

תכנות PHP 5 Power

◆ פיתוח אפליקציות Java על Linux קארל אלבינג ומיקל שורץ ◆ תכנות GUI ב-C++ עם Qt 3 ג'סמין
בלאנשט, מארק סאמרפייד ◆ ניהול מערכות Linux עם Webmin: ניהול מערכת ופיתוח מודולים ג'ימי
קמרון ◆ הבנת ניהול הזיכרון הווירטואלי של Linux מל גorman ◆ הטענת CIFS: מערכת קבצי האינטרנט
המשותפת כריסטופר הרטל ◆ פיתוח תוכנה למערכות מבוסצות (Embedded) עם eCos אנטוני מאסה ◆
פיתוח אפליקציות מהיר (RAD) עם Mozilla ניגל מקפרלן ◆ פלטפורמת הפיתוח של Linux: הגדרה,
שימוש ותחזוקה של סביבת תכנות מלאה רפיק או רחמן, כריסטופר פול ◆ זיהוי חידרות עם SNORT:
טכניקות IDS מתקדמיות באמצעות PHP, MySQL, Apache, MySQL ו-ACID SNORT, Apache, MySQL רפיק או רחמן ◆ המדריך
הרשמי ל-HOWTO Samba-3: HOWTO Samba-3: מדריך ייחוס ג'ון ה. טרפסתה, ילמר ר. רונuai, עורךים ◆
דוגמה: תרגילים מעשיים לפရישה מוצלת ג'ון ה. טרפסתה
המחברים והמציא לאור נקטו משנה זהירות בהכנות ספר זה, אך אין מספקים אחריות מפורשת אן
משתמעת מכל סוג שהוא ואין נושאים באחריות לשגיאות או השמטות. לא תהיה כל חבות לנזקים נלוויים או
תוצאותיים בקשר לשימוש במידע או בתוכניות הכלולים בספר זה או הנובעים משימוש זה.

מציאו לאור: ג'ון ויט

עורך ראשי: דון או'הגן

עורך רכישות: מארק ל. טאוב

עזרהת מערצת: נורין רג'ינה

עורכת פיתוח: ג'אנט ואלאד

מנהל שיווק: רובין אובריאן

מעצבת עטיפה: נינה סקודרי

עורכת管理: ג'ינה קנאיס

עורכת פרויקט בכירה: קריסטי הארט

עריכת לשון: Specialized Composition

مفצת אינדקס: ליסה סטומפף

סדרנית בכירה: ג'ורייה שורייק

קניין יצור: דן אוריג

המציא לאור מציע הנחות מציניות על ספר זה בהזמנה כמותית לרכישות מרכזות או מכירות מיוחדות, אשר
עשויות לכלול גרסאות אלקטרוניות /או עטיפות ותכנים מותאמים אישית לעסק שלך, לעדי ההכשרה, למיקוד
השיווקי ולAINטראים המיתוגיים שלך. למידע נוסף, אנא צרו קשר עמך:

מכירות למזר העסקי והממשלה בארה"ב

(800) 3419-382

corpsales@pearsontechgroup.com

למכירות מחוץ לארה"ב, אנא צרו קשר עמך:

מכירות בינלאומיות

international@pearsoned.com

בקרו אותנו באינטרנט: www.phptr.com

נתוני קטלוג בספריית הקונגרס:

2004107331

זכויות יוצרים © Pearson Education, Inc 2005

ניתן להפץ חומר זה רק בכפוף לתנאים ולהתנויות המפורטים ברישיון הפרסום הפתוח (Open Publication License), גרסה 1.0 ומעלה (הגדרה העדכנית ביותר זינה עת בכתובת <http://www.opencontent.org/openpub>).

Pearson Education, Inc

One Lake Street

Upper Saddle River, NJ 07458

נעsha כל מאמץ ליצור קשר ולתת קרדיט לכל בעלי זכויות היוצרים. שימוש בחומר ללא קרדיט מתאים אין מכין.

X ISBN 0-131-47149

הטקסט הודפס בארצות הברית על נייר ממוחזר ב-**Phoenix** שבאגראטאון, מרילנד.

הדפסה ראשונה, [אוקטובר 2004]

ליפעת, אשתי וחברתי הטובה ביותר, שסבלה בסבלנות את המעורבות שלי ב-PHP מההתחלה, ועודדה ותמכה بي בכל צעד ושלל.

אנדי גוטמןנו

למריאן, על הסבלנות והעידוד.

סטייג סאטר באקי

להורי, שדואגים לי גם כשאני לא בסביבה;

ול-42, התשובה לחימם,

ליקום ולכל השאר.

דריך רתאננו

תוכן עניינים

קדמה מאת זאב سورסקי

פתח דבר: מבוא ורקע

פרק 1: מה חדש ב-5 ?PHP

פרק 2: שפת הליבה של 5 PHP

פרק 3: תכונות מונחה עצמים (OO) ב-5 PHP

פרק 4: תכונות מונחה עצמים מתקדם ותבניות עיצוב (Design Patterns) ב-5 PHP

פרק 5: כיצד לכתוב אפליקציית אינטרנט עם PHP

פרק 6: בסיסי נתונים עם 5 PHP

פרק 7: טיפול בשגיאות

פרק 8: XML עם 5 PHP

פרק 9: הרחבות נפוצות (Mainstream Extensions)

פרק 10: שימוש ב-PEAR

פרק 11: חבילות PEAR חשובות

פרק 12: בניית רכיבי PEAR

פרק 13: ביצוע המעבר (לגרסת 5)

פרק 14: ביצועים

פרק 15: מבוא לכתיבה הרחבות ל-5 PHP

פרק 16: כתיבת סкриיפטים למעטפת (Shell Scripting) ב-5 PHP

א. אינדקס חבילות PECL ו-PEAR

ב. רפרנס פורמט phpDocumentor

ג. מדריך התחלת מהירה ל-Zend Studio

אינדקס (מפתח עניינים)

פירוט הפרקים

1. מה חדש ב-5 ?PHP

1.1 מבוא

1.2 מאפייני שפה

1.2.1 מודל מונחה עצמים חדש

1.2.2 מאפיינים מונחים עצמים חדשים

1.2.3 מאפייני שפה חדשים נוספים

1.3 שינויים כלליים ב-PHP

1.3.1 XML ושירותי רשת (Web Services)

1.4 מאפיינים חדשים נוספים ב-5 PHP

1.4.1 מנהל זיכרון חדש

1.4.2 הוסף תומכה ב-Windows 95

1.5 סיכון

2. שפת הליבה של PHP 5

2.1 מבוא

2.2 הטעמה בתוך HTML

2.3 הערות (Comments)

2.4 משתנים

2.4.1 הפניות עקיפות למשתנים

2.4.2 ניהול משתנים

2.4.3 סופר-גLOBליים (Superglobals)

2.5 סוג נתונים בסיסיים

2.5.1 מספרים שלמים (Integers)

2.5.2 מספרים בנקודה צפה (Floating-Point)

2.5.3 מחרוזות (Strings)

2.5.4 בוליאנים (Booleans)

2.5.5 Null

2.5.6 משאבים (Resources)

2.5.7 מערכיים (Arrays)

2.5.8 קבועים (Constants)

2.6 אופרטורים

2.6.1 אופרטורים ביןאריים

2.6.2 אופרטורי השמה
2.6.3 אופרטורי השוואה
2.6.4 אופרטורים לוגיים
2.6.5 אופרטורים על סיביות (Bitwise)
2.6.6 אופרטורים אונאריים
2.6.7 אופרטורי שלילה
2.6.8 אופרטורי קידום/הפחתה (Increment/Decrement)
2.6.9 אופרטורי המרה (Cast)
2.6.10 אופרטור ההשתקה (Silence Operator)
2.6.11 האופרטור הטרנארי הייחד במיניו
2.7 מבני בקרה
2.7.1 מבני בקרה מותנים (Conditional)
2.7.2 מבני בקרה של לולאות
2.7.3 מבני בקרה להכלהת קוד
2.8 פונקציות
2.8.1 פונקציות המוגדרות על ידי המשתמש
2.8.2 טווח הכרזה של פונקציות (Scope)
2.8.3 החזרת ערכים לפי ערך (By Value)
2.8.4 החזרת ערכים לפי הפניה (By Reference)
2.8.5 הכרזה על פרמטרים של פונקציה
2.8.6 משתנים סטטיים
2.9 סיכום
3. תכונות מונחה עצמים (OO) ב-PHP
3.1 מבוא
3.2 אובייקטים
3.3 הכרזה על מחלקה (Class)

3.4 מילת המפתח new ובנאים (Constructors)

3.5 מפרקם (Destructors)

3.6 גישה לMETHODS ומאפיינים באמצעות המשתנה `this`

3.6.1 מאפייני `private`, `protected` ו-`public`

3.6.2 METHODS ו-`public`, `protected` ו-`private`

3.6.3 מאפיינים סטטיים

3.6.4 METHODS סטטיות

3.7 קבועי מחלקה

3.8 שכפול אובייקטים (Cloning)

3.9 פולימורפיזם (רב-צורתיות)

self:: ו-parent::

3.11 אופרטור instanceof

3.12 METHODS ומחלקות מופשטות (Abstract)

3.13 ממשקים (Interfaces)

3.14 הורשה של ממשקים

3.15 METHODS final

3.16 מחלקות final

3.17 методה `toString__()`

3.18 טיפול בחיריגות (Exception Handling)

3.19 autoload__()

3.20رمצי סוג (Type Hints) בפרמטרים של פונקציות

3.21 סיכון

4. תכונות מנחה עצמים מתקדם ותבניות עיצוב

4.1 מבוא

4.2 יכולות העמשה (Overloading)

4.2.1 העמתת מאפיינים וMETHODS

4.2.2 העמותת תחבר גישה למערך

4.3 איטרטורים (Iterators)

4.4 תבניות עיצוב (Design Patterns)

4.4.1 תבנית האסטרטגיה (Strategy Pattern)

4.4.2 תבנית הסינגלטן (Singleton Pattern)

4.4.3 תבנית המפעל (Factory Pattern)

4.4.4 תבנית הצופה (Observer Pattern)

4.5 רפלקציה (Reflection)

4.5.1 מבוא

4.5.2 ממשק תכונות היישומים (API) של הרפלקציה

4.5.3 דוגמאות לרפלקציה

4.5.4 מימוש תבנית ההאלזה (Delegation) באמצעות רפלקציה

4.6 סיכום

5. כיצד לכתוב אפליקציית אינטרנט עם PHP

5.1 מבוא

5.2 הטעמה בתוך HTML

5.3 קלט משתמש

5.4 טיפול בטוח בקלט משתמש

5.4.1 טעויות נפוצות

5.5 טכניקות להפיכת סקריפטים ל"בטוחים"

5.5.1 אימות קלט (Validation)

5.5.2 HMAC אימות

5.5.3 הספרייה PEAR::Crypt_HMAC

5.5.4 מסנן קלט

5.5.5 עבודה עם סיסמות

5.5.6 טיפול בשגיאות

5.6 עוגיות (Cookies)
5.7 סשןים (Sessions)
5.8 ה <ul style="list-style-type: none">להת קבצים
5.8.1 5.8.1 טיפול בקובץ המועלה הנוכחי
5.9 ארכיטקטורה
5.9.1 5.9.1 סקריפט אחד משרת את כל
5.9.2 5.9.2 סקריפט אחד לכל פונקציה
5.9.3 5.9.3 הפרדת לוגיקה מעיצוב (Layout)
5.10 סיכון
6. בסיסי נתונים עם PHP
6.1 מבוא
6.2 MySQL
6.2.1 MySQL נקודות חזק וחולשה של MySQL
6.2.2 MySQL ממשק ה-PHP
6.2.3 MySQL נתוני דוגמה
6.2.4 MySQL חיבורים
6.2.5 MySQL שאלות מאוחסנות (Buffered) לעומת לא מאוחסנות
6.2.6 MySQL שאלות
6.2.7 MySQL הצהרות מרובות (Multi Statements)
6.2.8 MySQL מצב שיליפה (Fetching Modes)
6.2.9 MySQL הצהרות מוכנות (Prepared Statements)
6.2.10 MySQL טיפול ב-BLOB
6.3 SQLite
6.3.1 SQLite נקודות חזק וחולשה של SQLite
6.3.2 SQLite תחומי השימוש הטוביים ביותר
6.3.3 MySQL ממשק ה-PHP

PEAR DB 6.4

PEAR DB 6.4.1

6.4.2 יתרונות וחסרונות של הפשתה בסיס נתוניים (Abstraction)

6.4.3 אילו מאפיינים מופשטים?

6.4.4 חיבורו בסיס נתונים

6.4.5 ביצוע שאילות

6.4.6 שילפת תוצאות

6.4.7 סדרות (Sequences)

6.4.8 מאפייני ניידות (Portability)

6.4.9 שגיאות מופשטות

6.4.10 מתודות נוחות

6.5 סיכום

7. טיפול בשגיאות

7.1 מבוא

7.2 סוג שגיאות

7.2.1 שגיאות תכנות

7.2.2 סמלים לא מוגדרים (Undefined Symbols)

7.2.3 שגיאות ניידות

7.2.4 שגיאות זמן ריצה (Runtime)

7.2.5 שגיאות PHP

7.3 שגיאות PEAR

7.3.1 המחלקה PEAR_Error

7.3.2 טיפול בשגיאות PEAR

7.3.3 מצב שגיאה של PEAR

7.3.4 טיפול אדיב בשגיאות (Graceful Handling)

7.4 חריגות (Exceptions)

7.4.1 מהן חריגות?

throw-i try, catch 7.4.2

7.5 סיכום

הקדמה

במהלך השנים האחרונות, PHP צמחה והפכה לפלטפורמת הרשת הנפוצה ביותר בעולם, כשהיא פועלה ביותר משלוש משרות האינטרנט ברחבי העולם. הצמיחה של PHP אינה רק כמותית אלא גם איכותית. יותר ויותר חברות מרשות Fortune, מסתמכות על PHP כדי להריץ את האפליקציות הקרייטיות לעסקיהן, דבר היוצר מקומות העבודה חדשים ומגדיל את הדרישת לפתח PHP. גרסה 5, הצפיה לשחרר בעידן הקרוב מאוד, נשאת הבטחה גדולה אף יותר.

בעוד שהמורכבות של תחילת העבודה עם PHP נותרה ללא שינוי ונמנוכה מאוד, המאפיינים שמצויה PHP כיום מאפשרים למפתחים להציג הרבה מעבר לאפליקציות HTML פשוטות. מודל העצמים המעודכן מאפשר כתיבת פרויקטים בקנה מידת גדול בקורס יייל, תוך שימוש בתמודוגיות סטנדרטיות של תכונות מונחה עצמים. תמיכת-h XML החדשת הופכת את PHP לשפה הטובה ביותר הקיימת לעיבוד XML, יחד עם התמיכת החדשה ב-SOAP, היא מהויה פלטפורמה אידיאלית לייצירה ושימוש בשירותי רשת (Web Services).

ספר זה, שנכתב על ידי עמית, אנדי גוטמןס, ושני מפתחי PHP בולטים מאוד, סטייג באין ודריק רתאנס, מחזק בפתח לפיתוח העושר של PHP. הוא מכסה ביסודות את כל המאפיינים של הגרסה החדשה, והוא פריט חובה לכל מפתחי PHP המעניינים לחקר את יכולות המתקדמות של PHP 5.

דאב סורסקי

פתח דבר

"הבטיחון הטוב ביותר ביחס נגד מהפכה טמונה בתיקון מתמיד של עולות ובהכנסות שיפורים נחוצים. ההזנחה של תיקון בזמן היא שהופכת בנייה חדש לנוחזה." — ריצ'רד וייטלי

בראשית

זה היה לפני שנים שנות, כשהראשונים לדריך החלו לארונה לפתח את IFI/PHP. הוא לא יכול היה לדמיין שיצירתו תוביל בסופו של דבר לפיתוח של PHP כפי שהוא מכיר אותו כיום, הנמצאת בשימוש של מיליוני אנשים. הגרסה הראשונה של "IFI/PHP", שנקרהה Personal Homepage Tools/Form Interpreter הייתה הרגשה הראיפה ב-Perl בשנת 1995. אחד המאפיינים הבסיסיים היה שפה דמוית Perl לטיפוף בשילוח טפסים, אך חסרה בה מאפייני שפה נפוצים ושימושיים רבים, כמו לולאות for.

2 PHP/IFI

שכתב הגיע עם 2 PHP/IFI בשנת 1997, אך באותה עת הפיתוח נוהל כמעט בלבד על ידי רשומים. לאחר שחרורה בנובמבר של אותה שנה, אנדי גוטמןס וזאב סורסקי נתקלו ב-IFI/PHP בזמן שהייתה חזקה כלפי שנראתה, פתרו מසחר אלקטורי כפרויקט אוניברסיטאי. הם גילו ש-IFI/PHP לא הייתה חזקה כלפי שנראתה, ופתחה חסירה מאפיינים נפוצים רבים. אחד הבעיות המעניינות ביותר כלל את הדרך שבה מומשו לולאות while, for, ה Soruk lexical scanner (lexer) שנבנה בעבודת יד היה עובר על הסקריפט, וכשהיה נתקל במילת המפתח while, היה זכר את מיקומה בקובץ. בסוף הלולאה, מצביע הקובץ היה חוזר למיקום ששומר, וכל הלולאה נקראה ובוצעה מחדש.

PHP 3

זאב ואנדי החליטו לשכתב חלוטין את שפת הסקריפטים. לאחר מכן הם חברו לראסמוס כדי לשחרר את PHP 3, והוא הגיע גם בשם חדש: PHP: Hypertext Preprocessor, כדי להציג ש-PHP היא מוצר שונן ולא מתאימה רק לשימוש אישי. זאב ואנדי גם תכננו ומימושו API חדש להרחבות. API זה אפשר לתמוך בפתרונות בהרחבות נוספות לביצוע משימות כגון גישה לבסיס נתונים, בודקי איות וטכנולוגיות אחרות, דבר שימושי מפתחים רבים שלא הילך מקבוצת "הילבה". PHP. בזמן שחרור 3 PHP בימי 1998, בסיס התקנות המוערך של PHP עמד על כ-50,000 דומיינים. PHP 3 הוצאה את תחילת הפירצה האמיתית של PHP, והייתה הגרסה הראשונה שהגיעה לבסיס התקנות של למעלה ממילון דומיינים.

PHP 4

בשלהי 1998, זאב ואנדי הבינו לאחר עובdotם ב-3 PHP והרגישו שהם יכולים לכתוב את שפת הסקריפטים אפילו טוב יותר, ולכן החלו בשכתוב נוסף. בעוד ש-3 PHP עדין פירשה (parsed) את הסקריפטים בקצב תוך כדי ביצועם, 4 PHP הגיע עם פרדיגמה חדשה של "קוד הידור, אחר כך ביצוע". בשלב ההידור (compilation) אינו מהדר סקריפטים של PHP לקוד מכוון; במקרה זאת, הוא מהדר אותו ל"קוד ב-Byte" (byte code), שבוצע לאחר מכן על ידי מנוע Zend (Zend). הוא ראשי תיבות של & Andi (Andi), הלב החדש של 4 PHP. בשל דרך ביצוע חדשה זו של סקריפטים, הביצועים של 4 PHP היו טובים בהרבה מאשר של 3 PHP, עם פגעה מזערית בלבד בתאמיות לאחר. בין השיפורים האחרים היה API משופר להרחבות לביצועי זמן-ריצה טובים יותר, שכבת הפשטה לשרת אינטרנט המאפשרת ל-4 PHP לרוץ על רוב שרתיו האינטראקטיביים, ועוד הרבה יותר. PHP 4 שוחררה רשמית ב-22 במאי 2000, וכיום בסיס התקנות שלו עבר את רף ה-15 מיליון דומיינים.

ב-3 PHP, מספר הגרסה המשני (הספרה האמצעית) מעולם לא הייתה בשימוש, וכל הגרסאות מסויפות כ-3.x. זה השתנה ב-4 PHP, ומספר הגרסה המשני שמש לציין שינויים חשובים בשפה. השינוי החשוב הראשון הגיע ב-4.1.0 PHP, שהציג את ה"סופר-גלובליים" (superglobals) כגון `$_GET` ו-`$_POST`. ניתן לגשת לסופר-גלובליים מתוך פונקציות מוביליה להשתמש במילת המפתח `global`. תכונה זו נוספת יכולה לאפשר את כיבוי אפשרות ה-INI שנקראת `register_globals`.

היא תכונה ב-PHP הממיר אותה אוטומטית משתני קולט כמו `"foo=bar"`? בכתיבה `register_globals` כשי השתנה PHP הנקרא <http://php.net/?foo=bar> מכיוון שאנשים רבים אינם בודקים משתני קולט נכון, לאפליקציות פותחו חורי אבטחה, מה שההפר את עיקיפת קוד האבטחה והאימונות לקללה למדי.

עם הSUPER-גלובליים החדשניים במקומם, ב-22 באפריל 2002, שוחררה גרסה 4.2.0 של PHP כשי `register_globals` נקבע כברירת מחדל. PHP 4.3.0, גרסה ה-4 PHP המשמעותית האחרון, שוחררה ב-27 בדצמבר 2002. גרסה זו הציגה את ממשק שורת הפקודה (CLI), שכבת קולט/פלט (O/I) מוחדשת של קבצים ורשת (הנקראת streams), ווסףית GD מובנית. למחרת שלרוב התוספות הללו אין השפעה ממשית על משתמשי הקצה, מספר הגרסה הראשוני הולא בשל החדשניים הגדולים בלבית ה-PHP.

PHP 5

זמן קצר לאחר מכן, הביקוש לתוכנות מונחות- עצמים נפוצות יותר גדול מאוד, ואנדי הaga את הרעיון לשכתב את החלק מונחה-העצמים של מנוע ה-Zend. זאב ואנדי כתבו את המסמך "II Zend Engine": סקירה תוכנות ועיצוב" והציגו דיוונים סוערים על עתידה של PHP. למרות שפה הבסיס נשארה זהה, תכונות רבות נוספו, הושמו וושנה עד ש-5 PHP הבשילה. לדוגמה, מרחבי שמות namespaces (namespaces) והורשה מרובה, שהוזכרו במסמך המקורי, מעולם לא נכנסו ל-5 PHP. הורשה מרובה הושמטה לטובת ממשקים (interfaces), ומרחבי שמות הושמו וושמו לחלוטין. תוכלם למצוא רשימה מלאה של תוכנות חדשות בפרק "מה חדש ב-5 PHP".

PHP 5 צפיה לשומר ו אף להגדיל את ההובלה של PHP בשוק פיתוח הרשת. היא לא רק מחוללת מהפכה בתמייה מונחית-העצמים של PHP, אלא היא מכילה גם תכונות חדשות רבות שהופכות אותה לפלטפורמת פיתוח הרשת האולטימטיבית. פונקציונליות ה-XML המשוכבת ב-5 PHP מעמידה אותה בשורה אחת עם טכנולוגיות רשות אחרות בתחוםים מסוימים ועוקפת אותן באחרים, במיחוד הדות להרחבת ה-SimpleXML החדש MySQLi החדש SOAP ה-JSON ועוד. מגוון הרחבות אחרות הם אבני דרך ממשמעותיות בתמייה של PHP בטכנולוגיות נוספות.

קhal_haYud

ספר זה הוא מבוא לתכונות המתקדמות החדשנות ב-5 PHP. הוא נכתב עבור מתכנת PHP שעוסקים את המעבר ל-5 PHP. למחרת שפרק 2, "שפת הליבה של 5 PHP", מכיל מבוא לתחביר 5 PHP, והוא מיועד כרענון למתכנת PHP ולא כמדריך למתכנתים חדשים. עם זאת, מפתחי רשות עם ניסיון בתכונות בשפות אחרות ברמה גבוהה עשויים אכן למצוא שמדריך זה הוא כל מה שהם צריכים כדי להתחיל לעבוד ביעילות עם PHP 5.

סקירה פרקים

פרק 1, "מה חדש ב-5 PHP?": דן בתכונות החדשנות ב-5 PHP. רוב התכונות הללו עוסקות ביכולות מונחית-עצמים חדשות, כולל דוגמאות קטנות לכל תוכנה. הוא גם נותן סקירה של הרחבות החדשנות ב-5 PHP. רוב הנושאים המוזכרים בפרק זה מוסברים בפירוט רב יותר בפרקדים מאוחרים יותר.

פרק 2, "שפת הליבה של 5 PHP": מציג את תחביר ה-5 PHP לקוראים שאינם מכירים את PHP. כל מבני השפה הבסיסיים וסוגי המשתנים מוסברים יחד עם דוגמאות פשוטות כדי לחתך לדורא את אבני הבניין החדשנות לבניית סקריפטים אמיתיים.

פרק 3, "תכונות מונחה עצמים ב-5 PHP": מוביל לחקר את התחביר של 5 PHP, תוך התמקדות בפונקציונליות מונחית-העצמים שלו. פרק זה מכסה יסודות, כגון מאפיינים ומתרומות, ומתקדם לנושאים מסובכים יותר, כגון פולימורפיזם, ממשקים, חריגות (exceptions) ועוד הרבה יותר.

פרק 4, "תכונות מונחה עצמים מתקדם ותבניות עיצוב ב-5 PHP": משתמש בפרק הקודם הקודם כבסיס ומכסה כמה מהתכונות המתקדמות ביותר של מודל העצמים של 5 PHP. לאחר לימוד תכונות אלו, כולל ארבע תבניות עיצוב נפוצות ויכולות הרפלקציה של PHP, תהפכו במהרה לאשי'O.

פרק 5, "יכיז לכתוב אפליקציית אינטרנט עם PHP": כתעת כשתם מכירים את התחביר ותכונות השפה, פרק זה מכניס אתכם לעולם של כתיבת אפליקציות רשות. המחברים מראים לכם יסודות, כמו טיפול בקלט דרך משתני טפסים וטכניקות בטיחות, אך הפרק כולל גם נושאים מתקדמים יותר, כגון ניהול Sessions (sessions) עם עוגיות והרחבת ה-classes של PHP. תמצאו גם כמה טיפים על פרישת קוד המקור עבור אפליקציות הרשות שלכם.

פרק 6, "בסיס נתונים עם 5 PHP": מציג שימוש ב-MySQL, SQLite, Oracle MySQL או מተוך PHP, אך מתקדם בעיקר בפרטים הספציפיים ל-5 PHP של גישה לבסיס נתונים. עבור כל בסיס נתונים, תלמדו על כמה מנוקדות החזק והחולשה שלו, כמו גם על סוג האפליקציות שבהן כל אחד מצויין. כמובן, תלמדו כיצד להתמסך איתם באמצעות הפונקציות המובנות של PHP או באמצעות PEAR DB.

פרק 7, "טיפול בשגיאות": כל הסקריפטים יכולים להעלות שגיאות, אך כמובן שאינכם רוצים שהן יופיעו באתר שלכם ברגע שהאפליקציה נתקעה את שלב הפיתוח. פרק זה עוסק בסוגים שונים של שגיאות קיימות, כיצד לטפל בשגיאות אלו עם PHP וכייז לטפל בשגיאות עם PEAR.

פרק 8, "XML עם PHP": מכיוון שאחת התכונות החדשנות והחשיבות ב-5 PHP היא התמיכה המוחדשת ב-XML, לא ניתן היה לוותר על פרק זה. הוא עוסק באסטרטגיות שונות של ניתוח XML (parsing) והמרת XML לפורמטים אחרים עם XML. מוצגים XML-RPC ו-SOAP כדי להראות לכם כיצד להטמע שירות רשת בשתי הטכניקות.

פרק 9, "הרחבות נפוצות": למרות שאין ספציפיות ל-5 PHP, חמש הרחבות המרכזיות שפרק זה מכסהן הן חשובות מספיק כדי לזרכות במקום בספר זה. החלק הראשון, "קבצים וזרמים" (Files and Streams), מסביר על טיפול בקבצים וזרמי רשות. זרם (stream) אינו אלא דרך לגשת לנוטונים חיצוניים. החלק השני, "ביטויים רגולריים", מסביר את התחברות של מנוע הביטויים הרגולריים (PCRE). ב"טיפול בתאריכים", אנו מסבירים את הפונקציות השונות המשמשות לנתחות ועיצוב מחוזות תאריך ושעה. ב"מניפולציה גרפית GD", אנו מראים לכם דרך שני תרחישים מהחיים האמתיים את הפונקציות הבסיסיות ליצירה ועריכת גרפייה. החלק האחרון, "מחוזות רב-ביטיות ומערכות", מסביר את מערכי התווים השונים והפונקציות לטיפול בהם.

פרק 10, "שימוש ב-PEAR": מציג את PEAR, מאגר הרחבות והאפליקציות של PHP. הפרק מראה כיצד להשתמש ב-PEAR ולהזדקק את החבילות המותקנות מקומית.

פרק 11, "宦宦 PEAR חשובות": נותן סקירה של חברות ה-PEAR החשובות ביותר, יחד עם דוגמאות. החבילות המכוונות כוללות מערכות תבניות (Template Systems), חבילת Auth לביצוע אימרות, טיפול בטפסים וחבילה לפישוט ذיכרון מטמן (caching).

פרק 12, "בנייה רכיבי PEAR": מסביר כיצד ליצור חבילת PEAR משלכם. תקן הקידוד של PEAR ופורמת הגדרת החבילה package.xml יובילו אתכם בדרך להשלמת חבילת ה-PEAR הראשונה שלכם.

פרק 13, "ביצוע המעבר": עוסק בכמה שינויים שאינם תואמים לאחר שהוזגו בין 4 PHP ל-5 PHP. פרק זה אומר לכם באלו דברים עליים לטפל כשאתם גורמים לאפליקציה שלכם לעבד על 5 PHP.

פרק 14, "ביצועים": מראה לכם כיצד לארום לסקריפטים שלכם לעבוד טוב יותר. הפרק מציע טיפים על שימוש סטנדרטי ב-PHP, שימוש בכל עזר חיצוניים (APD ו-Xdebug) למציאת בעיות, ומיצי PHP.

פרק 15, "מבוא לכתיבת הרחבות ל-PHP": מסביר כיצד ל כתוב הרחבה PHP מותאמת אישית משלכם. אנו משתמשים בדוגמה פשוטה כדי להסביר את הדברים החשובים ביותר כמו ניתוח פרמטרים וניהול משאבים.

פרק 16, "כתיבה סקריפטית למעטפת (Shell Scripting)": מראה לכם כיצד ל כתוב סקריפטים למעטפת ב-PHP, מכיוון ש-PHP שימושית יותר מאשר רק אפליקציות רשות. אנו מסבירים את ההבדלים בין קבצי CGI ו-CLI.

הערה על סגנונות קידוד

ישנם כמעט כמספר סגנונות הקידוד כמספר המתכנתים. דוגמאות ה-PHP בספר זה עוקבות אחר תקן הקידוד של PEAR, כאשר הסוגרים המסורתיים הפותחים נמצאים בשורה שמתחת שם הפונקציה.

אודות התוכנה

מצורף בגב הספר קישור מיוחד <https://www.google.com/search?q=%D7%9C-Zend.com>, שם תוכנו להוריד גרסת ניסיון מלאה ל-90 יום של Zend Studio IDE. סביבת הפיתוח של Zend (ZDE) היא כל' נוח המשלב עורך, דיבאגר ומנהל פרויקטים כדי לעזור לכם לפתח, לנהל ולנפות באגים בקוד שלכם.

עדכונים, טעויות והורדות

עדכניםם, תיקוני טעויות (errata) והעתקים של תוכניות הדוגמה המשמשות בספר זה ניתן למצוא בכתובת:
<http://php5powerprogramming.com>

תודות

ספר זה לא היה יכול להכתב ללא המשוב של הסוקרים הטכניים שלו, לפיק, בראצנוו להודות למרכזי בוגר, סטף פוקס, מרטין יאנסן ורוב ריצ'רדס על העורתייהם והמשוב המצוין שלהם. מלבד ארבעת הסוקרים הללו, ישנים עוד כמה אנשים שעוזרו להשב על מספרשאלות במהלך כתיבת הספר, ובאופן ספציפי יותר: קריסטיין סטוקר על העזרה בפרק ה-XML, וז פורלונג ושרה גולמן על המענה לשאלות בנושא שכבת הזרמים (streams layer), פיר-אלן ג'וי על שסיפק תובנות לגבי העבודה הפנימית של ספריית GD, ובאופן כללי יותר קהילת PEAR על תמייניהם ומסירותם למ Lager נדר של רכיבי PEARI שימושים. חלק מהסעיפים בספר זה נכתבו על ידי מחברים-שותפים; גאורג ריכטר תרם את סעיף MySQLi בפרק בסיסי הנתונים, וזאת סורסקי הוסיף את הסעיף על חבילת הביצועים של Zend (Zend's Performance Suite).

בראצנוו להודות גם למאرك ל. טאוב ולצאות המערכת של PTR על הדברים שהם טובים בהם: ארגון, תכנון ושיווק של ספר זה, ודאגה לכך שהכל ישתלב יחד. תודה לגאנט ואלאד על תמיכת עריכת הפיתוח המועילה, ולעורכת הפרויקט שלמו קרייטי הארט, שעזרה לנו לסיים את הספר תחת לחץ והוסיפה לו את הנגינות האחרונות.

תודה!

אנדי, סטיג ודרייק

פרק 1

מה חדש ב-5 ?PHP

"הדרך הטובה ביותר להיות מוכן לעתיד היא להמציא אותו." — ג'ון סקאלין

1.1 מבוא

רק הזמן יגיד אם השחרור של 5 PHP יהיה מוצלח כמו שני קודמי (3 PHP ו-4 PHP). התוכנות והשינויים החדשניים שואפים להיפטר מכל חולשה שהיתה ל-PHP ולהבטיח שהיא תישאר בראש כספת תכונות הרשות הטובה בעולם.

ספר זה מפרט על 5 PHP ועל תכונותיה החדשניות. עם זאת, אם אתם מכירים את 4 PHP ומשתוקקים לדעת מה חדש ב-5 PHP, הפרק הזה הוא בשביבכם. בסיום קריית פרק זה, תלמדו על:

- תכונות השפה החדשניות
- חדשותות בקשר להרחבות PHP
- שינויים ראויים לצוין אחרים בגרסה האחורה של PHP

1.2 תוכנות שפה

1.2.1 מודל מונחה-עצמים חדש

כשزادב סורסקי הוסיף את התחריר מונחה-העצמים בימי 3 PHP, הוא נוסך כ"סוכר תחבירי לגישה לאוספים". מודל ה-Object Oriented (OO) תמרק גם בהורשה ואפשר למחלקה (ולאובייקט) לאגד מתודות ומאפיינים, אך לא הרבה מעבר לכך. כשزادב ואנדי גוטמן שכתבו את מנוע הסקריפטים עבור 4 PHP, זה היה מנוע

חדש לחלוטין; הוא רץ הרבה יותר מהר, היה יציב יותר והציג יותר תכונות. עם זאת, במודל ה-OOP שהוצע לראשונה ב-3 PHP כמעט ולא נגעו.

למרות שלמודל האובייקטים היו מגבלות רציניות, נעשה בו שימוש נרחב ברחבי העולם, לעיתים קרובות באפליקציות PHP גדולות. השימוש המרשימים זהה בפדריגמת ה-OOP עם 4 PHP, למורות חולשותיה, הוביל לכך שהיא הפכה למקד העיקרי של שחרור 5 PHP.

از מה היו חלק מהמגבלות ב-3 PHP ו-4? המגבלה הגדולה ביותר (שהובילה למגבלות נוספות) הייתה העובדה סטנדרטית ההעתקה של אובייקטים הייתה לזו של סוגי פרימיטיבים (native types). כיצד זה השפיע על מפתח ה-PHP? בעת השמת משתנה (המצביע על אובייקט) למשתנה אחר, נוצר עותק של האובייקט. לא רק שזה השפיע על הביצועים, אלא שזה בדרך כלל הוביל להתנהגות מעורפלת ובאגים באפליקציות 4 PHP, מכיוון שפתחים רבים חשבו שני המשתנים יצביעו על אותו אובייקט, מה שלא היה המקרה. המשתנים הצביעו במקום זאת על עותקים נפרדים של אותו אובייקט. שינוי של אחד לא משנה את השני.

לדוגמה, הקוד הבא ב-4 PHP היה מופיע "Andi" במקום "Stig", מכיוון שהאובייקט \$person מועבר לפונקציה ()`changeName`(*value*) לפי ערך (*by*), ולכן הפונקציה עבדת על עותק.

ב-5 PHP, התשתית של מודל האובייקטים שוכנבה לעבודה עם **ידיות אובייקטים** (object handles). אלא אם תשכלו אובייקט במפורש באמצעות מילת המפתח `clone`, לעומת לא תיצור עותקים "מאחורי הקלעים".

1.2.2 תוכנות מונחות- עצמים חדשות

התכונות החדשניות רבות מכדי לפרטן כאן (ראו פרק 3), אך הנה העקרונות שבהן:

- **שינויי גישה (Access Modifiers):** הוספה `protected`, `public` ו-`private` לשילטה על הגישה למתחדות ומאפיינים.
- **שם בניאי אחד:** שימוש ב-`__construct` במקום שם המחלקה.
- **מפרק (Destructor):** תמיינה במתודה `__destruct` שרצה בעת השמדת האובייקט.
- **marshalls (Interfaces):** היכולת של מחלקה למשוח חזזים מרובים.
- **OPERATOR instanceof:** תמיינה ברמת השפה לבדיקת סוג האובייקט.
- **מתודות ומחלקות final:** מניעת דרישת של מתחדות או הורשה של מחלקות.
- **SCPOL אובייקטים מפורשי:** שימוש במילת המפתח `clone`.
- **קבועי מחלקה:** הגדרת קבועים בתוך מחלקה באמצעות `const`.
- **מתודות ומאפיינים סטטיים:** גישה דרך המחלקה עצמה ללא צורך ביצירת מופע (`instance`), שימושי מאוד בתבניות כמו `Singleton`.
- **מחלקות ומתחדות מופשטות (Abstract):** מחלקות שלא ניתן ליצור מהן מופע ומשמשות כבסיס להורשה בלבד.
- **رمזי סוג (Class Type Hints):** אפשרות להגדיר שפרמטר של פונקציה חייב להיות מסווג מחלוקת מסוימת.
- **Dereferencing של אובייקטים:** היכולת לקרוא למתחודה ישירות מאובייקט המוחזר ממתודה אחרת: `obj->method2$()`.
- **Iterators (Iterators):** PHP 5 מאפשרת הן למחלקות PHP והן למחלקות של הרחבות PHP למשוח ממشك `Iterator`. לאחר שימוש ממشك זה, ניתן לעבור על מופעים של המחלקה באמצעות מבנה השפה `foreach ($obj as $value` :
- `PHP`

```
;()obj = new MyIteratorImplementation$  
} } (foreach ($obj as $value
```

```
;"print "$value  
{
```

- לדוגמה מלאה יותר, ראו פרק 4, "תכונות מונחה עצמים מתקדם וtabooites עיצוב ב-5 PHP".
- (__autoload()

פתחים רבים הכותבים אפליקציות מונחה-עצמים יוצרים קובץ מקור PHP אחד לכל הגדרת מחלקה. אחת המטרות המזיקות ביוטר היא הצורך לכתוב רשימה ארוכה של הכלות (inclusions) בתחילת כל סקריפט. ב-5 PHP זה כבר לא נחוץ. ניתן להגדיר פונקציית ()__autoload() שנקראת אוטומטית במקרה שבו מנשים להשתמש במחלקה שטרם הוגדרה, מה שמאפשר להעתין את הקובץ המתאים רגע לפני שהמנוע קורס עם שגיאה.

1.2.3 תכונות שפה חדשות נוספות

- טיפול בחיריגות (Exception Handling): PHP 5 מוסיפה את היכולת להשתמש בפראגמה המוכרת של try/throw/catch. ניתן לזרוק רק אובייקטים הירושים ממחלקה ה-Exception.
- foreach עם הפניות (References): PHP 4 לא ניתן היה לשנות ערכי מערך תוך כדי איטרציה עליו. PHP 5 תומכת בכך על ידי הוספת סימן -& (הפניה) בולולאות ה-foreach, כך שכל שינוי בערך משפיע ישירות על המערך המקורי.
- ערכי ברירת מחדל לפרמטרים המועברים בהפניה: כתוב ניתן להגדיר ערכי ברירת מחדל גם לפרמטרים שמוגדרים עם &.

1.3 שינויים כלליים ב-PHP

1.3.1 XML ושירותי רשת (Web Services)

עדכוני ה-XML ב-5 PHP הם أولי המשמעותיים והמרגשים ביותר.

- התשתיות: ב-4 PHP התמיכה ב-XML התבססה על ספריות שונות (Expat, Sablotron, libxml2), מה שהוביל לתחזקה לקויה וחוסר תאימות. ב-5 PHP, כל הרחבות ה-XML שוכתו לשימוש בערכת הכלים המעלוה libxml2, המציעה שימוש יעיל ועшир בתכונות לתקן ה-XML.
- DOM ו-SAX: מימוש ה-DOM עבר "מתיחת פנים" יסודית והוא כתת תואם לתקן W3C (למשל, שמות פונקציות בשיטת studlyCaps).
- SimpleXML: הרחבה זו מחוללת מהפכה בעבודה עם XML ב-PHP על ידי ייצוג קובץ ה-XML כובייקט PHP טבעי. הגישה לאלמנטים ותכונות הופכת לפשטota ביותר.
- SOAP: ב-5 PHP נוסףת תמיכה רשמית ב-SOAP כהרחבת C מובנית, מה שմשפר משמעותית את הביצועים לעומת מימוש PHP קודמים (כמו ב-PEAR).

1.3.1.7 הרחבת MySQL החדש

בעור 5 PHP, חברת MySQL AB כתבה הרחבה חדשה המאפשרת לנצל את כל היכולות של MySQL 4.1 ומעלה, כולל ממשק מונחה-עצמים, ה策חות מוכנות (prepared statements), חיבור SSL ועוד.

1.3.1.8 הרחבת SQLite

ספריית SQL מובנית שאינה דורשת שרת, מתאימה לאפליקציות קטנות או לסביבות ללא שרת SQL. ב-5 PHP היא מציעה ממשק OO ותמיכה באיטרטורים.

1.4 מאפיינים חדשים נוספים ב-5 PHP

- **מנהל זיכרון חדש:** מנגנון Zend כולל מנגנון זיכרון חדש התומך טוב יותר בסביבות מרובות-תהליכיים (multi-threaded) ומשחרר זיכרון בייעילות רבה יותר בסיום כל בקשה.
- **הפסקת תמייה ב-5 Windows:** עקב חוסר בתמייה טכנית של מיקרוסופט ומגבלות פונקציונליות, PHP 5 אינה תומכת עוד במערכת הפעלה זו.

1.5 סיכום

כמויות השיפורים ב-5 PHP מרשימה ביותר, והפרק הבא יcosa אותם לעומק.

פרק 2

שפת הליבה של 5 PHP

2.1 מבוא

PHP שואלת חלק מהתחביר שלה משפות כמו Perl, C, Shell, Java ואפיו. זהו שפה היברידית הולוקחת את התכונות הטובות ביותר לצירוף שפת סקריפטים עצמאית וקלת לשימוש. פרק זה תלמדו על מבנה השפה, הטעמה ב-HTML, העורות, ניהול משתנים, בסיס נתונים, מבני בקרה ופונקציות.

אם אתם מפתחי 4 PHP מנוסים, יתכן שתרצו לדלג ישירות לפרק הבא העוסק בשינויים המשמעותיים במודל מונחה-העצמים.

2.2 הטעמה בתוך HTML

הדבר הראשון שעיליכם ללמידה על PHP הוא כיצד היא מוטמעת בתוך HTML:

```
HTML
<html>
<head>
<title>סקריפט PHP לדוגמה</title>
<head/>
<body>
הקוד הבא מדפס "
Hello, World"
php?>
;"print "Hello, World
<?
<body/>
<html/>
```

בדוגמה זו, ניתן לראות שקוד ה-PHP שלכם מוטמע בתוך ה-HTML. בכל פעם שמספר ש-PHP מגיע לתג פתיחה שלirk(?) (PHP), הוא מתחילה לבצע את הקוד עד שהוא נתקל בתג הסגירה (?). לאחר מכן

מחליפה את קוד ה-PHP בפלט שלו (אם קיים), בעוד שכל טקסט שאינו PHP (כגון HTML) מועבר כפי שהוא לדף. לכן, הרצתה הסקריפט הנ"ל תוביל לפלאט הבא:

```
HTML
<html>
<head>
<title>סקריפט PHP לדוגמה</title>
<head/>
<body>
הקוד הבא מדפסו
:"Hello, World"
Hello, World
<body/>
<html>
```

טיפ: ניתן להשתמש גם בתג פתיחה מקוצר `<?`. עם זאת, שימוש זה אינו מומלץ מכיוון שלא בכל השרטים אפשרות זו מופעלת (ניתן להפעילה דרך הגדרת ה-`short_open_tag` בקובץ `php.ini`).

