פרק 16 – פונקציות

הקדמה

מדעי המחשב הוא תחום שנוצר בעיקר על ידי מתמטיקאים, אנשים העוסקים באלקטרוניקה וחשמל, ופה שם גם כמה פיסיקאים תועים. באלקטרוניקה יש מושג שנקרא "קופסא שחורה". בכדי לבנות מערכת גדולה, גדולה יותר ממה שבן אדם אחד יכול לפתח ולעקל, יש צורך בחלוקה של עבודה וחלוקה של העבודה מתבצעת באמצעות הקופסאות השחורות. קופסא שחורה הוא חלק שמבצע משהו מבלי שלאדם שמשתמש בו יהיה איכפת איך הוא עושה זאת. כשאני כותב כעת אני לא מתעניין בשאלה איך הדברים קורים, ואיך מעבד התמלילים מזיח את השורות או מציג נכונה את המלל על פי הגופן הנבחר, כל שאיכפת לי הוא כיצד אני משתמש במעבד התמלילים, מה הוא עושה, לא איך הוא עושה זאת.

זו הקדמה ארוכה לפחות יחסית לעובדה שאני מניח שאתה הקורא יודע פונקציות מהן עוד מ ActionScript. החדשות הטובות הן (ושוב, אין חדשות רעות) שפונקציות ב PHP דומות מאוד לפונקציות ב ActionScript והמעבר לבניית פונקציות ב PHP אמור להיות קל.

בפרק הקודם ראינו מספר פונקציות לטיפול במערכים, בפרק הזה אנו נלמד לבנות את הפונקציות האישיות שלנו שיעשו כרצוננו.

הגדרה של פונקציה

פונקציה חדשה מוגדרת על פי אותו התחביר של ActionScript. ראשית אנו כותבים את מילת המפתח function, לאחר מכן אנו כותבים את שם הפונקציה, שם הפונקציה ילווה בזוג סוגריים עגולים שביניהם ניתן להכניס ארגומנטים. לאחר מכן אנו פותחים זוג סוגריים מסולסלים ובו כותבים את הקוד שיתבצע בכל קריאה לפונקציה. הבט בדוגמא הבאה:

```
function sayHello()
{
     echo "Hello There";
}
```

בקוד הגדרנו פונקציה בשם sayHello שתפקידה הוא להציג לדפדפן את המחרוזת "Hello There" בעזרת הפקודה echo.

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

כמו ב ActionScript גם ב PHP ניתן לקרוא לפונקציה בשורה עליונה יותר מאשר בשורה שבה הפונקציה הוגדרה. כך לדוגמא הקוד הבא יציג את המחרוזת "Hello There" למרות שהקריאה לפונקציה התבצעה לפני הגדרת הפונקציה:

```
sayHello();

function sayHello()
{
     echo "Hello There";
}
```

תכונה זו של PHP התווספה רק מגרסה 4.

אני נוהג להציב את ההגדרות של הפונקציות בסוף הקובץ כך שהן לא מפריעות לי לראות מה קורה, מהי התמונה הגדולה, כבר עם הכניסה לקובץ.

פונקציה יכולה לקבל ארגומנטים שונים בין הסוגריים העגולים ולהשתמש בהם בכדי לבצע פעולות שונות בגוף הפונקציה, לדוגמא:

```
sayHello("Orit");
sayHello("Gil");

function sayHello($name)
{
     echo "Hello There $name<br>";
}
```

בקוד הגדרנו פונקציה בשם sayHello שמקבלת כפרמטר מחרוזת שבבלוק הקוד של הפונקציה תקרא $name. הפונקציה מציגה ברכת שלום דינאמית, דינאמית מהבחינה הזו שהערך של $name מהווה חלק מהברכה.

