTML** δεν αναγνωρίζεται και μπαίνει σε εισαγωγικά ως αλφαριθμητική σταθερά (κείμενο).

Στη δεύτερη λύση ανοίγουμε και κλείνουμε πολλαπλά **scripts** τα οποία βέβαια από την **PHP** αντιμετωπίζονται σαν ένα μεγάλο **script** όταν πάει να τα εκτελέσει. Δηλαδή μεταβλητές και εντολές του ενός script αναγνωρίζονται και συνεχίζονται στο επόμενο. Σε αυτή την αντιμετώπιση θεωρούμε ότι οποιοδήποτε στοιχείο βρίσκεται εκτός των **scripts** είναι HT**M**L και θα αποδοθεί στην έξοδο ως έχει. Σε όποιο σημείο αυτού του **HTML** κώδικα θέλουμε να βάλουμε κάτι μεταβλητό με τη βοήθεια της **PHP** ανοίγουμε ένα καινούργιο **script**.

Ακολουθεί ένα παράδειγμα για τη δημιουργία ενός HTML πίνακα όπου εμφανίζονται 100 τυχαίοι αριθμοί (που θα παράγει η rand από το 1 έως το 1000), μέσα σε ένα πίνακα 10X10=100 θέσεων. Για τη δημιουργία του πίνακα αυτού θα χρησιμοποιήσουμε δυο επαναληπτικές εντολές τη μια μέσα στην άλλη. Αυτή η δομή ονομάζεται εμφωλευμένη επανάληψη. Η εξωτερική επανάληψη θα χρησιμοποιείται για τη παραγωγή των 10 γραμμών και η εσωτερική για τη παραγωγή των 10 κελιών της κάθε γραμμής.

```
<!-- εμφάνιση αριθμών 100 τυχαίων αριθμών
 μέσα σε HTML πίνακα 10 σειρών και 10 στηλών->
<table>
<?php
for ($t = 1;$t <= 10;$t++) {
      echo "<tr>";
      for ($g = 1;$g <= 10;$g++) {
            echo "<td>" . rand(1,1000) . "</td>";
      }
      echo "</tr>";
}
?>
</table>


<table>
<?php for ($t = 1;$t <= 10;$t++) { ?>
      <tr>
      <?php for ($g = 1;$g <= 10;$g++) { ?>
            <tr>
            <td><?php echo rand(1,1000); ?></td>
            </tr>
      <?php } ?>
      </tr>
<?php } ?>
</table>
```

Και εδώ βλέπουμε δυο λύσεις, μια όπου η **HTML** παράγεται από την **PHP** με **echo** και μια όπου η **PHP** μπαίνει μέσα στην **HTML** με τη μορφή πολλών **scripts**.