2.3 הערות

כתבת הערות בקוד חשוב מאוד כדי להסביר מה הקוד עושים. PHP תומכת במספר סגנונות של הערות:

- **הערות שורה אחת:** סגנון C++ (`//`) או סגנון Shell (.#)
- **הערות מרובות שורות:** סגנון C (* ... */).

דוגמאות:

```
PHP
php?>
// זהוי הערה בשורה אחת
# גם זו הערה בשורה אחת
/* זהוי הערה
משתרעת על פני מספר שורות */
?<
```

2.4 משתנים

ב-PHP, כל שמות המשתנים מתחילה בסימן דולר (\$). שם המשתנה (לאחר ה-\$) חייב להתחיל באות או בקו תחתון (_), יכול להכיל אותיות, מספרים או קווים תחתוניים. PHP היא שפה **Case-Sensitive** (רגישה לרישיות), כלומר המשתנה `$foo` ו-`$FOO` הם משתנים שונים.

2.4.2 ניהול משתנים

ישנם שלושה מבני שפה (Language Constructs) בסיסיים לניהול משתנים: `isset()`, `empty()` ו-`unset()`.

(empty) 2.4.2.1

הfonקציה `empty()` בודקת אם משתנה הוגדר והוא איננו `NULL`. הfonקציה `empty()` בודקת אם הערך של המשתנה נחשב ל"ריק" (כמו מחוץ ריקה, `0`, `false`, או `NULL`).

ניתן להשתמש ב-`empty()` גם על איברי מערך או מאפייני אובייקט:

PHP

```
} ((["if (isset($arr["offset  
...  
{  
} ((if (isset($obj->property  
...  
{
```

שימוש לב שבשתי הדוגמאות לא היינו צריכים לבדוק אם `$arr` או `$obj` מוגדרים בעצמם; `empty()` תחזיר `false` אוטומטית אם הם אינם קיימים.

היא היחידה שמקבלת מספר בלתי מוגבל של פרמטרים:

```
;isset($var1, $var2, $var3
```

היא תחזיר `true` רק אם כל המשתנים הוגדרו. זה שימושי מאוד לבדיקת קלט שנשלח מהלוקו.

(unset) 2.4.2.2

הfonקציה `unset()` " לבטל" הגדרה של משתנה שהוגדר בעבר, ומשחררת את הזיכרון ששימש אותו (אם אין משתנים אחרים שמצביעים על ערכו). קריאה ל-`unset()` על משתנה שעבר `empty()` תחזיר `false`.

PHP

```
,"name = "John Doe$  
;(unset($name  
} ((if (isset($name  
// זה לא יודפס  
';print '$name is set  
{
```

(empty) 2.4.2.3

ניתן להשתמש ב-`empty()` כדי לבדוק אם משתנה לא הוצאה או שערכו הוא `false`. מבנה שפה זה משמש בדרך כלל לבדיקה אם משתנה טופס לא נשלח או אינו מכיל נתונים. בעת בדיקת ערך האמת של משתנה,

ערך מומר תחילה לboolani (Boolean) לפי הכללים המפורטים בסעיף הבא, ולאחר מכן הוא נבדק אם הוא אמת או שקר (true/false).

לדוגמה:

```
PHP
}
} ((if) (empty($name
; 'print 'Error: Forgot to specify a value for $name
{
}
}

קוד זה ידפיס הודעה שגיאה אם $name אינו מכיל ערך שמתפרש כ-true.
```

2.4.3 סופר-גלובליים (Superglobals)

ככל PHP אינה תומכת במשתנים גלובליים (משתנים שניית לגשת אליהם אוטומטית מכל טווח הכרזה). עם זאת, משתנים פנימיים מיוחדים מסוימים מותנהגים כמו משתנים גלובליים בדומה לשפות אחרות. משתנים אלו נקראים **סופר-גלובליים** והם מוגדרים מראש על ידי PHP לשימושם. כמה דוגמאות לסופר-גלובליים הן:

- **\$_GET**: מעיר הכלול את כל משתני ה-GET ש-PHP קיבל מדפסן הלוקה.
- **\$_POST**: מעיר הכלול את כל משתני ה-POST ש-PHP קיבל מדפסן הלוקה.
- **\$_COOKIE**: מעיר הכלול את כל העוגיות (cookies) ש-PHP קיבל מדפסן הלוקה.
- **\$_ENV**: מעיר המכיל את משתני הסביבה.
- **\$_SERVER**: מעיר המכיל את ערכי המשתנים של שרת האינטרנט.

ניתן לגשת למשתנים אלו בכל מקום בסקריפט – בין אם בתחום פונקציה, מתודה או בטווח הגלובלי – ללא צורך להשתמש במילת המפתח **global**.

2.5 סוגי נתונים בסיסיים

ב-PHP קיימים שמונה סוג נתונים שונים: חמישה מהם סקלאריים (scalar) ולכל אחד משלושת הנוגרים ייחודיות משלו. המשתנים יכולים להכיל ערכים מכל סוג הנתונים הללו מבלי להציגו במפורש על סוגם. המשתנה "מתנהג" בהתאם לסוג הנתונים שהוא מכיל.

2.5.1 מספרים שלמים (Integers)

מספרים שלמים הם מספרים שלמים ללא שבר. ברוב המכוונות הנפוצות (כמו Intel Pentium), מדובר במספר שלם של 32 סיביות עם טווח שבין -2,147,483,648 ל-2,147,483,647.

ניתן לכתוב מספרים שלמים בשיטה עשרונית, הקסדצימלית (עם הקידומת 0x) או אוקטליית (עם הקידומת 0).

2.5.2 מספרים בנקודה צפה (Floating-Point Numbers)

אלו הם מספרים ממשיים הקיימים נקודה עשרונית. ברוב הפלטפורמות, גודל הטיפוס הוא 8 בתים וטווחו הוא בערך בין 1.8×10^{-308} ל- 3.4×10^{308} .

2.5.3 מחרוזות (Strings)

מחרוזות ב-PHP הן רצף של תוים מסוימים פנימית בתו **string**. בניית לשפת C, PHP זוכרת את אורך המחרוזת פנימית, מה שמאפשר טיפול קל בנתונים בינאריים. אורך המחרוזת המקסימלי תלוי בפלטפורמה. אך ניתן לצפות לתמיהה של לפחות 2GB.

ניתן להגדיר מחרוזות בשלוש דרכים:

1. **מירכאות כפולות ("")**: תומכות בתווים מיlot (כמו \ch לשורה חדשה) ובשילוב משתנים ישירות בתווים המחרוזת (למשל: "The result is \$result").
2. **מירכאות יחידות ()**: פשוטות יותר; אין מבצעות החלפת משתנים ותומכות רק בתווים מיlot בסיסיים ביותר (כמו \W או \n).
3. **Here-Docs**: מאפשרים להטמייע קטעי טקסט גדולים ללא צורך במילוט מירכאות. הם מתחילה ב-<><> ולאחריו מזהה (identifier) ומסתיימים באותו מזהה בתחילת שורה.

גישה לתווים מחרוזת

ניתן לגשת לתווים בודדים במחרוזת באמצעות התחביר `str{offset}`.

לדוגמה:

```
PHP
      ;"str = "A$"
      ;"str{2} = "d$"
      ;"str{1} = "n$"
      // שרשור "str = $str . "i$"
      Andi // ידפיס ;print $str
```

הערה: ב-4 PHP השתמשו בסוגרים מרובעים [] לגישה למחרוזות. ב-5 PHP מומלץ להשתמש בסוגרים מסולסלים {} כדי להבדיל בין מחרוזת למערך.

2.5.4 בוליאים (Booleans)

ערך בוליани יכול להיות `true` (אמת) או `false` (שקר). PHP ממיר סוג נתוני לבוליאים באופן אוטומטי בהתאם (כמו `if`). למשל, המספר 0 יומר ל-`false`, בעוד מספר אחר יומר בדרך כלל ל-`true`.

טבלת ערכי אמת (Boolean conversion)

להלן האופן שבו PHP ממיר סוג נתוני שונים לערך בוליани:

סוג נתון	ערך שקר (False)	ערך אמת (True)
מספרשלם (Integer)	0	כל ערך שאינו אפס
מספר צפה (Float)	0.0	כל ערך שאינו אפס

כל שאר המחרוזות	מחרוזת ריקה (""), המחרוזת אפס ("0")	מחרוזות (Strings)
אף פעם לא אמת	תמיד שקר	Null
אם הוא מכיל לפחות איבר אחד	אם הוא אינו מכיל איברים	מערך (Array)
תמיד אמת	אף פעם לא שקר	אובייקט (Object)
תמיד אמת	אף פעם לא שקר	משאב (Resource)

2.5.7 מערכים (Arrays)

מערכות ב-PHP הם למעשה מפות מסודרות (Ordered Maps) שמקשרות בין מפתחות (keys) לערךים. המפתח יכול להיות מספרשלם או מחרוזת.

2.5.7.1 ייצרת מערכים

ניתן ליצור מערך באמצעות המבנה `array()`. המפתח הוא אופציונלי; אם הוא לא מצוין, PHP מקצה אוטומטית מפתח מספרי הגדל ב-1 מההמפתח הגבוה ביותר שהוא עד כה (מוחילה ב-0).

דוגמאות:

- `array(0 => 1, 1 => 2, 2 => 3)` – זהה ל-`array(1, 2, 3)`
- `array("name" => "John", "age" => 28)` – מערך אוטומטי עם מפתחות מחרוזת.
- `array(1 => "ONE", 2 => "TWO", 3 => "THREE")` – יהי שווה ל-`array("ONE", "TWO", "THREE")`.

2.5.7.2 גישה לאיירி מערך

הגישה נעשית באמצעות התחביר `[$key]arr`. שימושם במחרוזת קבועה כמפתח, חובה להשתמש במירכאות: `["key"]arr`.

2.5.7.3 שינוי ויצירת איברים

ניתן להוסיף איברים לערך קיים או ליצור מערך חדש שלב אחר שלב:

```
PHP
$arr2[0] = 1$ // מוסיף איבר במפתח הפנוי הבא (0)
$arr2[1] = 2$ // מוסיף איבר במפתח 1
```

2.5.7.6 מעבר על מערכים (Traversing) באמצעות `foreach`

זהה הדרך האלגנטית ביותר לעبور על מערך. התחביר הוא:

```
(foreach($array as [$key =>] [&] $value
```

השימוש ב-& לפני ה-\$value מאפשר לשנות את ערכי המערך המקורי תוך כדי הלולאה (תמונה חדשה ב-5 PHP).

דוגמה לשימוש ב-& לשינוי ערכים:

PHP

```
} (foreach ($people as &$person
} (if ($person["age"] >= 35
;"$person["age group"] = "Old$"
} else {
;"$person["age group"] = "Young$"
{ {
```

בדוגמה זו, המערך המקורי \$people יעדכן ויתווסף לו המפתח "age group" לכל אדם.

2.5.7.7 שימוש ב-`list()` ו-`each()`

שיטה ישנה יותר (פחות מומלצת מ-`foreach`) למעבר על מערך היא שילוב של `each()` ו-`list()`.

- `reset()`: מאפס את המצביע הפנימי של המערך לתחילה.
- `each()`: מחזיר את צמד המפתח/ערך הנוכחי ומגדם את המצביע.
- `list()`: מבנה המאפשר להציג מספר ערכים מערך מסוים מעתנים בפקודה אחת:
`;list($name, $age) = $person_data`

2.5.8 קבועים (Constants)

קבועים הם שמות לערכים פשוטים שלא משתנים במהלך הריצת הסקריפט. ביגוד למשתנים, הם **אינם** מתחדלים בסימן דולר (\$) והם נגישים מכל מקום בתוכנית (Global Scope).

הגדרת קבוע:

```
;(define("CONSTANT_NAME", value
```

- נהוג לכתוב קבועים באותיות גדולות (UPPERCASE).
- קבועים יכולים להכיל רק ערכים סקלריים (לא מערכים או אובייקטים).

דוגמה:

PHP

```
;(define("MY_OK", 0
;(define("MY_ERROR", 1
```

```
 } (if ($error_code == MY_ERROR  
;"print("There was an error\n{
```

2.6 אופרטורים (Operators)

PHP מכילה שלושה סוגים אופרטורים: אופרטורים אונאריים (על אופרנד אחד), ביןאריים (על שניים) ואופרטור טרנארי אחד.

PHP יכולה לבצע פעולות ביןאריות רק על שני אופרנדים מאותו סוג. אם הם מסוגים שונים, PHP מבצעת המרה אוטומטית לפי הכללים הבאים:

- **בוליניים, Null ומשאבים:** מתנהגים כשלמים (False=0, True=1, Null=0).
- **שלם ונקודה צפה:** השלם מומר לנקודה צפה.
- **שלם ומחרוזת:** המחרוזת מומרת למספר. אם המחרוזת מייצגת מספר ממשי, גם השלם יומר ממשי.

2.6.1 אופרטורים ביןאריים

1. **אופרטורים חשבוניים:** + (חיבור), - (חיסור), * (כפל), / (חילוק) ו-% (מודולו - שארית החילוק). כולם עובדים על ערכים מספריים.
2. **אופרטור השרשור (.):** לחבר שתי מחרוזות לאחת. אם אחד האופרנדים אינו מחרוזת, הוא יומר לאחת.
דוגמה: .2000;print "The year is " . "The year is "

2.6.2 אופרטורי השמה (Assignment)

האופרטור הבסיסי הוא =. בנווף, קיימים אופרטורים משולבים המבצעים פעולה והשמה בצעד אחד:

```
.counter = $counter + 2$ counter += 2$
```

אופרטורים נפוצים: +=, -=, *=, /=, .=.

השמה בהפניה (&): יוצרת כינוי (Alias) למשתנה אחר. שינוי באחד ישפיע על השני.

PHP

```
; "name = "Judy$  
; name_alias =& $name$  
; "name_alias = "Jonathan$  
Jonathan // ידפיס $name
```

2.6.3 אופרטורי השוואة

אפשררים לקבוע את היחס בין שני אופרנדים ומחזירים ערך בולוגי.

- `==` (שווה ל-): בודק שווין לאחר המרת סוגים במידה הצורך.
- `===` (זהה ל-): בודק שווין **כולל סוג הנתונים**. ללא המרת אוטומטית.
- `!=` (לא שווה), `!=` (לא זהה), `>` (קטן מ-), `<` (גדול מ-), `=<` (אנו מוציא).

2.6.4 אופרטורים לוגיים

משמשים לחברת תנאים: `&&` או `and` (וגם), `||` או `or` (או), `xor` (או מוציא).

הערכתה בקיצור דרך (Short-Circuit Evaluation): PHP חכמה מספיק כדי לעצור הערכה אם התוצאה כבר ידועה. למשל בתנאי `1 && 0`, המנווע יראה את ה-0 (שקר) ויעצור מיד כי התוצאה חייבת להיות שקר, מבלי לבדוק את הצד השני.

2.6.5 אופרטורים ברמת הביט (Bitwise)

פועלים על הייצוג הבינארי של המספרים: `<<` (AND), `|` (OR), `^` (XOR), `~` (NOT) & (&) (הזהה ימינה), `>>` (הזהה שמאליה).

2.6.6 אופרטורי קידום והפחתה (Increment/Decrement)

אופרטורים אלו ייחודיים כי הם משנים את ערך המשתנה עצמו.

- `++$var`: מגדם ב-1 ואז מחזיר את ערך הערך החדש.
- `$var++`:מחזיר את הערך הנוכחי ואז מגדם ב-1.

דוגמה להמחשה:

```
PHP
;num = 5$ // יופיע 5, אבל הערך של num יփוך ל-6
++echo $num // הערך יփוך ל-7 וידפס 7
echo ++$num
```

2.6.8.1 קידום מחרוזות

מחרוזות שאין נומריות (מספריות) מקודמות בדומה לשפת Perl: אם האות الأخيرة היא אלף-נומרית, היא מקודמת ב-1. אם היא הייתה 'ז', היא תהפוך ל-'א' והטו שלפנייה יקודם גם הוא.

- **הערה:** לא ניתן לבצע פעולה הפחתה (`--`) על מחרוזות שאין נומריות.

2.6.9 אופרטורי המרת (Cast Operators)

ניתן לכפות המרת סוג נתונים (Casting) בדומה לשפת C. ההמרה משנה את סוג הערך באותו רגע, אך לא את סוג המשתנה המקורי.

- `(int)`, `(integer)` — המרת מספרשלם.
- `(string)` — המרת למחרוזת.
- `(bool)`, `(boolean)` — המרת לבוליאני.
- `(array)` — המרת למערך.

2.6.10 אופרטור השתקה (Silence Operator)

הסימן `@` לפניו ביטוי מסוים הודיעות שגיאה שאפשרות להיווצר במהלך המהלך.

2.6.11 האופרטור הטרנארי (Ternary Operator)

אופרטור אלגנטי המבצע בדיקה והחזרת ערך בשורה אחת: `.expr2 : expr1 ? truth_expr`.

אם התנאי אמת, יוחזר `expr1`, אחרת יוחזר `expr2`.

PHP

```
;'message = isset($a) ? '$a is set' : '$a is not set$
```

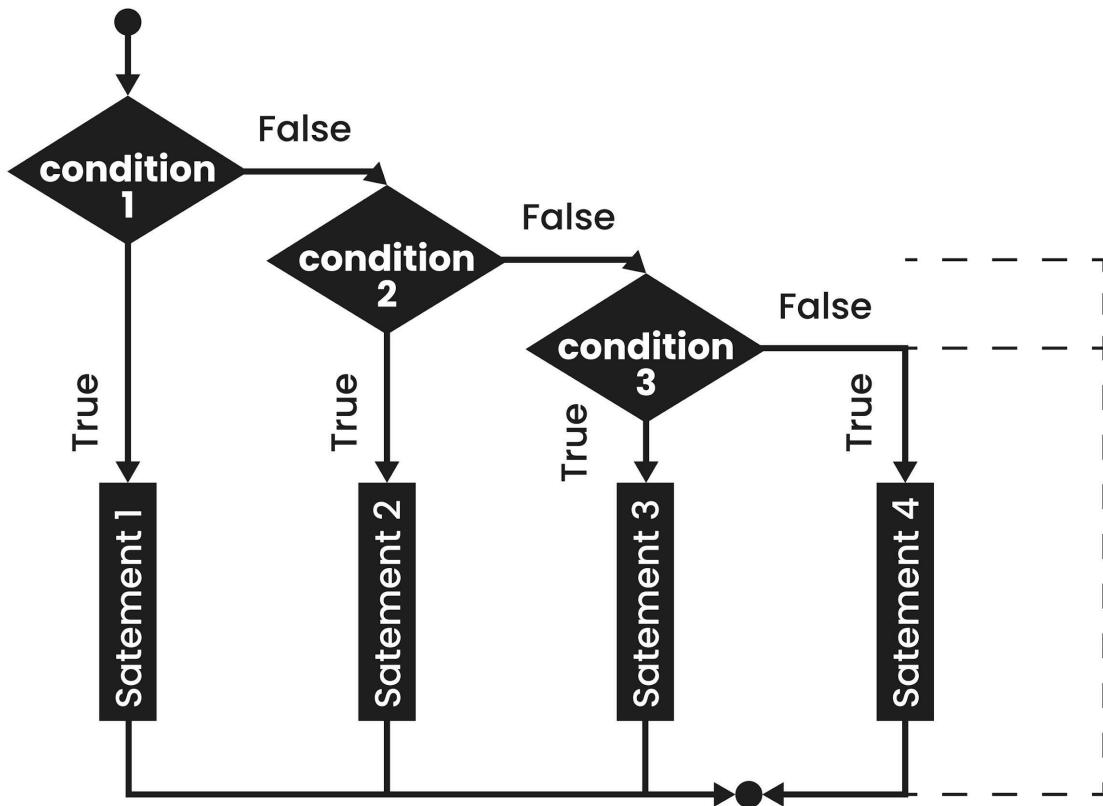
2.7 מבני בקרה (Control Structures)

מבנה הבקרה מחולקים לשתי קבוצות: מבני תנאי (משפיעים על כיוון התוכנית) ומבנה לולאה (חוזרים על קוד).

2.7.1 מבני תנאי

1. **if Statements**: המבנה הנפוץ ביותר. אם "ביטוי האמת" הוא `true`, הקוד יבוצע. ניתן להוסיף `else` במקרה שהתנאי לא מתקיים, ו-`elseif` לבדיקות נוספות.

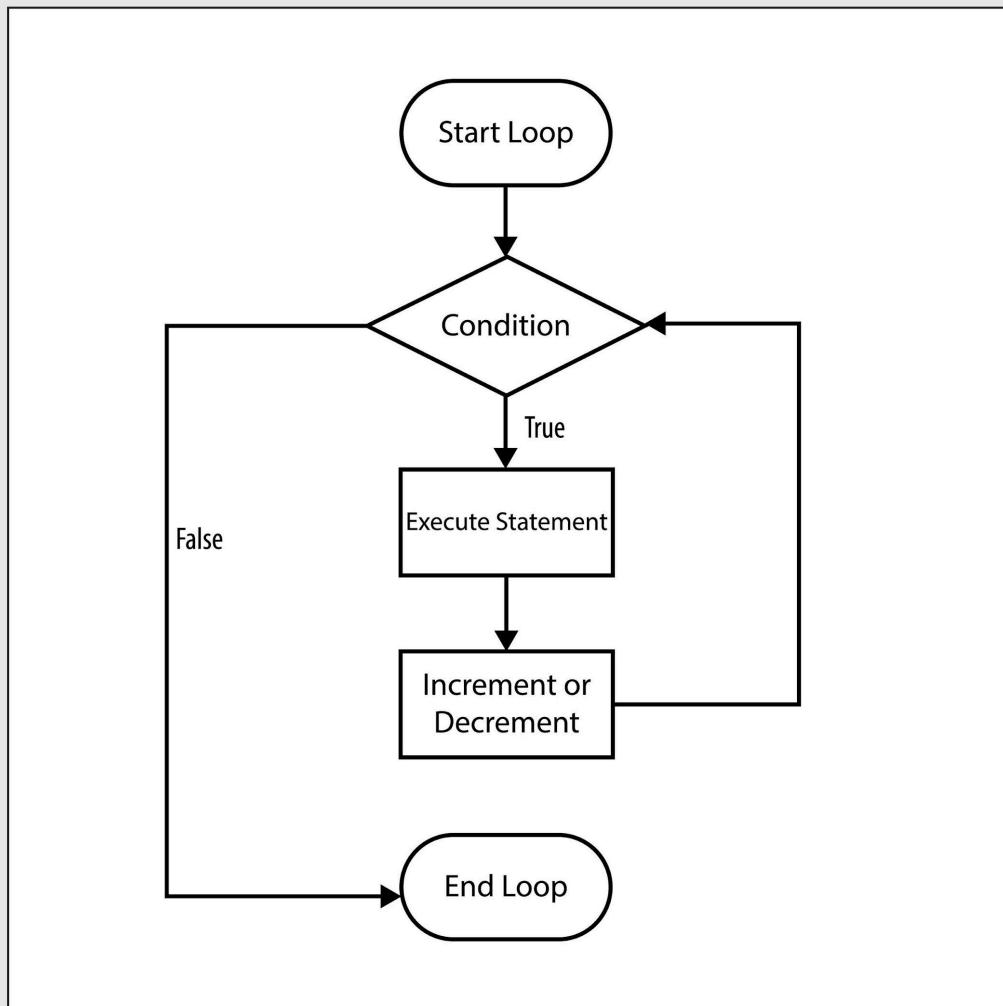
IF-ELSE-IF STATEMENT



.2 Shutterstock .3
.2 switch Statements : מחליף סדרת if/elseif/arocha בצורה אלגנטית. הוא משווה ביטוי מול מספר מקרים (case). חשוב להשתמש ב-break כדי לעצור את הביצוע לאחר שנמצאה התאמה, אחרת PHP תמשיך לבצע את המקרים הבאים. במקרה default יבוצע אם לא נמצאה אף התאמה.

2.7.2 מבני לולאה

1. while loops : הלולאה פשוטה ביותר. התנאי נבדק **בתחילת כל איטרציה**. אם הוא אמת, הלולאה ממשיכה.



.2

Shutterstock .3

Explore .4

:continue ו-break .2

`break` עוצר את הלולאה לחלוטין ויוצא ממנה.

`continue` עוצר את האיטרציה הנוכחיות וועבר מיד לבדיקה התנאי של האיטרציה הבאה.

ניתן להוסיף מספר (למשל 2 `break`) כדי לצאת ממספר לולאות מקוננות.

3. While Loops דומה ל-`for`, אך התנאי נבדק **בסוף** האיטרציה. המשמעות היא שהלולאה תרוץ לפחות פעם אחת תמיד.

4. for Loops לולאות בסגנון שפת C המקבילות שלושה ארגומנטים:

`(for (start_expressions; truth_expressions; increment_expressions`

ביטוי התחלה: מבוצע פעם אחת בלבד בכניסה לולאה.

ביטוי אמת: נבדק לפני כל סיבוב.

ביטוי קידום: מבוצע בסוף כל סיבוב.

דוגמאות לולאות `for`:

```
PHP
}
} ) ++for ($i = 0; $i < 10; $i
,"print "The square of $i is " . $i*$i . "\n"
{
```

כמו בשפת C, ניתן לספק יותר מביטוי אחד לכל אחד משלשות הארגומנטים על ידי הפרדתם בפסיקים. הערך של כל ארגומנט יהיה הערך של הביטוי הימני ביותר. ניתן גם להשאיר ארגומנטים ריקים; ארגומנט ריק ייחסב כ-`true`. למשל, `(;;)` יוצר לולאה אינסופית.

טיפ ביצועים: PHP אינה מבצעת אופטימיזציה לערכיהם קבועים בתוך הלולאה. למשל, בביטוי `($array); $i = 0; $i <= count($array);`, הפונקציה `count` תרוץ בכל סיבוב. עדיף לחשב את האורך מראש במשתנה מחוץ ללולאה.

2.7.3 מבני בקרה להכללת קוד (Inclusion)

מבנהו אלו חיוניים לארגון הקוד וליצירת יחידות בנייה שנייה לשימוש חוזר.

2.7.3.1 `include`, `require` ו-

פקודת `include` מאפשרת לפצל את קוד המקור למספר קבצים. בעת הביצוע, PHP קוראת את הקובץ, מקומפלט ומריצה אותו בתוך אותו "טווח משתנים" (scope) של הסקריפט הקורא.

- **include**: אם הקובץ לא נמצא, תונפק אזהרה (Warning) אך הסקריפט ימשיך לירוץ.
- **require**: אם הקובץ לא נמצא, תיווצר שגיאה קריטית (Fatal Error) והסקריפט ייעצר.
- **include_once / require_once**: הגרסאות המומלצות ביותר. הן מוגדרות שהקובץ ייכלל פעם אחת בלבד, גם אם נקראו שוב בהמשך. זה מנע הגדרות כפולות של פונקציות ומחלקות.

ניתן להשתמש בנתיבים ייחודיים או מוחלטים. מפתחים רבים משתמשים בנתיב מוחלט דרך `$_SERVER["DOCUMENT_ROOT"]`.

2.7.3.2 `()eval`

פונקציה זו דומה ל-`include`, אך במקום לקבל קוד מקובץ, היא מקבלת קוד PHP בתוך מחזוזת ומריצה אותו.

אזהרה: לעיתים אל תעבירו קולט משתמש `eval()`, שכן הדבר מהוות פרצת אבטחה חמורה המאפשרת הרצת קוד זדוני.

8 פונקציות (Functions)

פונקציה ב-PHP יכולה להיות מובנית (של השפה) או מוגדרת על ידי המשתמש. הקראיה לשתיهن זהה: `func(...args)`.

2.8.1 פונקציות מוגדרות משתמש

הגדרה בסיסית:

```
PHP
{
    } (function function_name ($arg1, $arg2
        // קוד
        // החזרת ערך ויציאה מהפונקציה
        return $value
    }
}
```

2.8.2 טווח משתנים בפונקציה (Function Scope)

כל פונקציה יש "ארגז חול" משלها. משתנים שהוגדרו מחוץ לפונקציה אינם נגישים בתוכה כברירת מחדל, וממשנים שנוצרים בתוכה "מתים" כשהיא מסתiya.

- **גישה למשתנים גלובליים:** ניתן להשתמש במערך המובנה `[$GLOBALS["var_name"]]` כדי לגשת למשתנה מחוץ לפונקציה.
- **מילת המפתח global:** קיימת דרך להציג על משתנה כגלובלי בתוך הפונקציה (`global $var;`), אך המחברים אינם מלאיצים להשתמש בה עקב בעיות תחזוקה והتنהגות לא צפיה עם הפניות.

2.8.3 החזרת ערכים לפי ערך (By Value)

כברירת מחדל, פקודת `return`מחזירה עותק של הערך. אם תשנו את הערך המוחזר מחוץ לפונקציה, המשתנה המקורי שנמצא בתוך הפונקציה (או במערך הגלובלי שהוחזר) לא ישתנה.

2.8.4 החזרת ערכים לפי הפניה (Returning by Reference)

PHP מאפשרת להחזיר משתנים לפי הפניה. המשמעות היא שאינכם מוחזרים עותק של הערך, אלא את "כתובת" המשתנה עצמו. הדבר מאפשר לשנות את המשתנה המקורי מתוך הטווח שקרה לפונקציה.

כדי להשתמש בכך, יש להוסיף את הסימן & לפני שם הפונקציה בהגדרה, וגם בעת הקראיה לה (בזמן ההשמה למשנה).

```
PHP
{
    } (function &get_global_variable($name
        ;[return $GLOBALS[$name
    }

    ;num = 10$ // השמה לפי הפניה
    ("value =& get_global_variable("num$ // העתק את הערך במקומות אליו, מה שעלול ליצור באגים קשים לאיתור.
    ;value = 20$ // יופיע 20
    20 // ;print $num
}
```

זהירות: השתמשו בטכניקה זו רק כנדרש. שכחה של סימן ה-& בזמן ההשמה תגרום ל-PHP להעתיק את הערך במקומות להפנות אליו, מה שעלול ליצור באגים קשים לאיתור.

2.8.5 הצהרה על פרמטרים של פונקציה

קיימות שתי דרכי להעביר ארגומנטים (פרמטרים) לפונקציה:

1. **העברה לפי ערך (By-Value):** זהה הדרך הנפוצה ביותר. הפונקציה מקבלת עותק של הערך, וכל שינוי בו בתוך הפונקציה אינם משפיע על המשתנה המקורי בוחוץ.
2. **העברה לפי הפניה (By-Reference):** הפונקציה מקבלת "קיצור דרך" למשתנה המקורי. כל שינוי בפרמטר בתוך הפונקציה ישנה את המשתנה שנשלח אליו בטוחה החיצוני.
דוגמה: `(function square($num){$num += $increment});` תגרום למשתנה `$num` לשולח אליה להפוך לערך הריבועי שלו.

2.8.5.3 פרמטרים כברירת מחדל (Default Parameters)

בדומה ל-`++C`, ניתן לקבוע ערך קבוע לפרמטר אם הוא לא נשלח בעת הקראיה לפונקציה. הערך חיב להיות קבוע (מספר, מחזור או מערך קבוע).

```
PHP
} (function increment(&$num, $increment = 1
                      ;$num += $increment$ {
{
}
```

- **כלל חשוב:** אם החלטתם להשמיט ארגומנט עם ערך ברירת מחדל בעת הקראיה, עליכם להשמיט גם את כל הארגומנטים האחרים. בהתאם, בהגדרת הפונקציה, כל הפרמטרים שאחרי פרמטר עם ברירת מחדל חייבים להיות גם הם בעלי ברירת מחדל.

2.8.6 משתנים סטטיים (Static Variables)

PHP תומכת במשתנים מקומיים מסוג `static`. אלו משתנים ששומרים על מצבם בין קראיות שונות לפונקציה, אך הם נגישים רק מתוך הפונקציה שבה הם הוגדרו. האתחול שלהם מתבצע רק בפעם הראשונה שבה מגיעים להצהרה עלייהם.

זהו כלי מעולה להרצת קוד הגדרות ראשוני (Initialization) שצריך ל��ורט רק פעם אחת:

```
PHP
} ()function do_something
{
    static $first_time = true
}
} (if ($first_time
      // קוד שיוציא רק בקריאה הראשונה לפונקציה
      ;$first_time = false$ {
{
    // הלוגיקה העיקרית שתורצ בקריאה
}
```

פרק זה כיסה את תוכנות היסוד של PHP: משתנים, מבני בקרה ופונקציות. ידע זה מספק לכם את כל הכלים הדרושים כדי להיות פרודוקטיביים בכתיבת קוד פונקציוני.

הפרק הבא יעסוק בהמשך של PHP: המודל מונחה העצמים (Object-Oriented).

פרק 3

3.1 מבוא

ב-5 PHP, מודל האובייקטים עוצב מחדש לחלוטן. בנוסף לארסאות קודמות, 5 PHP מציעה סט עשיר של תכונות המאפשרות בניית אפליקציות מורכבות ומסודרות. הפרק יcosa את יסודות המודל, יצירת אובייקטים, בנאים (Constructors), הרשות גישה (public, private, protected) וירושה.

3.2 אובייקטים (Objects)

ההבדל העיקרי בין תוכנות מונחה עצמים (OOP) לתוכנות פונקציוני היא שנתונים וקוד ארזים יחד לשות אחת הנקראת אובייקט.

- **מאפיינים (Properties):** המשתנים השייכים לאובייקט (הנתונים שלו).
- **מתודות (Methods):** הפונקציות שהאובייקט תומך בהן.
- **מחלקה (Class):** ה"תבנית" (Template) שגדירה אילו מתודות ומאפיינים יהיו לאובייקטים מסווג זה.
- **מוףע (Instance):** אובייקט ספציפי שנוצר מתוך המחלקה.

דוגמא למחלקה Person:

אם יש לנו שני אנשים באפליקציה, "ג'די" ו"ג'ו", נוצר שני מופעים נפרדים של המחלקה Person. לכל אחד מהם יהיה משתנה \$name.

```
PHP
}
} class Person
;private $name

} (function setName($name
;this->name = $name$
{
}

} ()function getName
;return $this->name
{
;

;()judy = new Person$
;("judy->setName("Judy$
```

```
;()joe = new Person$  
;("joe->setName("Joe"$  
  
print $judy->getName(); // Judy
```

3.4 מילת המפתח new ובנאים (Constructors)

כדי ליצור אובייקט חדש, משתמשים במילת המפתח new.

מיד לאחר יצירת האובייקט, PHP קוראת אוטומטית למетодה מיוחדת שנקראת הבניי (`construct`). הבניי משמש בדרך כלל לאותחול מאפיינים.

PHP

```
} class Person  
;private $name  
  
{(function __construct($name  
;this->name = $name$  
{  
;  
;"judy = new Person("Judy$
```

3.5 מפרקם (Destructors)

הפור מהבניין, המפרק (`destruct`) נקרא כאשר האובייקט נהרס (למשל כאשר יותר הפניות אליו או כשהסקריפט מסתיים). זהו מקום טוב לסגור חיבורים למסד נתונים או לכתוב לוג.

3.6 גישה באמצעות המשתנה this\$

בתוך מתודה של אובייקט, המשתנה המוחדר `this$` מייצג את האובייקט הנוכחי. דרכו ניתן לגשת למאפיינים ולמתודות של אותו מופע (למשל: `($this->name`).

3.6.1 הרשות גישה: Public, Protected, Private

זהו אחד העקרונות החשובים ב-OP (אנקפסולציה):

1. **public**: ניתן לגשת למאפיין/מתודה מכל מקום (גם מחוץ למחלקה).
2. **protected**: הגישה מותרת רק מתוך המחלקה עצמה או ממחליות שיורשות ממנה.
3. **private**: הגישה מותרת רק מתוך המחלקה שבה הוגדר המאפיין.

3.6.3 מאפיינים סטטיים (Static Properties)

בעוד שמאפיין רגיל שיר לモפע ספציפי (לאובייקט), מאפיין סטטי שיר למחלקה עצמה. יש רק עותק אחד שלו לכל האובייקטים מאותו סוג.

ניגשים אליו באמצעות השם של המחלקה והסימן ::. בטור המחלקה, ניתן להשתמש ב-self::.

דוגמה לשימוש (מונה מזהה ייחודי):

PHP

```
} class MyUniquedClass  
;static $idCounter = 0  
;public $uniqued
```

```
} ()function __construct  
;++self::$idCounter  
;this->uniqued = self::$idCounter$  
{  
}
```

בכל פעם שניצור אובייקט חדש, ה-\$idCounter הכללי יעלה ב-1, וכל אובייקט יקבל מספר רץ ייחודי.

3.6.4 מethodות סטטיות (Static Methods)

מетодות סטטיות שייכות למחלקה עצמה ולא למופע (אובייקט) ספציפי.

- איך קוראים לה? באמצעות ClassName::methodName().
- גישה לנוטונים: מכיוון שהן לא קשורות לאובייקט, המשתנה \$this לא זמין בתוכן. כדי לגשת למאפיינים או מетодות סטטיות אחרות מוצאה מחלקה, משתמשים ב-self::.
- שימוש: מתאים לפונקציות עזר (Utility) שאינן דורשות מידע השמור באובייקט ספציפי.

3.7 קבועים במחלקה (Class Constants)

ב-5 PHP ניתן להגדיר קבועים בתחום מחלקה באמצעות מילת המפתח const.

- בדומה למאפיינים סטטיים, הם שייכים למחלקה.
- הם תמיד רגישים לריביות (Case-sensitive).
- לא ניתן לשנות אותם לאחר ההגדרה. נהוג לכתוב אותם באותיות גדולות (למשל: const RED = "Red").

3.8 שכפול אובייקטים (Cloning)

ב-5 PHP, המשתנה אובייקט הוא למעשה "מזהה" (Handle) לאובייקט בזיכרון.

- העתקה רגילה: אם תכתבו \$obj1 = \$obj2, שניהם יצביעו על אותו אובייקט. שינוי אחד ישפיע על השני.
- שכפול אמיתי: כדי ליצור עותק נפרד, משתמשים במילת המפתח clone.
- מетодת __clone(): אם תגדירו מетодה כזו במחלקה, היא תופעל אוטומטית על העותק החדש שנוצר. זה שימושי לביצוע "העתקה عمוקה" (למשל, לפתוח קובץ חדש עבור העותק במקום לחולק את אותו קובץ עם המקור).

3.9 פולימורפיזם (Polymorphism) וירושא

פולימורפיזם (ריבוי צורות) הוא המושג החשוב ביותר ב-OO. הוא מאפשר לנו להתייחס לאובייקטים מסוימים שונים דרך משך מסו�탍.

- **ירושא (extends):** מחלקה "בנ'" ירושת את כל המאפיינים והmethodות של מחלקה ה"אב".
- **הקשר:** נוצר קשר של "הואה-סוג-של" (a-is). לדוגמה, Dog הוא סוג של Animal.

במקרה לכתוב המונן תנאי `if` כדי לבדוק אם חיה קיבלנו, אנחנו פשוט קוראים למתחודה `makeSound()`. כל חיה תבצע את הגרסה שלה למתחודה (נבייה או יללה), אך הקוד שקורא להן נשאר זהה ופשוט.

::self-ו ::parent 3.10

- `::self` – מתייחס למחלקה הנוכחי (בעיקר עבור סטטיים וקבועים).
- `::parent` – מתייחס למחלקה האב. משמש בעיקר כדי לקרוא לבניי של האב או למתחודות שהאב מגדר וסדרנו בכך.

3.11 האופרטור instanceof

משמש לבדיקה האם אובייקט שייר למחלקה מסוימת או ירוש ממנו. מחזיר `true` או `false`.

3.12 מחלקות ומתחודות מופשטות (Abstract)

לפעמים נרצה להגדיר מחלקה כללית מדי כדי ליצור ממנה אובייקטים (למשל "צורה").

- **מחלקה מופסטה (abstract class):** לא ניתן ליצור ממנה מופיע (instance) עם `new`. היא קיימת רק כדי שאחרים יירשו منها.
- **מתחודה מופסטה (abstract function):** הגדירה של מתחודה ללא קוד (לא סוגרים מסולסלים). היא "מחייבת" את כל הילדים שיורשים מהמחלקה למש את המתחודה הזה בעצם.

3.13 ממשקים (Interfaces)

משחק הוא "חוצה" שהמחלקה מתחייבת לו. הוא מכיל רק חתימות של פונקציות (לא קוד).

- מכיוון ש-PHP לא תומכת בירושה מרובה (מחלקה לא יכולה לרשף משני אבות), ממשקים הם הפתרון: מחלקה יכולה למש (implements) כמה ממשקים שהוא רוצה.
- **דוגמה:** מחלקה Person יכולה למש ממשק Loggable, שמחייב אותה להגדיר פונקציה שמחזירה מחרוזת לוג.

final 3.16-ו 3.15

- **מתחודה final:** לא ניתן לדרוס אותה במחלקות ירושות.
- **מחלקה final:** לא ניתן לרשף ממנה בכלל. זהו כלי להגנה על הקוד מפני שינויים לא רצויים בעתיד.

3.17 מתחודה ()toString

מתודת `print_r` מוחדרת שמאפשרת לקבוע מה יודפס כשננסה להדפס את האובייקט (למשל עם `$obj`). היא חיבת להחזיר מחזרת.

3.18 טיפול בחריגות (Exception Handling)

במוקם לבדוק שגיאות עם המונח `try / catch`, משתמשים במבנה `try / catch`:

1. `try`: הקוד שעלול להיכשל.
2. `throw`: זריקת אובייקט שגיאה (Exception) כשםשו משתבש.
3. `catch`: תפיסת השגיאה וטיפול בה.

זה מפריד את הלוגיקה העסקית מהטיפול בשגיאות והופך את הקוד לנקי הרבה יותר.

3.19 הטענה אוטומטית (`__autoload`)

כשיש פרויקט עם המונח `autoload`, קשה לנו לא כל ה-`include`. פונקציית `__autoload($class_name)` נקראת אוטומטית על ידי PHP ברגע שאתם מנסים להשתמש במחלקה שהיא עוד לא מכירה. בתוכה תוכלו לכתוב קוד שיטוען את הקובץ המתאים "בדיקה בזמן".

3.20 רמייזות טיפוס למחלקות בפרמטרים של פונקציות (Class Type Hints)

למרות ש-PHP אינה שפה בעלת "טיפוסיות חזקה" (Strictly Typed), בה נדרש להציג על סוג המשתנה בכל פעם, היא מאפשרת לכם לציין במפורש איזה מחלקה אתם מצפים לקבל בפרמטרים של פונקציה או מתודת.

בעבר, כדי להבטיח שפונקציה תקבל אובייקט מסוים, היה עליו לכתוב קוד בדיקה ידנית:

```
PHP
} (function onlyWantMyClassObjects($obj
    } ((if (!$obj instanceof MyClass
        ("MyClass")("ניתן לשולח לפונקציה זו רק אובייקטים מסווג die
    {
        // ... המשך הקוד
    }
```

כדי לחסוך עבודה זו, PHP מאפשר להוסיף את שם המחלקה ישירות לפני שם הפרמטר בהגדרת הפונקציה:

```
PHP
} (function onlyWantMyClassObjects(MyClass $obj
    ...
    {
```

דגשים חשובים:

- **בדיקה אוטומטית:** בעת הקריאה לפונקציה, PHP מבצעת אוטומטית בדיקת instanceof. אם האובייקט שנשלח אינו שיר למחלקה המבוקשת (או יורש ממנה), התוכנית תעוצר עם הודעה שגיאה.
- **תמייה בירושה:** ניתן לשלוח כל אובייקט שמקיים קשר של "הוא-סוג-של" (a-is) עם המחלקה המוגדרת.
- **יתרונות בפיתוח:** תוכנה זו עוזרת לוודא שאינכם מעבירים בטעות אובייקטים לפונקציות שלא תוכננו לטפל בהם, מה שמנוע באגים קשים לאיתור בזמן ריצה.

3.21 סיכום הפרק

פרק זה סקר את מודל האובייקטים החדש של PHP. למדנו על:

- ההבדל בין מחלקות (התבנית) לאובייקטים (המופעלים).
- פולימורפיזם וירושה המאפשרים שימוש חזזר בקוד.
- מנגנוני הגנה כמו private ו-protected.
- טיפול מתקדם בשגיאות באמצעות חריגות (Exceptions).

אם הגעתם לשפות כמו Java או C++, ודאי תזהו ש-5 PHP הפכה לשפה מודרנית ובעל הרבה יותר. אם כתבתם ב-4 PHP, התכונות הללו הן "敖oir לנשימה" עבור פיתוח פרויקטים גדולים ומורכבים.

פרק 4

4.1 יכולות "העמסה" (Overloading)

ב-PHP, המושג "Overloading" שונה משפות כמו Java או C++. כאן הכוונה היא ליכולת לירט (Intercept) גישה למאפיינים או למетодות שאינם קיימים במחלקה.