בזוג השורות הראשונות קראנו ל sayHello פעם עם המחרוזת "Orit" ופעם עם המחרוזת "Gil". כפי שבודאי תוכל לנחש פלט הקוד יהיה:

Hello There Orit
Hello There Gil

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

החזרת ערך מפונקציה

כמו ב ActionScript, גם פונקציות ב-PHP יכולות להחזיר ערכים בעזרת מילת המפתח return.
הבט לדוגמא בקוד הבא:

```
echo maxNum(9,-1);

function maxNum($a, $b)
{
     if ($a>$b) return $a;
     return $b;
}
```

בקוד הגדרנו פונקציה בשם maxNum שמקבלת כפרמטר שני מספרים $a ו-$b ומחזירה את הגדול ביניהם. בכדי להחליט איזה מספר הוא הגדול. הפונקציה נעזרת במשפט התניה, משפט ההתניה בודק אם ערכו של $a גדול מערכו של $b, במקרה הזה הפונקציה מחזירה את הערך $a ומסיימת את תפקידה. אם הפונקציה ממשיכה לשורה השנייה אנו יכולים להיות בטוחים ש $a איננו גדול מ $b, הוא קטן ממנו או שווה לו ולכן נוכל להחזיר את $b. אם היינו משתמשים ב else התוצאה הייתה זהה.

בשורה הראשונה קראנו ל maxNum עם המספרים 1- ו 9 ואת התוצאה החזרנו ל echo שדאג להציג את המספר שהפונקציה maxNum מחזירה, במקרה הזה המספר הוא 9.

נתבונן בדוגמא קצת יותר מורכבת. נניח ואנו בונים פונקציה שמתנהגת כמו חלק שקיים בכספומט עם אינסוף שטרות של 20 ושל 50 שקלים. הפונקציה מקבלת מספר כארגומנט שמציין כמה כסף המשתמש מעוניין למשוך ואנו צריכים להחליט אם הדבר אפשרי או לא עם שטרות של 20 ושל 50 שקלים. אם הדבר אפשרי אנו נחזיר את הערך true, אחרת נחזיר את הערך false. הקוד הבא מבצע את העבודה:

```
function getMoney($cash)
{
     $temp = $cash;

     while ($temp >= 0)
     {
          if (($temp%20 == 0) || ($temp%50 == 0))
               return true;

          $temp -= 50;
     }
     return false;
}
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

הפונקציה getMoney מקבלת כפרמטר משתנה מסוג מספר בשם $cash. בשורה הראשונה בגוף הקוד של הפונקציה מוגדר משתנה השייך לפונקציה בשם $temp, משתנה זה מאותחל לערכו של $cash.

לאחר הגדרת $temp אנו רואים לולאת while שבודקת אם $temp מתחלק ב-20 או ב-50, כלומר אם ניתן לספק את $temp בשטרות של 20 ושל 50, אם כן אז הפונקציה מחזירה true, אחרת הפונקציה מורידה מ $temp 50 (למה?). הלולאה מתקיימת כל עוד $temp חיובי.

אם יצאנו מהלולאה ועדיין נשארנו בפונקציה סימן שאי אפשר לפרוט את $cash לשטרות של 20 ו-50 ולכן נחזיר false בעזרת return. בכדי לבדוק את הקוד נעזרתי בלולאה הבאה שרצה על כל הערכים האפשריים בין 0 ל-200:

```
for($i=0; $i<=200; $i++)
{
     if (getMoney($i)) echo "$i ";
}
```

הפלט נראה כך:

0 20 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200

טווח הכרה של משתנים בהקשר לפונקציות

ישנם שני חוקים שכדאי לזכור בכל הנוגע לטווח הכרה של משתנים בהקשר לפונקציות:

- משתנה שהוגדר מחוץ לפונקציה לא קיים בתוך גוף הפונקציה.
- משתנה שהוגדר בתוך פונקציה לא קיים מחוץ לפונקציה.

זה כאילו מדובר בשני עולמות נפרדים – מגוף הפונקציה לא ניתן לגשת אל משתנים מחוץ לפונקציה אלא אם כן מציינים זאת במפורש כפי שאסביר בהמשך. משתנים שנמצאים מחוץ לפונקציה גם כן לא יכולים לגשת למשתנים שנמצאים בתוך פונקציה – בכלל, וטוב שכך. הבט בדוגמא הבאה:

```
$num = 13;
echo "num = $num ";
myFunc();
echo "num = $num ";

function myFunc()
{
     $num = 10;
     echo "num = $num ";
}
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