4.1.1 העמסת מאפיינים ומethods

באמצעות שימוש "magic methods" (Magic Methods), ניתן לשולט במקרה שקויה כשםישחו מנסה לגשת למשתנה או פונקציה שלא הוגדרו:

- `__get($property)`: מופעלת כשמנסים לקרוא מאפיין לא קיים.
- `__set($property, $value)`: מופעלת כשמנסים לכתוב למאפיין לא קיים.
- `__call($method, $args)`: מופעלת כשמנסים לקרוא למethod לא קיימת.

דוגמה לשימוש (Delegation): ניתן להשתמש ב-`call` כדי להעביר קריאות לאובייקט אחר באופן אוטומטי (כמו עוזר אישי שמעביר שיחות למנהל).

4.2.2 העמסת גישה כמערך (Array Access)

לפעמים נרצה שאובייקט יתנהג כמו מערך (למשל: `[5] John"][$userMap]`). PHP מאפשר זאת על ידי שימוש הממשק `ArrayAccess`.

זה שימושי מאוד כשרוצים לגשת למסד נתונים עצום: במקומות לטען את כל המידע **למערך**, האובייקט יבצע שאילתת למסד הנתונים רק ברגע שנבקש מפתח ספציפי באמצעות סוגרים רבים.

4.3 איטרטורים (Iterators)

כברית מחדל, לולאת `foreach` רצה על כל המאפיינים הציבוריים של אובייקט. אבל מה אם האובייקט מייצג קובץ הגדרות או תוצאות השאילתת?

PHP מאפשרת להציג התנהלות מותאמת אישית ללואה על ידי שימוש הממשקים `Iterator` או `IteratorAggregate`.

הmethodות שיש למש:

- | | |
|---|--|
| 1 | <code>rewind()</code> : חזרה לתחילת המידע. |
| 2 | <code>current()</code> : הערך הנוכחי. |
| 3 | <code>key()</code> : המפתח הנוכחי. |
| 4 | <code>next()</code> : מעבר לאיבר הבא. |
| 5 | <code>valid()</code> : האם נשאר עוד מידע? |

4.4 תבניות עיצוב (Design Patterns)

תבניות עיצוב מקובלות לביעות תוכנות שחוירות על עצמן. הן מספקות שפה משותפת למפתחים.

4.4.1 תבנית האסטרטגיה (Strategy Pattern)

משמשת כשרוצים להחליף אלגוריתם אחד אחר בקלות.

דוגמה: שרת הורדות שבוחר אם לנק לקובץ `.tar.gz` (למשתמשי חלונות) או `.tar`. (למשתמשי לינוקס). הקוד הראשי לא משתנה, רק "האסטרטגיה" של יצירת הלינק מוחלפת בזמן ריצה.

4.4.2 תבנית הסינגלטון (Singleton Pattern)

mbטיחה שקיים רק **מופיע אחד ויחיד** של מחלוקת בכל האפליקציה (למשל: אובייקט לניהול לוגים או חיבור למסד נתונים).

- **איך עושים זאת?** הופכים את הבנייה (`construct`) ל-`private` ויוצרים מתודה סטטית בשם `getInstance` שמנהלה את יצירת המופיע.

4.4.3 תבנית המפעל (Factory Pattern)

במקומות ליצור אובייקטים ישירות עם `new`, משתמשים ב"מפעל" (מתודה סטטית) שמקבל קלט ומחליט איזה סוג של אובייקט לייצר עבורנו.

דוגמה: `UserFactory::Create("Andi")` ייצור אובייקט מסוג `AdminUser` או `GuestUser` בהתאם להרשאות שלו במסד הנתונים.

4.4.4 תבנית הצופה (Observer Pattern)

מאפשרת לאובייקטים "להירשם" לירועים. כשميدע משתנה באובייקט אחד, כל האובייקטים שנרשמו אליו מקבלים הודעה אוטומטית.

דוגמה: כSSHUR ה-chlifin של הדולר מ-**מ-עדכן**, כל מוציא ה-chnot (ה-**צופים**) מעדכנים את המ-חיר המ-זג שלם באופן אוטומטי.

4.5 רפלקציה (Reflection / Introspection)

4.5.1 מבוא

יכולות הרפלקציה מאפשרות לכם לבצע אינטראוספקציה (בחינה פנימית) של הסקריפט בזמן ריצה. אתם יכולים לבחון פונקציות, מחלקות, متודות ופרמטרים, ואף לחש את אליהם או להפעיל אותם דרך המידע שנמצא (Metadata). בעוד ש-PHP תמיד אפשרה קראיה עקיפה לפונקציות (כמו `$func(...)`), ה-**API** של הרפלקציה הרבה יותר עשיר ומובנה.

Reflection API 4.5.2

ה-**API** מורכב מסדרה של מחלקות "יעודיות", שכל אחת מהן מטפלת בישות אחרת בשפה.

הפריטים העיקריים ב-**API**:

- **ReflectionClass**: מאפשרת לקבל מידע על מ-**מחלקה** (מתודות, מאפיינים, קבועים, האם היא מופשטת וכו').
- **ReflectionMethod**: בוחנת מתודות בתוך מ-**מחלקה** (האם היא ציבורית, סטטית, סופית וכו').
- **ReflectionFunction**: בוחנת פונקציות רגילים.
- **ReflectionProperty**: בוחנת משתנים (מאפיינים) של מ-**מחלקה**.
- **ReflectionParameter**: בוחנת פרמטרים שנשלחים לפונקציה או מ-**תודה** (שם הפרמטר, האם הוא אפשרי null).

4.5.3 דוגמאות לשימוש ברפלקציה

4.5.3.1 חילוץ מידע אוטומטי

ניתן להשתמש ב-**תודה** הסטטית `ReflectionClass::export()` כדי להדפיס את כל המבנה של מ-**מחלקה** מסוימת (כולל מ-**מחלקות פנימיות** של PHP). זהו כל' מעולה לדיביגינג או ליצור תיעוד אוטומטי של הקוד שלו.

4.4 שימוש בתבנית האצלה (Delegation) באמצעות רפלקציה

לעתים נרצה ש-**מחלקה** אחת תבצע את כל מה ש-**מחלקה** אחרת עשו, אך מוביל לשימוש בירושה (כי הקשר אין קשר של "הוא-סוא-של" או כי המ-**מחלקה** כבר ירושת ממשהו אחר). במקרה זהה משתמש בתבנית **האצלה** (Delegation).

איך זה עובד בדוגמה מהספר?

1. יוצרים מ-**מחלקה** בשם `ClassOneDelegator`.
2. משתמשים ב-**תודה** הקסם `()__call` כדי לירות כל קראיה ל-**תודה** שאינה קיימת.
3. בתוך `()__call`, עוברים על רשות אובייקטים רשומים ומשתמשים ב-**ReflectionClass** כדי לבדוק: "האם לאובייקט זהה יש מ-**תודה** בשם שביבשנו? האם היא ציבורית? האם היא לא מופשטת?".
4. אם התשובה חיובית, מפעילים אותה באמצעות `()invoke`.

4.6 סיכום

פרק זה כיסה את התכונות המתקדמות ביותר של PHP. בזכות הכלים הללו (העמסה, איטרטורים, תבניות עיצוב ורפלקציה), PHP הפכה לשפה המסוגלת להריץ אפליקציות ברמת Enterprise בקנה מידה גדול.

המלצתה: הספר מלילץ להעמק בתכונות עיצוב דרך המקורות הקלאסיים, כמו הספר של " Gang of Four" (Gang of Four), כדי לשדרג את רמת התוכן שלכם.

פרק 5

כיצד לכתוב אפליקציית אינטרנט ב-PHP

"הבטחה האולטימטיבית היא ההבנה שלך את המציאות." — ה. סטנלי ג'אד

5.1 מבוא

השימוש הנפוץ ביותר ב-PHP הוא בניית אתרי אינטרנט. PHP הופכת אפליקציות אינטרנט לדינמיות ומאפשרת למשתמשים ליצור אינטרנט אינטראקטיבי עם האתר. האפליקציה אוספת מידע מהמשתמש באמצעות טפסי HTML ומעבדת אותו. חלק מהמידע שנוסף רגיש, מה שהופך את האבטחה לנושא מרכזי. PHP מספקת תוכנות המאפשרות לאסוף מידע ולאבטח אותו.

לאחר קראת פרק זה, תלמדו:

- כיצד להטמע PHP בתוך קבצי HTML.
- כיצד לאסוף מידע ממבקרים באמצעות טפסי HTML.
- טכניקות נפוצות לתקיפת אתרים ואייר להציגם מפנה.
- כיצד לטפל בשגיאות בקלט של משתמשים.
- שימוש ב-Cookies וב-Sessions לשימירת נתונים לאורך האפליקציה.
- העלאת קבצים מהמשתמש.
- ארגון נכון של אפליקציית האינטרנט.

5.2 הטעמה (Embedding) בתוך HTML

ניתן ליצור קובץ PHP ללא HTML בכלל, אך בניית אפליקציות אינטרנט לרוב משלבים בין השניים. PHP פותחה במקור כ"שפה תבניות" (Templating language) המוטמעת ב-HTML. קבצים אלו יקבלו לרוב הסיומת .php.

כאשר הטקסט להדפסה פשוט, ניתן להשתמש ב-`echo`. אך כאשר הטקסט מכיל גרשיים (בודדים או כפולים), הקוד הופך למסורבל. דוגמה לכתיבה לא נקייה:

```
PHP
<? } ('php if ($user == 'Jerry'?>
      שלום ג'רי סינפלד!
<? { php else?>
```

כתיבה זו מפירה את העיקרון: "הفرد לוגיקה מתוכן". פתרון טוב יותר הוא שמירת התוכן במשתנה והדפסתו בסוף.

PHP תומכת גם בתגית מקוצרת: <? variable\$ =?> שהוא קיצור ל-<?php echo \$variable; ?>. עם זאת, מומלץ להשתמש בתגיוט המלאות <?php ... ?> כדי להבטיח תאימות לכל השרטים, שכן התג המקוצר עשוי להיות כבוי בהגדרות השרת.

5.3 קלט משתמש (User Input)

כדי לאסוף נתונים (כמו במערכת רישום), נשתמש בטופס HTML.

שיטות שלילה (Methods):

- **GET:** מקודדת את הנתונים בתחום ה-URL. הם נראים בשורת הכתובת ונitin לשמר אותם בסימניות (Bookmarks). לא מתאים לנוטונים ורגישים כמו סיסמאות.
- **POST:** שולחת את הנתונים בגוף הבקשה (HTTP Body). הנתונים לא מופיעים בכתובת ה-URL. זו השיטה המועדף למידע רגיש.

גישה לנוטונים ב-PHP:

PHP משתמש במערכות גלובליים מיוחדים ("סופר-גלובליים"):

- `$_GET` – לנוטונים שנשלחו ב-GET.
- `$_POST` – לנוטונים שנשלחו ב-POST.
- `$_REQUEST` – מכיל את שניהם (וגם Cookies).

ניתן לבדוק אם טופס נשלח על ידי בדיקה אם כפתור השליחה (Submit) קיים במערך:

```
{if (isset($_POST['register'])) ... }
```

5.4 טיפול בטוח בקלט משתמש

כל בrzil: אל תסמור על אף אחד, בטח לא על המשתמשים. משתמשים עשויים להזין נתונים לא צפויים בטענות או בזדון.

5.4.1 טעויות נפוצות

1. **משתנים גלובליים (register_globals):** בעבר PHP הפעלה אוטומטית כל פרמטר ב-URL למשתנה גלובלי. זהו פרצת אבטחה חמורה. בגרסאות מודרניות זה כבוי כברירת מחדל, ויש להשתמש רק במערכות המפורטים (`$_POST` וקדם).
2. **הזרקת קוד XSS - Cross-Site Scripting:** תוקף עשוי להזין קוד JavaScript לתוכה שדה טקסט. אם תדפיסו את הטקסט הזה ישירות ל-HTML ללא סינון, הקוד יירוץ בדף של משתמשים אחרים ויכל לגרום עוגיות.
 - **פתרון:** שימוש בפונקציות כמו `strip_tags()` או `htmlspecialchars()`.

3. **הזרקת SQL (SQL Injection):** תוקף מזמן תווים מיוחדים (כמו גרש ') כדי לשנות את שאלתת מסד הנתונים שלכם ולקוף את מערכת האימות.
- **פתרונות:** שימוש ב-`addslashes()` או בטכנייקות מתקדמות יותר כמו `Prepared Statements`.

5.5 טכניקות להפיכת סקריפטים ל"בטוחים"

5.5.1 אימות קלט (Input Validation)

עליכם לבדוק כל פיסת מידע שmagie מהמשתמש. אם אתם מצפים למספר (ID), וודאו שהוא אכן מספר:

```
$_GET['prod_id'] = (int) $prod_id$;  
כל ערך שאינו מספר יהפוך ל-0.
```

5.5.2 אימות HMAC

כשאתם מעבירים פרמטרים רגילים ב-URL, תוקף יכול לשנות אותם. כדי למנוע זאת, ניתן להוסיף "חתימה" (Hash). שיטת ה-HMAC משתמש ב מפתח סודי כדי ליצור חתימה שאפשר לשייך ללא המפתח.

5.5.5 עבודה עם סיסמות

לעולם אל תשמרו סיסמות כטקסט פשוט (Plain Text) במסד הנתונים. אם האקרים יפרוץ במסד הנתונים, כל הסיסמות ייחשפו.

יש להשתמש בפונקציות גיבוב (Hashing) כמו `sha1()` או `md5()` (היום מומלץ להשתמש ב-`password_hash()`). חתימות אלו אינן הפוכות – אין אפשרות לשחזר את הסיסמה המקורי מהחתימה.

5.6 עוגיות (Cookies)

עוגיות מאפשרות לשמור נתונים בין דפים שונים, שכן האינטרנט כשלעצמו הוא "חסר מצב" (Stateless).

שליחת עוגיה מתבצעת באמצעות:

```
;(setcookie('name', 'value', expiry_time, path
```

חשוב: עוגיות (וכותרות HTTP אחרות) חייבות להישלח לפני שנשלח TOKEN HTML כלשהו לדפדפן. אם תנסו לשלוח עוגיה אחרי שכבר הדפסתם טקסט, תקבלו שגיאת "Headers already sent". כדי לעקוף זאת, ניתן להשתמש ב-`ob_start()` (Output Buffering) שואגר את כל הפלט בזיכרון ושולח אותו רק בסוף.

5.7 סשןים (Sessions)

סשן (Session) ב-PHP מאפשר לאפליקציה לאחסן מידע עבור ה"מפגש" הנוכחי, שניתן להגדירו כמשתמש אחד המחבר לאפליקציה שלכם. סשן מזוהה על ידי מזהה ייחודי (PHP Session ID). PHP יוצרת מזהה סשן

שהוא מחרוזת הקסדצימלית המורכבת מגיבוב (MD5 Hash) של כתובת ה-IP המרוחקת, הזמן הנוכחי ומידה נוספת של אקריאיות. ניתן להעביר את זהה הסשן הזיה בעוגייה (Cookie) או להוציאו לכל הכתובות (URLs) בעת הניות באפליקציה. מטעמי אבטחה, עדיף לאלץ את המשתמש להפעיל עוגיות מאשר להעביר את זהה הסשן בכתובת ה-URL (מה שבדרכך כלל גנעת ידנית על ידי הוסף ?=PHP_SESSID?) או על ידי הפעלת session.use_trans_sid בקובץ php.ini, שם הוא עשוי להופיע ביוםני שירות האינטרנט HTTP_REFERER או להימצא על ידי גורם עין המנטר את התעבורה שלכם. אותן גורם עין יכול לראות את נתוני עוגית הסשן, כמובן, אך מומלץ להשתמש בשרת המאובטת ב-SSL כדי להיות בטוחים בהחלט.

כדי להמשיך בדיקון על שנים, נכתוב מחדש את דוגמת העוגיות הקודמת באמצעות שנים. ניצור קובץ בשם session.inc המגדיר כמה ערכי שן, כפי שמוצג בדוגמה הבאה, ונכלול (Include) קובץ זה בתחילת כל סקירה מהו חלק מהסשן:

בשורה הראשונה, פרמטר ההגדירה session.use_cookies מוגדר ל-1, מה שאומר שעוגיות ישמשו להעברת זהה הסשן. בשורה השנייה, session.use_only_cookies מוגדר ל-1, מה שאומר שזהה סשן שיועבר בכתובת ה-URL לסקריפט יידחלה. הגדרה זו מחייבת שימושם יהיו עוגיות מופעלות כדי להשתמש בסשן. אם איןכם יכולים להסתמך על כך שלאנשים יש עוגיות מופעלות, תוכלו להסיר שורה זו או לשנות את הערך ל-0.

טיפ: ניתן להגדיר את המיקום שבו PHP תஅחסן את קבצי הסשן באמצעות הגדרת session.save_path. הפונקציה session_start() חייבת להופיע לאחר שכל ההגדירות הקשורות לשן בוצעו עם session_set(). פונקציה זו מתחילה את מודול הסשן, מגירה כתורות (Headers) מסוימות (כמו עוגיות זהה סשן וכותרות למניעת זיכרון ממושן), וכן היא חייבת להופיע לפני שנשלח פלט כלשהו לדפפן. אם אין זהה סשן זמן הקראיה ל-session_start(), נוצר מזהה חדש והשן מאותחל עם מערך \$_SESSION ריק. הוספה איבורם למועד זה היא פשוטה, כפי שמוצג בדוגמה הבאה.

טיפ: ניתן לקרוא ל-session_name('NAME') לפני הקראיה ל-session_start() כדי לשנות את שם ברירת המחדל של עוגיית הסשן מ-PHP_SESSID.

תחילה אנו יכולים את הקובץ inc.session. הוספה משתנה הסשן 'uid' לשן מתבצעת בקלות על ידי הגדרת איבר ה-\$uid במערך הסופר-גlobeil \$_SESSION לערך של \$uid. ביטול הגדרה של משתנה שן יכול להתבצע עם unset(\$SESSION['uid']).

טיפ: אם עליים לעבד כמות גדולה של נתונים לאחר שינוי משתני הסשן, מומלץ לקרוא session_write_close(), דבר המבוצע בדרך כלל אוטומטית בסוף הסקריפט. פעולה זו כתובה את קובץ הסשן לדיסק ומחזרת את הנעליה שלו למערכת הפעלה כך שסקרייפטים אחרים יוכל להשתמש בו.

התנטיקות (Logout) זהה להשמדת הסשן והנתונים הקשורים אליו: אנו עדין צריכים לאותחל את הסשן עם session_start(), ולאחר מכן נוכל לנוקוט אותו על ידי הגדרת מערך \$_SESSION לערך ריק. לבסוף, אנו משמידים את הסשן על ידי קראיה ל-session_destroy().

הגישה למשתני שן נעשית דרך המערך הסופר-גlobeil \$_SESSION. בסקריפט php.index.php שלנו, העברתו את הוצאה ה-if שבודקת אם המשתמש מחובר לפונקציה מיוחדת בתוך הקובץ inc.session:

```
PHP
{
    function check_login()
    {
        if (!isset($_SESSION['uid']) || !$SESSION['uid'])
        {
            /* אם אין UID בעוגייה, אנו מפנים לדף ההתחברות */
        }
    }
}
```

```
;('header("Location: http://kossu/session/login.php
{
}
```

אנו קוראים לפונקציה `check_login()` בכל דף שבו אנו דורשים מהמשתמש להיות מחובר. עליינו לוודא שקובץ `session` נכלל לפני שנוצר פלט כלשהו, מכיוון שייתכן שהיא עליו לשלוח כותרות (Headers) לדף.

5.8 העלאת קבצים (File Uploads)

טרם CISINO סוג אחד של קלט – ה

עלאת קבצים
. ניתן להשתמש בתוכנות ה

עלאת קבצים
 של PHP כדי להעלות תמונות או חומרים רלוונטיים. מכיוון שהדף צריך לעשות קצת יותר מאשר רק לשלוח בקשה POST עם הנתונים, עליכם להשתמש בטופס שנבנה במיוחד להעלאת קבצים.

הבדל בין טפסים להעלאת קבצים לטפסים רגילים הוא קרייטי. ראשית, אטיריבוט `enctype="multipart/form-data"` מורה לדפדף לשלוח סוג אחר של בקשה POST. זהו POST רגיל, אלא שהגוף המכיל את הקבצים המוקודדים שונה לחולטי. במקום תחביר פשוט של `field=var&field2=var2`, נשלח גוף הדומה למיל המכיל טקסט-HTML, כאשר כל חלק הוא שדה בטופס.

שדה ה

עלאת קבץ
 עצמו הוא מטיפוס `file`, המציג שדה קלט וכפתור "עיוון" (Browse) המאפשר למשתמש לחפש קובץ במערכת הקבצים. שדה הקלט הנsteller שלו `MAX_FILE_SIZE` לדפדף, המגדיר את הגודל המקסימלי המותר לקובץ המועלה, אך רוב הדפדףים מתעלמים ממנו, ולכן עליכם לבדוק זאת בסקריפט הטיפול.

5.8.1 טיפול בקובץ המועלה

המערך `$_FILES` מכיל מידע על כל קובץ שהועלה. סקריפט הטיפול ניגש למידע באמצעות שם שדה הקובץ כמפתח. המשתנה `$_FILES['book_image']` מכיל את המידע הבא:

מפתח	ערך	тип	תיאור
<code>name</code>	מחרוזת	מחרוזת	שם המקורי של הקובץ במחשב של המשתמש.
<code>type</code>	מחרוזת	מחרוזת	סוג ה-MIME של הקובץ (למשל <code>jpeg/jpg</code>).
<code>tmp_name</code>	מחרוזת	מחרוזת	שם הקובץ הזמני במערכת הקבצים של השרת.
<code>error</code>	מספר	מספר	קוד השגיאה (0 אם הצליח).

size	מספר	גודל הקובץ בbytes.
------	------	--------------------

כמו שגיאות אפשריות במהלך העלאת קובץ הקשורות לרוח לגודלו. לכל קוד שגיאה יש קבוע תואם:

#	קבוע	תיאור
0	UPLOAD_ERR_OK	הקובץ הועלה בהצלחה.
1	UPLOAD_ERR_INI_SIZE	הקובץ חרג מהגודל המוגדר ב- <code>php.ini</code> .
2	UPLOAD_ERR_FORM_SIZE	הקובץ חרג מהגודל המוגדר בשדה <code>MAX_FILE_SIZE</code> בטופס.
3	UPLOAD_ERR_PARTIAL	הקובץ הועלה באופן חלק בלבד.
4	UPLOAD_ERR_NO_FILE	לא נבחר קובץ להעלה.

בסקריפט העלאת הקובץ, אנו בודקים אם הגודל מקובל (למשל, לא יותר מ-50KB) ואם הקובץ מהסוג הנכון (למשל JPEG או PNG). מכיוון שלא ניתן להסתמך על ה-`MAX_FILE_SIZE` שמספק המשתמש, علينا תמיד לבדוק את הגודל בעצמו באמצעות `$_FILES['book_image']['size']`. אנו משתמשים בפונקציה המויהדת `uploaded_file_move()` כדי להעביר את הקובץ ליעדו הסופי. פונקציה זו בודקת אם הקובץ הוא באמת קובץ שהועלה ולא ניסיון הונאה לגשת לקבצים רגילים בשורת.

5.9 ארכיטקטורה

בחלק זה נדון בכמה דרכים לארגון הקוד באפליקציית האינטרנט שלכם.

5.9.1 סקריפט אחד משרת את כלם (One Script Serves All)

הkoncept הוא שסקריפט אחד, בדרך כלל `index.php`, מטפל בכל הבקשות עבור הדפים השונים. תוכן שונה מועבר כפרמטרים כתובות ה-URL, למשל `?page=register`. מומלץ לטעון מודולים באופן דינמי מספרייה ייעודית.

5.9.2 סקריפט אחד לכל פונקציה

אלטרנטיבה היא אחסון כל פונקציה בסקריפט נפרד הנגיש דרך ה-URL שלו (למשל `about.php`). בשיטה זו אין סקריפט "מנהל", ויש לכלול את הבסיס (כמו טיפול בסשן) בכל קובץ בנפרד.

5.9.3 הפרדת לוגיקה מעיצוב (Layout)

בכל גישה שתבחרו, תמיד עליו לשאוף להפריד את הלוגיקה מהעיצוב של הדפים. ניתן לעשות זאת באמצעות מנوعי תבניות (Templates). קובץ התבנית יהיה החלק ה"סטטי" המכיל HTML עם הצהרות PHP פשוטות להדפסת משתנים הממולאים על ידי הלוגיקה בסקריפט הראשי.

5.10 סיכום

PHP מוטמעת בקלות בטור קובצי HTML, ומציג טפסים האוספים נתונים שהזנו על ידי משתמשים וקבצים שהועלו על ידם. איסוף מידע משתמש מציב סוגיות אבטחה עבור האתר האינטרנט ועבור כל מידע של משתמש המאושר באתר. לצורך אבטחה, יש להגדיר ב-PHP את `register_globals` במצב Off. כדי לתקוף את האתר שלכם או לגנוב את הנתונים שלכם, ה"רעים" משתמשים בטכניקות כמו Cross-site scripting (הרצה קתעה) סקריפט בצד הלקוח באתר שלכם) ו-`SQL injection` (הזרקת קוד זמני לתוך שאלות המורוצות על מסד הנתונים שלכם). כדי להגן מפני התקפות, עליהם להטיל ספק בכל הנתונים שמקורם במשתמשים. עליהם לאמת בקפידה את כל הנתונים שאתם מקבלים ולבדק אותם ביסודות כדי לוודא שהם בטוחים ואין מסוכנים לאתר.

ניתן להגן על האתר בעת העלאת קבצים על ידי בדיקת גודל הקובץ וסוג הקובץ המועלה. בנוסף, ניתן להגן על המידע הציבורי בחילון הכתובות של הדף – מידע המועבר ב-URL – על ידי גיבובו (Hashing) באמצעות אחד מספר שיטות, כולל מחלוקת PEAR בשם Crypt_HMAC. גיבוב שימושי גם להגנה על סיסמאות המאוחסנות לצורכי אימות משתמשים. אמצעי שימושי נוסף להגנה על האתר מפני טעויות משתמש או התקפות הוא פיתוח מטפל שגיאות (Error handler) משלכם, שיזהה מתי משהו אינו כשר ויטפל בבעיה.

כדי שאפליקציה אינטרנט תהיה שימושית, נתונים האפליקציה חייבות להיות זמינים לכל דפי האינטרנט באפליקציה במהלך סשן (Session) של משתמש. דרך אחת להעביר נתונים מדף אחד לשני הוא היא באמצעות עוגיות (Cookies). כאשר המשתמש ניגש לדף, מוצג דף התוצאות, והחשבון והסיסמה שהזנו נבדקים מול הנתונים המאוחסנים. אם המשתמש מאומת, מוגדרת עוגייה. המידע בעוגייה מועבר אוטומטית עם כל דף מבקש. שיטה שנייה לשמירה על המשכיות הנתונים היא שימוש בתכונות הסשן של PHP. ברגע שמתחלים סשן, ניתן לאחסן משתנים הזמינים לסקריפטים אחרים באותו סשן.

לאחר הכרת כל החלקים הדרושים לאפליקציה, עליהם לארגן אותם למכלול שימושי. שיטה נפוצה אחת לארגן נקראת "סקריפט אחד משרת את כולן" (all one script serves one), שמשמעותה היא ש-`index.php` מטפל בכל הבקשות לדפים השונים. ארגון נפוץ אחר הוא "סקריפט אחד לכל פונקציה". עיקרונו כללי הוא להפריד בין העיצוב (Layout) ללוגיקה. לאחר ארגון החלקים לאפליקציה מキפה, אתם מוכנים לצאת לדרך.

פרק 6: מיסדי נתונים עם PHP

6.1 מבוא

חלק בלתי נפרד מכל ספר PHP הוא נושא מסדי הנתונים והמשק בינם לבין PHP. ספר זה אינו שונה, פשוט מכיוון שרוב האנשים שכותבים אפליקציות PHP מועוניים להשתמש במסדי נתונים.

פרק זה מציג את השימוש ב-MySQL ו-SQLite מתוך PHP, אך מתמקד בעיקר בפרטים הספציפיים ל-5 PHP בכל הנוגע לממשק מסדי נתונים. בסוף פרק זה תלמדו:

- נקודות חזק וחולשה של MySQL ו-SQLite.
- עבודה עם MySQL באמצעות הרחבה `mysqli` החדשה.
- שימוש בהרחבת `sqlite` המובנית של 5 PHP.
- שימוש ב-PEAR DB לכתיבת קוד מסד נתונים נייד (Portable) יותר.

הערה לגבי גרסאות: פרק זה מתמקד בתכונות הקישוריות החדשנות של 5 PHP, ובפרט בהרחבות `mysqli` ו-`sqlite`. כדי להנốt מהfonקציונליות המתוארת, עליו להשתמש בגרסאות עדכניות: MySQL 4.1.2 ומעלה, SQLite 4.1.2 כפוי שמקור ל-5.0.0 PHP ומעלה, ו-PEAR DB 1.6 ומעלה.

MySQL 6.2

MySQL ו-PHP הפכו ל"לחם והחמה" של בני אפליקציות אינטרנט. זהו השילוב שסייע להניח שתפגשו חיים ובשנים הבאות. פרק זה מתמקד בהרחבת `mysqli` (או MySQL 4.1.2 המצורפת ל-5 PHP, הדורשת לפחות גרסה 4.1.2 של שרת ה-MySQL Improved).

6.2.1 MySQL נקודות חזק וחולשה של

- חזק: חDIRה عمוקה לשוק. ל-MySQL נתח השוק הגדול ביותר מבין מסדי הנתונים בקוד פתוח. כמעט כל חברת אחסון אתרים מספקת גישה ל-MySQL.
- חזק: קלות התחלת. ניהול מסדי נתונים הוא פשוט וישנים כלים כמו phpMyAdmin לניהול נוח.
- חזק: רישיון קוד פתוח לרוב המשתמשים. ניתן להשתמש ב-MySQL תחת רישיון GPL כל עוד איןכם מפיצים אותה מסחרית.
- חזק: מהירות. MySQL תמיד הייתה מהירה יחסית בשל פשוטותה, ועם השנים הוסיף תוכנות של "Enterprise Class" מוביל להתאפשר על ביצועים.
- חזק: סקלibility (יכולת צמיחה) סבירה. MySQL התפתחה כך שהיא כמעט מושתוות למערכות מסחריות גדולות כמו Oracle מבחינות אمنיות, אך עדין ניתנת להגדרה לשימוש "קל".
- חולשה: רישיון מסחרי להפצה מסחרית. אם אתם מגדים את MySQL עם מוצר מסחרי בקוד סגור, עליו לרכוש רישיון.

6.2.2 משק PHP

הרחבה `mysqli` נכתבת מהיסוד כדי לתמוך בתכונות החדשנות של MySQL 4.1 ו-5.0. השיפורים לעוממת הרחבה `mysqli` הישנה כוללים:

- פונקציונליות Native bind/prepare/execute.
- תמיכה ב-Cursors.
- קוד שגיאה בתקן SQLSTATE.
- הרצת מספר הצהרות (Statements) בשאלתה אחת.

לכל פונקציית mysqli יש מקבילה כמתודה או כמאפיין (Property) בגישה מונחית עצמאים.

6.2.4 (Connections)

טבלה 6.1 מציגה את פונקציות וمتודות החיבור.

טבלה 6.1: פונקציות וمتודות חיבור ב-`mysqli`

- `mysqli_connect(...)` / `new mysqli(...)`: פותח חיבור חדש לשרת MySQL.
- `mysqli_close(...)` / `$mysqli->close()`: סגור חיבור פתוח.
- `mysqli_connect_errno()`:מחזיר את קוד השגיאה של ניסיון החיבור האחרון שנכשל.

6.2.5 שאילותות נאגרות מול שאילותות לא נאגרות (Buffered vs. Unbuffered)

ללקוח MySQL יש שני סוגים של שאילותות:

1. שאילותות נאגרות (Buffered): התוצאות נשלפות ומאותחות בזיכרון של צד הלוקה.
 - יתרון: ניתן לנוט בטור התוצאות בחופשיות (Seeking).
 - חיסרון: דורש זיכרון נוסף ועלול לעכב את החזרת הפלט עד שכל התוצאות נשלפו.
2. שאילותות לא נאגרות (Unbuffered): מגבילות אתכם לגישה סדרתית בלבד לתוצאות, אך אין דורשות זיכרון נוסף לאחסון כל סט התוצאות.
 - יתרון: חסכו בזכרון בסיטים גדולים של נתונים; ניתן להתחיל לעבוד שורות מיד כשהשרות מתחילה להחזירן.

הבחירה בסוג השאלתה תלויה במצב: שאילותות לא נאגרות חוסכות זיכרון כשההתוצאה גדולה מאוד, בעוד שאילתות נאגרות נוחות יותר בזכות יכולת הניוט בתוצאות.

6.2.6 שאילותות (Queries)

חלק זה מתאר פונקציות וمتודות לביצוע שאילותות (ראו טבלה 6.3).

טבלה 6.3: פונקציות שאילתה ב-`mysqli`

תיאור	שם הפקציה
שולחת שאלתה לבסיס הנתונים ומחזירה אובייקט תוצאה. פרמטרים: חיבור (בפקציה בלבד), שאלתה (מחרוזת), מצב (נאגר או לא נאגר).	(...) mysqli_query
שולחת ומעבדת מספר שאלות בחת אחת. פרמטרים: אובייקט חיבור (בפקציה בלבד), שאלתה (מחרוזת).	(...) mysqli_multi_query

הפקציה **mysqli_query()** מחזירה אובייקט של סט תוצאות. במקרה של כישלון, השתמשו בפקציה **mysqli_error()** או במאפיין **\$conn->error** כדי לקבוע את סיבת הכישלון:

PHP

```
;("result = $conn->query("SELECT Name FROM City$"
} ()while ($row = $result->fetch_row
;"print $row[0] . "<br>\n"
{
;()result->free$"
;()conn->close$
```

לאחר ביצוע השאלתה, מוקצה זיכרון לצד הלוקום כדי לאחזר את סט התוצאות המלא. כדי להשתמש בסט תוצאות שאינו נאגר (**Unbuffered**), עליו לציין את הפרמטר האופציוני **MYSQLI_USE_RESULT**:

PHP

```
;($result = $conn->query("SELECT Name FROM City", MYSQLI_USE_RESULT$"
} ()while ($row = $result->fetch_row
;"print $row[0] . "<br>\n"
{
;()result->free$"
;()conn->close$
```

6.2.7 הצהרות מרובות (Multi Statements)

הרחבת **mysqli** מאפשרת לכם לשולח מספר הצהרות SQL בקריאה אחת לפקציה באמצעות **mysqli_multi_query**. מחרוזת השאילה מכילה הצהרת SQL אחת או יותר המופרדות בנקודה-פסיק בסוף כל הצהרה. אחזר סטם של תוצאות מהצהרות מרובות הוא מעט מורכב, כפי שמדגימה הדוגמה הבאה:

PHP

```
} ((if ($mysql->multi_query($query
} do
```

```

} ()if ($result = $mysqli->store_result
} ()while ($row = $result->fetch_row
;([printf("Col: %s\n", $row[0
{
;
;)result->close$}
{
;())while ($conn->next_result {
{
;()conn->close$}

```

6.2.8 מנגנון שליפה (Fetching Modes)

קיימות שלוש דרכים לשולוף שורות של תוצאות, בדומה להרחבת `mysql_fetch` הישנה: כמערך ממושך, כמערך אסוציאטיבי או אובייקט (ראו טבלה 6.4).

טבלה 6.4: פונקציות שליפה ב-`mysqli`

תיאור	שם הפונקציה
שולפת שורה כמערך ממושך. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).	(...) <code>mysqli_fetch_row</code>
שולפת שורה כמערך אסוציאטיבי. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).	(...) <code>mysqli_fetch_assoc</code>
שולפת שורה לתוכן אובייקט. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).	(...) <code>mysqli_fetch_object</code>

6.2.9 הצהרות מוכנות (Prepared Statements)

אחד היתרונות המרכזיים של הרחבה `mysqli` בהשוואה ל-`mysql` הוא "הצהרות מוכנות". אלו מספקות למפתחים את יכולת ליצור שאילתות מאובטחות יותר, בעלות ביצועים טובים יותר ונוחות יותר לכתיבתה.

ישנם שני סוגי של הצהרות מוכנות: כאלו המבצעות מניפולציה נתונים (Data Manipulation) וכאלו המבצעות שליפת נתונים (Data Retrieval). הצהרות אלו מאפשרות לחבר (Bind) משתני PHP ישירות לקלט ולפלט.

תהליך היצירה פשוט: בניית שאילתת נוצרת ונשלחת לשרת MySQL. השרת מתקף אותה, מנתח (Parses) אותה ומאהסן אותה במבנה מיוחד. לאחר מכן הוא מחזיר "ידית" (Handle) מיוחדת שניתן להשתמש בה לאחר יותר כדי להתייחס להצהרה המוכנה.

6.2.9.1 קישור משתנים (Binding Variables)

ישנם שני סוגי משתנים קשורים: משתני קלט (Input) הנקשרים להצורה, ומשתני פלט (Output) הנקשרים לסת התוצאות. עבור משתני קלט, עליהם לציין סימן שאלה כמצ"ן מקום (Placeholder) בהצרת ה-SQL:

?=SELECT Id, Country FROM City WHERE City
משתני קלט חייבים להיקשר **לפניהם** ביצוע ההצירה, בעוד משתני פלט נקשרים **לאחר** הביצוע.

תהליך משתני הקלט:

1. הכנה (nitoch) של ההצירה.
2. קישור משתני הקלט.
3. הקצאת ערכים למשתנים הקיימים.
4. ביצוע (Execute) ההצירה המוכנה.

תהליך משתני הפלט:

1. הכנה של ההצירה.
2. ביצוע ההצירה המוכנה.
3. קישור משתני הפלט.
4. שיליפת (Fetch) הנתונים לתוך משתני הפלט.

טבלה 6.5: פונקציות להצרות מוכנות ב-`mysqli`

תיאור	שם הפונקציה
מכינה הצהרת SQL לביצוע.	(...)(<code>mysqli_prepare</code>
קושרת משתנים להצירה (קלט). מצינית טיפוסים (o=מספר, s=מחרוזת וכו').	(...)(<code>mysqli_stmt_bind_param</code>
מבצעת את ההצירה המוכנה.	(...)(<code>mysqli_stmt_execute</code>
קושרת משתנים לסת התוצאות (פלט).	(...)(<code>mysqli_stmt_bind_result</code>
שולפת את הנתונים לתוך המשתנים הקיימים.	(...)(<code>mysqli_stmt_fetch</code>

דוגמה לשיליפת נתונים עם קישור פלט:

```

;("stmt = $mysqli->prepare("SELECT year, model, accel FROM alfas ORDER BY year$  

;                                ;()stmt->execute$  

;                                ;(stmt->bind_result($year, $model, $accel$  

;                                } ()while ($stmt->fetch  

;                                ;(printf("%d %s %.1f sec\n", $year, $model, $accel  

;                                {

```

BLOB 6.2.10

BLOB ראשי תיבות של Binary Large OBject ומתיחס לנתונים בינאריים, כגון תמונות JPEG המאוחסנות בסיס הנתונים.

6.2.10.1 הגדת נתונים BLOB

בעבר, נתונים BLOB הוכנסו ישירות כחלק מהשאלתה. ב-`mysql`, כאשר מכנים קבצים גדולים, על יותר להשתמש במתודה `data_send_long_(s)`. שיטה זו מאפשרת לשולח את הקובץ בחלקים (למשל 1KB בכל פעם) מוביל ש-`PHP` תצטרכ לאגור את כל הקובץ בזיכרון בבאת אחת.

SQLITE 6.3

`PHP` הציגה מנוע בסיס נתונים מובנה הזמן כברירת מחדל בשם SQLite 5.

6.3.1 חוזקוות וחולשות של SQLite

- **חזקקה: ללא שרת (Serverless) - SQLite** אינה משתמש במודול לקוח/שרת. היא מוטמעת באפליקציה ודורשת רק גישה לקבצי בסיס הנתונים.
- **חזקקה: קלות התחלת** - יצרת בסיס נתונים חדש אינה דורשת התערבות של מנהל מערכת.
- **חזקקה: קלת משקל ומהירה** - עבור רוב השאלות, הביצועים שלה משתווים או עולים על אלו של MySQL.
- **חולשה: בעיות הרחבה (Scaling)** - היעדר שרת מוביל לקשיים בנעילת קבצים וגבישה בו-זמןית של משתמשים רבים.
- **חולשה: טרנדזקציות נועלות את כל בסיס הנתונים - SQLite** נועלת את כל הקובץ בזמן כתיבה, מה שעלול לעכב קריאות בו-זמןיות.

6.3.3 ממשק PHP

`PHP` ב-5 מציעה גם ממשק פרוצדורלי וגם ממשק מונחה עצמים (OO). יצרת בסיס נתונים היא פשוט פתיחה קובץ; אם הוא לא קיים, `PHP` תיצור אותו.

טבלה 6.6: פתיחה וסגירה של בסיס נתונים

שם הפונקציה	תיאור
<code>(...)sqlite_open</code>	מחבר את הסקריפט לבסיס נתונים SQLite או יוצר אחד חדש.

מןתק את החיבור.	sqlite_close(...)
-----------------	--------------------------

6.3.3.2 שאילתות פשוטות

ב-SQLite ישנו רק שני טיפוסים פנימיים עיקריים: **INTEGER** (מספרים) ו-"**כל השאר**" (הדומה **VARCHAR** ויכול לאחסן טקסט ארוך מאוד). בኒיגוד להרחבות אחרות, SQLite מאפשר לבצע מספר שאילתות **CREATE TABLE** בקריאה אחת ל-**query**().

6.3.3.4 טרנסקציות

כברית מחדל, SQLite כתבת כל שאילתת לדייסק מיד, מה שולול להפוך הזנה של נתונים רבים לאייתית. כדי להאיץ את התהילה, ניתן להשתמש בטרנסקציות (**BEGIN** ו-**COMMIT**). בדוגמה של יבאו 638 אימילם, השימוש בטרנסקציה הוריד את זמן הייבוא מכ-60 דקות לפחות מ-2 דקות.

6.3.3.6 פונקציות מוגדרות משתמש (UDFs)

אחד התכונות המרשימות של SQLite היא יכולת לרשום פונקציות PHP כפונקציות SQL לכל דבר. ניתן להשתמש ב-**createFunction()** כדי לקשר פונקציית PHP לשאלילת SQL, ובכך לבצע עיבוד נתונים מורכב (כמו אינדוקס מילים) ישירות מתוך ה-SQL.

טבלה 6.10: פונקציות שאילתת ב-SQLite

תיאור	שם הפונקציה
מבצעת שאילתת פשוטה.	()sqlite_query
מבצעת שאילתת ללא אגירה בלוקו (חוסף זיכרון).	()sqlite_unbuffered_query
מבצעת שאילתת ומוחזירה רק את העמודה הראשונה מהרשומה הראשונה.	()sqlite_single_query

6.3.3.9 איטרטורים (Iterators)

ב-5 PHP ניתן לעבור על סט תוצאות של SQLite גם באמצעות איטרטור (**foreach** לולאת), דבר המהיר יותר מאשר שימוש בפונקציות שליפה (**fetch**) מסורתיות.

6.3.3.10 איטרציה עצמית (Homegrown Iteration)

כדי לראות בצורה ברורה יותר כיצד האיטרטור עובד פנימית, ניתן לבצע זאת גם ידנית (ללא הקסם של **foreach**), כפי שמצוג כאן בחלקו השני של הסקריפט:

```

" = details_query$"
(SELECT document_id, substr(doc.body, position - 20, 100
    FROM dictionary d, lookup l, document doc
        WHERE d.id = l.word_id
            ("AND word in ($words
                (AND document_id IN ($doc_ids
                    AND document_id = doc.id
                GROUP BY document_id, doc.body
                ;
";(result = $db->unbufferedQuery($details_query, SQLITE_NUM$
} } ()while ($result->valid
;()record = $result->current$;
:[list[$record[0]] = $record[1$;
;()result->next$;
{

```

כברית מחדל, ה-\$result מצביע על השורה הראשונה בעת תחילת האיטרציה, והמתודה `current()` מחזירה את הרשומה הנוכחית (באיינדקס לפי הסוג שצוין בפרמטר השני של `unbufferedQuery()`). באמצעות המתודה `next()`, ניתן להתקדם לרשומה הבאה בסט התוצאות. קיימות עוד מספר מתודות שנitin להשתמש בהן; הטבלה הבאה מציגה אילו מהן קיימות, וגם מונה את הפונקציות הפרוצדורליות המקבילות להן. הfrmeter הראשון בפונקציות הממשק הפרוצדורלי הוא תמיד ה"ידית" (Handle) של התוצאה, והוא אינו מופיע בטבלה .6.12

טבלה 6.12: פונקציות ומетодות לניצוט בסט תוצאות

תיאור	שם המתודה
מדרג לשורה מסוימת בסט התוצאות. הפרמטר היחיד הוא מספר הרשומה (מבוסס 0). ניתן לשימוש רק בשאלות נאגרות (Buffered).	<code>result->seek()</code> /\$ () <code>sqlite_seek</code>
מחזיר את מצביע התוצאה לרשומה הראשונה. ניתן לשימוש רק בשאלות נאגרות.	<code>result->rewind()</code> /\$ () <code>sqlite_rewind</code>
מתќדם לרשומה הבאה בסט התוצאות.	<code>result->next()</code> /\$ () <code>sqlite_next</code>
מחזיר את מצביע התוצאה לרשומה הקודמת. ניתן לשימוש רק בשאלות נאגרות.	<code>result->prev()</code> /\$ () <code>sqlite_prev</code>

מחזיר האם קיימות רשומות נוספות בסט התוצאות.	result->valid() /\$ (sqlite_valid)
מחזיר האם קיימת רשומה קודמת. לא ניתן לשימוש בשאלות לא נאגרות.	result->hasPrev() /\$ (sqlite_has_prev)

cutet נותר רק החלק האחרון של סקריפט החיפוש שלנו – החלק שבו אנו מוצאים את התוצאות בפועל:

PHP

```
        } (foreach ($rank as $record
        , "...echo $record[0], "\n=====\n";
        :"list[$record[0]], "...\\n-----\\n$"
        {
        <?
```

כאן, אנו פשוט עוברים שוב על תוצאות השאלה הראשונה ומשתמשים ב-ID של ההודעה כמפתח לסט התוצאות כדי להציג את החקקים הרלוונטיים מהאימיילים שנמצאו.