בשורת הקוד הראשונה הגדרנו משתנה בשם $num ואתחלנו את ערכו ל-13. בשורה השנייה הצגנו את ערכו לדפדפן, כך עשינו גם בשורה הרביעית. בשורה השלישית קראנו לפונקציה בשם myFunc שמאתחלת משתנה בשם $num לערך 10 ומציגה את ערכו. חשוב להבין שהמשתנה $num שב myFunc הוא משתנה שונה מהמשתנה $num שמחוץ להגדרת הפונקציה. הפלט נראה כך:

num = 13
num = 10
num = 13

ה-13 שהוצג בשורה הראשונה בא מה $num שהוגדר מחוץ לפונקציה, כך גם בשורה השלישית. בשורה השנייה ה-10 הוא ערכו של המשתנה $num שנמצא בתוך הפונקציה myFunc והוא שונה לגמרי, פרט לשמו, מהמשתנה $num שמחוץ לפונקציה. אני חוזר על הנקודה הזו הרבה בכדי שתפנים אותה היטב. הבט בקוד הבא בכדי להפנים עוד יותר כיצד טווח ההכרה של משתנים ב PHP פועל עם פונקציות:

```
$a = 10;
echo "Before doubling: $a<br>";
doubleIt($a);
echo "After doubling: $a<br>";

function doubleIt($num)
{
     $num *= 2;
     echo "After doubling but in function: $num<br>";
}
```

בקוד הגדרנו משתנה בשם $a ואתחלנו את ערכו ל-10. לאחר מכן הצגנו את הערך של $a לפני הקריאה לפונקציה בשם doubleIt שמקבלת כפרמטר את $a ומכפילה את ערכו. הערך המוכפל הוצג שוב בתוך הפונקציה. בשורה הרביעית הצגנו את ערכו של $a לאחר ש doubleIt סיימה את תפקידה. הפלט נראה כך:

Before doubling: 10
After doubling but in function: 20
After doubling: 10

אתה אולי שואל את עצמך מדוע בשורה השלישית ערכו של $a חזר להיות 10 ולא נשאר 20. הסיבה לכך היא ש PHP לא מאפשרת לפונקציה לשנות ערך של משתנים שנמצאים מחוץ לבלוק הקוד שלה. כשהעברנו את $a כפרמטר לפונקציה doubleIt כל שעבר הוא ערכו של $a, כלומר המספר 10, המקום בזיכרון בו $a נמצא איננו ידוע לפונקציה, כך שכל שינוי של הפרמטר בתוך גוף הפונקציה לא ישפיע על ערכו של המשתנה מחוץ לפונקציה.

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

מילת המפתח global

ההשלכות מהסעיף הקודם הן שפונקציה לא יכולה לשנות ערך של משתנה שנמצא מחוץ לפונקציה שזה דבר טוב ברוב המקרים, אבל יש מקרים שבהם דווקא נרצה לשנות את משתנים שנמצאים מחוץ לפונקציה, או לפחות לגשת אליהם. אין כל בעיה לעשות זאת אנו רק צריכים לומר ל PHP במפורש שזה רצוננו בעזרת מילת מפתח בשם global.

מילת המפתח global אומרת ל PHP שאנו לא מעוניינים להגדיר משתנה חדש אלא לאפשר גישה למשתנה שנמצא מחוץ לפונקציה.

```
$a = 5;

echo "Before calling tripleIt: $a<br>";
tripleA();
echo "After calling tripleIt: $a<br>";

function tripleA()
{
     global $a;
     $a *= 3;
}
```

אם נתמקד בשורה הראשונה בגוף הפונקציה tripleA נראה את מילת המפתח global ומיד אחריה שם משתנה שנמצא מחוץ לגוף הפונקציה. כעת אנו נוכל לגשת למשתנה הזה, במקרה שלנו $a, מתוך גוף הפונקציה, לגשת לערכו, לשנות את ערכו וכו'. בכדי להיווכח בכך בשורה השנייה הקוד מציג את ערכו של $a, שורה לאחר מכן אנו קוראים ל tripleA שמכפיל את ערכו של $a, ושורה לאחר מכן, בשורה הרביעית אנו מציגים שוב את ערכו של $a. הפלט נראה כך:

Before calling tripleIt: 5
After calling tripleIt: 15

ניתן לראות שהפונקציה שינתה את הערך של $a.

אם נרצה לגשת ליותר ממשתנה אחד נוכל להשתמש מספר פעמים ב global או להפריד את המשתנים השונים שאנו רוצים לגשת אליהם בפסיק, לדוגמא:

```
global $a,$b;
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

שינוי ערכי ארגומנטים (&)

כבר הספקנו לראות שפונקציה לא יכולה לגשת ובודאי לא לשנות ערכי משתנים שנמצאים מחוץ לבלוק הקוד שלה, אלא אם כן מציינים במפורש ל PHP, בעזרת global, שאנו כן מעוניינים לשנות ערך משתנה כלשהו שנמצא מחוץ לפונקציה. עוד הספקנו לראות כי הארגומנטים שמועברים לפונקציה אינם ניתנים לשינוי.

המשפט האחרון איננו לגמרי מדויק, כמו במקרה של משתנים שנמצאים מחוץ לבלוק הקוד של פונקציה, גם ארגומנטים ניתנים לשינוי אם רק נגיד ל PHP שאנו מעוניינים לעשות זאת. PHP מצילה אותנו מעצמנו.

הדרך בה אנו אומרים ל PHP שאנו רוצים לשנות ערך ארגומנט כלשהו הוא בעזרת הצבת הסימן & לפני שם הארגומנט, כך אנו אומרים ל PHP שאנחנו לא רוצים להעביר את ערכו של המשתנה לפונקציה אלא את המשתנה עצמו. הבט בקוד הבא:

```
$num = 5;

echo "Before: $num<br>";
addTen($num);
echo "After: $num<br>";

function addTen(&$num)
{
     $num += 10;
}
```

אם תקפוץ להביט בהגדרת הפונקציה תוכל לראות את הסימן & לפני שם הארגומנט $num. כך אנו מעבירים את המשתנה עצמו לפונקציה ולא רק את הערך שלו. להעברה של המשתנה עצמו לפונקציה קוראים – **העברה על פי ייחוס** בניגוד **להעברה על פי ערך**. הקוד דומה לדוגמאות הקוד הקודמות – אני מציגים את ערכו של המשתנה לפני ואחרי הקריאה לפונקציה. הפלט נראה כך:

Before: 5
After: 15

נביט בדוגמא נוספת – הפעם נביט בפונקציה שמקבלת שני ארגומנטים ומחליפה את הערכים שלהם, שם ראוי לפונקציה שכזו הוא swap:

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

```
$num1 = 26;
$num2 = 5;
echo "($num1,$num2)<br>";
swap($num1,$num2);
echo "($num1,$num2)<br>";

function swap(&$a, &$b)
{
     $temp = $a;
     $a = $b;
     $b = $temp;
}
```

הפונקציה ודאי מוכרת לך, ראינו כבר החלפה של ערכים בין שני משתנים באחד הפרקים הקודמים. בהגדרת הפונקציה ניתן לראות שהעברנו גם את $a וגם את $b על פי ייחוס, כלומר העברנו לפונקציה את המשתנים עצמם ולא רק את ערכם. כמו קודם – אנו מציגים את ערך המשתנים לפני ואחרי הקריאה לפונקציה. הפלט יראה כך:

(26,5)
(5,26)

דוגמאות ותרגול

הגענו פחות או יותר לאמצע הפרק, לפני שנמשיך נסה לכתוב פונקציות שיבצעו את הפעולות הבאות. בחן את הפונקציות!

- פונקציה שמקבלת שלושה מספרים ומחליפה את הערכים שלהם כך שהמשתנה הראשון יקבל את הערך הגדול ביותר, האמצעי את הערך האמצעי בגודלו, והמשתנה האחרון יקבל את הערך הקטן ביותר.
- פונקציה שמקבלת כפרמטר שני מספרים המהווים אורך ורוחב של מלבן, ומחזירה את השטח של המעגל החוסם את המלבן.
- פונקציה שמחשבת את העצרת של מספר שהועבר אליה כארגומנט. העצרת של מספר הוא מכפלת כל המספרים השלמים עד למספר הנתון, לדוגמא 5 עצרת, המסומן גם כך - $5!$ ערכו הוא: $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$.

פונקציה אחת שיכולה להיות מאוד שימושית בעבודה עם PHP ופלאש היא הפונקציה שמוסיפה למחרוזת כלשהי שאנו מעוניינים להעביר לפלאש עוד צמד שם-ערך. בדרך כלל אנו נאלצים לעשות זאת בעזרת שרשור למחרוזת קיימת באופן הבא:

```
$string .= "&name=Gil Cohen";
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

אמנם השורה הזו לא מורכבת מדי אבל הרבה יותר נוח יהיה לראות משהו שנראה כך:

```
append($string, "name", "gil");
```