6.3.3.11 פונקציות נוספות הקשורות לסט התוצאות

ניתן להשתמש בכמה פונקציות ומתקודמות נוספות על סיטים של תוצאות. המתודה `numFields()` (או `sqlite_num_fields()`) מחזירה את מספר השדות בסט התוצאות, והמתודה `fieldName()` (או `sqlite_field_name()`) מחזירה את שם השדה. הפרמטר היחיד למתודה זו הוא האינדקס של השדה בתוך סט התוצאות (מבועו 0). אם אתם מבוקעים `JOIN` בין מספר טבלאות, שימו לב שפונקציה זו מחזירה את שם השדה "כפי שהוא" מהשאלתה; לדוגמה, אם השאלה מכילה `"SELECT a.field1 FROM address a"` שם השדה שיוחזר יהיה `"a.field1"`.

mozuot נסافت עם שמות עמודות, התקפה גם לגבי מפתחות במערכות מוחזרים כאשר האופציה `SQLITE_ASSOC` מוגדרת, היא שהם תמיד מוחזרים באותו `Case` (אותיות גדולות/קטנות) שבו הם נוצרו בהצורת ה-`"CREATE TABLE"`. על ידי הגדרת האופציה `sqlite_assoc_case` ב-`iini.php` ל-1, אתם מאלצים את הרחבות SQLite להחזיר שמות עמודות באותיות גדולות (Uppercase). הגדרה ל-2 תאלץ אותיות קטנות (Lowercase). הגדרה ל-0 (ברירת המחדל) לא תיגע ב-`Case`-`Insensitive` של שמות העמודות כל.

המתודה `numRows()` או `sqlite_num_rows()` מחזירה את מספר הרשומות בסט התוצאות, אך עובדת רק עבור שאלות נאגרות (Buffered).

6.3.3.12 פונקציות אגרגציה מוגדרות משתמש (Aggregate UDFs)

מלבד UDFs רגילים בדומה לאלו שהשתמשנו בהם כדי ליצור את האינדקס שלנו מתוך Trigger, ניתן גם להגדיר UDF עבור פונקציות אגרגציה (צבירה). בדוגמה הבאה, אנו מחשבים את אורך המילים המופיעים במילון שלנו:

PHP

```
)db->createAggregate$
```

```

        , 'average_length'
        , 'average_length_step', 'average_length_finalize'
    );

```

המתודה `createAggregate()` יוצרת את פונקציית האגרגציה שלנו. הפרמטר הראשון הוא שם הפונקציה לשימוש בשאלות SQL; השני הוא הפונקציה שמתבצעת עבור כל רשותה (הנקראת גם `step`); והפרמטר השלישי הוא שם הפונקציה שרצה כאשר כל הרשותות נבחרו.

PHP

```

)avg = $db->singleQuery$
;"SELECT average_length(word) FROM dictionary"
;"echo "$avg\n"
<?

```

כאן, אנו פשוט מרים את השאלה תוך שימוש בפונקציה החדשה שהגדרנו ומדפיסים את התוצאה, שאמורה להיראות如下:

.Average over 28089 words is 10.038 chars

6.3.3.13 קידוד תווים (Character Encoding)

ל-SQLite יש תמייה בשני סטימ של תווים: 1-ISO, שהוא ברירת המחדל ומשמש לרוב השפות הערב-איורופיות, ו-UTF-8. כדי להפעיל מצב UTF-8, עליכם להוראות לפקודת `configure --enable-sqlite-utf8`. אופציה זו משפיעה רק על מיון התוצאות.

6.3.3.14 כוונון ביצועים (Tuning)

כבר רأינו שניתן להאיץ כמויות גדולות של הזרנות נתונים (Inserts) על ידי עטייפת השאלות בטור Transaction. אך ישנו עוד כמה טריקים. בדרך כלל, בעת הזרנות נתונים רבים, איןנו מעוניינים לדעת כמה שינויים היו בסט התוצאות. SQLite מאפשר לCBCות את ספירת השינויים, מה שיפור את המהירות בזמן הזרנה:

`PRAGMA count_changes = 0`

טריק נוסף הוא לשנות את הדרך שבה SQLite מפרקנת (Flushes) נתונים לדיסק. עם ה-`synchronous`, ניתן לעבור בין המצביעים הבאים, כפי שמוצג בטבלה 6.13.

טבלה 6.13: אופציות "PRAGMA Synchronous"

מצב	תיאור

sqlite3_open_v2(<i>filename</i> , & <i>db</i> , SQLITE_OPEN_READONLY , <i>open_flags</i>) sqlite3_exec(<i>db</i> , "PRAGMA cache_size=5000") sqlite3_close(<i>db</i>)	OFF
sqlite3_open_v2(<i>filename</i> , & <i>db</i> , SQLITE_OPEN_READONLY , <i>open_flags</i>) sqlite3_exec(<i>db</i> , "PRAGMA journal_mode=wal") sqlite3_exec(<i>db</i> , "PRAGMA cache_size=5000") sqlite3_close(<i>db</i>)	ON/NORMAL (ברירת מחדל)
sqlite3_open_v2(<i>filename</i> , & <i>db</i> , SQLITE_OPEN_READONLY , <i>open_flags</i>) sqlite3_exec(<i>db</i> , "PRAGMA cache_size=5000") sqlite3_exec(<i>db</i> , "PRAGMA synchronous=NORMAL") sqlite3_close(<i>db</i>)	FULL

במצבים שבהם יש הרבה קריאות מסיסד הנתונים, כדאי להגדיל את גודל המטמון (Cache). בעוד שברירת המחדל היא 2,000 דפיים (דף הוא 1,536 בתים), ניתן להגדיל זאת כך:

```
;PRAGMA cache_size=5000
```

הגדירה זו תקפה רק לsoon הנוכחי. אם ברכונכם שהוא תישמר, עליכם להשתמש ב-default_cache_size.

במקום.

6.3.3.15 טריקים נוספים

נותרו עוד כמה דברים לגבי SQLite – למשל, מהי המתודה לשאול מבנה מסד הנתונים? התשובה קלה – באמצעות השאלתה הבאה:

```
SELECT * FROM sqlite_master
```

שאילתה זו מחזירה איבר אחד לכל אובייקט במסד הנתונים (טבלה, אינדקס וטריגר) עם המידע הבא: סוג האובייקט, שמו, הטבלה אליה הוא מקשר, ID, ושאלחת ה-SQL לייצרת האובייקט.

6.3.3.16 דברי חכמה

לסימן, הנה כמה דברי חכמה מאת מחבר מנוע SQLite, בהם הוא משתמש במקום הودעת זכויות יוצרים:

- מי יתן ותעשה טוב ולא רע.
 - מי יתן ותמצא מחלוקת עצמן ותסלח לאחרים.
 - מי יתן ותשתף בחופשיות, ולעולם לא תיקח יותר مما שאתה נותן.
- ד. ריצ'רד היפ

PEAR DB 6.4

חבורה ה-PEAR הנפוצה ביותר לגישה למסדי נתונים היא PEAR DB Abstraction Layer (Layer) המספקת API אחיד לשאול רוב מסדי הנתונים הנתמכים על ידי PHP, כמו גם דברים ספציפיים למסדי נתונים בדרך ניידת (Portable), כגון רצפים (Sequences) וטיפול בשגיאות.

6.4.2 יתרונות וחסרונות של הפשתת מסדי נתונים

שני היתרונות המרכזיים בשימוש בשכבת הפשתה כמסדי PEAR DB הם:

1. **API אחד שקל לזכור.** אתם פרודוקטיים יותר כשאתם מבקזחים פחות זמן בחיפוש בתיעוד.
2. **API אחד המאפשר לרכיבים אחרים להשתמש ב-API DB** לגישה גנרטיב ל-DBMS, מבלי לדאוג לפרטיטים הספציפיים של ה-Back-end.

מכיון ש-DB ממומש ב-PHP, יתרונות אלו מגיעים עם מהירות:

- שכבה הכתובה ב-PHP איטית יותר משימוש בfonkcioot PHP מובנות.
- שכבת הקוד הנוסף מוסיף מורכבות ומרקוט פונקצייאליים לשגיאות.

6.4.4 חיבור מסד נתונים

PEAR שואל את המונח (DSN) Data Source Name (DSN) כדי לתאר כיצד פונים למסד נתונים. MySQL משתמשים בפורמט URI. זה דוגמה ל-DSN המתיחס למסד נתונים מסווג mysql ב-localhost בשם :"world"

`mysql://user:password@host/world`

הפורט המלא של DSN הוא הרבה יותר מפורט, ורוב השדות הם אופציונליים. למעשה, רק שם הרחבות מסד הנתונים הוא חובה עבור כל הדריברים. הדריברים המצויפים ל-DB הם: dbase, fbsql, ibase, ifx, .sybase, msql, mssql, mysql, mysqli, oci8, odbc, pgsql, sqlite

האם תרצה שימושirk עם תרגום קטעי הקוד של PEAR DB או הסברים על פונקציות הקישור שלו?

הנה תרגום הפרק לעברית (tekst מתרגם בלבד):

6.4.4.2 ייצרת חיבור

הנה דוגמה כיצד ליצור חיבור למסד נתונים באמצעות PEAR DB:

```
PHP
;
;"require_once "DB.php
;"dsn = "mysql://test@localhost/test$"
;(dbh = DB::connect($dsn$)
} ((if (DB::isError($dbh
;"print "Error connecting: " . $dbh->getMessage() . "\n
;"print "Error details: " . $dbh->getUserInfo() . "\n
;(exit(1
{
;"print "Connect ok!\n
```

סקריפט זה מתחבר למסד הנתונים "test" באמצעות הרחבה mysql. שרת מסד הנתונים רץ על localhost, והחיבור יפתח כמשתמש "test" ללא סיסמה.

`DB.php` הוא הקובץ היחיד שעליוכם לכלול כדי להשתמש ב-DB PEAR. המתודה `DB::connect()` היא מתודת "מפעלי" (Factory method) שכוללת אוטומטית את הקובץ המתאים עבור הדריבר שלכם. היא יוצרת אובייקט דרייבר, מתחילה אותו וקוראת לפונקציה המובנית לייצור החיבור בפועל. במקרה של CISLON, היא תעורר שגיאת PEAR.

עבור מסדי נתונים של **SQLite**, כל שיעלים לציון הוא את הרחבות ה-PHP ואת קובץ מסד הנתונים: `sqlite:///test.db`. במקרה זה, הקובץ יפתח מהתיקייה הנוכחיית. כדי לציין נתיב מלא, יש להוסיף לוכסן נוסף לפני שם הקובץ: `sqlite:///var/lib/sqlite/test.db`.

6.4.4.3 אפשרויות הגדרה (Configuration Options)

ניתן להגדיר חלק מהתנהגות ה-DB לכל חיבור באמצעות המתודה `setOption()`. האפשרויות הנתמכות הן:

- **persistent**: (בוליאני) האם להשתמש בחיבור קבוע (Persistent connection).
- **ssl**: (בוליאני) האם להשתמש בחיבור SSL מאובטחים.
- **seqname_format**: (מחרוזת) פורתט השם עבור רצפים (Sequences) מודומים. בירית המחדל היא `seq%_s`.
- **autofree**: (בוליאני) האם לשחרר אוטומטית את סטי התוצאות לאחר סיום השאלות.
- **portability**: (מספר שלם) קובע אילו תכונות ה-DB צריך לדמות עבור ניידות בין מסדי נתונים שונים.

6.4.5 ביצוע שאלות

ישנן ארבע דרכים להרצת שאלות עם PEAR DB, כלן מבוצעות על ידי קרייה למתודות באובייקט החיבור:

1. **query(\$query, \$params = array())**: הדרך הסטנדרטית. אם יש תוצאות, היא מחזירה אובייקט תוצאה; אחרת, מחזירה ערך בוליאני המעיד על הצלחה.
2. **limitQuery(\$query, \$from, \$count, \$params = array())**: זהה ל-`query()`, אך מאפשר להגביל את מספר התוצאות ולהגדיר היסט (Offset).
3. **execute() ו-prepare()**: משמשות להכנות שאלתה וביצועה עם פרמטרים. זה חוסך זמן ניתוח (Parsing) של השאלתה כאשר מריצים אותה פעמים רבות עם נתונים שונים (למשל בלולאת `INSERT`).
4. **simpleQuery(\$query)**: מיועדת לשאלות מניפולטיביות נתונים שאינם מחזירות תוצאות. היא בעלת תקורה (Overhead) נמוכה במיוחד.

6.4.6 שליפת תוצאות

הכיתה `DB_result` מציעה מספר מצבים של שליפה (Fetch Modes):

- **DB_FETCHMODE_ORDERED**:מחזיר מערך עם אינדקס מספרי (בירית מחדל).
- **DB_FETCHMODE_ASSOC**:מחזיר מערך אסוציאטיבי שבו שמות העמודות הם המפתחות.
- **DB_FETCHMODE_OBJECT**:מחזיר אובייקט שבו שמות העמודות הם המאפיינים (Properties).

ניתן להגדיר את מצב השליפה לכל החיבור באמצעות `setFetchMode()` או לשנות אותו נקודתית בכל קרייה ל-`fetchRow()` או `fetchInto()`.

6.4.7 רצפים (Sequences)

רצפים בסיסיים נתונים הם עניין מורכב לנידות, כיוון שככל מסד נתונים (כמו MySQL או Oracle) מטפל בהם אחרת. PEAR DB מציע API אחיד ליצור מספרי זיהוי (ID) ייחודיים באמצעות המתודות:

- **nextId(\$seqname)**:מחזירה את המספר הבא ברכף.
- **createSequence(\$seqname)**: יוצרת רצף חדש.
- **dropSequence(\$seqname)**: מוחקת רצף.

6.4.8 תכונות ניידות (Portability)

החל מגרסת 1.6, ניתן להפעיל או לנכבות תכונות ניידות ספציפיות כדי לאזן בין ביצועים ליכולת מעבר בין מסדי נתונים:

- **DB_PORTABILITY_LOWERCASE**: הופך את כל שמות העמודות לאותיות קטנות.
- **DB_PORTABILITY_RTRIM**: מסיר רווחים מיוטרים מסוף מחזרות.
- **DB_PORTABILITY_DELETE_COUNT**: מאפשר לדעת כמה שורות נמחקו גם במסדי נתונים שאינם תומכים בכך כברירת מחדל.
- **DB_PORTABILITY_ERRORS**: מתקן מיפוי שגיאות כך שייהי עקבי בין כל הדרישים.

6.4.9 שגיאות מופשטות (Abstracted Errors)

במקומות להסתמך על קוד שגיאה שונות מכל שירות (MySQL, Oracle, DB MySQL וכו'), PEAR DB משתמש בסט קבועים משלו, כגון:

- **DB_ERROR_ACCESS_VIOLATION** (בעיית הרשות)
- **DB_ERROR_NOSUCHTABLE** (טבלה לא קיימת)
- **DB_ERROR_SYNTAX** (שגיאת תחביר SQL)

6.4.10 מethodות נוחות (Convenience Methods)

PEAR DB כולל מethodות לקיצור תהליכי שליפה נפוצים:

- **getOne()**:מחזירה רק את הערך הראשון מהשורה הראשונה.
- **getRow()**:מחזירה את כל השורה הראשונה כמערך.
- **getCol()**:מחזירה עמודה ספציפית מכל השורות כמערך.
- **getAssoc()**:מחזירה מערך אסוציאטיבי שבו העמודה הראשונה היא המפתח.
- **getAll()**:מחזירה את כל הנתונים מכל השורות כמערך של מערכים.

6.5 סיכום

פרק זה הציג שתי הרחבות מסדי נתונים חדשות ב-PHP 5: **sqlite** ו-**mysql**, וכן את **PEAR DB** כשבית הפשטה. למדתם:

- את יתרונות והחרונות של SQLite MySQL מול MySQL.
- מתי כדאי להשתמש בשכבה הפשטה למסדי נתונים.
- כיצד לבצע שאילתות, לשולף תוצאות ולטפל בשגיאות בכל אחת מהשיטות.
- את ההבדל בין שאילתות נאגרות (Buffered) ללא נאגרות.
- כיצד להשתמש בטריגרים ובפונקציות מוגדרות משתמש (UDF) ב-SQLite.

אם תרצה שאסביר בהרחבה על אחת המethodות של **PEAR DB** או אולי אראה דוגמה מורכבת יותר לשימוש ב-**prepare/execute**?

פרק 7

7.1 מבוא
טיפול בשגיאות

ניתן לצמצם את מספר השגיאות באפליקציה על ידי שימוש בפרקטיקות תכנות טובות; עם זאת, גורמים רבים הגורמים לשגיאות נמצאים מעבר לשילטתנו בסקריפט. הפסוקות רשת, דיסקים קשיחים מלאים, כשל בחומרה, באגים ברכיבי PHP אחרים או בתוכנות שהאפליקציה מתממשקת איתן – כל אלה יכולים לגרום לשגיאות שאין נובעות מכל אשמה בקוד ה-PHP שלו.

אם לא תעשו דבר כדי לטפל בשגיאות כאלה, התנהגות ברירת המחדל של PHP היא להציג את הودעת השגיאה למשתמש, יחד עם קישור לדף במדריך המתאר את הפונקציה שנכשלה, וכן את שם הקובץ והשורה בקוד שעוררו את השגיאה. עבור רוב השגיאות, PHP ממשיכה לרווח לאחר הצגת הודעה זו. ראו אior 7.1.

איור 7.1 הودעת שגיאה של PHP.

הודעת שגיאה זו מיועדת למעשה לכם, המפתחים, ולא למשתמשי האתר. המשתמשים יעריכו דף המסביר במונחים פשוטים מה השتبש, ואין להם עניין בקישורים לтиיעוד או במיקום הקוד שלכם בשורת.

PHP מספקת מספר אפשרויות לטיפול בשגיאות כאלה בצורה טובה יותר. לאחר שתסינוו לקרוא פרק זה, תלמדו:

- את סוג השגיאות השונים שבهم המשתמשים שלכם להיתקל.
- אילו אפשרויות עומדות בפניכם כמפתחים בתוך PHP לטיפול בהן.
- כיצד לכתוב מטפלי שגיאות (Error handlers) משלכם.
- המرة בין מנגנון דיווח שגיאות שונים.

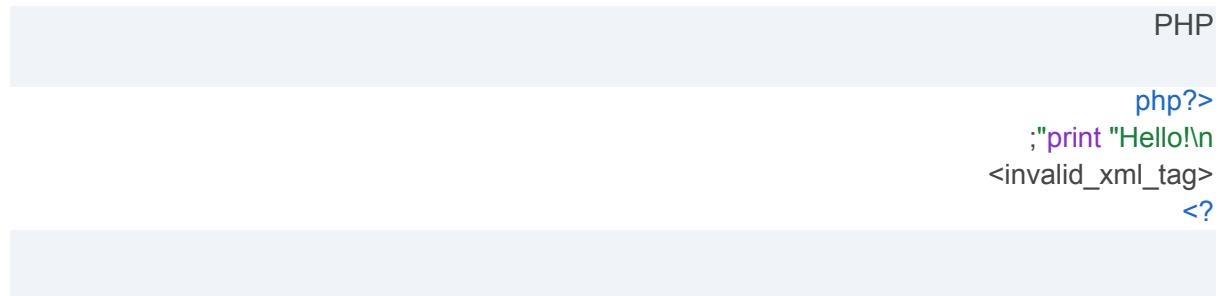
7.2 סוגי שגיאות

7.2.1 שגיאות תכנות

לפעמים שגיאות מתרחשות עקב טעויות בקוד שלנו. במקרים מסוימים, אלו השגיאות שהכי קל לטפל בהן מכיוון שניתן לחושף אותן לרוב על ידי בדיקות ישירות, פשוט על ידי ניסוי כל הפעולות שהאפליקציה מספקת. הטיפול בהן הוא פשוט עניין של תיקון הקוד.

7.2.1.1 שגיאות תחביר/ניתוח (Syntax/Parse Errors)

שגיאות תחביר ושגיאותניתוח הן נתקשות כאשר הקובץ עבר קומpileציה (הידור), עוד לפני ש-PHP מתחילה להריץ אותו בכלל.



```
PHP
php?>
;"print "Hello!\n<invalid_xml_tag>
?>
```

דוגמה זו מכילה תגית XML במקומ שבו PHP מצפה למצוא קוד. הריצת קוד זה תגרום לשגיאה:

Parse error: parse error in test.php on line 4

כפי שניתן לראות, הסקריפט אפיו לא הדפיס "!Hello" לפני הצגת הודעת השגיאה, כיוון ששגיאת התחביר הtgtلتה בזמן הקומPILEציה.

Eval 7.2.1.2

כל שגיאות התħħebir נတפסות בזמן הקומפילציה, למעט שגיאות בקוד המורץ באמצעות eval(). במקרה של eval, הקוד עובר קומפילציה במהלך הרצת הסkrift.

```
PHP
php?>
;"print "Hello!\n
;("eval("<invalid_xml_tag
<?

הפעם הפלט שונה:
!Hello
```

Parse error: parse error in /home/ssb/test.php(4) : eval()'d code on line 1

הפעם השגיאה הוצאה בזמן הריצה, כיון שהקוד המורץ ב-eval() אינו עבר קומפילציה עד שפעולת ה-eval()
עצמה מתבצעת.

Include / Require 7.2.1.3

אם הסkrift שלכם כולל קובץ אחר שיש בו שגיאת ניתוח (Parse error), הקומפילציה תיעצר בשגיאה. קוד והצהרות שקדמו לשגיאה יעברו קומפילציה, ואילו אלו שאחריה ייזרקו. זה אומר שתקבלו קובץ שחציו עבר קומפילציה.

7.2.2 סמלים לא מוגדרים (Undefined Symbols)

בזמן הריצה, PHP עשוי להיתקל בשמות של משתנים או פונקציות שהיא אינה מכירה. במקרה תħħebir, שגיאות לגבי סמלים לא מוגדרים מתרחשות בזמן שהקוד רץ.

7.2.2.1 משתנים וקבועים

משתנים וקבועים לא מוגדרים אינם גורמים ל垦יסה דרמטית, אלא רק להתראה (Notice):

```
PHP
php?>
;(var_dump($undefined_variable
;(var_dump(UNDEFINED_CONSTANT
;"print "Still alive!\n
<?

הפלט יהיה התראה על משתנה לא מוגדר שיקבל ערך NULL, והתראה על קבוע לא מוגדר ש-PHP תניח  
שהתכוונתם למחזרת בשם הקבוע. שימוש במשתנים לא מוגדרים אינם שגיאה טכנית ב-PHP, אלא פרקטיקת תכנות רשנית. אנו ממליצים להשייר את הדיווח על התראות (Notices) פעיל ולתקן את
```

המשתנים. באשר לקבועים לא מוגדרים – אלו הם באגים; אל تستמכו על כך ש-**PHP** הופכת אותם למחוזות, והקפידו לשים מחוזות בмирיאות.

7.2.2.2 אינדקסים במערך

אם ניגשים לאינדקס במערך שאינו קיים (למשל `$_name[GET]` כשלא נשלח פרמטר), **PHP** תציג: **Notice: Undefined index**.

7.2.2.3 פונקציות ומחלקות

בעוד ש-**PHP** ממשיכה לזרץ אחריו היטקלות במשתנה לא מוגדר, היא מסויימת מיד (**Abort**) כאשר היא נתקלת בפונקציה או מחלקה לא מוגדרת. זהה שגיאה קритית (**Fatal error**).

יצא דוף אחד למחלקות הוא הפונקציה `autoload`, הנקראת כאשר **PHP** נתקلت במחלקה לא מוכרת, ומאפשרת לטען אותה בזמן אמת.

7.2.3 שגיאות ניידות (Portability Errors)

7.2.3.1 הבדלים במערכות הפעלה

למרות ש-**PHP** רצה על פלטפורמות רבות, זה לא הופך את הקוד לאוטומטי לבלי תלויה בפלטפורמה. יש להתחשב בתווים המפרדים בין נתיבי קבצים (סלאש מול בקסלאש), פונקציות שצミニות רק בLINUX או רק בWIN32, ושירותים חיצוניים.

7.2.3.2 הבדלים בהגדרות PHP

קל להסתבר כמשמעות הנחות לגבי הגדרות `-no_magic_quotes_gpc`. דוגמה נפוצה היא `magic_quotes_gpc`; אם היא פעללה, **PHP** מוסיפה לוכנים אוטומטית לנוטונים חיצוניים. הדרך הנכונה היא לבדוק את ההגדרות בזמן ריצה באמצעות `get_magic_quotes_gpc()` ולהתאים את הקוד.

רשימת הגדרות חשובות לנידות:

- **register_globals**: קובע אם משתני GET/POST יוכנסו כמשתנים גלובליים. מומלץ להימנע מכך ולהשתמש ב-`$_GET` וחבריו.
- **magic_quotes_gpc**: מוסיף לוכנים אוטומטי. מומלץ לכבות זאת, אך קוד ניד חייב לדעת לטפל בכך בעזרת `stripslashes()`.
- **allow_url_fopen**: מאפשר לפונקציות קבצים לקרוא כתובות URL. אם זה כבוי, פועלות אלו ייכשלו.

האם תרצה שאפרט על פונקציות ספציפיות ב-Table 7.1 **শמייעדות לשרת Apache לעומת ממשק CLI**?

טבלה 7.1: פונקציות ספציפיות ל-SAPI

פונקציה	שכבות SAPI שמגדירות אותה

	apache_hooks	(מחולקת) ApacheRequest
	apache, apache_hooks, apache2filter	apache_lookup_uri
	apache, apache_hooks, apache2filter	apache_request_headers
	apache, apache_hooks, apache2filter	apache_response_headers
	apache, apache_hooks, apache2filter	apache_note
	apache, apache_hooks, apache2filter	apache_setenv
	apache, apache_hooks	apache_getenv
	apache, apache_hooks	apachelog
	apache, apache_hooks	apache_child_terminate
	apache, apache_hooks	apache_exec_uri
aolserver, apache, apache_hooks, apache2filter		getallheaders
milter		smfi_setflags
milter		smfi_settimeout

	milter	smfi_getsymval
	milter	smfi_setreply
	milter	smfi_addheader
	milter	smfi_chgheader
	milter	smfi_addrcpt
	milter	smfi_delrcpt
	milter	smfi_replacebody
apache, apache_hooks, apache2filter		virtual

7.2.3.4 התמודדות עם ניידות

שגיאות ניידות יכולות להיות קשות לאיתור מכיוון שהן דורשות בדיקה יסודית של הקוד בתוצאות שונות ועל מערכות שונות. עם זאת, בדיקות נאותות וסיקירות קוד הן הדריכים הטובות ביותר למציאת בעיות ניידות.

כמובן, אם אתם כתבים ומפיצים את כל הקוד שלכם על אותה פלטפורמה עם תצורה אחידה, יתכן שלא יהיה לא תקין במערכות ניידות. מודעות לנושאי ניידות היא דבר חיוני בכל מקרה; היא מאפשרת לכם לכתוב קוד טוב יותר, עמיד יותר ונitin לשימוש חוזר.

תיקון שגיאות ניידות עשוי להיות פשוט, כמו בדיקת הגדרות ה-`h`-וּן בדוגמה הקודמת של `magic_quotes_gpc`, אך הוא יכול להיות גם מורכב יותר. יתכן שתצטרכו לנתח פלט של פקודה בצוරה שונה עבור מערכות הפעלה שונות, או לספק מימוש חלופי ב-PHP עבור רכיב זמן רק בחלק מהפלטפורמות.

במקרים מסוימים, מה שאתם מנסים לעשות פשוט אינו אפשרי בדרך ניידת. באופן כללי, הגישה הטובה ביותר לביעות ניידות היא הסתרת פרטי מערכת הפעלה או ה-`SAPI` בתוך שכבת קוד המפשיטה (`Abstracting`) את הבעיה. דוגמה אחת להפשתה צזו היא מחלוקת `PEAR System`, המספקת מימוש PHP לכמה פקודות UNIX נפוצות ופעולות אחרות שהן ספציפיות למערכת הפעלה.

7.2.3.5 כל נידות

- מחלקת System: PEAR: מחלקת זו זמין חלק מהתקנת הבסיס של PEAR.
- מחלקת OS_Guess: מחלקת זו משתמשת בפונקציה `uname_r()` כדי לקבוע על איזו מערכת הפעלה היא רצה. היא גם מספקת דרכים להכללה והשווואה של חתימות מערכות הפעלה:

```
PHP
php>
;"require_once "OS/Guess.php
;os = new OS_Guess$(
;"print "OS signature: " . $os->getSignature() . "\n
} ((if ($os->matchSignature("linux-*i386
;"print "Linux running on an Intel x86 CPU\n
{
<?
```

פלט לדוגמה:

OS signature: linux-2.4-i386-glibc2.1

Linux running on an Intel x86 CPU

7.2.4 שגיאות זמן ריצה (Runtime Errors)

ברגע שהקוד פועל, שגיאות זמן ריצה שאין קритיות (Non-fatal) הן סוג השגיאה הנפוץ ביותר ב-PHP. המונח "זמן ריצה" מתייחס לשגיאות המתרחשות במהלך הקוד, לרבות אין טיעיות תכונות אלא גורמות מגורמים מחוץ ל-PHP עצמו, כגון פעולות דיסק, רשת או קריאות למסד נתונים.

ל-PHP יש מנגנון דיווח שגיאות המשמש לכל השגיאות שמתעוררות בתחום PHP, בין אם בזמן הידור הסקריפט או בעת ביצוע פונקציה מסוימת. ניתן להשתמש במנגנון זה גם מתוך סקריפט, אם כי קיימות דרכים חזקות יותר לדיווח על שגיאות (כגון חריגות - Exceptions).

דוגמאות לשגיאות זמן ריצה מתרחשות כאשר `fopen()` נכשל כי קובץ חסר, כאשר `mysql_connect()` נכשל בגלל שם משתמש שגוי, או אם ניסיתם להזין שורה לטבלה מבלי לספק ערך לעמודה שמוגדרת כ-null-Not.

7.2.5 שגיאות PHP

מנגנון השגיאות ב-PHP משמש את כל הפונקציות המובנות. כבירית מחדל, מנגנון פשוט זה מדפיס הודעה שגיאה עם שם הקובץ ומספר השורה, ויצא מהתוכנית.

7.2.5.1 רמות שגיאה (Error Levels)

שגיאות PHP מחולקות לרמות, החל מהתראות (Notices) ועד שגיאות קritisיות (Fatal errors). רמת השגיאה מעידה על חומרתה.

- **E_ERROR**: שגיאה קריטית שלא ניתן להתאושש ממנה (למשל חריגת זיכרון).
- **E_WARNING**: הסוג הנפוץ ביותר. מעיד שימושו שניסיתם לעשות השتبש (למשל חסר פרמטר בפונקציה או חלוקה באפס).
- **E_PARSE**: שגיאת ניתוח בזמן ההידור, הגרמת ל-PHP לעזoor לפני הריצה.
- **E_STRICT**: נועד להקל על המעבר מ-4 PHP ל-5 PHP על ידי הצעות לשיפור הקוד.
- **E_NOTICE**: מצין שהקוד עשה משהו שעלול להיות לא נכון (כמו קריית משתנה לא מוגדר).
- **E_CORE_ERROR / E_COMPILE_ERROR**: שגיאות פנימיות בזמן העלייה או ההידור של PHP.
- **E_USER_ERROR / WARNING / NOTICE**: שגיאות המוגדרות על ידי המשתמש/הסקריפט ולא על ידי PHP עצמה.

7.2.5.2 דיווח שגיאות (Error Reporting)

מספר הגדרות ב-`ini.php` שולטות אילו שגיאות יוצגו ויכיז:

- **error_reporting**: קובע אילו רמות של שגיאה ידועו (למשל `E_ALL`).
- **display_errors**: שולט האם להציג שגיאות כחלק מהפלט בדף.
- **log_errors**: שולט האם לתרום שגיאות בקובץ לוג. הכתובת נקבעת ב-`error_log`.
- **html_errors**: קובע האם להוסיף פורמט HTML להודעות השגיאה.

7.2.5.3 מטפיל שגיאות מותאמים אישית

במוקם ש-PHP תדפיס את השגיאה, ניתן לרשום פונקציה שתופעל בכל שגיאה. כך תוכלן לתרום שגיאות לממד נתונים או לשלוח התראות בדואיל.

7.2.5.4 השתקת שגיאות (Silencing Errors)

ניתן להשתיק שגיאות באמצעות התו `@` לפני ביטוי. זה יגרום לכך שההודעת השגיאה לא תוצג, אך המנגנון המובנה עדין יופעל. יש להיזהר בשימוש בזה עם מטפיל שגיאות מותאמים אישית.

7.3 שגיאות PEAR

ל-PEAR מנגנון דיווח משלו המבוסס על העיקרונות ששגיאות הן אובייקטים שונים להעיבר כערכים.

```
PHP
php?>
;"require_once "DB.php
;("dbh = DB::connect("mysql://test@localhost/world$"
} ((if (PEAR::isError($dbh
;("die("Connection failed (" . $dbh->getMessage() . "\n"
{
;"print "DB::connect ok!\n
<?
```

בדוגמה זו, אם החיבור נכשל, המתוודה מחזירה אובייקט שגיאת PEAR. אנו בודקים זאת בעזרת `PEAR::isError()` ושולפים את ההודעה באמצעות `getMessage()`.

()PEAR::isError 7.3.0.2

מתודת סטטית המחזירה אמת אם המשתנה שהועבר לה הוא אובייקט מסווג PEAR_Error או תת-מחלקה שלו.

7.3.0.3 העלאת שגיאות (Raising Errors)

בטרמינולוגיה של PEAR, שגיאות "מעולות" (Raised), אם כי הדריך הקלה ביזור להעלות שגיאת PEAR היא להחזיר את ערך החזרה ממתודה שנקראת throwError. זאת פשוט מפני ש-throwError היא גרסה מופשטת של המתודה המקורית PEAR raiseError משתמשת במונח "הعلاה" כדי למנוע הבלבול עם חריגות (Exceptions) של PHP, אשר אותן "זורקים" (Thrown).

העלות היחסית של העלאת שגיאת PEAR בהשוויה לעירור שגיאת PHP היא גבוהה, כיוון שהיא קרוכה ביצירת אובייקט ומספר קריאות לפונקציות. המשמעות היא שעיליכם להשתמש בשגיאות PEAR בזיהוות – שמרו אותן לכשלים שלא אמרו לkerot בדרכן כלל. העדיף להשתמש בערך חזרה בוליאני פשוט למקרים הרגילים. עצה זו תקפה גם לגבי שימוש בחריגות ב-PHP, כמו גם ב-C++, Java או שפות אחרות.

כאשר אתם משתמשים בחבילות PEAR בקוד שלכם, עליכם להתמודד עם שגיאות שהועלו על ידי החבילה. ניתן לעשות זאת באמצעות דרכיהם: כתלות בשאלת האם אתם נמצאים בהקשר של אובייקט (Object), והאם המחלקה הנוכחית שלכם יורשת ממחלקה ה-PEAR (context).

אם הקוד שלכם אינו רץ בהקשר של אובייקט, למשל מtopic הטווח הגלובלי (Global scope), תוך פונקציה רגילה או במתודה סטטית, עליכם לקרוא לממתודה הסטטית ()PEAR::throwError:

```
PHP
php?>

:require_once 'PEAR.php

} (((if (PEAR::isError($e = lucky
;"die($e->getMessage()) . "\n

{

;"print "You were lucky, this time.\n

} ()function lucky

} (if (rand(0, 1) == 0
;(!return PEAR::throwError("tough luck
{
{
<?
```

כאשר שגיאות מועלות באמצעות קריאות למתודה סטטית, מוחלות הגדרות ברירת המחדל שנקבעו עם `PEAR::setErrorHandling()`. הדרך השניה להעלאת שגיאות היא כאשר המחלקה שלכם יורשת מ-PEAR ויהood שלכם מבוצע בהקשר של אובייקט:

PHP

```
php?>

'require_once 'PEAR.php

class Luck extends PEAR

}

} ()function testLuck

} (if (rand(0, 1) == 0

;(!return $this->throwError('tough luck

{

;!"return "lucky

{

{

;luck = new Luck$

;()test = $luck->testLuck$

} ((if (PEAR::isError($test

;("die($test->getMessage() . "\n

{

;"print "$test\n

<?
```

כאשר `throwError()` נקראת בהקשר של אובייקט, הגדרות ברירת המחדל שנקבעו באותו אובייקט עם `object->setErrorHandler()` מוחלות תחילה. אם לא נקבעו ברירות מחדל לאובייקט, יחולו ברירות המחדל הגלובליות.

PEAR::throwError() 7.3.0.4

[[object PEAR::throwError([string message], [int code], [string userinfo])

מתודת זו מעלה שגיאת PEAR תוך החלת הגדרות בירית המחדל לטיפול בשגיאות. אילו ביריות מחדל מוחלחות בפועל תלוי באופן הקריאה למתודה: בסטטיות (גלובלי) או מתור אובייקט (מוקמי לאובייקט).

אם איןכם בקיאים בסמנטיקה של `this` ב-PHP, אתם עשויים להיתקל בהפטעות. אם תקראו למתודה סטטית מתור אובייקט (שבו ל-`this` יש ערך), הערך של `this` יהיה מוגדר גם בתוך המתודה שנקראה סטטית. המשמעות היא שאם תקראו ל-`PEAR::throwError()` מתור אובייקט, `this` יתיחס לאותו אובייקט והגדירותיו המקומיות יחולו לפני הגלובליות.

PEAR::raiseError() 7.3.0.5

object PEAR::raiseError([string message], [int code], [int mode], [mixed options], [[string userinfo], [string error_class], [bool skipmsg]

מתודה זו שקופה ל-`throwError()` אך עם פרמטרים נוספים. בדרך כלל לא תזדקקו לכל האפשרויות הללו, אך הן עשויות להועיל אם אתם בונים מערכת שגיאות משלכם המבוססת על PEAR.

- **error_class**: המחלקה שתשתמש לאובייקט השגיאה (בירית מחדל "PEAR_Error").
- **skipmsg**: פרמטר המורה להתעלם מהודעת השגיאה, שימושי למנגנוני המבוססים על קוד שגיאה בלבד.

PEAR_Error 7.3.1

מחלקה זו היא בסיס דיווח השגיאות של PEAR. ניתן להרחיב אותה לצרכים ספציפיים, ו-`PEAR::isError()`. הפרמטרים כוללים:

- דרך כלל לא יוצרים שגיאות PEAR עם `new`, אלא דרך `throwError()` או `raiseError()`. הפרמטרים כוללים:
- **message**: הודעת השגיאה.
 - **code**: קוד שגיאה (מספרשלם). מומלץ תמיד לציין הקוד שגיאה לטיפול נקי יותר.
 - **mode**: מצב השגיאה (למשל: RETURN, PRINT, DIE, TRIGGER, CALLBACK).
 - **options**: פרמטר המשתנה לפ' המצב (למשל, פורט `printf` או שם פונקציית CALLBACK).
 - **userinfo**: מידע נוסף (כמו שאלתת ה-SQL שנכשלה).

:PEAR_Error בחרות ב-

- **getMessage() / getCode()**: החזרת ההודעה או הקוד.
- **getUserInfo()**: החזרת המידע הנוסף.
- **getBacktrace()**: החזרת היסטוריית הקרייאות (Stack trace) מרגע יצירת השגיאה.

PEAR 7.3.2

בירית מחדל, שגיאות PEAR רק מוחזירות אובייקט. ניתן לקבוע "מצב שגיאה" שיחול על כל השגיאות הבאות:

```
php?>
```

```
;require_once 'DB.php
```

```
;("PEAR::setErrorHandling(PEAR_ERROR_DIE, "Aborting: %s\n
```

```
;('dbh = DB::connect('mysql://test@localhost/test$
```

```
;'"print "DB::connect ok!\n
```

```
<?
```

בדוגמה זו השתמשנו ב-`PEAR_ERROR_DIE`, מה שגורם לתוכנית להפסיק את השגיאה ולהיעזר מיד במקרה של כישלון בחיבור.

PEAR::setErrorHandling() 7.3.2.1

קובע את ברירות המחדל לטיפול בשגיאות. קריאה סטטית קבועה ברירות מחדל גלובליות, וקריאה מתוור אובייקט (כמו `$dbh->setErrorHandling()`) קבועת אותו עברו אותו אובייקט בלבד.

7.3.3 מצבים של PEAR (Error Modes)

- .1 **PEAR_ERROR_RETURN**: לא עושה דבר מלבד החזרת אובייקט השגיאה (ברירת המחדל).
- .2 **PEAR_ERROR_PRINT**: מדפיס את ההודעה אוטומטית.
- .3 **PEAR_ERROR_DIE**: מדפיס את ההודעה ויוצא מהתוכנית.
- .4 **PEAR_ERROR_TRIGGER**: מעביר את ההודעה לפונקציית `trigger_error()` המובנית של .PHP
- .5 **PEAR_ERROR_CALLBACK**: מפעיל פונקציה חיצונית שהוגדרה מראש.

7.3.4 טיפול אלגנטיבי (Graceful Handling)

PEAR::pushErrorHandler() 7.3.4.1

דוחף מצב טיפול בשגיאות חדש לראש "מחסנית" המתפלים. המצב הזה יהיה פועל עד שתיקרא המתודת `popErrorHandler()`. זה שימושי לשינוי זמני של התנהגות השגיאות (למשל, מכיבוי התוכנית לחזרת ערך בלבד) בזמן ביצוע פעולה ספציפית.

PEAR::expectException() 7.3.4.3

אפשר להגדיר קוד שגיאה או הודעות ספציפיות שאנו "מצפים" להם. עבור שגיאות אלו, מצב הטיפול ייכפה ל-`PEAR_ERROR_RETURN` באופן זמני, מה שמאפשר לטפל בהן ידנית בקוד מביל' להפעיל את המתפל הגלובלי (כמו `DIE`).

7.4 חריגות (EXCEPTIONS)

7.4.1 מהן חריגות?

חריגות הן מנגנון שגיאות מובנה ברמה גבוהה שהוצע ב-5 PHP. חריגות הן אובייקטים שנייתים ("לזרוק Throw"). אם משהו מוקן "لتפוע" (Catch) את החריגה, היא מטופלת; אם לא, PHP עוצרת עם שגיאה קритית ומציגה Stack trace.

throw ,catch ,try 7.4.2

שלושה מבני שפה משמשים בחריגות:

1. **try**: בלוק קוד שבו PHP "מחפשת" חריגות.
 2. **catch**: בלוק קוד המבוצע אם נזרקה חריגה מסווג מסוים בתוך ה-**try**.
 3. **throw**: הצהרה המשמשת ליצור וזריקת חריגה חדשה.

```
PHP  
 } try  
 ;("fp = @fopen($filename, "r"  
 } ((if (!is_resource($fp  
 ;("throw new FileException("could not read '$filename  
 {  
 } (catch (FileException $e {  
 ;((die($e->getMessage  
 {
```

ניתן להגדיר תת-מחלקות של `Exception` כדי להפריד בין סוגים שונים (למשל `Exception_O` לעומת `Exception_X`). אם חריגה אינה נתפסת באף בLOC `catch`, היא עוברת לפונקציה הקוראת. אם היא לא נתפסת כלל, ניתן להגדיר מטפל סופי באמצעות `set exception handler` (set).

7.5 סיכום

בפרק זה למדתם על סוגי השגיאות השונים ש-PHP ו-PEAR יכולות לחול ולנהל. למדתם כיצד להתאים אישית את הטיפול בשגיאות באמצעות הקובץ `error.php`, לכתוב מטפל שגיאות משלכם ולהמיר שגיאות PHP לשגיאות PEAR או לחריגות (Exceptions).

למדתם על הביעות שועלות להיגרם כתוצאה מהבדלים בין ממשק השרת (מודולי SAPI) ומערכות הפעלה שונות, ועל כמה דרכים להתמודדות עם סוגיות הניידות (Portability).

לבסוף, למדתם כיצד להשתמש בצורה הטובה ביותר בחריגות ב-PHP ואת הפרטים הספציפיים לגבי שימוש
בחריגות עם PEAK.

בזמן כתיבת שורות אלו, קהילת PEAR עדין מגבשת את הדרך הטובה ביותר להציג ולהשתמש בחיראות בקורס PEAR, ולכן השימוש בחיראות עם PEAR הושם במכoon מהדורה זו של הספר. עקבו אחר עדכונים באתר האינטרנט של הספר בכתובת <http://php5powerprogramming.com>!

פרק 8

PHP 5 עם XML

8.1 מבוא

XML צוברת תאוצה כשפה אוניברסלית לתקשורת בין פלטפורמות; יש המכנים אותה "מהפכת האינטרנט החדשה". לעיתים משתמשים ב-XML כדי לשמור נתונים לאחסון מסמכים, אך אחסון נתונים מעולם לא היה מטרתה העיקרית. היא פותחה כדי להעביר מידע ממכלול אחורת בפורמט משותף.

XML היא שפת תגיות (Tagged language). הנתונים עצם מוכלים בתוך אלמנטים מובנים ומתייחסים במסמך. יש לנתח (Parse) את מסמך-XML כדי להציג את המידע. לעיתים קרובות, יש להמיר את המידע לפורמט אחר. בפרק זה נתמקד בשימוש ב-PHP לקריאה ושינוי (Transformation) של מסמכי XML, ובשימוש ב-XML כפרוטוקול תקשורת מול שירותים מרוחקים.

לאחר סיום פרק זה, תלמדו:

- את המבנה של מסמך XML.
- את הטרמינולוגיה הדרישה לעבודה עם מסמכי XML.
- כיצד לנתח קובץ XML באמצעות שיטות המרכזיות: **DOM** ו-**SAX**.
- כיצד לנתח קובץ XML פשוט בדרך קלה יותר: הרחבת **SimpleXML** של PHP.
- כיצד להשתמש בחבילות PEAR שימושיות עבור XML.
- כיצד להמיר מסמך XML לפורמט אחר באמצעות **XSLT**.
- כיצד לשփר מידע בין מערכות באמצעות XML.

8.2 אוצר מילים

בעבודה עם מסמכי XML, תיתקלו במספר מונחים שעשויים להיות לא מוכרים. הדוגמה הבאה מציגה מסמך XML שהוא מסמך XHTML:

```
XML
<? "xml version="1.0" encoding="ISO-8859-1?">
  DOCTYPE html!
  "PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   <"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <title>XML Example</title>
      <head/>
    <"body background="bg.png">
      <p>
        .<Moved to <a href="http://example.org/">example.org</a
          </ br>
```

```
foo &bar;  
<p/>  
<body/>  
<html/>
```

- **הצהרת XML:** השורה הראשונה מצינית את גרטת-XML וקידוד הקובץ. שימוש לב שהיא מתחילה ב-<?. זה עשוי ליצור בעיה אם הגדרת short_open_tag ב-PHP פעללה, שכן PHP תחשוף שמדובר בתחילת קוד PHP. מומלץ לנכונות הגדרה זו ב-.ini.php.
- **הצהרת DOCTYPE:** מצינית את אלמנט השורש (html) ומפנהו לקובץ DTD (הגדרת סוג מסמך).
- קובץ DTD מתאר את מבנה המסמך.
- **Well-formed vs. Valid:** מסמך Well-formed (בנוי כהלכה) הוא צזה העומד בחוקי התחריר של XML. מסמך Valid (תקף) הוא צזה שגם توאמ את הכללים הספציפיים שהוגדרו ב-DTD שלו.
- **אלמנט שורש (Root Node):** האלמנט העוטף את כל המסמך (בדוגמה שלנו: <html>).
- **מרחב שמות (Namespace):** מוגדר ע"י xmlns. מאפשר לערבות סוגים שונים (כמו MathML בתוך XHTML) ללא התנגשויות.
- **תכונות (Attributes):** מידע נוספת בתוך התגית (כמו `background="bg.pgn.kc.pch"`). ב-XML חובה לכל תוכנה יהיה ערך והוא חייב להיות בתוך מירכאות.
- **ישויות (Entities):** תווים מיוחדים כמו > או & חיברים להיכתב כ-> ו-& בהתקאה, כדי שהמנתח לא יחשוב שמדובר בתגיות.

8.3 ניתוח XML (Parsing)

- קיימות שתי טכניקות עיקריות לניתוח XML ב-PHP:
1. **XSimple API for XML (SAX):** המנתח עובר על המסמך ויורה "אירועים" (Events) עבור כל תגית התחלה, סוף או תוכן. אתם מחליטים איך לטפל בכל אירוע. חסכו בזיכרון.
 2. **Document Object Model (DOM):** הקובץ יכול נתען לזכרון עצם של אובייקטים שניתן לנווט ביניהם. קל יותר לשימוש אך צורך זיכרון רב.

8.3.1 SAX

- ב-SAX, עליינו להגדיר "מטפליים" (Handlers) לפונקציות שלנו.
- `xml_parser_create`: יצירת אובייקט המנתח.
 - `xml_set_element_handler`: הגדרת הפונקציות שיירצו בתחילת ובסוף תגית.
 - `xml_set_character_data_handler`: הגדרת הפונקציה שתטפל בטקסט שבתוך התגיות.

בדוגמה שבספר, המנתח מציג את מבנה התגיות עם הזחחה (Indentation) לפי עומק התגית. מכיוון ש-SAX עשוי לקרוא את הטקסט בחלוקת, נהוג להשתמש ב"באפר" (Buffer) כדי לאוסף את כל הטקסט ורק אז להציגו בצורה נקייה.

8.3.2 DOM

ניתוח קובץ פשוט עם SAX דורש עבודה רבה. שיטת ה-DOM קלה בהרבה, אך המחיר הוא בצריכת הזיכרון. בעוד שזה לא מרגש בקבצים קטנים, ניתוח קובץ של 20MB בשיטת DOM עשוי להיות כבד מאוד. DOM יוצר עצ בזיכרון המציג את הקובץ.

כפי שניתן לראות באירור 8.1, עז ה-DOM מייצג את המבנה ההיררכית של הקובץ. אנו יכולים להציג את כל התוכן ללא התגיות על ידי מעבר על עצ האובייקטים. בדוגמה זו אנו עושים זאת על ידי מעבר רקורסיבי על כל הילדים של האמתים:

```
PHP
php?>

;();$dom = new DomDocument$()

;(;$dom->load('test2.xml$

;,$root = $dom->documentElement$

;

;(process_children($root

(function process_children($node

)

;,$children = $node->childNodes$

;

} (foreach ($children as $elem

} (if ($elem->nodeType == XML_TEXT_NODE

) (((if (strlen(trim($elem->nodeValue

;"echo trim($elem->nodeValue)."\n

{

} (else if ($elem->nodeType == XML_ELEMENT_NODE {

; (process_children($elem

{

{

{

<?
```

הפלט יהיה:

XML Example

Moved to
example.org

foo & bar

דוגמא זו מציגה שימוש DOM בסיסי מאד. בשורה 4 אנו שולפים את אלמנט השורש של המסמך. עבור כל אלמנט שאנו פוגשים, אנו קוראים ל-`process_children()` אשר עברת על רשימה של צומת הבן. אם הצומת הוא צומת טקסט, אנו מדפיסים את ערכו, ואם הוא אלמנט, אנו קוראים לפונקציה באופן רקורסיבי. הרחבות ה-DOM חזקה בהרבה ממה שמצוג כאן וממשת忿ען את כל הפונקציונליות של מפרט 2 DOM.

הדוגמה הבאה משתמשת במתודה `getAttributeNode()` כדי להחזיר את התכונה `background` של תגית `:body`:

PHP

`php?>`

```
;()$dom = new DomDocument$  
;('dom->load('test2.xml$  
;root = $dom->documentElement$  
  
;(process_children($root  
  
(function process_children($node  
}  
  
;$children = $node->childNodes$  
  
} (foreach ($children as $elem  
} (if ($elem->nodeType == XML_ELEMENT_NODE  
} ('if ($elem->nodeName == 'body  
;"echo $elem->getAttributeNode('background')->value . "\n  
{
```

```
:process_children($elem  
{  
{  
{  
<?
```

אנו עדין צריכים לחפש רקורסיבית בעץ, אך כיוון שאנחנו מכירים את מבנה המסמך, ניתן לפשט זאת:

```
PHP  
php?>  
;()$dom = new DomDocument$  
;('dom->load('test2.xml$  
;($body = $dom->documentElement->getElementsByTagName('body')->item(0$  
;"echo $body->getAttributeNode('background')->value . "\n  
<?
```

שורה 4 היא העיקרית: אנו מבקשים את כל אלמנטי הבן עם שם התגית `body`, לוקחים את הprixt הראשון ברשימה, ובשורה 5 שולפים ומציגים את ערך התכונה `.background`.

8.3.2.1 שימוש ב-XPath

באמצעות XPath נוכל לפשט את הדוגמה עוד יותר. XPath היא שפת שאלות למסמכי XML (בדומה ל-SQL למסדי נתונים):

```
PHP  
php?>  
;()$dom = new DomDocument$  
;('dom->load('test2.xml$  
;($xpath = new DomXPath($dom$  
;($nodes = $xpath->query("*[local-name()='body']", $dom->documentElement$  
;"echo $nodes->item(0)->getAttributeNode('background')->value . "\n  
<?
```

8.3.2.2 ייצירת עץ DOM

הרחבת ה-DOM יכולה גם ליצור מסמך XML מופיע על ידי בניית עץ אובייקטים ושמירתו:

PHP

php?>

```
;()$dom = new DomDocument$  
;("html" = $dom->createElement('html'$  
;("html->setAttribute("xmlns", "http://www.w3.org/1999/xhtml$  
;("html->setAttribute("xml:lang", "en$  
;("html->setAttribute("lang", "en$  
;($dom->appendChild($html$  
  
;('head' = $dom->createElement('head'$  
;($html->appendChild($head$  
;('title' = $dom->createElement('title'$  
;(("title->appendChild($dom->createTextNode("XML Example$  
;($head->appendChild($title$  
  
/* ייצירת גוף המסמך ותוכן נוסף באותה שיטה... */
```

;()\$echo \$dom->saveXML

<?

SimpleXML 8.4

הרחבת SimpleXML היא הדרך הקלה ביותר לעבוד עם XML ב-PHP. אין צורך לזכור API מסובך; פשוט ניגשים לנוטונים כאל מבנה נתונים. ארבעת הכללים הפחותים הם:

1. **מאפיינים (Properties)** מייצגים איטרטורים של אלמנטים.
2. **אינדקסים מספריים** מייצגים אלמנטים.

3. **אינדקסים שאינם מספריים מייצגים תוכנות (Attributes).**
4. **הمرة למחוזת מאפשרת גישה לנוטני טקסט (TEXT).**

8.4.1 ייצור אובייקט SimpleXML

ניתן ליצור אובייקט בשלוש דרכים: מקובץ (`simplexml_load_file`), ממחוזת (`simplexml_load_string`) או מאובייקט DOM (`simplexml_load_dom`).

8.4.2 דפוד בובייקטי SimpleXML

- `lolaha` על כל תגיות ה-`k` בתור ה-`$k`.
- `gisha` לתגית ה-`k` השנייה: `[$sx->body->p[1$]`.
- `gisha` לתוכנה `['background: $sx->body['background`.
- `hagat htekst` של תגיota: `[$echo $sx->body->p[1->p]`.

כדי לכלול צמי'i בן בטקסט, ניתן להשתמש ב-`XML(as)` בשילוב עם `strip_tags()`. למעבר על כל ידי צומת משתמשים ב-`children()`, ולמעבר על תוכנות משתמשים ב-`attributes()`.

PEAR 8.5

PEAR מכילה מחלקות לטיפול ב-XML כאשר הרחבות ה-DOM אין זמינות או למשימות ספציפיות:

- **XML_Tree:** שימושי לבניית מסמכי XML בצורה מהירה ופושאה יותר מ-DOM.
- **XML_RSS:** מחלקה ייעודית לניטוח קבצי RSS (עדכוני חדשות).

XML_RSS 8.5.2

RSS הוא אוצר מילימ של XML לתיאור פרטיה חדשות. מחלקה `XML_RSS` מאפשרת לקחת קובץ RSS (כמו זה של `net.php`), לנתח אותו ולשЛОפ בקהלת את הכותרות, הקישורים והתייאורים של הכתבות באמצעות `getChannelInfo()` ו-`getItems()`.

8.6 המרת XML

יתכן שתרצו להמיר מסמך XML למשהו אחר, כגון מסמך HTML, קובץ טקסט או קובץ XML בפורמט שונה. השיטה הסטנדרטית להמרת מסמך XML לפורמט אחר היא באמצעות **XSLT** (שפת גילונות סגנון מורחבת להתרמות). XSLT היא מורכבת, ולכן לא נעבר על כל פרטיו אוצר המילים של ה-XML. אם תרצה למדוד עוד על XSLT, תוכלם למצוא את המפרט המלא בכתבota <http://www.w3.org/TR/xslt>.

אם XSLT לא עשו את מה שאותם רוצים, יתכן שתצטרכו לפונקציות אחרות. מחלקה `XML_Transformer` היא פתרון אפשרי אחד. באמצעות `XML Transformer` תוכלם לבצע התמרות XML עם PHP ללא צורך ב-XSLT או בספריות חיצונית.

XSLT 8.6.1

כדי להשתמש בפונקציות ה-XSLT ב-PHP, עליו לתקן את הגרסה העדכנית ביותר של ספריית `itaxslslib`, המימוש את הפונקציות הדרושים להתרמות.

אם אתם משתמשים בוינדוז, תוכלם להעתיק את הקובץ `libxslt.dll` מוסףית ה-`zlib` של הפצת ה-PHP למקום בנתיב המערכת שלכם. הפעלת ההרחבה בוינדוז (XUNI) מטבחה על ידי הוספה `--with-xsl=` בshell לשורת `configure` והידור מחדש. משתמשי וינדוז יכולים להסיר את ה-`comment` מהשורה `.php.ini` בקובץ ה-`extension=php_xsl.dll`.

כפי שהוסבר קודם, ניתן להשתמש ב-XSLT כדי להתמיר את מסמכי ה-XML שלכם לפורמט אחר. אנחנו נתמיר קובץ הדומה לקובץ RSS לשמו לקובץ ML(HT)X על ידי החלת גילוונות סגנון על מסמר ה-XML.

gilinot ssgn (Stylesheets) משמשים לכל ההתמרות המבוצעות עם XSLT כדי למפות את האלמנטים בקובץ ה-XML המקורי לתבנית (template) עבור כל אלמנט. החלק הראשון של גילון הסגנון (XSL stylesheet) מכיל אפשרות לקלט ופלט. אנחנו רוצים להוציא את התוצאה כמסמך HTML עם mime-type ISO-8859-1/text/html. מרחב השמות (namespace) להצרת ה-XSL מוגדר כ-`xsl`, מה של '`xsl:/text/html`'. שואומר שלכל אלמנט הקשור ל-XSL יש את הקידומת `:xsl` לפני שם התגית (למשל, `xsl:output`).

התבניות עוקבות אחר חלק המבוא שהוזג קודם. תוכנת ה-`xsl:template` של אלמנט `match` משמשת לבחירת אלמנטים במסמר. בתבנית הראשונה, כל אלמנט ה-`pdf` במסמר יותאמו. מכיוון שהזהו אלמנט השורש של המסמר שלנו, התבנית מוחלת פעם אחת בלבד. כאשר אלמנט מותאם על ידי התבנית, תוכן ה-`xsl:template` מועתק למסמר הפלט, למעט אלמנטים השייכים למרחיב השמות XSL שיש להם משמעות מיוחדת.

התגית `<xsl:value-of select="מחזירה"` את הערך של אלמנט או תוכנה המצוינים בתוכנת ה-`select`. בתבנית המוצגת כאן, תוכן של הילך `title` של אלמנט ה-`channel` מוכנס לתוך תגית ה-`title` של ה-HTML.

כעת, כישיש לנו גם את גילון הסגנון וגם את קובץ המקור של ה-XML, נוכל להשתמש ב-PHP כדי להחיל את גילון הסגנון על מסמר ה-XML. השתמש בפונקציות ה-XSL עם הקבצים `php.net.xsl` ו-`rss.php.net-stripped`, ונדפיס את הפלט למסך:

PHP

`php?>`

```
;()dom = new domDocument$  
;("dom->load("php.net.xsl$  
;proc = new xsltprocessor$  
;(xsl = $proc->importStylesheet($dom$  
;()xml = new domDocument$  
;('xml->load('php.net-stripped.rss$  
;(string = $proc->transformToXml($xml$  
;echo $string  
<?
```

טיפ: ניתן להשתמש באותו גילון סגנון XSLT שנטען מ-\$load->dom() להמרתה של מספר מסמכי XML. הדבר חוסף את התקורה הכרוכה בניתוח (parsing) של גילון הסגנון.

בנוסף למתודה `transformToXml()`, זמינים שתי פונקציות עיבוד XML נוספת להמרת מסמכים: `transformToUri()` ו-`transformToDoc()`. הראשונה מוציאה אובייקט `DomDocument` שנייה להמשר לעבד אותו, והשנייה מרנדרת לכתובת URI המועברת כפרמטר השני.

8.7 תקשורת באמצעות XML

ישונים מתקשרים כו� דרך האינטרנט במספר דרכיהם. TCP/IP ו-UDP משמשים כפרוטוקול תחבורה בrama נמוכה. תקשורת בין מערכות היא קשה מכיוון שמערכות מאחסנות נתונים בזיכרון בשיטות שונות (למשל סדר בתים Little Endian לעומת Big). פתרון אחד הוא **RPC** (קריאה לפונקציה מרוחקת), אך הוא אינו קל לשימוש. XML הוא לעיתים קרובות הפתרון הטוב ביותר, שכן הוא מאפשר לשימושים במערכות שונות לתקשר באמצעות פורמט סטנדרטי מבוסס ASCII. **XML-RPC** ו-**SOAP** הם שניהם פרוטוקולים מבוססי XML.

8.7.1 XML-RPC

נתחיל בדרך הפשוטה ביותר לתקשר: XML-RPC.

8.7.1.1 הדרות: XML-RPC הוא פרוטוקול של בקשה-תגובה (request-response). עבור כל בקשה לשרת, מוחזרת תגובה. הבקשה וה答复 מקודדות כ-XML. המפרט מגדיר מספר סוגי סקלרים (ראו טבלה 8.1).

טבלה 8.1: סוגי נתונים ב-XML-RPC

סוג XML-RPC תיאור
--- ---
או <int> מספר שלם חתום (4 בתים)
<boolean> 0 שקר) או 1 (אמת)
<string> מחרוזת ASCII
<double> מספר נקודה צפה בדיק כפול
<dateTime.iso8601> תאריך/שעה
<base64> נתונים בינהירים בקידוד Base64
זמינים שני סוגי נתונים מורכבים: <array> למערכות שאיןם אסוציאטיביים, ו-<struct> למערכות אסוציאטיביים.

8.7.1.2 בקשה (Request): בקשות ב-XML-RPC הן בקשות POST רגילות לשרת HTTP עם כמה תוספות מיוחדות. ה-`Content-Type` הוא תמיד `text/xml`. גוף ההודעה מורכב ממסמך XML הכלול בתגית `<methodCall>` המכילה את שם הפונקציה (`<methodName>`) ופרמטרים (`<params>`).

8.7.1.3 תגובה (Response): כאשר קריאת הפונקציה מצליחה, מוחזרת תגובה מקודדת ב-XML תחת התגית `<methodResponse>`. היא תמיד תכיל אלמנט `<params>` עם ערך חזרה יחיד (שיכל להיות מערך).

8.7.1.4 שגיאה (Fault): כאשר משהו לא עובד כמצופה, מוחזר אלמנט `<fault>` במקומם `<params>`. הוא מכיל מבנה עם `faultString` (קוד שגיאה) ו-`faultCode` (תיאור השגיאה).

(The Client) 8.7.1.5

כתיבת לקוב נשתמש במחלקה `XML_RPC_Message` PEAR. אנו בונים אובייקט `XML_RPC_Client` עם שם הפונקציה והפרמטרים, ושולחים אותו באמצעות אובייקט `XML_RPC_Client`.

ניתן להשתמש ב-`XML_RPC_encode()` כדי להמיר אוטומטית משתנה PHP לסוג ה-`XML-RPC` המתאים, וב-`XML_RPC_decode()` לפועלה ההפוכה.

8.7.1.6 רטרוספקטיבית (Retrospection): זהה היכולת לשאול את השירות אילו פונקציות הוא מייצא. הפונקציה `system.listMethods` מhzירה רשימה של כל הפונקציות הזמיןות. ניתן להשתמש גם ב-`system.methodSignature` לתיאור הפונקציה וב-`system.methodHelp` לתיאור השירות אליו סוג פרמטרים היא מקבלת.

(The Server) 8.7.1.7

כתיבת השירות דומה לכתיבת הלקוב. אנו ממשמים את הפונקציות ב-PHP (למשל `hello` או `add`), מגדרים את ה"ח提ימה" שלהן (הסוגים שהן מקבלות ומוחזירות) בתוך מערך `methods$`, ומפעלים את השירות על ידי יצירת אובייקט `XML_RPC_Server($methods new)`. הקונסטרקטור של המחלקה מטפל בניתוח הבקשת ובקריה לפונקציות המתאימות באופן אוטומטי.

בזה נסימן את נושא ה-`XML-RPC`.

SOAP 8.7.2

עיף זה ידריך אתכם בשימוש ב-`SOAP`(Cloud API Google Web API) עבור ה-`SOAP` ובהטמעת שירות SOAP משלכם. מכיוון ש-`SOAP` מורכב אף יותר מ-`XML-RPC`, לרובו הצער לא יוכל לכלול את הכל.

PEAR::SOAP 8.7.2.1

גוגל הוא מנוע חיפוש נחמד ומהיר. האם זה לא יהיה הנדר לו היה לכם מנוע חיפוש משלכם בשורת הפוקודה הכתוב ב-`PHP`? עיף זה מסביר כיצד לעשות זאת.

Google: כדי לישוט שימוש ב-`SOAP` API שגוגל מייצאת, אתם צריכים לחשבו, אותו תוכלו ליצור בכתובות בעט `http://www.google.com/apis/.php`. בעת ההרשמה, תקבלו מפתח (Key) בדוור אלקטרוני, בו תשתמשו כאשר תקראו למתודת `SOAP`.

כדי שהדוגמה הבאה תעבור כראוי, עליכם להתקין את מחלוקת ה-`SOAP` של `PEAR` באמצעות הפוקודה `pear install SOAP`. לאחר ההתקינה, תוכל להתחיל עם הסקריפט פשוט הבא:

```
PHP
usr/local/bin/php!#
php?>
'require_once 'SOAP/Client.php
... // הגדרת פרמטרים ומפתח ...
)params = array$,
'key' => $key'
```

```

        ,q' => $query'
        ,start' => $start'
        ...
        // ... פרמטרים נוספים ...
        ;
        ;('client = new SOAP_Client('http://api.google.com/search/beta2$'
            )response = $client->call$'
            , 'doGoogleSearch'
            , params$'
            ('array('namespace' => 'urn:GoogleSearch'
            );

```

בדוגמה זו, אנו מניחים שקריאה החיפוש החזרה משזה מועיל. ה-API של גוגל מחזיר טקסט עם ישויות XML מומarrant וכמה תגיות `
` מוכנסות. אנו ממירם את הישויות חזרה לתווים רגילים באמצעות `strip_tags()` ומסירים את כל התגיות בעזרת `html_entity_decode()`.

שרת SOAP: ראשית נכלול את מחלקה `SOAP_Server` של PEAR. לאחר מכן נגידר מחלקה (`Example`) עם שתי הפונקציות שאנו רוצים ליצא. במתודה `hello()` משתמש בהירה משתמש מסוגי PHP לסוג `SOAP`; במתודה `add()` נגידר במפורש את סוג ה-`SOAP` (`float`):

```

PHP
php?>
;require_once 'SOAP/Server.php'
} class Example
} (function hello($name
;"!return "Hi $name
{
} (function add($a, $b
;(return new SOAP_Value('return', 'float', $a + $b
{
{
;server = new SOAP_Server$
;()soapclass = new Example$
;('server->addObjectMap($soapclass, 'urn:Example$'
;('server->service($HTTP_RAW_POST_DATA$
<?

```

לקוח SOAP: הלוקח דומה מאוד לлокוח של גוגל, פרט לכך שהשתמשו בטיפוס מפורש (`explicit typing`) עבור הפרמטרים בקריאה למетодה `add()` באמצעות `SOAP_Value`.

8.7.2.2 הרחבת ה-`SOAP` של PHP

PHP 5 מגיע גם עם הרחבת `SOAP` מובנית (`ext/soap`), שהיא בעלת תכונות רבות יותר מ-`PEAR::SOAP`, וכתובה ב-C (מה שהופך אותה למהירה יותר).

הרחבת ה-`SOAP` תומכת גם ב-`WSDL` (נוהga "ויז-דיל"), אוצר מילים של XML המשמש לתיאור שירותים (Web Services). באמצעות קובץ `WSDL` זה, הרחבת יודעת פרטים כמו נקודת הקצה (`endpoint`), הפרזידורות וסוגי ההודעות.

כאשר משווים את השימוש ב-WSDL לשיטה הקודמת, רואים שקריאה למתודת SOAP היא הרבה יותר קלה – שתי שירותות בלבד!

שרות SOAP: פיתוח שרת SOAP וקובץ ה-WSDL הנלווה אליו אינם קשא במיוחד; הבעיה האגדולה ביותר היא יצירת קובץ תיאור ה-WSDL עצמו. קוד השירות מחבר את המחלקה המספקת את המתודות לשרת ה-SOAP בעזרת קובץ ה-WSDL, והמתודה `handle()` דואגת לעיבוד המידע כאשר ללקוח מבקש קריאה למתודה.

לקוח SOAP: הלוקו משתמש ב-`SoapClient` וביכולות הטיפול בשגיאות של `try-catch` (PHP 5). אם ננסה לקרוא למתודה ללא פרמטרים נדרשים, תיזרק חריגת `SoapFault`.

8.8 סיכום

XML תוכנן בעיקר לשימוש בחילופי מידע בין מערכות. ל-XML טרמינולוגיה משלו המתארת את מבנה המסמכים, כאשר המידע עטוף בתגיוט המזהות אותו בצורה מובנית. כדי להשתמש במידע, עליו לנתח (Parse) את המסמכים.

PHP מספקת שני מנתחים עיקריים:

- **SAX** (Simple API for XML) – מנתח כל אלמנט במסמך תוך כדי תנועה.
- **DOM** (Document Object Model) – יוצר עצ היררכי בזיכרון של כל מבנה המסמך.

PHP 5 מספקת גם את הרחבה SimpleXML הקלה יותר לניתוח מסמכים פשוטים. לצורך המרת מסמכי XML לפורמטים אחרים (כמו HTML), השיטה הסטנדרטית היא XSLT, המשמשת בಗילונות סגנון ובתבניות.

لتקשורת בין "ישומים" במערכות שונות, PHP מציעה שני פתרונות מבוססי XML:

- **XML-RPC** – מאפשר ללקוח להריץ מתודות על שרת.
- **SOAP** – מגדר פורטט להחלפת נתונים רחב ומורכב יותר, עם פוטנציאל רב ליישומים עתידיים.

פרק 9

הרחבות מרכזיות (Mainstream Extensions)

"הדבר החשוב הוא לא להפסיק לשאול." — אלברט איינשטיין

9.1 מבוא

הפרקים הקודמים CISO את הרחבות הנפוצות ביותר. פרק זה מציג הרחבות מרכזיות וערכיות נוספות. הסעיף הראשון מתאר קבוצת פונקציות שהן חלק מליבת ה-PHP ולא הרחבה נפרדת. שאר הסעיפים דנים במספר הרחבות פופולריות ושימושיות שאין חלק מהליבת.

לאחר סיום פרק זה, תלמדו:

- לפתח, לקרוא ולכתוב קבצים מקומיים ומרוחקים.
- לתקשר עם תהילכים ותוכניות.

- לעבוד עם תזריםים (Streams).
- לבצע התאמת טקסט, אימות קלט, החלפת טקסט, פיצול טקסט ומיניפולציות טקסט אחרות באמצעות ביטויים רגולריים (Regular Expressions).
- לטפל בניתוח ופרמטר של תאריכים ושעות, כולל סוגיות של שעון קיז.
- ליצור תמונות עם הרחבה GD.
- להלץ מידע מטא (Meta Information) מתמונות דיגיטליות באמצעות הרחבה Exif.
- להמיר בין סטימ של תווים בבית-בודד (Single-byte) למורובי-בתים (Multi-byte).

9.2 קבצים ותזרים (Files and Streams)

הגישה לקבצים השתנתה באופן דרמטי. לפני גרסת PHP 4.3.0, לכל סוג קובץ (מקומי, דחוס, מרוחק) היה מיומש שונה. עם זאת, מאז הצגת התזרים, כל אינטראקציה עם קובץ עשויה שימוש בשכבות ה-**Streams**, שכבה המפשיטה את הגישה לפרטי המימוש של סוג ה"קובץ" הגרפי. שכבות התזרים מאפשרות ליצור אובייקט תמונה ב-GD ממוקור HTTP באמצעות תזרים URL, לעבוד עם קבצים דחוסים, או להעתיק קובץ למקום. ניתן להחיל המרות משלכם במהלך הפעלה על ידי שימוש תזרים-משתמש (User-stream) או מסנן (Filter).

9.2.1 גישה לקבצים

נתחיל בפונקציות הבסיסיות לגישה לקבצים. במקור, פונקציות אלו עבדו רק על קבצים רגילים, ולכן שמן מתחילה באות "f", אך PHP הרחיבה זאת כמעט לכל דבר. הפונקציות הנפוצות ביותר הן:

- `fopen()` – פותחת ידית (Handle) לקובץ מקומי או לקובץ מכתובת URL.
- `fread()` – קוראת בלוק של נתונים מקובץ.
- `fgets()` – קוראת שורה בודדת מקובץ.
- `fwrite() / fputs()` – כתבת בלוק נתונים לקובץ.
- `fclose()` – סגירת את ידית הקובץ הפתוחה.
- `feof()` – מחזירה אמת (True) כאשר מגעינו לסוף הקובץ.

העובדת עם קבצים היא פשוטה, כפי שמראה הדוגמה הבאה:

```
PHP
php?>

;(+fp = fopen("counter.dat", "r$"
;(count = (int) fgets($fp, 20$
;++count$
;(rewind($fp
;(fwrite($fp, $count
;(fclose($fp
<?
```

בשורה 3, ידית קובץ (`$fp`) משוויכת לתזרים, והזריםם משוויר לקובץ `counter.dat` של הדיסק. הפעמטר השני ב- `fopen()` הוא **ה-mode** (מצב), המציין האם התזרים נפתח לקריאה, כתיבה, שניהם או הוספה (Append):

- **z**: קריאה בלבד. המצביע בתחילת הקובץ.
- **+z**: קריאה וכתיבה. המצביע בתחילת הקובץ.
- **w**: כתיבה בלבד. תוכן הקובץ נמחק והצביע בתחילתיה. אם הקובץ לא קיים, המערכת תנסה ליצור אותו.
- **+w**: קריאה וכתיבה. הקובץ נמחק והצביע בתחילתיה.
- **a**: כתיבה בלבד (Append). המצביע בסוף הקובץ. אם הקובץ לא קיים, המערכת תנסה ליצור אותו.
- **+a**: קריאה וכתיבה. המצביע בסוף הקובץ.

ניתן להוסיף את המגדיר **b** כדי לציין שהקובץ הוא בינארי. מערכות Windows מבוחנות בין קבצי טקסט לבינארים; ללא שימוש ב-**b**, הקובץ עלול להישחת. לכן, כדאי להשתמש ב-**b** תמיד כדי להבטיח ניידות (Portability).

9.2.2 קלט/פלט של תוכניות

בדומה לפונקציה של UNIX שבה "כל קלט/פלט הוא קובץ", ב-PHP הפרדיגמה היא "כל קלט/פלט הוא תזרים". כשרוצים לעבוד עם קלט ופלט של תוכנית חיצונית, פותחים תזרים לאותה תוכנית באמצעות `proc_open()` או `popen()`.

9.2.2.1 `(popen()`

פונקציה פשוטה המספקת קלט/פלט חד-כיווני בלבד (רק **z** או **w**). פתיחת תזרים כזו נקראת Pipe (צינור). ניתן להשתמש בפונקציות קבצים רגילים כדי לקרוא ממנו.

9.2.2.2 `(proc_open()`

מאפשרת שימוש אינטראקטיביות יותר. ניתן לקשר את כל ערכיו הקלט והפלט של התהיליך לצינורות (Pipes) או לקבצים. הפונקציה דורשת שלושה פרמטרים: פקודה להרצה, מערך הגדרות (descriptorspec,) ומערך צינורות.

9.2.2.3 מתארים קבצים (File Descriptors)

מתאר קובץ הוא מספר המיצג ערך תקשורת. כברית מחדל, 0 הוא קלט סטנדרטי (`stdin`), 1 הוא פלט סטנדרטי (`stdout`), ו-2 הוא שגיאת סטנדרטית (`stderr`). בעזרת `proc_open()` ניתן לנתח ערכים אלו לקבצים או למשתנים בתוך הסקריפט, מה שמאפשר שליטה מלאה בתהליכי מערכת חיצוניים.

9.2.2.4 צינורות (Pipes)

במקום להשתמש בידית קובץ עבור קלט ופלט לתהיליך הבן של PHP, כדי שהווג בsurface הקודם, ניתן לפתוח צינורות (pipes) לתהיליך הבן המאפשרים לכם לשולט בתחום שנון מטור הסקריפט שלכם. הסקריפט הבא שולח קוד PHP מטור הסקריפט עצמו למפרש ה-PHP שנוצר. הסקריפט כותב את הפלט של פקודת `echo` לפלט הסטנדרטי, תוך הפעלת `urlencode` על מחזורת הפלט "Hello you!".

PHP

```
php?>
;((descs = array(0 => array('pipe', 'r'), 1 => array('pipe', 'w$
```

```

;(res = proc_open("php", $descs, $pipes$)
} ((if (is_resource($res
;(<? ;"fputs($pipes[0], '<?php echo "Hello you!\n
;([fclose($pipes[0
} ) ([while (!feof($pipes[1
;([line = fgets($pipes[1$(
;echo urlencode($line
{
;(proc_close($res
{
<?

```

הפלט הוא: A%021%you+Hello

9.2.2.5 קבצים

ניתן להעביר קובץ כמעבד עבור מתארי הקבצים (file descriptors) של התהיליך שלכם, כפי שown בדוגמה הבאה:

PHP

```

php?>
)descs = array$(
,'array('pipe', 'r <= 0
,('array('file', 'output', 'w <= 1
('array('file', 'errors', 'w <= 2
;
;(res = proc_open("php", $descs, $pipes$(
} ((if (is_resource($res
;(<? ;"!fputs($pipes[0], '<?php echo "Hello you
;([fclose($pipes[0
;(proc_close($res
{
<?

```

קובץ הפלט מכיל כתעת you Hello ! וקובץ ה-'errors' ריק.

בנוסף לצינור הקלט [0] וצינור הפלט [1] [קופיק] שהוצגו, ניתן להשתמש בציורות אחרים כדי לנכתב מחדש את כל מתארי הקבצים של התהיליך הבן. בדוגמה לעיל, ניתבנו את כל הודיעות השגיאה (מתאר 2) לצינור [2] [קופיק], הקוביץ האינדקס במערך \$descs אינו מוגבל ל-2-0, אך שניתן לשימוש בכל מתאר קוביץ קריאות עניות (אם כי ב-Windows אינדקסים מעל 2 טרם נתמכים ב-PHP).

9.2.3 תזריםி קלט/פלט (I/O Streams)

ב-PHP ניתן להשתמש ב-stdin, stdout ו-stderr כקבצים. "קבצים" אלו, המושרים לתזריםי התהיליך, נגישים באמצעות מצין הפרטוקול php:// בקריאה ל-fopen(). תוכנה זו שימושית במיוחד בעבודה עם מסך שורת הפקודה (CLI).

קיים שני תזרימי קלט/פלט נוספים: `php://input` ו-`php://output`. באמצעות `php://input` ניתן לקרוא נתונים POST גולמיים (Raw Data), דבר השימושי בעיבוד בקשות WebDAV, XML-RPC או SOAP.

כפי שניתן לראות באIOR 9.1, לעיתים נתונים ה-POST הגולמיים ב-`php://input` מכילים מידע רב יותר מאשר מערכת ה-`$_POST` האוטומטי, במיוחד כאשר ישנו שדות בעלי שמות שאינם מוגדרים כמערך. התזרים מופיעים `php://output` מאפשר כתיבה ישירה למאגרי הפלט של PHP (בדומה ל-`echo`).

9.2.4 תזרימי דחיסה

PHP מספקת עיטופות (wrappers) עבור פונקציות דחיסה. קרייה וכתיבה מקובץ דחוס ב-`gzip` או `bzip` או `//compress.bzip2://compress.zlib` באמצעות המציגים `//compress`.

תזרימי `Gzip` תומכים במדדים נוספים כמו רמת דחיסה (9-1) ושיטות דחיסה (f למסון, h להופמן בלבד). ניתן אפילו לשרר עיטופות, למשל פתיחת קובץ דחוס שירות מכתובת URL:

```
compress.zlib://http://www.example.com/foobar.gz
```

9.2.5 תזרימי משתמש (User Streams)

שכבת התזרמים ב-5 PHP מאפשרת להגדיר "תזרימי משתמש" – עיטופות תזרים הממומנשות בקוד PHP באמצעות מחלוקת. עבור כל פעולה קובץ (פתיחה, קרייה וכדומה), יש למשם מתודה תואמת:

- `stream_open`: נקראת בעת `fopen`.
- `stream_read`: מוחזירה נתונים בעת `fread` או `fgets`.
- `stream_write`: נקראת בעת כתיבה לתזרים.
- `stream_seek` / `stream_tell`: לנויוט ודיווח על המיקום בתזרים.
- `stream_eof`: בודקת האם הגיעו לסוף התזרים.

9.2.6 תזרימי URL

אלו תזרמים שהנתיב שלהם מזכיר URL רגיל. התזרמים הבסיסיים הנתמכים הם:

- `//` ו-`://`: עבור קבצים בשרת HTTP (דורש OpenSSL עבור SSL).
- `://ftp` ו-`://ftps`: עבור קבצים בשרת FTP.

עבור אימות (Authentication), ניתן להוסיף את שם המשתמש והסיסמה בתוך ה-URL:

```
;('fp = fopen ('ftp://username:password@ftp.example.com', 'wb$
```

```
PHP  
php?>  
;(((context = stream_context_create(array('ftp' => array('overwrite' => true$  
;(fp = fopen('ftp://secret@ftp.php.net', 'wb', false, $context$  
<?
```

איור 9.2: **התקשרות באמצעות phpSUCK**

הדוגמה הבאה מדגימה קריית קובץ משרת HTTP ושמירתו כקובץ דחוס. דוגמה זו מציגה גם פרמטר רביעי לкриאה ל-fopen() המגדיר **Context** (הקשר) עבור התזרים. באמצעות פרמטר זה, ניתן להגדיר אפשרויות מיוחדות, כמו למשל **notifier** (מדוח). פונקציית ה-**callback** זו תיקרא באירועים שונים במהלך התקשרות:

```
PHP
usr/local/bin/php!#
php?>
/* מיפוי חומרת הודעות */
$severity_map = array(
    'info' <= 0,
    'warning' <= 1,
    'error' <= 2
);(



/* פונקציית Callback עבור אירועי תזרים */
(function notifier($code, $severity, $msg, $xcode, $sofar, $max
)
{
    global $term_sol, $severity_map, $max_kb_sec, $size
    /* אל תדפיס קידומת סטטוס כאשר מתקבל אירוע */
    if ($code != STREAM_NOTIFY_PROGRESS
        ;
        echo $severity_map[$severity
    }

    } (switch ($code
        :case STREAM_NOTIFY_CONNECT
            ;printf("Connected\n
        ;GLOBALS['begin_time'] = time() - 0.001$
```

```

;break

:case STREAM_NOTIFY_AUTH_REQUIRED
;((printf("Authentication required: %s\n", trim($msg
;break

:case STREAM_NOTIFY_AUTH_RESULT
;((printf("Logged in: %s\n", trim($msg
;break

:case STREAM_NOTIFY_MIME_TYPE_IS
;(printf("Mime type: %s\n", $msg
;break

:case STREAM_NOTIFY_FILE_SIZE_IS
;(printf("Downloading %d kb\n", $max / 1024
;size = $max$
;break

:case STREAM_NOTIFY_REDIRECTED
;(printf("Redirecting to %s...\n", $msg
;break

:case STREAM_NOTIFY_PROGRESS
/* חישוב מד התקדמות */
} (if ($size
;(stars = str_repeat (*", $sofar * 50 / $size$
} else {
;" = stars$
{
;((stripe = str_repeat ('-', 50 - strlen($stars$

/* חישוב מהירות הורדה */

```

```

;kb_sec = ($sofar / (time() - $GLOBALS['begin_time'])) / 1024$


/* שהיית הסקירהפט אם המהירות גבוהה מהמקסימום שהוגדר */

} (while ($kb_sec > $max_kb_sec

;(usleep(1

;kb_sec = ($sofar / (time() - $GLOBALS['begin_time'])) / 1024$


{



;(printf("${term_sol}[%s] %d kb %.1f kb/sec", $stars.$stripe, $sofar / 1024, $kb_sec
;break

:case STREAM_NOTIFY_FAILURE
;(printf("Failure: %s\n", $msg
;break

{



/* קביעת שם הקובץ לשמירה */

;([url_data = parse_url($argv[1$
;(['file = basename($url_data['path$
{ ;"if (empty($file)) { $file = "index.html
;("printf ("Saving to $file.gz\n
;:"fil = "compress.zlib://$file.gz$


/* יצירת הקשר והגדרת ה-notifier */

;()context = stream_context_create$
;(("stream_context_set_params($context, array ("notification" => "notifier

```

```

/* פתיחת כתובות היעד */

;(fp = fopen($url, "rb", false, $context$

} ((if (is_resource($fp

;(fs = fopen($fil, "wb9", false, $context$

} ((if (is_resource($fs

} ((while (!feof($fp

;(data = fgets($fp, 1024$

;(fwrite($fs, $data

{

;(fclose($fs

{

;(fclose($fp

printf("${term_sol}[%s] Download time: %ds\n", str_repeat('*', 50), time() -
;(['$GLOBALS['begin_time

{

<?

```

- ¹ ניתן ל-**טפל** באירועים שונים בתוך פונקציית `notify`. להלן הרשימה המלאה:
- **STREAM_NOTIFY_CONNECT**: נורה כאשר נוצר חיבור עם המאוב.
 - **STREAM_NOTIFY_AUTH_REQUIRED**: מופעל כאשר נדרשת הרשאה.
 - **STREAM_NOTIFY_AUTH_RESULT**: מופעל בסיום תהליך האימוס (הצלחה או כישלון).
 - **Content-Type**: מופעל ב-HTTP כאשר כוורתה-`Content-Type` זמינה.
 - **STREAM_NOTIFY_FILE_SIZE_IS**: מופעל כאשר גודל הקובץ מזוהה (ב-FTP או דרך `Content-Length`).
 - **STREAM_NOTIFY_REDIRECTED**: מופעל בעת זיהוי הפניה (Redirect).
 - **STREAM_NOTIFY_PROGRESS**: נורה ברגע שmagieva חבילת נתונים (משמש למדדי התקדמות).
 - **STREAM_NOTIFY_FAILURE**: מופעל כאשר מתרחשת תקלת (כמו פרטיה התחרבותות שגויות).

בעת כתיבה לקבצים שייתכן ונקראים על ידי סקריפטים אחרים בו-זמנית, עלולות להיווצר בעיות (קריאת קובץ חלק). הפתרון לכך נקרא **נעילה** (Locking).

PHP מאפשרת להגדיר נעילות באמצעות הפונקציה `flock()`. היא מונעת מסקריפט קורא לגשת לקובץ בזמן שסקריפט אחר כותב אליו, בתנאי שני הסקריפטים ממשים את מגנון הנעילה.

9.2.8 שינוי שם ומחיקת קבצים

PHP מספקת את הפונקציה `chunlink()` למחיקת קובץ (ניתוקו מהספרייה). במערכות UNIX, הקובץ ימחק פיזית מהדיסק רק לאחר שלל התוכניות המשתמשות בו יסגרו את הידית שלהן (`fclose()`).