הפונקציה append היא פונקציה שמיד נבנה שמוסיפה לסוף המחרוזת $string את השם-ערך המתקבלים כארגומנטים השני והשלישי בהתאמה. הנה המימוש של הפונקציה append:

```
function append(&$str, $name, $value)
{
     $str .= "&$name=$value";
}
```

הדבר הראשון שאני רוצה שתשים לב אליו הוא לעובדה שהארגומנט הראשון, המחרוזת, מועבר על פי ייחוס בכדי שהפונקציה תוכל להוסיף למחרוזת מידע נוסף על הקיים. שני הפרמטרים האחרים מועברים על פי ערך כי ערכם הוא כל מה שאנו רוצים.

שים לב ש & משמש גם כדרך של פלאש להבדיל בין שני משתנים במחרוזת שהיא מקבלת, אל תתבלבל עם התפקיד הזה של & לבין התפקיד של העברה על פי ייחוס.

עבודה עם מספר לא ידוע של ארגומנטים

החל מגרסה 4 של PHP אנו יכולים לקרוא לפונקציה עם מספר אינו קבוע מראש של ארגומנטים. העבודה עם הארגומנטים הללו מתבצעת בעזרת 3 פונקציות קלות לשימוש שפועלות רק מתוך גוף של פונקציה אחרת.

הפונקציה הראשונה שאדבר עליה היא הפונקציה func_num_args, הפונקציה מחזירה את מספר הארגומנטים שהועברו לפונקציה כלשהי. הבט בקוד הבא:

```
myFunc(5,10);
myFunc("gil");
myFunc();
myFunc("a","b","c","d","e");

function myFunc()
{
     echo func_num_args()." ";
}
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

בשורה החמישית הגדרנו פונקציה בשם myFunc שלכאורה איננה מקבלת ארגומנטים – הסיבה לכך שלא כללנו ארגומנטים בסוגריים העגולים של myFunc היא שמדובר בפונקציה עם מספר לא ידוע של ארגומנטים. כן כדאי, כהרגל, להציב הערה בתוך הסוגריים שמציינת שמדובר בפונקציה שמקבלת מספר לא קבוע של ארגומנטים. כל שהפונקציה עושה הוא להציג בעזרת הפונקציה func_num_args כמה ארגומנטים העוברו אליה.

בארבעת השורות הראשונות קראנו ל myFunc עם מספר ארגומנטים שונה בכל פעם, הפלט נראה כך:

2 1 0 5

הפונקציה fun_num_args מאפשרת לנו לדעת כמה ארגומנטים הועברו לפונקציה, זוהי פונקציה מאוד יעילה במקרה השכיח בו אנו רוצים לעבור על כל הארגומנטים כאילו היו במערך. אבל כיצד ניתן לגשת לארגומנטים? ובכן לשם כך PHP מספקת את הפונקציה func_get_arg שמקבלת כפרמטר מספר המציין את מספר הארגומנט. הפונקציה תחזיר את הארוגמנט המבוקש. הבט בקוד הבא:

```
echo Average(1, 10, 4);

function Average( /* args */ )
{
     $numArgs = func_num_args();
     $sum = 0;

     if ($numArgs == 0) return 0;