הজת קובץ עם הפונקציה `rename()` היא פעולה **אטומית** (Atomic) אם הקובץ נשאר באותה מערכת קבצים. פעולה אטומית מבטיחה שהההילר לא יופרע במהלך. אם עוברים בין מערכות קבצים שונות, בטוח יותר להעתיק את הקובץ לשם זמני ורק אז לשנות את שמו לשם הסופי באותה מערכת קבצים, כדי להבטיח שהקובץ לעולם לא יופיע שם במצב חלק.

9.2.9 קבצים זמניים

במידה ואתם מעוניינים ליצור קובץ זמני, הדרך הטובה ביותר לעשות זאת היא באמצעות הפונקציה `tmpfile()`. פונקציה זו יוצרת קובץ זמני עם שם אקראי ייחודי בספרייה הנוכחיית ופותחת אותו לכתיבה. קובץ זה יסגר (וימחק) אוטומטית כאשר תסגורו אותו עם `fclose()` או כשהסקריפט יסתום.

אם אתם רוצים שליטה רבה יותר על המיקום שבו נוצר הקובץ הזמני ועל שמו, תוכלו להשתמש בפונקציה `tempnam()`. ברגע של-`tmpfile()`, קובץ זה **לא** יוסר אוטומטית מהדיסק:

הפרמטר הראשון לפונקציה מצין את הספרייה שבה יוצר הקובץ, והשני הוא **קידומת** (prefix) שתתווסף לשם הקובץ האקראי.

9.3 ביטויים רגולריים (Regular Expressions)

למרות שביטויים רגולריים הם חזקים מאוד, הם קשים לשימוש, במיוחד למתחילים. לכן, לפני שנצלול לפונקציות ה-PHP, נcosa קודם את תחביר התבניות (Pattern Syntax). אם PCRE מופעל, תוכלו לראות זאת בפלט של `phpinfo()` (או [יראה](#) 9.3).

9.3.1 תחביר (Syntax)

פונקציות PCRE בודקות האם מחרוזת טקסט תואמת לתבנית מסוימת. תחביר התבנית תמיד נראה כך: `magdirim [מפריד] [תבנית] [מפריד]`.

המגדירים (modifiers) הם אופציונליים. המפריד (delimiter) מפריד בין התבנית למגדירים. PCRE משתמש בטו הראשון של הביטוי כמפריד. נהוג להשתמש ב-`/`, אך ניתן להשתמש גם ב-`-` או `@`.

הפונקציה `preg_match()` משמשת להתקמת ביטויים רגולריים. היא מחזירה TRUE אם נמצאה התקמת FALSE אם לאו. ניתן להעביר פרמטר שלישי (אופציוני), משתנה אליו יוכנסו התוצאות שנמצאו.

9.3.1.2 תווים מיוחדים (Metacharacters)

התווים בטבלה 9.1 הם תווים מיוחדים המשמשים לבניית התבניות:

טבלה 9.1: תווים מיוחדים

\ תווים מיוחדים
--- ---
\ מילוט (Escape). משמש לביטול המשמעות המיוחדת של תווים מיוחדים אחרים.
. מתאים בדיק לטו אחד (פרט לטו שורה חדשה).
? מסמן שהטו או תת-המבנה שקדם לו הם אופציונליים (0 או 1 פעמים).
+ מתאים לטו הקודם פעם אחת או יותר.
* מתאים לטו הקודם לפחות פעם אחת או יותר.
{,}, } התאמה בין ולח פעמים.
^ מסמן את תחילת המחרוזת.
\$ מסמן את סוף המחרוזת.
[...] מגדר מחלקת תווים (למשל [0-9] עבר כל ספרה).
(...) יוצר תת-מבנה (Sub-pattern) לקבוצה, שתופיע בנפרד במערך התוצאות.
(:? ...) יוצר תת-מבנה ללא "לכידה" (לא תופיע בנפרד התוצאות בנפרד).

9.3.1.3 דוגמה 1: כתובות MAC

התבנית `/^(?:[0-9a-zA-Z]{5}|\d{2})(?:[0-9a-zA-Z]{2})$` מתאימה לכתובת MAC בפורמט `00:04:23:7c:5d:01`. היא מחפשת 5 קבוצות של שתי ספרות הקסדצימליות ונקודות, ואחריהן קבוצה שישית ללא נקודות.

9.3.1.4 דוגמה 2: כתובות אימייל פשוטה

התבנית `/^[\w+\.]+@[a-zA-Z0-9\-.]+\.[a-zA-Z]{2,}` משמשת להטאה של פורטט כמו `Derick Rethans <derick@php.net>`.

9.3.1.5 רצפי מילוט (Escape Sequences)

בנוסף לביטול סימנים מיוחדים, \ משמש לקייזרים (טבלה 9.2):

- \p: כל ספרה (כמו [0-9]).
- \D: כל תוו שאינו ספרה.
- \s: כל תוו "רווח" (רווח, טאב, שורה חדשה).
- \w: כל תוו "מילה" (אותיות, ספרות וקו תחתון).
- \b: גבול מילה (הנקודה שבין תוו מילה לטו שאינו מילה).

9.3.1.7 התאמה "עצלנית" (Lazy Matching)

כברית מחדל, אופרטורים כמו * ו+ הם "חמדניים" (Greedy) – הם ינסו להתאים כמה שיותר טקסט. הוספה ? אחריהם (למשל `*?`) הופכת אותם ל"עצלנים", כך שייצרו בנקודה הראשונה האפשרית.

9.3.2 פונקציות

קיימות שלוש קבוצות של פונקציות PCRE:

1. **פונקציות התאמת:** preg_match() (התאמת ראשונה) ו-preg_match_all() (מציאת כל הופעת).
2. **פונקציות החלפה:** preg_replace() – מחליפה טקסט שתוואם לתבנית בטקסט אחר.
3. **פונקציות פיצול:** preg_split() – מפצלת מחוזת למערך לפי התבנית.

פונקציית preg_grep() היא גרסה מעניתה: היא מקבלת מערך של מחוזות ומחזירה מערך חדש המכיל רק את האיברים שתוואים לתבנית.

9.3.2.2 פונקציות החלפה

בנוסף להטאמה שתוארה בסעיף הקודם, פונקציות הביטויים הרגולריים של PHP יכולות גם להחליף טקסט על בסיס התאמת התבניות. פונקציות אלו יכולות להחליף תת-מחוזות התואמת לתת-תבנית בטקסט אחר. בהחלפה, ניתן להתייחס להטאמות בתבנית באמצעות **back references** (סימוכין לאחר).

הנה דוגמה המשמשת ב-preg_replace() כדי להחליף קישור-דינה, תוך שימוש ב-[link](http://www.php.net) ב-[PHP](#):

```
PHP  
-----  
php?>  
  
;'.str = '[link url="www.php.net"]PHP[/link] is cool$  
;@[\pattern = '@\[link url="([^\"]+)"\](.*?)\[link$  
;' = '<a href="http://\1">\2</a$  
;(str = preg_replace($pattern, $replacement, $str$  
.a href="http://www.php.net">PHP</a> is cool';echo $str  
  
<?
```

התבנית מורכבת משתי תת-תבניות: ([^"]+) עבור ה-URL ו-(.*?) עבור הטקסט. מנוע ה-PCRE מוצא את תת-מחוזות הללו לסימוכין \\\1 ו-\\\2 במחוזת ההחלפה. מומלץ תמיד לשים את מחוזות ההחלפה בגרשיים בודדים (' ') כדי להימנע מביעיות מילוט של סלאשים או סימני دولار.

טיפ: אם מחוזות ההחלפה צריכה להיות סימוכין הצמוד למספר, ניתן להשתמש בפורמט \${1}.

ניתן להשתמש ב-preg_replace() על מערך של מחוזות בו-זמןית. אם נרצה לבצע מניפולציה מורכבת יותר (כמו הפיכת האות הראשונה של כל מילה לגודלה), יוכל להשתמש במגדר /e (המריץ את מחוזות ההחלפה קוד PHP) או בפונקציה preg_replace_callback().

9.3.2.3 פיצול מחוזות

הפונקציה preg_split() מאפשרת לפצל מחוזות לתת-מחוזות באמצעות ביטוי רגולרי כמפריד (delimiter). היא גמישה הרבה יותר מ-explode(), אם כי איטית ממנה.

PHP

php?>

```
;" .() str = "This is an example for preg_split$  
;(ops = preg_split('/[\s.]+/', $str$  
;)(print_r($ops  
<?
```

ניתן להעביר פרמטרים נוספים כמו **limit** (הגבלה מספר האיברים המוחזרים) ודגלים (flags):

- **PREG_SPLIT_NO_EMPTY**: מונע החזרת איברים ריקים.
- **PREG_SPLIT_DELIM_CAPTURE**: מוחזיר גם את הטקסט ששימש כמפריד (אם הוא מוקף בסוגרים בתבנית).
- **PREG_SPLIT_OFFSET_CAPTURE**: מוחזיר מערך דו-ממדי הכלול גם את המיקום (offset) של כל איבר במחוזת.

9.4 טיפול בתאריכים

PHP מציעה מגוון פונקציות לטיפול בתאריך ושעה. רובן עובדות עם **timestamp UNIX** – מספר השניות החלפו מאז ה-1 בינואר 1970. בغالל מגבלות של 32-ビט, טווח התאריכים ברוב המערכות הוא בין 1970 ל-2038.

9.4.1 שילוט במידע על תאריך ושעה

- `time()`: מוחזירה את חותמת הזמן הנוכחי בשניות.
- `microtime()`: מוחזירה זמן בדיקות גבוהה יותר (כולל מיקרו-שניות).
- `gettimeofday()`: מוחזירה מערך עם פרטי הזמן הנוכחי ומיקרו-שניות.
- `getdate()` ו-`localtime()`: מקבלות חותמת זמן ומוחזירות מערך מפורט (שניות, דקות, שעות, יומם בשבוע וכך).

טבלה 9.5: רכיבים במערכי `localtime()` ו-`getdate()`

האלמנט `tm_isdst` ב-`localtime()` הוא הדך היחידה ב-PHP לבודק אם השרת נמצאת כרגע בשעון קיץ (DST).

`gmstrftime()` ו-`mktime()` יוצרות חותמת זמן מפרמטרים נפרדים (שעה, דקה, חודש וכו'). ההבדל הוא ש-`gmstrftime()` מתיחסת לפרמטרים זמן גראניץ' (GMT), בעוד `mktime()` מתיחסת אליהם זמן מקומי לפי הגדרות השרת.

9.4.2 פרמטר תאריך ושעה

לצורך הצגת התאריך בפורמט קרייא, משתמש ב-`date()` (זמן מקומי) או ב-`gmdate()` (זמן GMT). שתי הפונקציות משתמשות במחוזות פורמט (Format String). הן תמיד מוחזירות שמורות באנגלית. כדי להציג תאריכים בשפה המוגדרת במערכת (Locale), יש להשתמש ב-`strftime()` ו-`gmstrftime()`.

דוגמה 1: מספרי שבועות לפי ISO 8601

לפי תקן ISO 8601, השבוע הראשון של השנה הוא השבוע שבו יש לפחות ארבעה ימים. אך, ה-1 בינואר עשוי לפעמים להיות שירך רשמי לשבוע 53 של השנה הקודמת.

9.4.2.2 דוגמה 2: בעיות בשעון קיץ (DST)

בכל שנה בסביבות אוקטובר, מודיעים עשרות באגים שהם יום מסוים מופיע פעמיים בתצוגה. למעשה, היום שופיע פעמיים הוא התאריך שבו מסתיים שעון הקיץ.

הסיבה לכך היא שבמעבר משעון קיץ לחורף, יש למעשה 25 שעות בין חצות של ה-31 באוקטובר לחצות של ה-1 בנובמבר, ולא 24 שעות כפי שנוסף בלולאה פשוטה. ניתן לפתור זאת בשתי דרכים:

1. בחירת שעה אחרת ביום (למשל צהרים), כך שהסת�性 של שעון אחורית לא תחצה את גבול חצות.
2. "שימוש לרעה" מושכל בפונקציה `mktime()`: במקום להוסיף שניות (שמשתנות בגלל DST), מושכים את מספר הימים ישירות לפרק הזמן `mktime()`. הפונקציה תדע לחשב נכון את המעבר לחודש הבא ולטפל בשעות ה-DST באופן אוטומטי.

9.4.2.3 דוגמה 3: הצגת זמן מקומי באזורי זמן אחרים

לפעמים תרצו להציג זמן בפורמט מקומי עבור אזור זمان שונים. איור 9.4 מציג פלט של סקריפט המציג את ה-1 במרץ במדינות כמו ארה"ב, נורווגיה, הולנד וישראל.

הערה: לשם כך יש לוודא שה-Locales והגדירות אזורי הזמן מותקנים במערכת הפעלה שלכם.

9.4.3 ניתוח (Parsing) של פורטט תאריך

הפעולה ההופוכה לפרטט היא ניתוח מחרוזת טקסטואלית והפיכתה לחותמת זמן (timestamp). הפונקציה `strtotime()` מטפלת במגוון רחב של פורטטים.

9.7: פורטט תאריך/זמן נתמכים (חלקי)

- **1970-09-17:** פורטט ISO מועדף.
- **next Thursday 7pm:** יום חמישי הבא בשעה שבע בערב.
- **year 2 days ago 1:** שנה ויוםיים לאחר מכן הנוכחי.
- **yesterday / tomorrow:** אתמול /מחר.

9.5 מניפולציה של גרפיקה עם GD

במקום לעבור על כל הפונקציות,ណון בשני מקרים נפוצים של ספריית GD:

1. **טפסים מוגנִי-בוטים (CAPTCHA):** יצירה תמונה עם קוד שקשה לקריאה ממוחשבת על ידי עיוותים וטשטוש.
2. **תרשיימי עמודות (Bar Charts):** יצירה גרף הכלול צירים,תוויות, רקע וטקסט `TrueType`.

כדי להשתמש ב-GD, יש לוודא שהוא מופעלת ב-PHP (איור 9.5).

9.5.1 מקרה 1: טפסים חסיני בוטים

השלבים לייצרת תמונה צוֹרְכוּלִים:

- **יצירת קבבָּס**: שימוש ב-`imagecreatetruecolor()` לייצרת מרחבעובדת במילוני צבעים.
- **הקצת צבעים**: שימוש ב-`imagecolorallocate()` להגדרת צבעי רקע, גבול וטקסט (בפורמט RGB).
- **ציר טקסט**: שימוש ב-`imagettfttext()` המאפשר לצייר אותיות בגודלים ובזווית משתנות כדי להקשות על תוכנות זיהוי טקסט.
- **הוספת עיוותים**: ציר של מאות קווים קטנים אקראיים (אילם) והפעלת Anti-aliasing להחלהקת קצוות (איור 9.6).
- **פלט לדפסן**: שילוח כותרת `Content-type: image/png` ולאחריה פקודה `imagepng()` כדי להציג את התמונה.

9.5.2 מקרה 2: תרשימים עמודות

בגרפים מורכבים נשתמש בטכניקות מתקדמות יותר:

- **Alpha Blending**: שילוב ערוץ אלפא (שקיפות) כך שהעמודות יהיו שקופות למחצה וניתן יהיה לראות את הרקע דרך.
- **טיענת תמונה רקע**: שימוש ב-`imagecreatefrompng()` לטיענת תמונה קיימת כביסיס.
- **חישוב תיבת טקסט (Bounding Box)**: שימוש ב-`imagegetbbox()` כדי לבדוק מהן המידות של מחורצת טקסט לפני הציג, מה שמאפשר למרכז כותרות ותוויות לצורה מושלמת (איור 9.9).

Exif 9.5.3

Exif היא שיטה לשימרת מטא-דאטא (כמו זמן הצילום, מפתח צמצם ומהירות טריס) בתוך קובץ התמונה. PHP מאפשרת לקרוא נתונים אלו באמצעות פונקציית `exif` (איור 9.11), מה שמאפשר להציג מידע טכני על התמונה באופן אוטומטי.

שימוש ב-PEAR

10.1 מבוא

ספר זה הזכיר את PEAR מספר פעמים ב פרויקטים הקודמים. PEAR, קיצור של **PHP Extension and Application Repository**, היא מערכת חבילות עבור PHP. במהלך גרסה 4 של PHP, מספר משתמשים זינק, ואיתו גם כמה קטעי הקוד שניתן היה להוריד מאתרים שונים. חלק מהאתרים יצאו קוד שהוא עלייכם להעתיק ולהדיבוק לעורך שלכם, בעוד אחרים אפשרו להוריד ארכיבונים עם קבצי מקו. זה היה שימושי, אך נוצר צורך בדרך טוביה יותר לשיתוף ושימוש חוזר בקוד PHP, בדומה ל-CPAN של Perl.

פרויקט PEAR הוקם כדי לפתור בעיה זו על ידי אספקת כל התקנה ותחזקה וסטנדרטים לניהול קוד וגרסאות. כיום, PEAR מספק:

- **PEAR Installer** (כל לניהול חבילות)
- חבילות עם קוד ספריית PHP
- חבילות עם הרחבות PHP (הנקראות **PECL**)
- סטנדרטים של כתיבת קוד ב-PEAR, כולל סטנדרט גרסאות

הת-קבוצה של PEAR, אך כיום הוא מנוהל בנפרד, עם אתר ורשימות תפוצה משלה. עם זאת, PECL חולקים כלים ותשתיות: שניהם משתמשים ב-PEAR Installer, באותו פורמט חיבורו ובאותו

סטנדרט גרסאות. ההבדל הוא בסטנדרט הכתיבה: PECL עוקב אחרי הסטנדרט של PHP (לקוד C), בעוד-L-PEAR יש סטנדרט משלה.

בפרק זה נלמד על המושגים של PEAR וכייד להשתמש ב-Installer כדי להתקין ולנהל חבילות.

PEAR מושגי 10.2

סעיף זה מסביר מושגים כמו חבילות (packages), גרסאות (releases) ושיטת מספר הגרסאות.

10.2.1 חבילות (Packages)

כשרוצים להתקין משהו מ-PEAR, מורידים ומתקינים גרסה (release) מסוימת של חבילה. לכל חבילה מצורף מידע:

- שם החבילה (למשל: HTML_QuickForm)
- תקציר, תיאור וכתובת אתר הבית
- מתחץ אחד או יותר
- מידע על רישוי
- מספר גרסאות שהופצו

חבילות PEAR דומות לפורמטים אחרים כמו RPM של לינוקס או חבילות DBIAN. ההבדל העיקרי הוא שהן תוכננו להיות **בלתי תלויות בפלטפורמה**. ניתן להתקין אותן על כל פלטפורמה ש-PHP תומכת בה: UNIX, לינוקס, Windows ו-Mac OS X.

10.2.2 הפיצות (Releases)

בדומה ל-PHP עצמה, הקוד מופץ בקובץ tar.gz או zip יחד עם הוראות התקינה הנקבעות על ידי Installer. כל הפיצזה כוללת:

- מספר גרסה
- רשימה של קבצים והוראות התקינה
- מצב ההפיצה (snapshot, stable, beta, alpha, devel)

כברית מחדל, כשתקינו חבילה, תקבלו את הגרסה היציבה (stable) האخונה. למשל:

```
pear install XML_Parser $
```

הפקודה תוריד ות התקין את הגרסה היציבה האחרון (לדוגמה 1.1.0).

10.2.3 מספרי גרסאות

סטנדרט הגרסאות של PEAR מגדיר כיצד לפרש מספר גרסה וכייד להשוות בין שני מספרים. הוא מיושם באמצעות הפונקציה `version_compare()` של PHP.

10.2.3.1 פורמט מספר הגרסה

מספר גרסה יכול להיות פשוט כמו "1" או מורכב כמו "PEAR 8.1.1.2.9b2". PEAR מתחקדת לרוב בשלושה מספרים וחלק נוסף בסוף למקרים מיוחדים. התchapיר הוא:

[[Major [. minor [. patch]]] [dev | a | b | RC | pl | N

מספר גרסאות	Major	Minor	Patch	מצב (State)
1.2.1	1	2	1	—
1.0a1	1	0	—	alpha 1

10.2.3.2 השוואת מספרי גרסאות

ההשוואה מתחילה במספר ה-Major. אם הוא זהה, עוברים ל-Minor, ואז ל-Patch. אם כולם זהים, בודקים את ה-State.

- מצבים כמו a, dev (אלפא), b (בטא) ו-RC נחשבים **ישנים** יותר מגרסה ללא סימנת.
- מצב patch level (b) נחשב **חדש** יותר.

:Patch Minor לעומת Major

- **Patch**: תמיד בטוח לשדרוג (תיקוני באגים בלבד).
- **Minor**: הוספת תכונות חדשות; ה-API עשוי להשתנות מעט. כדאי לקרוא את ה-changelog.
- **Major**: שינוימשמעותי. לא מובטחת תאימות לאחר. הקוד עשוי להכתב מחדש.

PEAR השגת 10.3

ניתן להתקין את PEAR כחלק מהפצת PHP או דרך האתר go-pear.org

10.3.1 התקינה בLinux/UNIX

החל מגרסת PHP 4.3.0, PEAR מותקן כברירת מחדל עם PHP. עם זאת, אם אתם מקמפלים את PHP מהמקור, שימו לב לא להשתמש ב--- disable-pear או -- disable-pear, שכן ה-Installer זוקק למסמך שורת הפקודה (CLI) ולתומיכה ב-XML.

Windows 10.3.1.1

התקינו את גרסת ה-binary של PHP מהאתר הרשמי. כברירת מחדל, PHP תותקן ב-`C:\PHP\`.

10.3.2 Windows Installer

עליכם לוודא שה-`include_path` בקינפוג של PHP מוגדר נכון. ערכו את קובץ ה-`ini.php` שלכם וchno את הנתיב `PEAR\php\pear\c:` כדי ש-PHP יוכל למצוא את החבילות המותקנות.

10.3.3 go-pear.org

זהו אתר עם סקריפט PHP ייחיד שנitin להוריד ולהריץ כדי להתקין את הגרסה היציבה האחרונה של PEAR. הוא עובד על כל הפלטפורמות ונitin להריץ אותו משורת הפקודה או מהשרת.

דרישות מוקדמות:

מכוון sh-pear-go כתוב ב-PHP, אtmp זוקים לארסת CLI או CGI של PHP מותקנת. בדקו זאת על ידי הרצת -v.php.

הפעלת :go-pear

ניתן להריץ את התקינה יישורת מהאינטרנט (למשל עם lynx או curl) ולהעביר את הפלט ל-PHP:

```
$ lynx -source http://go-pear.org | php
```

האם תרצה שאprt על פקודות ספציפיות לניהול חבילות ב-PEAR, כגון חיפוש או הסרת חבילת?

איור 10.2 דוגמה לשינויים בקובץ config.in.

cut את מוכנים להשתמש ב-go-pear.

go-pear.org 10.3.3

go-pear.org הוא אתר אינטרנט המכיל סקריפט PHP ייחיד שנitin להוריד ולהריץ כדי להתקין את הגרסת היבנה האחורה של מתקין PEAR ושל מחלקות היסוד של PHP (PFC). go-pear PHP חוצה פלטפורמות ונitin להריץ אותו משורת הפקודה או משרת האינטרנט שלכם.

הפיצות PHP מגיעה עם מהדרה מסויימת של מתקין PEAR; מנגד, go-pear-go מספק לכם את מהדרות PEAR היוצאות חדשות ביותר. עם זאת, go-pear-amnmc מכיר את מבנה הספריות שלכם, אך הוא מתאים מאוד כדי לפעול אותו וינסה להתאים את התקנת ה-PEAR שלכם למבנה זה.

בסעיף זה תלמדו כיצד להשתמש ב-go-pear-go משורת הפקודה ומשרת האינטרנט, הן ב-XUNI והן ב-Windows.

10.3.3.1 דרישות קדם

מכוון sh-pear-go כתוב ב-PHP, דרושה לכם גרסת CGI או CLI של PHP כדי להריץ אותו מחוץ לשרת האינטרנט. כבירת מחדל, גרסת CLI מותקנת יחד עם מודול PHP של שרת האינטרנט. נסו להריץ php -v כדי לראות אם היא זמינה לכם:

```
Plaintext
PHP 5.0.0 (cli), Copyright (c) 1997-2004 The PHP Group
Zend Engine v2.0, Copyright (c) 1998-2004 Zend Technologies
```

כברית מחדל, פקודה php מותקנת בספריה /usr/local/bin/, או php-c: ב-XUNI, או php-c: ב-Windows, גרסת CLI של PHP עשויה להיקרא גם php-cli; במקרה זה, עליכם להקליד php-cli בכל דוגמה שבה כתוב פשוט .php.

10.3.3.2 PEAR מפעילים את

אם התקנת ה-PHP שלכם לא כוללת את PEAR, תוכלם להשתמש ב-go-pear-go ("מניע" (bootstrapper) אוניברסלי ל-PEAR. כל מה שאתה צריך זה גרסת CLI או CGI של PHP מותקנת איפשהו).

ניתן להוריד את הסקריפט go-pear ולהריץ אותו, או להריץ הכל בפקודה אחת:

```
lynx –source http://go-pear.org | php $
```

פקודה זו פשוט לוקחת את התוכן של <http://go-pear.org> ושולחת אותו ל-PHP לביצוע. אם אין לכם אולא, נסוי דרך חלופית:

באמצעות wget:

```
wget –O- http://go-pear.org | php $
```

באמצעות (ב-FreeBSD) fetch:

```
fetch –o – http://go-pear.org | php $
```

ב-Windows, איןCLI "משוק URL" מובנה, אך יתכן שתוכלו להשתמש בזרמי ה-URL של PHP (ויליאם ש-uses url אינו מבוטל ב-*php.ini*):

```
";(C:\> php-cli –r "include('http://go-pear.org
```

אם שום דבר לא עובד, פתחו את הכתובת בדף, שמרו את התוכן כ-*go-pear.php* והריצו:

```
C:\> php go-pear.php
```

הפלט יראה כך:

```
Plaintext  
  
!Welcome to go-pear  
. Go-pear will install the 'pear' command and all the files needed by it  
...  
:If you wish to abort, press Control-C now, or press Enter to continue
```

10.4 התקנת חבילות

סעיף זה עוסק בתחזוקת אוסף החבילות המותקנות שלכם.

למתקין PEAR יש ממשקים שונים (front-ends). ברירת המחדל היא ממשק שורת הפקודה (CLI).

10.4.1 שימוש בפקודת pear

פקודת *pear* היא CLI ההתקינה הראשי. הפקודה הראשונה שכדי להכיר היא *help*. הפקודה *pear help* תציג טקסט עזרה קצר *<subcommand>*.

אפשרויות (Options):

ניתן לציין אפשרויות (כמו -h) גם לפקודת *pear*-*subcommand* עצמה וגם לתת-הפקודה.

- *-v*: העלאת רמת הפירוט (Verbosity). רמה 0 היא שקטה, רמה 3 היא פלט לניפוי שגיאות (Debug).

- `-C` / `-c`: ציון קובץ הגדרות (Configuration).
- `-D` / `-d`: הגדרת פרמטר הגדרות עבור הריצה זו בלבד.
- `-S` / `-s`: שימרת (Store) שינויים הגדרות שהתבצעו עם אפשרות `-p`.

10.5 פרמטרים של הגדרות (Configuration Parameters)

למתקין PEAR יש פרמטרים רבים. החשובים שבהם הם ספריות היעד להתקינה:

פרמטר הגדרה	שם משתנה	ערך לדוגמה
ספריית PEAR ראשית	<code>php_dir</code>	<code>usr/share/pear/</code>
ספריית קבצי הרצה	<code>bin_dir</code>	<code>usr/bin/</code>
ספריית תיעוד	<code>doc_dir</code>	<code>usr/share/pear/docs/</code>
מצב חבילת מועדף	<code>preferred_state</code>	<code>stable</code>

שכבות הגדרה (Configuration Layers):

כל פרמטר יכול להיות מוגדר בשלוש שכבות:

1. **User (משתמש)**: הגדרות פרטיות של המשתמש.

2. **System (מערכת)**: הגדרות לכל המערכת.

3. **Default (ברירת מחדל)**: ערכיהם מובנים במתקין.

המתקין בודק תחילה את שכבת המשתמש; אם לא נמצא ערך, הוא עובר למערכת, ולבסוף לברירת המחדל.

שינויי ההגדרות:

כדי לשנות פרמטר לצמויות, השתמשו ב-`<pear config-set <parameter> <value>`. כדי לראות ערך הנוכחי, השתמשו ב-`<pear config-get <parameter>`.

שימוש לב: אם תשנו את `php_dir`, המתקין לא "יראה" יותר את החבילות שהותקנו בנטייב הישן, מכיוון שבספרייה זו נשמר בסיס הנתונים של החבילות המותקנות (תחת תיקיית `registry`).

10.6 פקודות PEAR

בוסף זה תלמדו את כל פקודות מתקין PEAR עבור התקינה ותחזקה של חבילות מערכת שלכם. עבור כל פקודה, יוצג הפלט של command `pear help` והסביר מפורט על כל אפשרות (option) שהפקודה מציעה. אם תבחינו בפקודות המופיעות בטקסט העזרה אך אין מקומות כאן, דעו כי אלו פקודות המשמשות מתחזקי חבילות PEAR במהלך הפיתוח. פקודות הפיתוח מקומות בפרק 12.

pear install 10.6.1

פקודה זו לוקחת את התוכן של קובץ חבילה ומתקינה את הקבצים בספריות ה-PEAR שהגדרכם. ניתן לציין את החבילה להתקנה כקובץ מקומי, כשם חבילה בלבד או כתובות HTTP מלאה. להלן טקסט העזרה עבור `:pear install`

```
pear help install $
```

```
... <pear install [options] <package>
```

מתקין חבילת PEAR אחת או יותר. ניתן לציין חבילה להתקנה באربע דרכים:

- **"Package-1.0.tgz"**: התקינה מקובץ מקומי.
- **"http://example.com/Package-1.0.tgz"**: התקינה מכל מקום ברשת.
- **"package.xml"**: התקנת החבילה המתוארת בקובץ package.xml. שימושי לבדיקות, או לעתיפת חבילת PEAR בתוך ניהול חבילות אחר כמו RPM.
- **"Package"**: תשאל השרת המוגדר שלכם (pear.php.net) והורדת החבילה החדשה ביותר באיכות/מצב המעודף (stable).

ניתן לציין יותר מחבילה אחת בו-זמנית, ונitin לערבות בין ארבע הדרכים הללו.

אפשרויות (Options):

- **--force-f**, **-f**: ידריס חבילות מותקנות חדשות יותר. שימושי לתקן התקנות פגומות או במהלך בדיקות.
- **--nodeps-n**: הטעלות מתלוויות (dependencies), התקינה בכל זאת. השתמשו בה רק אם אתם מבינים את ההשלכות, שכן החבילה עלולה לא לעבוד כלל.
- **--register-only-z**: רישום החבילה כמותקנת בלבד, ללא התקנת הקבצים בפועל. נועד לאפשר למנהל חבילות חיצוניים (כמו RPM) לבדוק את מרשם ה-PEAR.
- **--soft-s**: התקינה "רכה" – שדרוג אם החבילה קיימת, או התקינה אם לא.
- **--nobuild-B**, **-B**: אל תבנה הרחבות C. אם החבילה כוללת קוד C ואיינכם מעוניינים לקמפל אותו, השתמשו באפשרות זו.
- **--nocompress-Z**: בקש קבצים לא דחוסים בעת ההורדה (שימושי אם אין תמייה ב-gzip ב-PHP שלכם).
- **R DIR, --installroot=DIR-**: ספירת שורש שתשתמש להתקינה (בדומה ל-`INSTALL_ROOT` של PHP).
- **--ignore-errors--**: כפיה התקינה גם אם היו שגיאות. השתמשו בהירות!
- **--alldeps-a**: התקנת כל התלוויות הנדרשות והאופציונליות באופן אוטומטי.
- **--onlyreqdeps-o**: התקנת תלויות חובה בלבד, ללא תלויות אופציונליות.

דוגמאות לשימוש טיפוסי:

התקינה רגילה של חבילה ללא תלויות:

```
pear install Console_Table $
```

התקינה עם תלויות חובה בלבד (דילוג על אופציונליות):

```
pear install -o HTML_QuickForm $
```

התקנת חבילה וכל תלויותיה, בחיפוש אחר גרסאות במצב beta ומעלה:

```
pear -d preferred_state=beta install -a Services_Weather $
```

pear list 10.6.2

הפקודה `pear list` מציגה את רשימת החבילות המותקנות במערכת שלכם או את רשימת הקבצים בתוך חבילה ספציפית.

כדי לראות את כל החבילות המותקנות:

```
pear list $
```

כדי לבדוק אילו קבצים הותקנו עבור חבילה מסוימת (למשל `Net_Socket`):

```
pear list Net_Socket $
```

הפלט יראה את סוג הקובץ (`php`, `data`, `script`) ואת נתיב ההתקינה שלו.

pear info 10.6.3

הפקודה `pear info` מציגה מידע על חבילה מותקנת, קובץ חבילה (tarball) או קובץ הגדרת חבילה (XML).

המידע כולל תיאור, מתחזקים, רישוי, גרסה והערות שחרור (Release Notes).

ניתן להריץ את הפקודה על חבילה מותקנת:

```
pear info XML_RPC $
```

או על קובץ שטרם הותקן:

```
pear info XML-RPC-1.1.0.tgz $
```

pear list-all 10.6.4

בעוד `pear list` מציגה רק מה שמותקן אצלכם, `pear list-all` מציגה רשימה אלפביתית של **כל** החבילות הקיימות בשרת ה-PEAR, ביצירוף הגרסה העדכנית ביותר והגרסה המותקנת אצלכם (אם יש צזו).

pear list-upgrades 10.6.5

הפקודה `pear list-upgrades` משווה בין הגרסאות המותקנות אצלכם לבין הגרסאות החדשות ביותר הזרומות בשרת (בהתאם להגדרת `preferred_state`). היא תציג טבלה עם הגרסה המקומית, הגרסה המרוחקת וגודלה ההורדה.

pear upgrade 10.6.6

הפקודה `pear upgrade` מחליפה חבילה מותקנת אחת או יותר בגרסה חדשה יותר.

```
pear upgrade Log $
```

לפקודה זו יש את אותן אפשרויות כמו לפקודה `install` (למעט האפשרות `-s`). אם ברצונכם לשדרג בכל זאת לגרסה קיימת, השתמשו ב-`--force` (`f`).

pear upgrade-all 10.6.7

לנוחיותכם, פקודה זו משלבת את `list-upgrades`, `upgrade` ו-`i`, ומשרגת באופן אוטומטי את כל החבילות שיש להן גרסה חדשה יותר זינה.

```
pear upgrade-all $
```

pear uninstall 10.6.8

כדי למחוק חבילה, עליכם להסיר אותה מההתקנה (`uninstall`). להלן דוגמה:

```
pear uninstall Cache $
```

```
'Warning: Package 'services_weather' optionally depends on 'Cache  
uninstall ok: Cache
```

לפקודת ההסרה יש שלוש אפשרויות עיקריות:

```
... <pear uninstall [options] <package
```

ניתן להסיר חבילה אחת או יותר בו-זמנית.

:
אפשרויות (Options) :

- `n`, `--nodeps`: הטעלים מתלות, הסרה בכל זאת.
- `r`: אל תסיר את הקבצים בפועל, רק רשום את החביבה ככזו שאינה מותקנת.
- `R DIR`, `--installroot=DIR`: הגדרת ספריית שורש (בדומה ל-`INSTALL_ROOT` של PHP).
- `ignore-errors`: כפיה הפעולה גם אם התגלו שגיאות.

pear search 10.6.9

אם תרצו להתקין חבילה אך איןכם זוכרים את שמה המדויק, או אם אתם תוחים אם קיימת חבילה שמבצעת פעולה מסוימת (למשל X), תוכלנו לחפש אותה. הפקודה מבצעת חיפוש של מחרוזת בתוך שמות החבילות.

לדוגמה, חיפוש חבילות הקשורות ל-XML:

```
pear search xml $
```

הפלט מציג ארבע עמודות: שם החביבה, הגרסה העדכנית ביותר ברשות, הגרסה המותקנת מקומית (ריך אם אינה מותקנת), ותיאור קצר.

pear remote-list 10.6.10

פקודה זו מוצגה רשימה של כל החבילות והగרסאות היציבות הזמיןות במאגר החבילות המרוחק:

```
pear remote-list $
```

הבדל בין `list-all` והוא ש-`list-all` מוצגה רק את הגרסה האخונה הזמיןה, בעוד ש-`list` מוצגה גם את מה שモתקן אצלם. פקודה זו מציתת להגדרת ה-`preferred_state` שלכם (שכברית מחדל היא `stable`).

ניתן לשנות זאת זמנית עבור פקודה אחת כדי לראות גם גרסאות אלו:

```
pear -d preferred_state=alpha remote-list $
```

pear remote-info 10.6.11

כדי להציג מידע מפורט על חבילה שטרם התקנתם, השתמשו ב-`remote-info`. המידע (רישוי, קטגוריה, תיאור מלא) נלקח מהגרסה החדשה ביותר של החביבה בשורת.

pear download 10.6.12

פקודת `install` אינה שומרת את קובץ החביבה המקורי. אם אתם זקוקים לקובץ ה-`tarball` עצמו (להתקינה מאוחרת יותר או לכל מטרה אחרת), השתמשו ב-`download`:

```
pear download DB $
```

pear config-get / 10.6.14 pear config-set 10.6.13

כפי שראינו, פקודות אלו משמשות לציפוי ו שינוי של פרמטרי ההגדדות.

- `config-get`: מציג ערך (ניתן לציין שכבה ספציפית: `user`, `system` או `default`).
- `config-set`: משנה ערך. כברית מחדל השינוי מבצע בשכבות המשתמש (`user`).

pear config-show 10.6.15

מציג את כל הגדדות התצורה הנוכחות המישימות במערכת.

10.6.16 קיצורי דרך (Shortcuts)

לכל פקודה ב-PEAR יש קיצור דרך כדי לחסוך בהקלדה. ניתן לראותם באמצעות `pear help shortcuts`.

לדוגמה:

- `install` במקומם
- `upgrade` במקומם
- `uninstall` במקומם
- `config-set` במקומם
- `cs` במקומם

10.7 ממשקי מתקין (Front-Ends)

10.7.1 ממשק שורת הפקודה (CLI)

זהו המשק שבו השתמשנו עד כה, המציג פלט טקסטואלי בטרמינל.

10.7.2 ממשק Gtk (גרפי)

PEAR מציעת גם ממשק גרפי מובוסס Gtk (נפוץ בLinux, אך קיים גם ל-Windows). הוא דורש התקינה של PEAR_Frontend_Gtk ו-[php-gtk](#).

שימוש במתיקין הגרפי:

מפעילים אותו עם הפקודה: `G-pear`

הأشك הגרפי מאפשר ניווט נוח בין קטגוריות, סימון חבילות להתקינה או שדרוג באמצעות תיבות סימון (checkboxes), וצפיה בהגדירות התצורה בלשונות (tabs) מעוצבות. הקוד המבצע את התקינה בפועל הוא אותו קוד של גרסת שורת הפקודה; רק התצוגה משתנה.

10.8 סיכום

מטרת פרק זה הייתה להציג את התשתיית של PEAR ולהראות כיצד להתקין חבילות לשימושכם. בפרק הבא נלמד על חבילות חשובות ספציפיות וכיום לשילן בקוד שלכם.

פרק 11

宦官 PEAR חשובות

11.1 מבוא

בפרק זה נסקור דוגמאות לכמה מה宦官 PEAR הפופולריות ביותר. הספר אינו יכול להכיל דוגמאות לכל宦官 ו宦官, אך פרק זה יספק לכם לפחות מושג בסיסי לנושא.

11.2 שאלות מסדי נתונים

ראו פרק 6, "מסדי נתונים עם PHP", למבוא בנושא DB-PEAR.

11.3 מערכות מבניות (Template Systems)

מערכות מבניות הן רכיבי PHP המאפשרים להפריד בין הלוגיקה האפליקטיבית (Application Logic) לבין לוגיקת התצוגה (Display Logic), ומיציאות פורמט מבנית פשוט יותר מ-PHP עצמה.

זהו מצב אירוני ש-PHP, שהחלה את דרכה כשפה לבניית מבניות, משמשת כיום ליישום מערכות מבניות. אך ישנן סיבות טובות לכך מעבר להפרדת הקוד מההתצוגה, כגון מתן פורמט סימוני פשוט יותר למעצבים אחרים ושיפור השליטה של המפתחים ביצירת הדפים. לדוגמה, מערכת מבניות יכולה לתרגם טקסטים באופן אוטומטי או למילא טפסים בערבי בירית בלבד.

11.3.1 טרמינולוגיה של מבניות

לפני שנצלול למערכות השונות, כדאי להכיר את המושגים העיקריים:

טבלה 11.1: מיליון מונחים לתבניות

מילה משמעות
--- ---
תבנית (Template) תוכנית המתאר של הפלט; מכילה "ממלאי מקום" (Placeholders) ובלוקים.
הידור (Compile) התמרת התבנית לקוד PHP.
Placeholder (ממלא מקום) מחרוזת תחומה המוחלפת בנתונים בזמן הריצה.
Block (בלוק / מקטע) חלק בתבנית שנייה לחזור עליו עם נתונים שונים.

HTML_Template_IT 11.3.2

מערכת התבניות הראשונה של PEAR היא IT, HTML_Template, או בקיצור IT. זהה החברה הפופולרית ביותר ב-PEAR, אך היא גם האיטית ביותר מאחר שהיא מנתחת (parses) את התבניות בכל בקשה חדשה ואני מהדרת אותן לקוד PHP.

טיפול: **חבילת HTML_Template_Sigma** מספקת API תואם ל-IT, אך מהדרת את התבניות לקוד PHP לביצועים טובים יותר.

Placeholder 11.3.2.1

IT משתמש בסוגרים מסולסלים כתווי תייחום: {PageTitle}

11.3.2.2 דוגמה: בניית IT בסיסית

```
PHP
php?>

;"require_once "HTML/Template/IT.php
;("tpl = new HTML_Template_IT('./templates$"
;("tpl->loadTemplateFile('hello.tpl$"
;("tpl->setVariable("title$"
;(" tpl, 'שלום עולם!');
;!HTML_Template_IT', 'זהו בדיקה של IT->setVariable("body$"
;();tpl->show$
```

בקוד זה אנו טוענים קובץ התבנית, מדיררים משתנים (Variables) התואמים ל-Placeholders בקובץ -.tpl, ובסיום הפענקציה show() מבצעת את החלפות ומציג את הפלט.

HTML_Template_Flexy 11.3.3

הביבלה הבאה היא HTML_Template_Flexy (או פשוט Flexy). למרות שתכניות פשוטות של IT יעבדו בה, שתי היכולות שונות מאוד. Flexy עובדת על אובייקטים ומשתני מחלקה (Members) במקומם על מערכים אוטומטיים.

11.3.3.1 דוגמה: תבנית Flexy בסיסית

דורשת הגדרת ספריית תבניות וספריית הידור (Compile Directory) שחיבת להיות בעלת הרשאות כתיבה לשורת.

PHP

```
)tpl = new HTML_Template_Flexy(array$  
, 'templateDir' => 'templates'  
, 'compileDir' => 'compiled'  
);()  
;('tpl->compile('hello.tpl'$  
;view = new StdClass$  
;שלום עולם!';  
;(tpl->outputObject($view$
```

11.3.3.3 פורמט הסימון של Flexy

Flexy תומכת בתגים מתקדמים המאפשרים לוגיקה בסיסית בתחום התבנית:

טבלה 11.2: Tags סימן ב-Flexy

- {variable} – הצגת משתנה (כולל קידוד HTML כברירת מחדל).
- {method}() – קראיה למתודה מהאובייקט.
- {if:variable} – הצהרת תנאי פשוטה.
- {foreach:arr,val} – לולאה על מערך (מקביל ל-foreach ב-PHP).

11.3.3.4 טיפול במאפייני HTML

אחד היכולות המעניינות של Flexy היא שימוש במאפיינים (Attributes) בתחום ה-HTML עצמו לשיליטה בלוגיקה, למשל:

```
<li flexy:foreach="list_entries,text">{text}</li>
```

11.4 אינומת זהות (Authentication)

חברית PEAR Auth היא שכבה אימות מופשטת המשמשת ב-"Containers" כדי להתmeshק למערכות אימות שונות: קבצי סיסמה, מסדי נתונים, LDAP, IMAP ועוד.

11.4.1 סקירה כללית

החברה משתמש בבקשות POST להעברת שם משתמש וסיסמה. ברגע שהאימות מצליח, Auth משתמש ב-Sessions כדי לעקוב אחר המשתמש. היתרון הוא שהבדיקה מול מסד הנתונים (שעשוי להיות "יקרה" במקרים מסוימים) מתבצעת פעם אחת בלבד בתחילת הסשן ולא בכל בקשה HTTP.

11.4.2 דוגמה: Auth עם קובץ סיסמאות

```
PHP
;require_once 'Auth.php'

// הגדרת אימות מול קובץ htpasswd.
;("auth = new Auth("File", ".htpasswd", "login_function$"

;();auth->start$

{
    if (!$auth->getAuth

        // אם לא מאומת, הפונקציה login_function תופעל והסקריפט ייעזר
        ;echo "אתה מחובר!";
    }

}

;$auth->start$()
```

הקריאה `-$auth->start()` מנהלת את הסשן באופן אוטומטי, בודקת משתני POST ומיצגה את טופס ההתחברות מיידית הצורך.

11.4.3 דוגמה: Auth עם מסד נתונים ונתוני משתמש

דוגמה זו מרחיבת את הקוד המקורי שימוש במסד נתונים לאחסון שמות משתמשים וסיסמאות, ואינה מספקת טופס ההתחברות בהתאם אישית אלא משתמש בטופס מובנה.

בנוסף, תלמדו כיצד לצרף מידע נוסף הרלוונטי למשתמש לסשן ההתחברות, וכךיך לישם שיטות שפוקעים באופן טוב יותר כדי לסייע בהתחברות נשמרם בסשן, להלן דוגמה לנתוני סשן של Auth:

```
PHP
)SESSION["_authsession"] = array_$
    ,("data" => array"
        ,("registered" => 1"
            ,("username" => "guest"
                ,("timestamp" => 1075642673"
                    ,("idle" => 1075643017"
```

(

משתנה ה-PHP שמחזיק את סשן ה-Auth נקרא תמיד `$_SESSION['authSession']`. המפתחות בתוך מערך זה מוצגים בטבלה 11.5.

הערה: הסיסמה אינה נשמרת בסשן. אין בכך צורך — המשתמש כבר אומת. הסשן מכיל רק מידע שנשלף לאחר אימות מוצלח, ומידע שמתעדכן כל הזמן (כמו `$_SESSION['idle']`).

טבלה 11.5: משתני סשן של Auth

שם מפתח תיאור
--- ---:
data מקום לאחסון נתונים שסופקו על ידי המשתמש, או נתונים שנטענו ממסד הנתונים.
registered מוגדר תמיד כ-TRUE כאשר המשתמש מחובר.
username מכיל את שם המשתמש.
timestamp מכיל את הזמן (<code>time()</code>) בו המשתמש התחבר.
idle מכיל את זמן הפעולות האחרונות בסשן.

ראית המבנה הזה עוזרת להבין כיצד Auth עובד. לדוגמה, כדי להפסיק התחברות לאחר N שעות, Auth בודק את המשתנה `$_SESSION['idle']`. כדי להפסיק התחברות לאחר N דקות של חוסר פעילות, הוא בודק את המשתנה `$_SESSION['timestamp']`.

11.4.4 שיקולי אבטחה ב-Auth

סוגיית האבטחה הגדולה ביותר היא ההסתמכות על Sessions PHP. מזהה הסשן (Session ID) הוא סודי, אך הוא כל מה שימוש זדוני צריך כדי להשתלט על חשבונו.

11.4.4.1 טיפ אבטחה 1: ביטול sid_trans_session

תמונה זו מוסיפה את מזהה הסשן לכל קישור בדף כפרמטר GET. זה שוקל להצבת שם המשתמש והסיסמה בכתובת ה-URL. קיים סיכון שהמזהה ידלוּך דרך כוורת ה-HTTP Referer לאתרים חיצוניים. יש לבטל אפשרות זו כדי למנוע דילפה.

11.4.4.2 טיפ אבטחה 2: שימוש ב-HTTP_Hash

אם ברצונכם לתמוך במשתמשים ללא עוגיות (Cookies), התקינו את HTTP_Hash. חבילה זו מחליפה את טופס ההתחברות בחילון קופץ סטנדרטי של דפדן לאימות HTTP.

11.4.4.3 טיפ אבטחה 3: שימוש ב-HTTPS

שימוש ב-HTTPS מגן על שמות משתמש, סיסמות ומזהי סשן מפני ציתות לחבילות מידע בראשת (Sniffers).

11.4.5 שיקולי סקלibility (Scalability)

마חר sh-Auth משמש בסשנים המאוחסנים כבירת מחדל בקבצים מקומיים, תיתקל בעיה באתר המבוזר בין מספר שירותים (Load Balancing). אם משתמש התחבר בשרת 1, והבקשה הבאה שלו תגיע לשרת 2, לשרת 2 לא ימצא את קובץ הסשן המקומי.

פתרונות אפשריים:

1. **Load-Balancing לפי ID Session**: הבטחה שכל הבקשות מאותו שן יגיעו תמיד לאותו שרת.
2. **אחסון שן משותף**: שימוש במסד נתונים או מערכת כמו msession לשיתוף נתונים הסשן בין כל שרת האינטרנט.

11.5 טיפול בטפסים (Form Handling)

בנייה טפסי HTML ידנית היא פשוטה, אך כשלעצמה הוצרך באימות נתונים (Validation), טפסים מרובי דפים או שימוש בתבניות, עדיף להשתמש במכלול טפסים כמו **HTML_QuickForm**.

חביבה זו מייצגת כל אלמנט בטופס כאובייקט. ניתן להגדיר חוקי אימוט שיבוצעו לצד הלוקה (JavaScript) או לצד השרת באופן אוטומטי.

דוגמה לקבלת נתונים:

הmethod validate()מחזירה true אם נשלחו נתונים וهم עומדים בכל החוקים שהוגדרו:

PHP

```
{ ()if ($form->validate  
// עיבוד הנתונים  
{ }
```

11.6 מטמון (Caching)

PEAR מציעה שתי חבילות: **Cache_Lite** ו-**Cache**. כפי שהיא מרמז, **Cache_Lite** מתוכננת להיות מהירה וקלה יותר על חשבון גמישות מסוימת.

Cache_Lite 11.6.1

החבורה מציעה מטמון מבוסס קבצים עבורו:

- נתונים כלליים.
- פלט (Output) של PHP.
- ערכי חזרה של פונקציות.

היא בינה כך שלא תטען מחלקות PEAR נוספות אלא אם יש בכך צורך, מה ששומר על זמן תגובה נמוך במיוחד (Latency).

11.7 סיכום

פרק זה הציג כמה מה宦官 הידועות הנפוצות ביזור ב-PEAR. הכוונה הייתה להזכיר אתכם לעניינים כדי שתוכלו להמשיך לחקור בעצמכם את התיעוד המקורי בכתב: <http://pear.php.net/manual>.

פרק 12

12.1 מבוא

בפרק 10 ו-11 למדתם כיצד להשתמש במתќין PEAR ו宦官 PEAR בקוד שלכם. בפרק זה תלמדו כיצד לבנות宦官 PEAR משלכם — בין אם לשימוש פנימי בארגון ובין אם לפרוסום ברישוין קוד פתוח דרך שרת ההפצה של PEAR.

12.2 סטנדרטים של PEAR

תקן הקידוד של PEAR (או PCS) נועד בעיקר למפתח宦官, אך הוא שימושי לכל מי שמשתמש בהן כדי להבין את המבנה והשמות הגרפיים.

12.2.1 מתן שמות לסמליים (Naming)

- **קבועים (Constants):** אוטו-יות גדולות בלבד, עם שם החביבה כתחילה.
 - דוגמה: `PEAR_ERROR_DIE`, `AUTH_EXPIRED`.
 - ב-5 PHP ניתן להשתמש ב-`const`, `class`, `Auth::EXPIRED`.
- **משתנים גלובליים:** רצוי להימנע מהם ב-5 PHP (לטובת משתני محلקה סטטיים). אם חייבים, התבונתי היא `$_Package_Name_variable`.
- **פונקציות:** שם החביבה כתחילה, ולאחריו שם הפונקציה בשיטת camelCase (המתחילה באות קטנה).
 - דוגמה: `Package_Name_functionName()`.
 - פונקציה "פרטית" (פנימית לחביבה) תתחיל בקיצור `Package_Name_privateFunction_()`.
- **מחלקות (Classes):** שם החביבה משמש כתחילה (למשל `Name_Class`). חrieg נפוץ הוא מחלקות של דרייברים (כמו `DB_mysql`), שם החלק המשתנה עשוי להיות באותיות קטנות.
- **METHODS (Methods):** אותן קטנה בהתחילה, וכל מילה נוספת גודלה (למשל `toHTML()`). ב-5 PHP יש להשתמש במילוי המפתח `private/public`, בעוד שבקוד תואם 4 משתמשים בקוו תחתון לפני שם המתודה כדי לסייע שהיא פרטית.

12.2.2 הזחה (Indentation)

PEAR משתמש בהזחה של ארבעה רווחים (לא שימוש ב-`\t`!). הסיבה לכך היא שתווי Tab מרונדרים באופן שונה בערבי טקסט שונים, מה שפוגע בעקבות הקוד עבור מפתחים אחרים.

12.3 ניהול גרסאות (Versioning)

- גרסה ייצבה ראשונה חייבת להיות **1.0.0**.

- גרסאות לפני היציבות יהיו בתבנית **0.x** (גרסאות פיתוח).
- גרסה **N.1** חייבת להיות תואמת לגרסה **M** (כאשר $M > N$).
- שנייה בגרסה הראשית (**Major**, למשל `m-1.x` ל-`2.x`) מאפשר שבירת תאימות לאחר. במקרה זה, לעיתים משנהים את שם החבילה ל-`2-Foo`.

12.4 סביבת שורת פקודה (CLI)

PEAR מאפשרת לכלול סקריפטים של שורת פקודה בחבילות. כדי לפתור בעיות של נתיבים למנוע ה-PHP, משתמשים בשתי סביבה כמו `BIN_PEAR_PHP_BIN`. ניתן להטמע קוד PHP בתוך סקריפטים של Shell (לינוקס) או קבצי Bat (וינדוס) בצורה חכמה שמתעלמת ממועד המעטפת ומריצה רק את ה-PHP.

12.5 יסודות

כasher בונים חבילה, יש להקפיד על שלושה עקרונות בשילוב קבצים:

1. **שימוש ב-`require_once` בלבד**: כדי למנוע שגיאות של הגדרה כפולה של מחלקות.
2. **מתאים בין שם המחלקה לשם הקובץ**: PEAR דוגלת בעיקרון של "מחלקה אחת לקובץ". שם הקובץ נוצר משם המחלקה על ידי הילפה קווים תחתונים בולוסניים.
 - o `XML_Parser` נמצא בנתיב `XML/Parser.php`.
3. **אנקופסולציה של `Includes`**: כל קובץ צריך להציג על המחלקות שבו הוא תלוי.

תרשים 12.1: תלויות משורשות (Nested dependencies).

הסמל היחיד שמתיחסים אליו ישירות בתוך `App.kph` הוא המחלקה `B` מתוך `App.kph.B`. הקובץ כולל איננו מתיחס למחלקה `C`. למעשה, עליהם להניח `App.kph.C` כולל איננו מודע לקיומו של `App.kph.C`. על ידי הקפדה על עיקרון זה, אינכם מסתמכים על המבנה הפנימי של חבילה `B`, עשוי להשתנות בעתיד. זה הופך את החבילה שלכם לעמידה יותר בפניו שינויים בחבילות אחרות.

12.5.2 טיפול בשגיאות

קוד PEAR מדווח וטופס שגיאות באמצעות ה-API לטיפול בשגיאות של PEAR, המפורט בפרק 7.

12.6 בניית חבילות

בחלק זה נלמד כיצד לבנות חבילות משלכם. להלן דוגמה לחבילה המכילה מחלקה PHP, סקריפט שורת פקודה, בדיקת רגרסיה וקובץ תיאור חבילה (`package.xml`).

PEAR: HelloWorld 12.6.1

זהו המקירה המינימלי — קובץ PHP בודד הממש מחלקה בשם `HelloWorld`:

```
PHP  
php?>  
**/  
HelloWorld.php*
```

* קובץ `HelloWorld.php`

```
/*
```

```
} class HelloWorld
```

```
} (function display($name = null
```

```
} (if ($name
```

```
; "print "Hello, $name!\n
```

```
} else {
```

```
; "print "Hello, World!\n
```

```
{
```

```
{
```

```
{
```

```
<?
```

בנוסף, נכין סקורייפט שורט פקודה וקובץ בדיקה בתבנית .phpt. קובץ .phpt מחולק למקטעים המתחילה ב---SECTION---.

טבלה 12.2: כותרות מקטעי בדיקה

מקטע תיאור

--- ---:

TEST* תיאור קצר של הבדיקה.

FILE* קוד הבדיקה בפועל.

*EXPECT הפלט המדויק שהקוד אמור להציג.

EXPECTF פלט צפיי הכלול "מלאי מקום" "(Placeholders) ..".

SKIPIF קוד שקובע אם לדלג על הבדיקה.

12.6.2 בנית קובץ Tarball

לאחר הרכבת הקבצים, ניצור את החבילה באמצעות הפקודה:

```
pear package $
```

התוצאה תהיה קובץ בשם HelloWorld-1.0.tgz. קובץ זה ניתן להתקינה בכל מכונה שיש בה מתקין PEAR.

12.6.3 אימות (Verification)

השתמשו בפקודה `pear package-validate` (או בקיצור `pk validate`) כדי לוודא שהחבילת תקינה. האימוטה ייכשל אם הגדרתם סמלים מוחוץ ל-`Namespace` של החבילת, אם חסרים אלמנטים ב-`package.xml`, או אם רשימת הקבצים אינה תקינה.

12.7 מבנה קובץ `package.xml`

זהו קובץ התיאור של החבילת. להלן האלמנטים העיקריים העיקריים:

- `<package>`: אלמנט השורש.
- `<name>`: שם החבילת (תליי רישיות).
- `<summary>`: תיאור בשורה אחת.
- `<description>`: תיאור מלא של החבילת.
- `<developer>`: רשימת המתזקקים (כולל שם, אימייל ותפקיד כמו `lead` או `maintainer`).
- `<maintainers>`: מילויים נוספים המתייחסים ל-
- `<release>`: מידע על הגרסה הנוכחית (תאריך, מצב כמו `stable`, והערות שחרור).

12.7.2.6 רשימת קבצים (`<filelist>`)

מכיל אלמנטים מסווגים `<dir>` (ספרייה) ו-`<file>` (קובץ).

לכל קובץ ניתן להגדיר תפקיד (Role):

- `php`: קובץ מקור.
- `test`: בדיקת רגסיה.
- `doc`: תיעוד.
- `script`: סקריפט להרצה.

12.8 תלויות (Dependencies)

אחד היתרונות המרכזיים בשימוש ב-PEAR הוא שימוש חוזר בקוד. עם זאת, כאשר משתמשים בקוד מחבילת אחרת, נוצרות תלויות בין החבילות. יש לציין תלויות אלו בתיאור החבילת כדי לידע את המשתמשים.

12.8.2 האלמנט `<dep>`

אלמנט זה מתאר תלות בודדת.

- **מאפיין type**: סוג התלות (למשל `php` עבור גרסת PHP, `pkg` עבור חבילת PEAR אחרת, או `ext` עבור הרחבה PHP ספציפית).
- **מאפיין rel**: מגידר את היחס לגרסתה (למשל `ge` עבור "גדול או שווה ל-", `lt` עבור "קטן מ-", או `eq` עבור "שווה ל-").
- **מאפיין optional**: מאפשר להגדיר שתלות מסוימת אינה חובה להתקינה, אלא רק מוסיפה פונקציונליות (ערךים: `yes` או `no`).

12.8.4 סיבות להימנע מטלויות

טלויות הן מנגןן הכרחי, אך הן עלולות להפוך ל-"טלויות דוחירות" (Run-away dependencies) – מצב שבו חבילה אחת גוררת אחרת עשרות חבילות אחרות שלא לצורך.

ניהול תלויות מורכבות גוזל זמן פיתוח יקר. אם ההבדל בין הוסףת תלות לבין כתיבת הקוד בעצמכם הוא רק כמה שורות, כדאי לשקלו שוב האם התלות הכרחית. בתרשים 12.2 (בספר) מוצגות "טלויות שעירות", שבהן חבילה A תלואה ב-B ו-C, אך אלו תלויות בתווך ב-D, E, F ו-G.

12.9 החלפות מחרוזות (String Substitutions)

ניתן להגדיר החלפות המבוצעות בקבצים בזמן ההתקנה. זה שימושי לסנכרון נתיבי ברירת מחדל עם הגדרות ה-PEAR של המשתמש.

- **האלמנט <replace>**: מגדר חילוף. למשל, ניתן להחליף תבנית כמו `@php_bin@` בתיב `"type="pear-config`.

12.10 הכללת קוד C

חבילת PEAR יכולה לכלול קוד C או C++. מתקין PEAR ירץ את תהליך הבנייה (build) באופן אוטומטי אם ישנה קבצים עם התפקיד `src="role=src"`. ניתן להגדיר אפשרויות הגדרה (configureoptions) שישאלו את המשתמש שאלות בזמן ההתקנה (למשל נתיב לספריות חיצונית).

12.11 תהליכי השחרור ב-PEAR

אם אתם מפרסמים חבילה דרך `pear.php.net`, עליום לעבור תהליכי קהילתית:

1. **הצעה חבילה (Proposal)**: הגשת הצעה באתר. הקהילה דנה בשם החבילה ובניהצotta.
2. **מצבעה**: מפתחי PEAR מצביעים על ההצעה.
3. **יצירת חבילה**: לאחר אישור, מנהל מערכת מאשר את יצירת החבילה בשרת.
4. **אריזה (Rolling a tarball)**: יוצרת קובץ ה-`.tgz` בעזרת הפקודה `pear package`.
5. **בדיקות (QA)**: הרצת בדיקות רגיסטריה.
6. **העלאה (Upload)**: פרסום המהדרה לעולם.

12.12 אריזה (Packaging)

ה-PEAR Packager מבצע מספר פעולות אוטומטיות:

- **ניתוח קוד המקור**: בודק תליות ונזווה מחלקות ופונקציות כדי לוודא עמידה בתקני הקידוד.
- **יצירת MD5 Checksum**: מחשב חתימה דיגיטלית לכל קובץ כדי לוודא שהקובצים לא הושחתו בזמן ההורדה.
- **עדכון package.xml**: יוצר קובץ הגדרות סופי הכלל את כל המידע שנאוסף.

12.13 העלה וסיום

לאחר הבדיקות, מעלים את החבילה לדף ה-Upload ב-PEAR. לאחר אימונות המידע, החבילה מפורסמת ומופיעה ברשימת "השחרורים האחרונים". הכרזה נשלחת אוטומטית לשימוש התפוצה של הקהילה.

12.14 סיכום

פרק זה סיפק לכם את הכלים להפוך מתורמים פסיביים למפתחים פעילים בקהילה ה-PEAR, או להקים תשתיות פיצה פנימית בארגון שלכם.

פרק 13

13.1 מבוא

"קידמה פירושה לא לעמוד במקום כשל השאר זה". — וודרו וילסן

עם כל כך הרבה תכונות חדשות, במיוחד בתחום ה-Object-Oriented Object-Oriented (תכנות מונחה עצמים), כמעט בלי אפשרי שכל סקריפט שנכתב ב-4 PHP יעבד ללא שינוי ב-5 PHP. צוות הפיתוח ניסה להפוך את המעבר לקל ככל האפשר, אך ישנן אי-תאמיות קטנות. פרק זה מכוסה את הדברים שעולמים להישבר וכייזד לתקן.

13.2 מודל האובייקטים

ב-5 PHP יש מודל אובייקטים חדש המשנה את האופן שבו אובייקטים מנוהלים. בחלוקת מהמקרים, ניתן להורות ל-5 PHP לחזור להתנהגות של 4 PHP באמצעות "מצב תאימות" (Compatibility Mode).

13.3 העברת אובייקטים לפונקציות

אחד השינויים הגדולים ביותר הוא שאובייקטים המועברים לפונקציה **אינם מועתקים יותר כברירת מחדל**. ב-4 PHP, העברת אובייקט יוצרה עותק שלו. ב-5 PHP, מועבר "ידיית" (Handle) לאובייקט המקורי, וכך שינוי בתוך הפונקציה משפיע על האובייקט המקורי.

פתרון: אם ברצונך לשמור על התנהגות 4 PHP, עליך להשתמש באופרטור `clone`:

```
$(display_quoted(clone $s
```

לחילופין, ניתן להפעיל ב-`ini_set` את ההגדירה `.zend.ze1_compatibility_mode = 1`.

13.4 מצב תאימות (Compatibility Mode)

הפעלת מצב זה משפיעה על מספר דברים מעבר להעברת אובייקטים:

1. **המרה (Casting):** ב-4 PHP, המרת אובייקט ריק ל-`Integer` ריק ל-`Integer` 0. ב-5 PHP זה תמיד מחזיר 1.
2. **השווות אובייקטים:** ב-4 PHP, האופרטור `==` החזיר אמת אם לכל התכונות היו אותם ערכים. ב-5 PHP, הוא מחזיר אמת רק אם מדובר באותו אובייקט (אותה ידיית).

13.5 שינויים נוספים

ישנם שינויים ש"מצב תאימות" לא פותר:

- **השמה `for`:** ב-5 PHP לא ניתן לבצע השמה למשתנה `this` (למשל: `this = new $()`). זה יגרום לשגיאה קטלנית. הפתרון הוא לעצב מחדש המחלקה, למשל `Singleton`.
- **הפונקציה `get_class()`:** ב-4 PHP שם המחלקה תמיד הוחזר באותיות קטנות. ב-5 PHP, השם מוחזר כפי שהוגדר (Case-preserved). כדי לשמור על תאימות, מומלץ להשתמש `((strtolower(get_class($obj))`.

13.6 שגיאות E_STRICT

PHP 5 הציגה רמת שגיאה חדשה להתריע על שימוש בתכונות מיושנות (Deprecated). כדי לראותן, יש להגדיר `error_reporting` לערך `E_ALL | E_STRICT` (או המספר 4095).

- **יצירת אובייקטים אוטומטיות:** השמה למשתנה שלא הוגדר כאובייקט (למשל `$name = $person->name`).
אקספליקט (`__construct`) תבעוד אך תעלה שגיאת `E_STRICT`.
- **Constructor:** ב-5 PHP הוכנס המבנה האחד `__construct()`. אם קיימת גם פונקציה בשם `__construct` (סוגנן יישן) וגם `__construct()`, תעלה שגיאת `E_STRICT`.
- **ירושה:** חתימת הפונקציה (הפרמטרים שהוא מקבלת) חייבת להיות זהה במחלקה הבן ובמחלקה האב.

13.7 בעיות תאימות אחרות

- **CLI (ממשק שורת פקודה):** בגרסת הוינדוס, הקובץ שונה מ-`php.exe` ל-`php-cgi.exe`.
- **MySQL:** הספרייה של MySQL אינה כוללה יותר כבירה מוחלט בטור 5 PHP. יש להשתמש בספרייה חיצונית (או בתוסף MySQLi החדש).

13.8 שינוי בפונקציות

1. **array_merge()**: ב-5 PHP פונקציה זו מקבלת רק **מערכות**. העברת משתנה רגיל (כמו מחזוזת או מספר) תגרום לשגיאת זהירה ותחזיר מערך ריק.
2. **strpos() ו-strlen()**: ב-5 PHP, חיפוש המיקום האחרון של מחזוזת בתוך מחזוזת משתמש בכל ה-`Needle` (מחוזות החיפוש), בניגוד ל-4 PHP שהשתמש רק בתו הראשון של מחזוזת החיפוש.

13.9 סיכום

המעבר ל-5 PHP מביא אליו עוצמה רבה אך דורש תשומת לב לפרטים קטנים במודול האובייקטים ובפונקציות הליבה. שימוש ב-`E_STRICT` הוא הדרכת הטובה ביותר לוודא שהקוד שלכם מוכן לעתיד.

13.1 מבוא

"קידמה פירשה לא לעמוד במקום כסכל השאר זז". — וודרו וילסון

עם כל כך הרבה תכונות חדשות, במיוחד בתחום Object-Oriented (תכונות מנחה עצמים), כמעט בלי אפשרות שכל סקריפט שנכתב ב-4 PHP יעבד ללא שינוי ב-5 PHP. צוות הפיתוח ניסה להפוך את המעבר קל ככל האפשר, אך ישנן einige תאיימות קטנות. פרק זה מכסה את הדברים שעולים להישבר וכיוצא בהם.

13.2 מודל האובייקטים

ב-5 PHP יש מודל אובייקטים חדש המשנה את האופן שבו אובייקטים מנוהלים. בחלוקת מהמקרים, ניתן להורות ל-5 PHP לחזור להתנהגות של 4 PHP באמצעות " מצב תאימות" (Compatibility Mode).

13.3 העברת אובייקטים לפונקציות

אחד השינויים הגדולים ביותר הוא שימוש אובייקטים המועברים לפונקציה **אינם מועתקים יותר כבירה מוחלט**. ב-4 PHP, העברת אובייקט יוצרה עותק שלו. ב-5 PHP, מועבר "ידיית" (Handle) לאובייקט המקורי, וכך שניי בתוך הפונקציה משפיע על האובייקט המקורי.

פתרון: אם ברצונך לשמר על התנהגות 4 PHP, עליך להשתמש באופרטור `clone`:

```
;(display_quoted(clone $s
```

```
.zend.ze1_compatibility_mode = 1
```

13.4 מצב תאימות (Compatibility Mode)

הפעלת מצב זה משפיעה על מספר דברים מעבר להעברת אובייקטים:

1. **המרה (Casting):** ב-4 PHP, המרת אובייקט ריק ל-`Integer` החזירה 0. ב-5 PHP זה תמיד מוחזר 1.
2. **השווות אובייקטים:** ב-4 PHP, האופרטור `==` החזיר אמת אם לכל התכונות היו אותן ערכים. ב-5 PHP, הוא מוחזר אמת רק אם מדובר באותו אובייקט (אותה ידית).

13.5 שינויים נוספים

ישנים שינויים ש"מצב תאימות" לא פותר:

- **השמה `this`:** ב-5 PHP לא ניתן לבצע השמה למשנה `this` (למשל: `$new = new this();`). זה יגרום לשגיאה קטלנית. הפתרון הוא לעצב מחדש המחלקה, למשל באמצעות `Singleton`.
- **הפונקציה `get_class()`:** ב-4 PHP שם המחלקה תמיד הוחזר באותיות קטנות. ב-5 PHP, השם מוחזר כפי שהוא (Case-preserved). כדי לשמר על תאימות, מומלץ להשתמש `((strtolower(get_class($obj))`.

13.6 שגיאות E_STRICT

PHP 5 הציגה רמת שגיאה חדשה שנועדה להתריע על שימוש בתכונות מיושנות (Deprecated). כדי לראותן, יש להגיד `error_reporting E_ALL | E_STRICT` (או המספר 4095).

- **יצירת אובייקטים אוטומטית:** השמה למשנה שלא הוגדר כאובייקט (למשל `$person->name = $x;`) תבעוד אך עליה שגיאת `E_STRICT`.
- **ב-5 PHP:** ב-5 PHP הוכנס המבנה האחד `__construct()`. אם קיימת גם פונקציה בשם המחלקה (סוגנן יישן) וגם `construct()`, עליה שגיאת `E_STRICT`.
- **ירושה:** חתימת הפונקציה (הפרמטרים שהוא מקבלת) חייבת להיות זהה במחלקה הבן ובמחלקה האב.

13.7 בעיות תאימות אחרות

- **CLI (ממשק שורת פקודה):** בגרסת הוינדוס, הקובץ שונה מ-`php.exe` ל-`php-cgi.exe`.
- **MySQL:** הספרייה של MySQL אינה כוללה יותר כבירה מיוחדת בטור 5 PHP. יש להשתמש בספרייה חיצונית (או בתוסף MySQLi החדש).

13.8 שינויים בפונקציות

- . 1. ()**array_merge()**: ב-5 PHP פונקציה זו מקבלת רק **מערכים**. העברת משתנה רגיל (כמו מחרוזת או מספר) תגרום לשגיאת אזהרה ותחזיר מערך ריק.
- . 2. **strpos() ו-stripos()**: ב-5 PHP, חיפוש המיקום האחרון של מחרוזת בתוך מחרוזת משתמש בכל ה-Needle (מחרוזת החיפוש), בניגוד ל-4 PHP שהשתמש רק בתו הראשון של מחרוזת החיפוש.

13.9 סיכום

המעבר ל-5 PHP מביא אליו עוצמה רבה אך דורש תשומת לב לפרטים קטנים במודל האובייקטיבים ובפונקציות הליבה. שימוש ב-**E_STRICT** הוא הדרך הטובה ביותר לוודא שהקוד שלכם מוכן לעתיד.

פרק 14

14.1 מבוא

לכל אפליקציה יש יעד ביצועים. תמיד יהיו מגבלות משאבים כמו מעבד (CPU), זיכרון, רוחב פס של הדיסק ועוד. אם האתר שלכם צפוי לתנועה משמעותית (מיליוני צפיות ביום), כדאי להקדים זמן לכיוון ביצועים (Performance Tuning).

פרק זה ילמד אתכם כיצד:

- לעצב אפליקציות PHP בעלות ביצועים גבוהים.
- להשתמש בסוגים שונים של שיטות מטמון (Caching).
- לבצע ניתוח ביצועים (Profiling) לקוד PHP.
- לייעל את הקוד ואת מסד הנתונים.
- לבצע אופטימיזציה לשרת האינטרנט ולמערכת הפעלה.

14.2 עיצוב לביצועים

התכוון לביצועים צריך להתחיל כבר בשלב העיצוב. עדיף להימנע מאופטימיזציה מאוחרת של הקוד, שעלולה להוביל לבאגים או לקוד שקשה לתקן.

14.2.1 טיפ עיצוב #1: היזרו ממצבים (State)

כדי לאפשר לאפליקציה שלכם לגדול (Scaling), עדיף להימנע ככל האפשר משמרות "מצב" לצד השרת בין בקשות. "מצב" הוא מידע שעובר מבקשת אחת לבאה אחרת (כמו שם משתמש או התקדמות בטופס).

ניהול Sessions: כבירות מחדל, PHP שומרת מייד על ה-**Sessions** בקבצים מקומיים. הבעה מתחילה כישיש יותר משרת אחד.

כפי שמצוג בתרשים 14.1 (בספר), אם משתמש נשלח לשרת A' בבקשת הראשונה ולשרת B' בשנייה, המידע שלו לא ימצא בשרת השני.

פתרון: בידוד הנתונים. ניתן לשמור את נתונים ה-**Session** במסד נתונים מרכזי או בשרת ייעודי. זה מאפשר להוציא שרתי אינטרנט ללא חשש לאובדן מידע.

14.2.2 טיפ עיצוב #2: שימוש במטמון (!Cache)

שימוש במתמון הוא דרך מצינית לזכור את זמן התגובה. כדי לתקן את האפליקציה בשכבות כדי שנייתן יהיה להוסיף מטמון בקלות.

ניתן להוסיף מטמון בין כל שכבה ושבה:

1. **מטמון שאלות מסד נתונים**: שבירת תוכאות של שאלות יקרות.
2. **מטמון קריואט לפונקציות**: שבירת ערך החזירה של פונקציה עבור פרמטרים מסוימים.
3. **מבנה מקומפלות (Compiled Templates)**: רוב מערכות התבניות הופכות אותן לקוד PHP
שניתן לשומר במתמונו אופקודה (Opcode).
4. **מטמון פלט (Output Caching)**: שבירת הפלט המודפס של סקריפטשלם או חלקים ממנו.

14.2.3 טיפ עיצוב #3: אל תגדימו בעיצוב (Over Design)

עם תכונות ה-OO החדשנות של 5 PHP, קל להתפתות לעיצובים מורכבים מדי.

- **עטיפת פונקציות מובנות**: אל תעטפו פונקציות מובנות של PHP במחלקות אלא אם כן זה מוסיף ערך אמיתי (כמו יצירת API אחד למסדי נתונים שונים).
- **הכללה (Generalize) בזהירות**: אל תיצור שכבות הפשטה סתם כך. זה מוסיף מורכבות ומאט את המערכת.
- **PHP היא לא Java**: ב-Java יצרת אובייקטים מהירה מאוד. ב-PHP עדיף להשתמש בטיפוסים המובנים (כמו מערכיים) במקומן לנסوت לחוקות מחלקות מורכבות מ-Java. השתמשו בחזקות של PHP.

14.3 בדיקות ביצועים (Benchmarking)

מה שחשוב בסופו של דבר הוא איך האתר שלכם מתפרק באופן כללי. דרך ייילה לבדיקת תכוניות ואיתור צווארי בקבוק היא להריץ המדומות תעבורת אמת (Production).

14.3.1 שימוש ב-ApacheBench

כל ה-ab (קיצור של Apache Benchmark tool) מגיע בדרך כלל עם שרת ה-Apache. הוא פועל על ידי סימולציה של מספר לקוחות השולחים בקשות לשירות ב-URL ספציפי.

לדוגמה: \$ ab -c 10 -n 10000 -H "Host: http://localhost/test.php"

פקודה זו תירה 10,000 שאלות, 10 בכל פעם. בסיום, הכללי יציג נתונים כמו בקשות לשניה (Throughput) וזמן תגובה באחוזנים (למשל, כמה זמן לוקח ל-99% מהבקשות להסתיים).

14.3.2 שימוש ב-Siege

החולשה של ab היא שהוא לא מאפשר להגדיר התפלגות בקשות מציאותית (הוא "דופק" על אותו URL שוכ ושוב). כל ה-ab מאפשר להגדיר קובץ עם רשימת כתובות URL מלאות ולבחר מהן באקרים.

14.3.3 בדיקה מול תעבורת אמיתית

הסנה בבדיקות אלו היא שהן לא תמיד משקפות את העולם האמיתי (גולשים עם מודמים איטיים, בוטים של מנוני חיפוש וכדומה). מומלץ ליצור קובץ בקשות המבוסס על לוגים (Logs) אמיתיים של האתר.

14.4 ניתוח ביצועים עם ה-Profiler של Zend Studio

אופטימיזציה צריכה להתמקדך רק בחלקים שגוזלים את רוב המשאבים. השקעת זמן בשיפור קוד שמהווה רק שבריר מהעומס היא בזבוז זמן, ולעיתים אף פוגעת בקריאות הקוד.

ה-Profiler של Zend Studio מספק מידע קרייטי: אילו חלקים לוקחים היכי הרבה זמן, כמה פעמים כל פונקציה נקראת ואיך נראה ה-"Call Trace" (ע"ז הקראיות).

כפי שניתן לראות בתרשימים 14.6 (בsofar), אם קובץ מסוים אחראי ל-53% מהעומס, עדיף להתרכז בו מאשר בקבצים אחרים.

14.5 ניתוח ביצועים עם APD

הוא תוסף Zend (APD) (Advanced PHP Debugger) שאוסף סטטיסטיות בזמן ריצה ושומר אותן לקובץ. לאחר מכן ניתן לנתח את הקובץ עם הכלי `pprof`.

14.5.1 התקנה והרצה

מתקנים דרך `pear install apd` (PECL) ומגדירים `-ini.php`. כדי להתחיל איסוף נתונים בסקריפט, קוראים לפונקציה `apd_set_pprof_trace()`.

- **טייפ:** ניתן להפעיל זאת סלקטיבית רק כמשמעותם פרמטר ל-URL (כמו `?_profile=apd`), אך יש להיזהר מחשיפת אפרשות זו בשרת ייצור.

14.5.2 ניתוח נתוני המקבב (Trace)

בניתוח של אתרים כמו `pear.php.net`, התגלה למשל `require_once` צורך 50% מהזמן. זהו סימן מובהק לכך ששימוש ב-Opcode Cache יחתור את זמן הריצה בחצי.

ב-5 PHP, ניתן ליעיל קבאים על ידי הפיכתם ל-`const` בתוך מחלוקת במקומם להשתמש ב-`define()`, וכך שקבוע מחלוקת נשמרים במתมอง האופקוד ואינם מחושבים מחדש בכל בקשה.

14.6 ניתוח ביצועים עם Xdebug

הוא תוסף נוסף לאיסוף נתונים. בעוד APD מתמקד בעיקר ביצועים, Xdebug מתמקד גם בניתוח שגיאות (Debugging). הוא מאפשר ליצור נתונים בפורמט **Cachegrind**, שאוטם ניתן להציג בצורה גרפית מרשימה בעזרת תוכנות כמו **KCachegrind**.

14.6.1 התקנת Xdebug

בדומה ל-APD, ניתן להתקין את Xdebug מ-PECL על ידי הרצת הפקודה `pear install xdebug`. לאחר ההתקינה, עליהם לטעון את Xdebug לתוך Zend ולהגידר אותו בהתאם למשימה. להלן דוגמה להגדירה ב-`ini.php` לטיענות Xdebug:

```
; "zend_extension = "/usr/lib/php/extensions/20040412/xdebug.so
```

או עבור שרת אינטרנט מרובי-תהליכיונים (Apache על Windows או IIS):

```
; "zend_extension_ts = "c:/php5/extensions/xdebug.dll"
```

הגדרת Xdebug תלולה במטרה אותה תרצו להשיג.

14.6.2 מעקב אחר הרצת סקרייפט (Tracing)

מעקב אחר קרייאות לפונקציות במהלך הרצת סקרייפט מאפשר לכם לבחון אילו פונקציות נקראו לפי הסדר, כולל פרמטרים אופציונליים וערך חזרה. קובץ המעקב (Trace) כולל לא רק את הקריימות עצמן, אלא גם מידע על תזמון ושימוש בזכרון. הגדרות התצורה האופטימלית ליצירת קבצי מעקב מוצגות בטבלה 14.2.

טבלה 14.2: הגדרות תצורה אופטימלית למעקב הרצת

תיאור	הגדרה
כאשר מופעל, טביעת הרgel של הזכרון גדלה בכ-33% והביצועים מואטים.	<code>xdebug.extended_info = 0</code>
הפעלת מעקב אוטומטי לאחר סקרייפטים.	<code>xdebug.auto_trace = 1</code>
הגדרת ספריית הפלט לקבצי המעקב.	<code>xdebug.trace_output_dir = /tmp/xdebug</code>
הכללת שמות הקבצים בקריאות <code>.include/require</code> .	<code>xdebug.collect_includes = 1</code>
מצגת ההבדל בשימוש בזכרון בין כל קריאה לפונקציה.	<code>xdebug.show_mem_delta = 1</code>
איסוף ערכי חזרה של פונקציות.	<code>xdebug.collect_return = 1</code>
איסוף פרמטרים המועברים לפונקציות.	<code>xdebug.collect_params = 1</code>

הערה: קבצי מעקב עלולים להפוך לעצומים (מעל MB100) בסקרייפטים מורכבים. ודאו שיש לכם מספיק שטח דיסק פנוי.

כasher כל ההגדרות מוכנות, `ngrepX` מייצר קובץ ספרייה שהוגדרה בשם כגון `xt.480204079.trace`. איור 14.9 מציג קובץ מעקב זה.

כל שורה מתחילה במדד זמן, כמוות הזיכרון שבשימוש, והשנייה בזיכרון לעומת השורה הקודמת. ההזחה (Indentation) מראה את הקשר בין הקריאה, ואחריה מופיעים שם הפונקציה, הפרמטרים, שם הקובץ ומספר השורה.

KCachegrind 14.6.3

בעוד שמעקב (Trace) שימושי לניתוח פשוט, הוא נדרש יותר כדי לניפוי שגיאות (Debugging). עבור ניתוח ביצועים טהור (Profiling), Xdebug מציע פונקציונליות ייעודית שאת תוכואתיה ניתן לנתח בתוכנת .KCachegrind

לאחר טיענת הקובץ ל-KCachegrind, יוצג מסך הדומה לאיור 14.10:

החלונית השמאלית מציגה את כל הפונקציות ממוגנות לפי הזמן שהושקע בהן. בבחירה פונקציה, ניתן לראות בחולונית הימנית-עלונה מי קרא לה, ובחולונית הימנית-תחתונה אילו פונקציות היא קראה.

ऋטיסיות **Call Map** מציגה דיאגרמה ויזואלית של הזמן המושקע בפונקציות. ככל שהשתוח של פונקציה גדול יותר, כך הושקע בה זמן רב יותר.

(APC) Advanced PHP Cache 14.7

אחד מבעליות הביצועים הגדולות ב-PHP היא שהזמן הנדרש לעיבוד הקוד גדול ככל שהקוד מורכב יותר. הפתרון לכך הוא **מטמון אופקוד (Opcode Cache)**. הוא שומר את הפלט של מהדר ה-*Zend* בזיכרון משותף, וכך שבקשות עוקבות אין צורך לנתח את אותו הקוד שוב ושוב.

APC הוא מטמון פופולרי בקוד פתוח הזמן דרך PECL:

```
shell$ pear install apc
```

כדי להשתמש בו, הוסיפו ל-*ini.php*:

```
Ini, TOML
```

```
apc.enable = yes
```

```
apc.shm_size = 4
```

לאחר הפעלה מחדש של שרת האינטרנט, תבחןנו שקריאות ל-*require/include* כמעט נעלמות מרשימה צרכני המעבד בביוץ ניתוח ביצועים (Profiling).

(ZPS) Zend Performance Suite 14.8

ZPS הוא מוצר מסחרי מבית Zend.com המספק כלים:

- **אופטימיזציה אוטומטית:** שיפור של כ-20% בBITS (בBITSים ללא שינוי קוד).
- **מטמון קוד מקומפל (Compiled-code Caching):** שיפור של 50% עד 300% BITS (בBITSים).
- **מטמון תוכן (Content-Caching):** האצה של עד פי 100 (10,000%) על ידי ביטול תקורה של BITS (בBITSים).
- **דוחיסת תוכן:** מצoom נפח HTML בערך ב-90% לשיפור מהירות הטעינה אצל המשתמש.

14.8.1 אופטימיזציה אוטומטית

PHP פועלת בשני שלבים: הידור הקוד לקוד בינים (Intermediate code) וביצועו.

לדוגמה, הקוד `$i++`; מומר לפעולות פשוטות. לפעמים נוצרות פעולות מיותרות (כמו שמיירת ערך זמני שאינו בשימוש). Zend Optimizer ה- **Zend Optimizer** מנתה את קוד הבינים ומחליפ תבניות לא שימושיות יותר (למשל, הפיכת post-increment ל-pre-increment במידת האפשר). התהיליך קורה בזיכרון ואני משנה את קוד המקרה.

14.8.2 מטמון קוד מקומפל

במצב רגיל, בסיום הרצת סקריפט, קוד הבינים מושמד. בבדיקה הבא, PHP תהדר אותו מחדש. במקרים פופולריים זהו צוואר בקבוק.

מודול ה-**Acceleration** של ZPS שומר את קוד הבינים בזיכרון משותף (Shared Memory). בפעם הבאה שהקובץ ידרש, המנווע ידלג על שלב ההידור ויעבור ישירות לביצוע. זה יעל במיוחד באפליקציות OO (מנוחות עצמאיים) הכוללות קבצים רבים.

14.8.3 מטמון תוכן דינמי

זו השיטה העילית ביותר: שמיירת תוצאה ה-HTML הסופית ושליחתה כפי שהיא לבקשת הבאות.

- **TTL (זמן חיים):** הגדרת מועד הזמן שהעוקק במטמון תקין.
- **תלוויות (Dependencies):** המטמון חייב לדעת להבחין בין פרמטרים שונים (למשל `7=id` לעומת `5=id`).
- **מטמון חלק/בלודי:** ניתן להחריג אזורים אישרים בדף (כמו שם המשתמש) או להגיש מטמון רק למשתמשים שאינם מחוברים (Exclusive caching).

14.9 אופטימיזציה של הקוד

14.9.1 מיקרו-בנץ'マーク (Micro-Benchmarks)

לפעמים עולה השאלה מה מהיר יותר: למשל `preg_replace()` או `str_replace()`. ניתן לכתוב סקריפט קטן המודד מזמן מעבד בעזרת פונקציית `getrusage()`.

חשוב: יש להריץ בדיקות אלו פעמים רבות כדי לקבל ממוצע אמין, שכן רצולוציית המדידה תלולה במערכת ההפעלה.

14.9.2 פרזידורילי לעומת מונחה עצמים (OO)

בדיקות מראות שקיימות לפונקציות רגילים (Procedural) מהירות בכ-11-12% מקרים למתודות בתוך אובייקטים. זה הופך לגורם משמעותי רק אם הקוד כולל המון פונקציות קטנות מאוד, שבהן תקופת ה"קראה" (Call overhead) מהוות חלק ניכר מזמן הריצה הכלול.

14.10 סיכום

עיצוב אפליקציות לביצועים גבוהים הוא נושא מורכב. המפתח הוא לשאוף לעיצוב רזה, יעיל ואלגנטית, תוך ביצוע בדיקות ביצועים (Profiling & Benchmarking) ללא הרף כדי למצוא את צוואר הבקבוק האמיתיים.