     for($i=0; $i < $numArgs; $i++)
     {
          $sum += func_get_arg($i);
     }

     return ($sum/$numArgs);
}
```

מרבית הקוד מגדירה פונקציה בשם Average שמקבלת מספר לא קבוע של ארגומנטים ותפקידה הוא להחזיר את ממוצע הארגומנטים. השורה הראשונה בקוד קוראת לפונקציה Average עם הארגומנטים 1, 10 ו-4 והפלט יהיה כמובן 5. הבה נביט מתחת למכסה המנוע של Average.

בשורה הראשונה בגוף הפונקציה הגדרנו משתנה חדש בשם $numArgs שמאותחל לערך שהפונקציה func_num_args מחזירה, זהו כאמור מספר הארגומנטים שהועברו לפונקציה. בשורה השנייה הגדרנו משתנה בשם $sum שאותחל לאפס. משתנה זה יאגור את סכום כל הארגומנטים.

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

שורה לאחר מכן אנו רואים משפט התנייה שבודק אם מספר הארגומנטים שהועבר הוא 0, במידה וזה המצב אנו מגדירים את הממוצע כ-0 ומחזירים 0 בעזרת return. הסיבה לכך שאנו דואגים למקרה הקצה הזה בו $numArgs שווה לאפס הוא בגלל שבכדי לחשב את הממוצע אנו צריכים לחלק במספר הארגומנטים ואסור לחלק ב-0.

מיד לאחר משפט ההתניה יושבת לה לולאת for שעוברת על כל הארגומנטים ובכל איטרציה מוסיפה את ערך הארגומנט הנוכחי ל $sum. הלולאה נעזרת בפונקציה func_get_arg בכדי לקבל את הארגומנט ה $i-י.

לבסוף אנו נעזרים במילת המפתח return בכדי להחזיר את המנה של סכום הארגומנטים במספר הארגומנטים - הממוצע שלהם.

בתחילת הסעיף אמרתי שיש 3 פונקציות המאפשרות לנו לעבוד עם מספר לא קבוע של ארגומנטים לפונקציה, אך שתי הפונקציות הקיימות – func_get_arg ו- func_num_args מאפשרות לנו לבצע כל פעולה שנרצה, אז מה תפקיד הפונקציה השלישית? ובכן הפונקציה השלישית מחזירה, בקריאה אליה, מערך המכיל את כל הארגומנטים. שם הפונקציה הנ"ל הוא func_get_args. הקוד הבא מממש את הפונקציה Average, הפעם בשימוש ב func_get_args:

```
function Average( /* args */ )
{
     $theArgs = func_get_args();
     $numArgs = count($theArgs);
     $sum = 0;

     if ($numArgs == 0) return 0;

     for($i=0; $i < $numArgs; $i++)
     {
          $sum += $theArgs[$i];
     }

     return ($sum/$numArgs);
}
```

בשורה הראשונה בגוף הפונקציה הגדרנו מערך בשם $theArgs שמקבל את המערך המוחזר מהפונקציה func_get_args – מערך המכיל את כל הארגומנטים שהועברו לפונקציה. השורה השנייה מגדירה משתנה בשם $numArgs, הפעם לא על פי func_num_args, למרות שהדבר אפשרי, אלא על פי count – כלומר על פי ספירת מספר האיברים במערך $theArgs. שאר הקוד זהה לדוגמא הקודמת פרט לגוף הלולאה, שם אנו לא קוראים לפונקציה func_get_arg אלא מושכים את הערך מהמערך $theArgs.

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

רקורסיה

אם אינך יודע כלל מה זה רקורסיה אני מציע שתדלג על הסעיף. אני כן אומר שרקורסיה זהו תהליך בו פונקציה מסוימת קוראת לעצמה עם פרמטרים שונים וב PHP הדבר לא שונה. המטרה שלנו הסעיף היא להציג את כל האיברים במערך שהאיברים שלו הם בעצמם מערכים, אך לפני שנעשה זאת הבה נביט בקוד פשוט יותר – בתרגול התבקשת לחשב, ללא רקורסיה, את העצרת של מספר נתון, ניתן לעשות זאת בלולאת for אך ניתן לעשות זאת גם באופן רקורסיבי ומסיבות פדגוגיות נתחיל משם – הבט בקוד הבא:

```
function Fact($n)
{
     if ($n == 1) return 1;
     return $n*Fact($n-1);
}
```

הפונקציה מורכבת בסך הכל מ-2 שורות, אבל יש מקום להרחיב את הדיבור עליה. הפונקציה Fact (התרגום הלועזי לעצרת הוא Factorial) מקבלת כארגומנט מספר ותפקידה הוא להציג את העצרת שלו. בשורה הראשונה הפונקציה בודקת אם ערך הארגומנט שהתקבל הוא 1, אם כן היא מחזירה 1, כי עצרת של 1 זה 1. אם הגענו לשורה השנייה נסיק כי $n שונה מ-1, במקרה הזה הפונקציה מחזירה את המכפלה של $n עם הערך שיוחזר מהפונקציה Fact, אותה פונקציה עצמה, עם ארגומנט שונה, הארגומנט $n-1.

כעת נעבור לדוגמא קצת יותר מעניינת – הצגה של איברי מערך, גם אם איברי המערך הם מערכים בפני עצמם, הרעיון הוא שהפונקציה תכנס לעומק האיברים במערך, ורקורסיה הוא הכלי המושלם לעבודה. ראשית הבט במערך איתו נעבוד:

```
$crazyArray =  array ( "2",
               "4",
               array ( "3",
                       "5"),
               "1",
               array ( "8",
                       array ( "1",
                               array("0","5"),
                               "9"),
                       "7"),
               "6");
```

למערך קראנו $crazyArray, ואתה בודאי לא תתקשה להבין מהיכן בא שמו. המערך מכיל מערכים בתוך מערכים כאיברים וזאת בכדי לבחון שהפונקציה שלנו אכן יודעת להתמודד עם מערך שכזה. נעבור לפונקציה:

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

```
function echoArray($array)
{
     echo " { ";

     for($i=0; $i<count($array); $i++)
     {
          if (is_array($array[$i]))
          {
               echoArray($array[$i]);
          } else {
               echo $array[$i]." ";
          }
     }
     echo "} ";
}
```

שם הפונקציה הוא echoArray, העיקרון שלה הוא כזה – אם האיבר שאנו מטפלים בו כרגע איננו מערך אז תציגי את ערכו, אחרת – תקראי לעצמי עם המערך המהווה את האיבר הנוכחי. הפונקציה מקשתת קצת עם סוגריים מסולסלים את תתי המערכים, הפלט נראה כך:

{ 2 4 { 3 5 } 1 { 8 { 1 { 0 5 } 9 } 7 } 6 }

אם ידע קודם ברקורסיות בודאי לא תתקשה ליישם את הידע גם ב PHP.

משתנים סטטיים - Static

לעיתים רחוקות אנו רוצים שפונקציה תשמור על ערכי המשתנים שלה גם אחרי שהיא סיימה את פעולתה, לתת לה מן זיכרון שלא פג בכל פעם מחדש, זאת ניתן לעשות בעזרת מילת המפתח static. הרעיון הוא להגדיר משתנה חדש ולאתחל אותו, האתחול של המשתנה הזה יעשה רק פעם אחת ובכל כניסה לפונקציה נוכל לגשת למשתנה עם הערך שהיה לו מתי שיצאנו מהפונקציה בפעם האחרונה. הבט בקוד הבא:

```
for($i=0; $i<4; $i++) remember();

function remember()
{
     static $called = 0;

     $called ++;
     echo "I've been called $called times<br>";
}
```

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il

בשורה השנייה הגדרנו פונקציה בשם remember. בשורה הראשונה בגוף הפונקציה נעזרנו במילה השמורה static בכדי להגדיר **משתנה סטטי** בשם $called שאותחל ל-0. השורה הזו תתבצע רק בפעם הראשונה שהפונקציה תקרא, בכל פעם אחרת שהפונקציה תקרא המשתנה $called לא יאותחל בחזרה ל-0. שורה לאחר מכן קידמנו את ערך המשתנה ב-1 והצגנו את ערכו.

לפונקציה קראנו ארבע פעמים בעזרת לולאה והרי התוצאות:

```
I've been called 1 times
I've been called 2 times
I've been called 3 times
I've been called 4 times
```

כפי שניתן לראות המשתנה $called, מהיות סטטי, שמר על ערכו גם בין הזמנים בהם הפונקציה בוצעה.

באופן מעשי כל פעולה שמשתנה סטטי מאפשר לנו לעשות, אפשר לעשות גם עם משתנים גלובליים כי גם משתנים גלובליים שומרים על ערכם מחוץ לפונקציה. אך באופן כללי קוד שמסתמך על משתנים גלובליים קשה יותר לתחזוקה מקוד שמנסה לעקוף משתנים גלובלים, בין היתר בעזרת משתנים סטטיים.

סיכום

בפרק זה ראית כיצד לכתוב פונקציות משלך ב PHP, נגעת בנושאים כמו החזרת ערכים מפונקציה, גישה לארגומנטים, העברת ארגומנטים על פי ערך והעברת ארגומנטים על פי ייחוס. למדת כיצד עובדים משתנים סטטיים ב PHP, כיצד ליישם רקורסיות, פרק לא קל. הפרק הבא יהיה קליל וקצר ויידבר על הכללת קוד מקבצים שונים בעזרת פונקציות כמו include.

כל הזכויות שמורות לגיל כהן, http://www.flashoo.co.il