פרק 15

15.1 מבוא

אחד הסיבות העיקריות להצלחתה של PHP היא היכולת העצומה של הרחבות (Extensions) הזמיןנות עבורה. לא משנה מה מפתח אינטרנט עשוי להזדקק לו, רוב הסיכויים שהוא ימצא זאת בהפצה של PHP: תמייה בסיסי נתונים שונים, פורמטים גרפיים, דחיסה, טכנולוגיות XML ועוד.

הפריצה הגדולה של PHP התרחשה בגרסה 3 עם הצגת ה-API להרחבות, שאפשר לכתילת המפתחים להרחיב את השפה בקלות. הרעיון היה להס提ר ככל האפשר את הקביבים של PHP ושל מגוון התסריטים (Scripting Engine) מכותב ההרחבה, ולדריש ממנה רק מיומנות ב-API עצמו.

ישנן שתי סיבות עיקריות לכתיבה הרחבה משלך:

1. **תמייה בטכנולוגיה חדשה:** למשל, אם יוצא לשוק מוצר חדש נתונים חדש בשם FooBase, יהיה עליך ליצור הרחבת PHP שעוטפת את ספריית ה-C של FooBase כדי לאפשר ממשק אליה מתוך PHP.
2. **ביצועים או פונקציונליות:** כתיבת לוגיקה עסקית ב-C כדי להשיג מהירות גבוהה יותר.

פרק זה ילמד אתכם כיצד ליצור הרחבות פשוטות באמצעות סקריפט UNIX בשם `ext_skel` (או `ext_skel_win32.php` לחילונות), שיוצר בלבד של הרחבה מתוך קובץ הגדרות.

15.2 מדריך מהיר (Quickstart)

במוקם להסביר לאט את כל אבני הבניין, נצלול ישיר לקוד.

תארו לעצמכם שאتمם זקנים לפונקציה שמשכפלת מחזורת `$n` פעמים. ב-PHP זה נראה כך:

```
PHP
} (function self_concat($string, $n
;"" = result$
} (++for ($i = 0; $i < $n; $i
;result .= $string$
{
;return $result
{
```

אם עלייכם לקרוא לפונקציה זו לעתים קרובות עם מחרוזות ארכוכת מאד וערך \$ גודלים, הדבר יגרום להרבה הקצות זיכרון ויאט את המערכת. ב-C נוכל להקצות מראש את כל הזיכרון הנדרש ולבצע זאת הרבה יותר מהר.

שלב 1: הגדרת הפונקציה

ניצור קובץ בשם myfunctions.def עם השורה הבאה:

```
h (string self_concat(string str, int
```

שלב 2: יצירת השלד

נՐץ את הסקרייפט:

```
ext_skel --exname=myfunctions --proto=myfunctions.def/.
```

זה ייצור ספרייה בשם myfunctions.so תחת ספריית ./ext.

שלב 3: קימפול

כדי ש-PHP יוכל בהרחבה, יש לעורק את הקובץ config.m4 ולהוסיף את סימני ההערה משורות buildconf, ./configure --enable-myfunctions/. PHP_ARG_ENABLE .make

שלב 4: כתיבת הלוגיקה ב-C

הסקרייפט יצר עבורנו קוד התחלתי ב-C. הפונקציה המרכזית להבנת הקולט היא zend_parse_parameters().

טבלה 15.1: מזהי טיפוסים (Type Specifiers)

מזהה	טיפוס C תואם	תיאור
I	long	מספר שלם (Integer).
d	double	מספר עשרוני (Floating-point).
s	char *, int	מחרוזת בינארית (כולל אורן).
b	zend_bool	ערך בוליани (1 או 0).

מערך אסוציאטיבי.	* zval	a
------------------	--------	---

הסביר על ה-zval:

ה-zval הוא כל קיבול הערכים של מנוע Zend. בין אם הערך הוא בוליאני, מחרוזת או אובייקט, המידע שלו נשמר בתוך מבנה ה-zval.

בדוגמה שלנו, כדי לקבל מחרוזת ומספר שלם, משתמש ב-"`as`:

```
C
(if (zend_parse_parameters(argc TSRMLS_CC, "sl", &str, &str_len, &n) == FAILURE
    ;return
```

שים לב ש-`s` דורש שני ארגומנטים ב-C: מצביע למחרוזת (`char*`) ומשתנה לאורך שלה (`int`). תמיד עדיף להשתמש באורך המחרוזת כדי שהקוד יהיה "Binary Safe" (בטוח לשימוש עם נתונים בינאריים שיכולים להכיל תוים ריקים).

15.2.1 ניהול זיכרון

ה-API של PHP להקצתה זיכרון מהזיכרון (Heap) כמעט זהה ל-API הסטנדרטי של שפת C. בעת כתיבת הרחבות, השתמשו בפונקציות ה-API הבאות התואמות למקבילותיהן ב-C:

<code>;malloc(size_t size</code>	•
<code>;(efree(void *ptr</code>	•
<code>;(calloc(size_t nmemb, size_t size</code>	•
<code>;(erealloc(void *ptr, size_t size</code>	•
<code>;(estrndup(const char *s</code>	•
<code>;(estrandup(const char *s, unsigned int length</code>	•

הפונקציה `estrandup()` היא היחידה המיועדת ל-PHP. היא מתנהגת כמו `strndup()`, אך מאפשרת לציין את אורך המחרוזת לשכפול (ללא ה-`\n` המסיים), וכן היא **Binary Safe** (בטוחה לשימוש ביןארי). מומלץ להשתמש בה על פני `estrandup()`.

יתרונות השימוש בפונקציות אלו:

1. **מניעת דיליפוט זיכרון:** כל זיכרון שהוקצה דרך ולא שוחרר בטעות, ישוחרר אוטומטית בסיום הבקשה (Request).
2. **יציבות:** PHP עלולה להתפרק אם תחזירו למנוע התסריט ערכים שלא הוקטו עם פונקציות אלו.
3. **ביצועים:** שיפור ביצועים בסביבות מרובות-תהליכיונים (Multi-threaded) ויזיהו השחתת זיכרון במצב Debug.
4. **בטיחות:** אין צורך לבדוק אם חזר `NULL`; במקרה של כשל בהקצאה, הפונקציות יפסיקו את הריצה עם שגיאת `E_ERROR`.

15.2.2 החזרת ערכים מפונקציות PHP

- ה-API כולל אוסף עשיר של מacroאים (Macros) להחזרת ערכיהם, המגיעים בשני סגנונות:
1. **retval_type()**: קובע את ערך החזרה אך ממשיר בביצוע קוד ה-C (שימושי לניקוי משאבים לפני סיום). לסיום יש להשתמש ב-`return;`.
 2. **return_type()**: קובע את הערך וגם מוחזר מיד את השליטה ל-PHP.

טבלה 15.2: מacroאים לערכי חזרה

הערות	macro (קביעה וסיום)	סוג ערך
מספרשלם.	(RETURN_LONG(Long
ערךאמת/שקר.	(RETURN_BOOL(b	Boolean
מחרוזת. אם <code>dup</code> הוא 1, המנווע ישכפל את המחרוזת.	(RETURN_STRING(s, dup	String
מחרוזת באורך <code>l</code> . מהיר יותר אם האורך ידוע מראש.	(RETURN_STRINGL(s, l, dup	String (אורך)
לא סוגרים.	RETURN_TRUE / RETURN_FALSE	True / False
מזהה משאב (כמו חיבור למוסד נתונים).	(RETURN_RESOURCE(r	Resource

15.2.3 השלמת הפונקציה `()self_concat`

להלן הקוד המלא של הפונקציה המממשת שכפול מחרוזת ב-C:

```
C
(PHP_FUNCTION(self_concat
)
;char *str = NULL
;()int argc = ZEND_NUM_ARGS
;int str_len
```

```

;long n
/* מצביע למחוזת התוצאה */ ;char *result
/* מצביע למיקום הבא להעתקה */ ;char *ptr
/* אורך התוצאה */ ;int result_length

(if (zend_parse_parameters(argc TSRMLS_CC, "sl", &str, &str_len, &n) == FAILURE
;return

;(result_length = (str_len * n
(result = (char *) emalloc(result_length + 1
; /* הקצתה זיכרון */ ;ptr = result
;ptr += str_len

} )--while (n
/* העתקת המחרוזת */ ;(memcpy(ptr, str, str_len
;ptr += str_len
{

/* תמיד סיום מחרוזת ב-null */ ;'ptr = '\0'

/* החזרה ל-PHP ללא שכפול נוסף */ ;(RETURN_STRINGL(result, result_length, 0
{

```

15.2.5 עטיפת ספריות צד-שלישי

המוטיבציה הנפוצה ביותר לכתיבת הרחבות היא עטיפת ספריות C קיימות כמו MySQL, ImageMagick ו-GD. בהמשך נדגים זאת על ידי עטיפת פונקציות הקבצים הסטנדרטיות של C (כמו fopen).

הרחבה תשתמש בטיפוס נתונים מופשט שנקרא **Resource** כדי לייצג את הקובץ הפתוח (*.FILE). מנוע ה-PHP אינו מבין מצביעי C ישירות, ולכן "משאבים" הם הדרך לקשר ביניהם.

המנשך שתכננו להרחבה החדש (myfile.def)

<pre>(resource file_open(string filename, string mode bool file_close(resource filehandle) string file_read(resource filehandle, int size) bool file_write(resource filehandle, string buffer) bool file_eof(resource filehandle)</pre>	<ul style="list-style-type: none"> • • • • •
---	---

לאחר הריצת myfile.ext_skel, קיבלו קוד ראשון. יתכן שתראו ש⁵ את קומpileציה בשורות הכלולות את המאקרו FETCH_RESOURCE(), שכן הסקריפט אינו יודע להשלים אותו לבדו—עלינו להגדיר לו כיצד לטפל במשאב החדש.

15.2.5.2 משאבי (Resources)

משאב הוא ערך פשוט שניתן להכיל כל סוג של מידע. כפי שצוין בעבר, מידע זה מורכב לעתים קרובות מנתונים כגון מצביעי קבצים (file handles), מבני התקשרות למסדי נתונים וטיפוסים מורכבים אחרים.

הסיבה העיקרית לשימוש במשאים היא שהם מנוהלים באמצעות רשיימה מרכזית המשמידה אוטומטית את המשאב במקרה שפתח ה-PHP לא עשה זאת במפורש בסקריפט שלו.

לדוגמה, חשבו על כתיבת סקריפט הפותח חיבור ל-MySQL באמצעות הקריאה `mysql_connect()`, אך אין קורא ל-`mysql_close()` כדי לסגור אותו ברגע שימוש החיבור למסד הנתונים אין בשימוש יותר. ב-PHP, מנגנון המשאים מזהה מתי יש להשמיד משאב זה, וישmid אותו (לכל המאוחר) בסוף הבקשת הנוכחות, ולעתים קרובות הרבה לפני כן. הדבר מספק מנגנון חסן לביטול האפשרות לדיליפות משאים. ללא מנגנון זה, לאחר מספר בקשות אינטרנט, שרת האינטרנט עשוי לשוב מדיליפה פוטנציאלית של משאים רבים, מה שעלול להוביל לקריסת שרת או לתפקיד לוקוי.

15.2.5.3 רישום טיפוסי משאים

כיצד משתמשים במשאים? מנווע Zend הפרק את העבודה עם משאים לפחות יחסית. הדבר הראשון שעליכם לעשות הוא לרשום את טיפוס המשאב שלום בmenoוע. פונקציית ה-API לשימוש היא:

```
int zend_register_list_destructors_ex(zend_dtor_func_t Id, rsrc_dtor_func_t pld, char (*type_name, int module_number
```

הפונקציה מחזירה מזהה (ID) של טיפוס המשאב, אותו הרחבה צריכה לשמר במשתנה גלובלי והוא יועבר לקריאות API אחרות של משאים בעת הצורך. `Id` היא פונקציית ה-`destructor` (המשמיד) ש צריכה להיקרא עבור המשאב זהה. `pld` משמש עבור משאים קבועים (persistent) שיכולים לשזר בין בקשות ולא יוכסו בפרק זה. `type_name` הוא מחרוזת עם שם תיאורי לטיפוס. `module_number` משמש פנימית את המנווע, וכאשר נקרא לפונקציה זו, פשוט נעביר את המשתנה `module_number` שכבר הוגדר.

בחזרה לדוגמה שלנו: נוסיף את הקוד הבא לקובץ המקור `myfile.c`. הוא כולל את ההגדלה לפונקציית `zend_dtor` שmovedה לפונקציית הרישום `zend_register_list_destructors_ex()` (יש להוסיף אותה בשלב מוקדם בקובץ כדי שתהייה מוגדרת בזמן הקריאה לרישום):

```
C
(static void myfile_dtor(zend_rsrc_list_entry *rsrc TSRMLS_DC
{
    FILE *fp = (FILE *) rsrc->ptr
    ;(fclose(fp
{
```

לאחר הוספה שורת הקריאה לפונקציית ה-`PHP_MINIT_FUNCTION()` שנוצרה אוטומטית, היא אמורה להיראות דומה לזה:

```
C
(PHP_MINIT_FUNCTION(myfile
```

```

        }

/* אם יש לכם רשומותINI, הסירו את העורות מהשורות הללו

;(ZEND_INIT_MODULE_GLOBALS myfile, php_myfile_init_globals, NULL
;()REGISTER_INI_ENTRIES

/*
le_myfile = zend_register_list_destructors_ex(myfile_dtor, NULL, "standard-c-file",
;(module_number

;return SUCCESS

{

```

* שימושו לב ש-myfile_le הוא משתנה גלובלי שכבר הוגדר על ידי הסקריפט .ext_skel.

הח疏ף להרחבתיהם. טבלה 15.3 מוצגת קצירה של הפונקציות הזמינות ויכן ניתן להשתמש בהן.

טבלה 15.3: מאקרוים להצחת פונקציות

סמנטיקה	מאקרו ההצחת פונקציה
פונקציית הסטארט-אפ של המודול נקראת על ידי המנווע כאשר PHP נתען ומאפשרת לבצע אתחולמים חד-פעמיים נחוצים, כגון רישום טיפוסי משאים, רישום ערכיINI ועוד.	()PHP_MINIT_FUNCTION
פונקציית הכיבוי של המודול נקראת על ידי המנווע כאשר PHP נסגר לשלוטין ומשמשת בדרך כלל לביטול רישום של רשומותINI.	()PHP_MSHUTDOWN_FUNCTION
פונקציית הסטארט-אפ לכל בקשה נקראת בתחילת כל בקשה המתופלת על ידי PHP, ומשמשת לניהול לוגיקה לכל בקשה.	()PHP_RINIT_FUNCTION

פונקציית ה- <code>cgi</code> לכל בקשה נקראת בסוף כל בקשה המטופלת על ידי PHP, ומשמשת לרוב לניקוי הלוגיקה של פונקציית הסטארט-אפ לכל בקשה.	<code>()PHP_RSHUTDOWN_FUNCTION</code>
פונקציית המידע של המודול נקראת במהלך פונקציית <code>phpinfo()</code> ומדפיסה את המידע של המודולים הללו.	<code>()PHP_MINFO_FUNCTION</code>

15.2.5.4 יצירה ורישום של משאבי חדשים

אנו עומדים למשם את הפונקציה `open_file()`. לאחר שנפתחה את הקובץ ונקבל `FILE *`, علينا לרשום אותו במנגנון המשאבי. המאקרו העיקרי להשגת מטרה זו הוא:

`:(ZEND_REGISTER_RESOURCE(rsrc_result, rsrc_pointer, rsrc_type`

ראו טבלה 15.4 להסביר על הארגומנטים של המאקרו.

טבלה 15.4: ארגומנטים של המאקרו Zend_Register_Resource

טיפוס פרמטר	ארגון המאקרו
<code>* zval</code> , אשר אמור להתעדכן עם מידע המשאב הרשום.	<code>rsrc_result</code>
מצבייע לנתחי המשאב שלנו.	<code>rsrc_pointer</code>
מזהה המשאב שהתקבל בעת רישום טיפול המשאב.	<code>rsrc_type</code>

15.2.5.5 פונקציות קבצים

cut כשותם יודעים כיצד להשתמש במאקרו `ZEND_REGISTER_RESOURCE()`, אתם כמעט מוכנים לכתוב את `open_file()`. נותר רק נושא אחד נוסף שעליינו לכסות.

מכיון ש-PHP רץ גם תחת שרתים מרובי-תהליכיים (multi-threaded), לא ניתן להשתמש בפונקציות הגישה לקבצים הסטנדרטיות של C. זאת מושם שסקרייפט PHP הרץ בתהילICON אחד עלול לשנות את תיקיית העבודה הנוכחיית (CWD), מה שוביל לכך שקריאה `fopen()` משתמשת בנטייב ייחודי בתהילICON אחר תיכשל בפתיחת הקובץ המזעדי. כדי למנוע בעיות כאלה, תשתיית PHP מספקת מאקרוואים של `VCWD` (Virtual Current Working Directory) שיש להשתמש בהם במקום כל פונקציית גישה לקבצים המסתמכת על

תיקיות העבודה הנוכחיית. (טבלה 15.5 מפרטת את המאקרוואים הזמינים). המאקרוואים מתנהגים אותו הדבר כמו הפונקציות שהם מחליפים, והכל מטופל עבורכם באופן שקו. פונקציות ספריית C סטנדרטיות שאינן זמינים בפלטפורמות מסוימות אין נתמכות על ידי תשתיית-h-Win32. לדוגמה, `chown()`, שאינה קיימת ב-Win32, לא תכלול מאקרו `VCWD_CHOWN()` תואם.

טבלה 15.5: רשימת מאקרואי VCWD

הערה	מאקרו VCWD	C ספריית סטנדרטית
	<code>()VCWD_GETCWD</code>	<code>()getcwd</code>
	<code>()VCWD_FOPEN</code>	<code>()fopen</code>
משמש Lagerst שני הפרמטרים.	<code>()VCWD_OPEN</code>	<code>()open</code>
משמש Lagerst שלושת הפרמטרים של <code>open()</code> .	<code>()VCWD_OPEN_MODE</code>	<code>()open</code>
	<code>()VCWD_CREAT</code>	<code>()creat</code>
	<code>()VCWD_CHDIR</code>	<code>()chdir</code>
	<code>()VCWD_GETWD</code>	<code>()getwd</code>
	<code>VCWD_REALPATH()</code> ²	<code>realpath()</code> ¹
	<code>VCWD_RENAME()</code> ⁴	<code>rename()</code> ³

	VCWD_STAT() ⁶	stat() ⁵
	VCWD_LSTAT() ⁸	lstat() ⁷
	VCWD_UNLINK() ¹⁰	unlink() ⁹
	VCWD_MKDIR() ¹²	mkdir() ¹¹
	VCWD_RMDIR() ¹⁴	rmdir() ¹³
	VCWD_OPENDIR() ¹⁶	opendir() ¹⁵
	VCWD_POPEN() ¹⁸	popen() ¹⁷

טבלה 15.5 רשימה פקודות מאקרו של VCWD

טבלה 15.5: רשימה מאקרו VCWD

ספריית C סטנדרטית	מאקרו VCWD
()access	()VCWD_ACCESS

()VCWD_UTIME	()utime
()VCWD_CHMOD	()chmod
()VCWD_CHOWN	()chown

15.2.5.6 כתיבת פונקציית ה-PHP הראשונה שלך מבוססת משאבים

מימוש `file_open()` אמור להיות קלCut, והוא אמור להירות ככך:

```
C
(PHP_FUNCTION(file_open
}

;char *filename = NULL
;char *mode = NULL
;()int argc = ZEND_NUM_ARGS
;int filename_len
;int mode_len
;FILE *fp

;if (zend_parse_parameters(argc TSRMLS_CC, "ss", &filename
} (filename_len, &mode, &mode_len) == FAILURE&
;return
{

;(fp = VCWD_FOPEN(filename, mode
} (if (fp == NULL
;RETURN_FALSE
```

{

;(ZEND_REGISTER_RESOURCE(return_value, fp, le_myfile

{

יתכן שתבחינו שהารוגומנט הראשון למקרו רישום המשאב הוא משתנה בשם `return_value`, שהופיע משם מקום. משתנה זה מוגדר אוטומטית על ידי תשתית החרחבות והוא מסוג `*zval` * המצביע לערך הזרה של הפונקציה. המאקרים שדנו בהם קודם המשפיעים על ערך הזרה, כגון `RETURN_LONG()` ו-`RETURN_BOOL()`, למעשה משנים את הערך של `return_value`. לכן, קל לנחש שהקוד רושם את מצביע הקובץ `fp` שקיבלנו ומגדיר את `return_value` למשאב הרשום.

15.2.5.7 גישה למשאב

כדי לגשת למשאב, علينا להשתמש במקרו הבא (ראו טבלה 15.6 להסביר על הארגומנטים שלו):

```
ZEND_FETCH_RESOURCE(rsrc, rsrc_type, passed_id, default_id, resource_type_name,
;(resource_type
```

טבלה 15.6: ארגומנטים של המקרו Zend_FETCH_RESOURCE

פרמטר	משמעות
rsrc	המשתנה אליו יוקצה ערך המשאב. עליו להיות מאותו טיפול של המשאב.
rsrc_type	הטיפוס של <code>rsrc</code> שישמש להמרת המשאב פנימית לטיפוס הנקון.
passed_id	ערך המשאב לחיפוש (בתור <code>zval</code> **).
default_id	אם ערך זה אינו -1, מזהה זה נלקח. משמש למימוש בירית מחדש למשאב.
resource_type_name	שם קצר לטיפוס המשאב המשמש בהודעות שגיאיה.

מזהה טיפוס המשאב של המשאב הרשום.	resource_type
----------------------------------	----------------------

בעזרת מקרו זה, נוכל כתוב ממש את ¹ `file_eof():`

```
C
(PHP_FUNCTION(file_eof
{
    int argc = ZEND_NUM_ARGS
    zval *filehandle = NULL
    FILE *fp

} if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE
    ;return
    {

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile
    ;RETURN_FALSE
    {

} if (feof(fp) <= 0
    /* החזר eof גם אם הייתה שגיאה */
    ;RETURN_TRUE
    {

;RETURN_FALSE
    {
}
```

15.2.5.8 הסרת משאב

כדי להסיר משאב, בדרך כלל תרצו להשתמש במאקרו הבא:

```
(int zend_list_delete(int id
```

טבלה 15.5: רשימת מאקרים VCWD

ספריית C סטנדרטית	מאקרו VCWD
()access	()VCWD_ACCESS
()utime	()VCWD_UTIME
()chmod	()VCWD_CHMOD
()chown	()VCWD_CHOWN

15.2.5.6 כתיבת פונקציית ה-PHP הראשונה שלך מבוססת משאים

מימוש `file_open()` אמור להיות פשוטCut, והוא אמור להירות כה:

```
C
(PHP_FUNCTION(file_open
{
    ;char *filename = NULL
    ;char *mode = NULL
    ;()int argc = ZEND_NUM_ARGS
    ;int filename_len
    ;int mode_len
```

```

;FILE *fp

,if (zend_parse_parameters(argc TSRMLS_CC, "ss", &filename
} (filename_len, &mode, &mode_len) == FAILURE&
;return

{

;(fp = VCWD_FOPEN(filename, mode
} (if (fp == NULL
;RETURN_FALSE

{

;(ZEND_REGISTER_RESOURCE(return_value, fp, le_myfile
{

```

יתכן שתבחינו שהארגומנט הראשון למקrho רישום המשאב הוא משתנה בשם `return_value`, שהופיע "משום מקום". משתנה זה מוגדר אוטומטית על ידי תשתיות ההרחבות והוא מסוג `* zval` * המצביע לערך החזרה של הפונקציה. המאקרואים שדנו בהם קודם המשפיעים על ערך החזרה, כגון `RETURN_LONG()` ו-`RETURN_BOOL()`, למעשה משנים את הערך של `return_value`. לכן, קל לנחש שהקוד רושם את מצביע הקובץ `fp` שקיבלו ומגדיר את `return_value` למשאב הרשם.

15.2.5.7 גישה למשaab

כדי לגשת למשaab, עליום להשתמש במקrho הבא (ראו טבלה 15.6 להסביר על הארגומנטים שלו):

```
ZEND_FETCH_RESOURCE(rsrc, rsrc_type, passed_id, default_id, resource_type_name,
;(resource_type
```

טבלה 15.6: ארגומנטים של המקrho Zend_FETCH_RESOURCE

פרמטר	משמעות

המשתנה שאליו יוקצה ערך המשאב. עליו להיות מאוות טיפוס של המשאב.	rsrc
הטיפוס של <code>rsr</code> ישמש להמרת המשאב פנימית לטיפוס הנכון.	rsrc_type
ערך המשאב לחיפוש (בטור <code>zval **</code>).	passed_id
אם ערך זה אינו -1, מזאה זה נלקח. משמש למימוש בירית חדש למשאב.	default_id
שם קצר לטיפוס המשאב המשמש בהודעות שגיאה.	resource_type_name
מזאה טיפוס המשאב של המשאב הרשום.	resource_type

בעזרת מקרו זה, נוכל כתוב ממש את ¹`file_eof():`

```
C
(PHP_FUNCTION(file_eof
}

;()int argc = ZEND_NUM_ARGS
;zval *filehandle = NULL
;FILE *fp

} (if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE
;return
{

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile
```

} (if (fp == NULL

;RETURN_FALSE

{

```
} if (feof(fp) <= 0
```

/* החזר true גם אם הייתה שגיאה */

;RETURN_TRUE

{

;RETURN_FALSE

{

הסרת משאב 15.2.5.8

כדי להסיר משאב, בדרך כלל תרצו להשתמש במקורות הבאים:

```
(int zend_list_delete(int id
```

למקרה מועבר המזהה (ID) של המשאב, והוא מחזיר SUCCESS או FAILURE. אם המשאב קיימ, לפני הסרתתו מרשימת המשאים של Zend, המנווע יקראה ל-**destructor** הרשות עבור טיפוס משאב זה. לכן, בדוגמה שלנו, אין צורך להשיג את מצביע הקובץ ולקראא ל-**(fclose()**) לפני הסרת המשאב; ניתן פשוט למחוק אותו.

שימוש במקאו זה מאפשר לנו למשם את `close(file)`

C

(PHP FUNCTION(file close)

}

;()int argc = ZEND_NUM_ARGS

:zval *filehandle = **NULL**

```
{if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE
```

:return

```

    }

}

} (if (zend_list_delete(Z_RESVAL_P(filehandle)) == FAILURE
        ;RETURN_FALSE

        {

;RETURN_TRUE

}

```

בזודאי שאלתם את עצמכם מה עושה `Z_RESVAL_Z()`. כאשר אנו שולפים את המשאב מרשימת הארגומנטים באמצעות `zend_parse_parameters()`, אנו מקבלים אותו בצורה `zval`. כדי לגשת למשאב `resource id`, אנו משתמשים במקירו `Z_RESVAL_P()`, ואז מעבירים אותו `zend_list_delete()`.

משפחה שלמה של מקראים מס'יעת בגישה לערכים המאוחסנים בתוך `zval` (ראו טבלה 15.7).

טבלה 15.7: מקראים לגישה ל-`zval`

טיפוס C	ערך הגישה	מקIRO
<code>Long</code>	ערך שלם (<code>Integer</code>)	<code>Z_LVAL, Z_LVAL_P, Z_LVAL_PP</code>
<code>zend_bool</code>	ערך בוליани	<code>Z_BVAL, Z_BVAL_P, Z_BVAL_PP</code>
<code>double</code>	ערך עשרוני	<code>Z_DVAL, Z_DVAL_P, Z_DVAL_PP</code>
<code>* char</code>	ערך מחוوظ	<code>Z_STRVAL, Z_STRVAL_P, Z_STRVAL_PP</code>
<code>int</code>	אוריך מחוوظ	<code>Z_STRLEN, Z_STRLEN_P, Z_STRLEN_PP</code>
<code>Long</code>	ערך משאב	<code>Z_RESVAL, Z_RESVAL_P, Z_RESVAL_PP</code>

* HashTable	מערך אסוציאטיבי	Z_ARRVAL, Z_ARRVAL_P, Z_ARRVAL_PP
Enumeration	טיפוס ה-zval	Z_TYPE, Z_TYPE_P, Z_TYPE_PP

15.2.5.9 מאקרים לגישה לערכי zval

לכל המאקרים יש שלוש צורות: אחת המקבלת `zval`, אחת עבור `* zval`, ואחת עבור `** zval`. ההבדל בשמות הוא שלראשונה אין סימנת `_P` (מצין pointer אחד), ול-`zval **` יש סימנת `_PP` (שני פינטרים).

כעת יש לכם מספיק מידע כדי להשלים את הפקציות `()file_write()` ו-`()file_read()` בעצמכם. להלן מימוש 2 אפשרי:

```
C
(PHP_FUNCTION(file_read
}

;()int argc = ZEND_NUM_ARGS
;long size
;zval *filehandle = NULL
;FILE *fp
;char *result
;size_t bytes_read

} (if (zend_parse_parameters(argc TSRMLS_CC, "rl", &filehandle, &size) == FAILURE
;return
{

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile
;(result = (char *) emalloc(size + 1
```

```
; (bytes_read = fread(result, 1, size, fp
; 'result[bytes_read] = '\0

;(RETURN_STRING(result, 0
{

(PHP_FUNCTION(file_write
}

;char *buffer = NULL
;()int argc = ZEND_NUM_ARGS
;int buffer_len
;zval *filehandle = NULL
;FILE *fp

if (zend_parse_parameters(argc TSRMLS_CC, "rs", &filehandle, &buffer, &buffer_len) ==
} (FAILURE
;return

;

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile

} (if (fwrite(buffer, 1, buffer_len, fp) != buffer_len
;RETURN_FALSE
{
;RETURN_TRUE
{
```

15.2.6 משתנים גלובליים

יתכן שתרצה להשתמש במשתני C גלובליים בהרחבה שלכם. מכיוון ש-PHP תוכננה לוחץ בסביבות מרובות-תהליכיים (multi-threaded), אין להגדיר משתנים גלובליים באופן רגיל. PHP מספקת מנגנון היוצר עותק של המשתנים לכל תהליכון בנפרד.

הסקריפט `ext_skel` יצר את הקוד הנחוצץ לכך בתוך `myfile.h`:

```
C
(ZEND_BEGIN_MODULE_GLOBALS(myfile
;int global_value
;char *global_string
(ZEND_END_MODULE_GLOBALS(myfile
```

כדי לגשת למשתנים אלו בקוד ה-C שלכם, עליהם להשתמש בmacro'ו `MYFILE_G(MYFILE)`. המנגנון מודא גישה בטוחה לכל תהליכון ללא צורך בניהול נעלילות (mutual exclusion) מצדכם.

בנוסף, עליהם להסיר את סימן ההערה מהשורה הבאה ב-`c-file`:

```
(ZEND_DECLARE_MODULE_GLOBALS(myfile
```

ניתן לאתחל את המשתנים הגלובליים בתחילת כל בקשה באמצעות המacro':

```
ZEND_INIT_MODULE_GLOBALS(module_name, globals_ctor, globals_dtor
```

15.2.7 הוספת הוראותINI מותאמות אישית

ミימוש קובץ ה-INI (`php.ini`) מאפשר להרחבות PHP לרשום ולהזין לרשומותINI משלמן. אם לרשומות אלו מוקצתה ערך דרך `php.ini`, קובץ `htaccess` של Apache או שיטות הגדרה אחרות, המשטנה הרשום תמיד יעדכן בערך הנכון.

הוראותINI נרשומות באמצעות המacro' `STD_PHP_INI_ENTRY` בין המacro'אים `STD_PHP_INI_BEGIN` ו-`STD_PHP_INI_END`. לדוגמה, ב-`c-file`, ב-`myfile.c` תראו משהו כזה:

```
C
()PHP_INI_BEGIN
STD_PHP_INI_ENTRY("myfile.global_value", "42", PHP_INI_ALL, OnUpdateInt, global_value,
(zend_myfile_globals, myfile_globals
STD_PHP_INI_ENTRY("myfile.global_string", "foobar", PHP_INI_ALL, OnUpdateString,
(global_string, zend_myfile_globals, myfile_globals
()PHP_INI_END
```

קיימים מאקרים נוספים, אך זהו הנפוץ ביותר. להלן הסבר על הפרמטרים שלו:

טבלה 15.9: פרמטרים של המאקרו STD_PHP_INI_ENTRY

פרמטר	משמעות
name	שם רשוםת ה-INI.
default_value	ערך בירית המחדל (תמיד כמחרוזת).
modifiable	היכן ניתן לשנות את הערך: PHP_INI_SYSTEM (ב-), PHP_INI_USER (ב-.htaccess), PHP_INI_PERDIR (בסקריפטים) או PHP_INI_ALL (בכל מקום).
on_modify	fonkציית Callback המתפלת בשינוי (למשל: OnUpdateString, OnUpdateBool).
property_name	שם המשתנה שיש לעדכן.
struct_type	סוג המבנה (Struct) בו נמצא המשתנה (למשל zend_myfile_globals).
struct_ptr	שם מבנה הגלובלים (למשל myfile_globals).

כדי שהמנגנון יעבד, עליו לנוסיר את סימני ההערכה מהקריאה ל-REGISTER_INI_ENTRIES() בתור ()UNREGISTER_INI_ENTRIES(). מಹקראייה PHP_MINIT_FUNCTION בתור PHP_MSHUTDOWN_FUNCTION.

הגישה למשתנים מתבצעת פשוט על ידי כתיבת `value_global_value` מכל מקום בהרחבה.

15.2.8 מאקרים של ניהול משאבי בטוח לתהיליכונים (TSRM)

בודאי שמתם לב לשימוש במאקרים המתחלים ב-**TSRM** (ראשי תיבות של Thread-Safe Resource Manager). אלו מאפשרים להרחבה להחזיק משתנים גלובליים משלה בסביבה מרובת-טהיליכונים.

כדי להשתמש במאקרו גישה (כמו `MYFILE_G()`), המידע על הקשר (`context`) ה-`TSRM` חייב להיות נכון בפונקציה. מנוע Zend מעביר את ההקשר זהה כפרמטר לכל האפשר מטעמי ביצועים. בתוך `PHP_FUNCTION()` הוא זמין אוטומטית. עם זאת, אם אתם קוראים לפונקציות C אחרות הזקוקות לגלובלים, עליו להעביר להן את ההקשר או לשולוף אותו (מה שאיתך יותר).

כדי לשולוף את ההקשר ידנית:

```
C
} ()void myfunc
;()TSRMLS_FETCH
;MYFILE_G(myglobal) = 2
{
```

עדיף להעביר את ההקשר כפרמטר באמצעות `TSRMLS_C` (בקריאה) ו-`D` (**בהצחה**):

```
C
} (void myfunc(TSRMLS_D
;MYFILE_G(myglobal) = 2
{
}
} (PHP_FUNCTION(my_php_function
;(myfunc(TSRMLS_C
{
```

(הערה: המאקרים עם סימנת `CC` או `DC` כוללים פסיק לפני ההקשר, לשימוש שיש ארגומנטים נוספים).

15.3 סיכום

פרק זה למדתם את היסודות החשובים לנכיבת והבנת הרחבות PHP. ה-API שמספק מנוע Zend עשיר מאוד ומאפשר גם כתיבת הרחבות מונחות עצמאיים. אין תחליף לבחינת קוד המקור של הרחבות הליבה המגיעות עם PHP כדי ללמידה טכנית מתקדמות.

מידע נוסף ניתן למצוא במדריך PHP תחת הפרק "Extending PHP", וכן מומלץ לעקוב אחרי כל חדש בשם `.ext_skel`, המציע תוכנות מתקדמות יותר מהסקריפט `PECL_Gen`.

פרק 16

16.1 מבוא

באופן מסורתי, PHP מושמת בסביבות אינטרנט להפקת קוד HTML שבו המשתמש צופה בדף. האינטראקציה בין PHP לשרת האינטרנט (Apache, AOLserver, IIS ועודמה) מתבצעת דרך שכבה הנקראת **SAPI** (קיצור של API Web Server). נדרשת גרסה בנייה נפרדת של PHP כדי להתmesh עם כל סוג של שרת אינטרנט דרך ה-SAPI.

בפרק זה תחקרו את ה-**CLI** (קיצור של Command Line Interface), ממשק שורת פקודה שהופך את PHP לשפת תסריט (Scripting) מסורתי. פרק זה מדגים שימוש ב-CLI לכתיבת כל שורת פקודה וכיישומיו שרת עצמאיים.

16.2 CLI של PHP (Shell)

גרסת ה-CLI של PHP نوعה לכנתיבת תסריטי של עצמאיים הרצים ללא תלות בשרת אינטרנט. החל מגרסתה 4.3.0, גרסת ה-CLI מותקנת כברירת מחדל לצד ממשק שרת האינטרנט שבחרתם.

16.2.1 CGI במה שונה CLI מ-

גרסת ה-CLI דומה לגרסה ה-CGI שעלייה התבססה בעבר, אך ההבדל העיקרי טמון באינטראקציה עם שרת האינטרנט. ב-CLI, PHP מקוצצת ליסודות בלבד: היא אינה מייבאת משתני טפסים (GET או POST), אינה מפיקת כותרות MIME בפלט, ובאופן כללי אינה מבצעת את הפעולות "מאחורி הקלעים" שביצועי אחרים עושים.

16.2.1.1 פרמטרים כבירית מחדל

ל-CLI יש ערכי בירית מחדל שונים עבור מספר אפשרויות שורת פקודה והגדירות .ini.php.

טבלה 16.1: אפשרויות בירית מחדל ב-CLI

תיאור	ברירת מחדל ב-CLI	הגדרה/אפשרייה
ביטול כותרות HTTP בפלט.	מאופסרת	אפשריה ♦-
PHP לא משנה את תקיות העבודה לו של התסריט.	מאופסרת	אפשריה ♦-

הודעות שגיאה יוצגו בטקסט פשוט ולא ב-HTML.	مبוטלת	html_errors
הפלט נשלח מיד ללא אגירה (Buffer).	מאופשרת	implicit_flush
המשתנים \$argc ו-\$argv נרשומים תמיד.	מאופשרת	register_argc_argv
זמן ריצה מקסימלי: 0 פירוחו ללא הגבלה.	0	max_execution_time

16.2.1.2 אפשרויות נוספת

ישנן אפשרות נוספת פקודה הקיימת ב-CLI אך לא ב-CGI.

טבלה 16.2: אפשרויות CLI נוספת

אפשריה	תיאור
-r	הרצה קוד ישירות מהטרמינל (לא Tagi <?>).

אם אתם כתובים תסריט שצורך לקרוא מקלט סטנדרטי (STDIN) ובו-贊נית לקבל קלט משתמש מהטרמינל, עליכם להשתמש ב-`/dev/tty` לInteraction עם המשתמש.

16.2.2 אורך חי הביצוע

בשרות אינטרנט, תסריטי PHP מס'ימים את העבודה מהר. ב-CLI, התסריט שלכם עשוי לרוץ לנצח (למשל Daemon).

השלכה מעשית אחת היא שוגנון קידוד מרושל, שהוא יחסית חסר נזק בבקשת אינטרנט קצרה, הופך לקרים בתסריט שרחץ זמן רב. אם פתחתם קובץ או חיבור למסד נתונים ולא סגרתם אותו במפורש, המשאב יישאר תפוס עד שהתסריט יסתום. לכן: סגורו קבצים, התנתנו ממסדי נתונים וורקנו מערכיים גדולים כשמיינטם איתם.

16.2.3(#!): הפיכת התסריט לבר-הרצה

במערכות דמויות UNIX, אם שני התווים הראשונים בקובץ הם `#!`, שאר השורה נחשבת לשם התוכנית שמריצה את הקובץ.

לדוגמה, תסריט שמתחליל כר:

usr/bin/php -Cq/!#

אפשר לך להריץ את הקובץ 'שירות מהלך' בקיליד `php` לפני.

16.2.5 ניתוח אפשרויות שורת פקודה (getopt)

כדי להפוך את כל שורת הפקודה שלכם למקצועיים, עליהם לתמוך בפרמטרים (כמו -v לגרסה או -h לעזרה).

הספרייה `Console_Getopt` (ambil PEAR) מאפשרת לנו זאת בקלות.

דוגמה לשימוש באופציות קצרות וארוכות:

הקוד משתמש במחזורת הגדרות שבה נקודתיים (:) אחריתו מציין שהאופציה דורשת ערך (למשל c: עבור קובץ קונפיגורציה). נקודתיים כפולות (::) מציינות ערך אופציוני.

PHP

// דוגמה לשימוש בסיסי ב-getopt

```
;(:options = Console_Getopt::getopt($argv, "qvhc$
```

הfonקציה `usage`() בדוגמה מדימה כתיבה ל- `stderr` (ערוץ השגיאות הסטנדרטי), מה שנחשב לפרקטיקה נכונה בכתיבה של כל שיל, כך שהודעות העזרה לא יתערבבו עם הפלט הרגיל של התוכנית.

16.2.4 פרקטיקות טובות

בעת כתיבת תסריטי של (shell scripts), עליך לעקוב אחר מספר פרקטיקות טובות כדי להקל על חירך ועל חייהם של אחרים שיישתמשו בתסריט שלך. לדוגמה, רוב משתמשי UNIX מצפים שהתוכניות שלהם ייגיבו ל-h- foo או -- help foo עם הودעת שימוש קצרה, או שהן ידפיסו שגיאות לערזץ השגיאות הסטנדרטי (standard error) במקום לפלט הסטנדרטי. סעיף זה מונה מספר פרקטיקות שהמחברים מחשבים כ"טובות". (™Good)

16.2.4.1 הودעת שימוש (Usage Message)

הציגו הודעת שימוש ב-standard error וצאו עם קוד שאינו 0 אם התסריט הופעל ללא הפרמטרים המצויפים, או אם הוא רץ עם אופציית -h (או -- help אם אתם משתמשים באופציות ארוכות). הודעת השימוש צריכה למנוע את כל הפרמטרים הנדרשים והאופציונליים.

קיימת גם מוסכמה סטנדרטית לרשום אופציות ופרמטרים:

- [-c] – עשוי להכיל את האופציה -c.
- {c} – חייב להכיל את האופציה -c עם פרמטר.
- [a | -b] – עשוי להכיל או את -a או את -b.
- ...file – חייב לקבל פרמטר אחד או יותר של קבצים.

16.2.4.2 קוד יציאה (Exit Code)

אם התסריט נכשל, צאו עם קוד שאינו 0 (פרט ל-255, השמור ל-PHP עצמה עבור שגיאות הידור/פונCTION). אם התסריט הצליח, צאו עם קוד 0.

16.2.4.3 הودעות שגיאה

הויספו את שם התסריט לפני כל הודעת שגיאה, כדי שהמשתמש יוכל לראות מאייה תסריט מקורה השגיאה. זה שימושי במיוחד אם התסריט מופעל מתוך תוכניתים או תוכניות אחרות.

16.2.5 בקרת תהליכיים (Process Control)

בעת הריצת תסריטי PHP ב-CLI, הרחבה `pcntl` מספקת פונקציות לשילטה בתהליך PHP.

16.2.5.1 תהליכיים (Processes)

תהליך הוא קטע קוד המבוצע על ידי מערכת הפעלה. ב-XUNI, תהליכיים מורכבים מקוד בר-ביצוע, משתני סביבה, זיכרון מחסנית (stack), זיכרון עירימה (heap), מתארים קבצים (file descriptors) ומאפייני אבטחה.

16.2.5.2 פיצול (Forking)

"Forking" הוא מונח ב-XUNI לייצרת תהליך חדש על ידי שוכפול של תהליך קיים. תהליך הבן (child) יירוש את הקוד, הסביבה, הזיכרון ומתארוי הקבצים של תהליך האב.

ההבדל הוא שבתוך תהליך האב, הקריאה ל-`fork` מחזירה את מזהה התהליך (PID) של הבן, בעוד שבתוך תהליך הבן הקריאה מחזירה 0.

Exec 16.2.5.3

כאשר תוכנית אחת מרצה תוכנית אחרת, מבוצע הליך דו-שלבי: ראשית התהיליך הקורא מתפצל (`fork`), ומיד לאחר מכן מבצע קריית `exec` כדי להחליף את הקוד והזיכרון שלו של התוכנית החדשה. הדבר משמש גם לייצרת "Daemons" (תהליכי רקע) על ידי ניתוק מהטרמינל.

16.2.5.4 אותות (Signals)

ב-XUNI, אותן הם מנגן בסיסי להעברת הודעות בין תהליכיים (כמו `SIGINT` כאשר לוחצים על `Ctrl-c`). ב-PHP ניתן להגדיר פונקציה שתטפל באאות אלו באמצעות `pcntl_signal()`.

טבלה 16.3: אותות נבחרים

- **SIGHUP**: ניתוק חיבור טרמינל.
- **SIGINT**: הפרעה (`Ctrl-c`).
- **SIGKILL**: סיום תהליך כפוי (לא ניתן לתפיסה).
- **SIGTERM**: בקשה סיום תהליך רגילה.
- **SIGCHLD**: תהליך בן מת או שינה מצבו.

16.2.6 דוגמאות

הפרק מציג מספר كلمים:

1. **PHP Filter Utility**: כדי לקרוא שורות מהקלט הסטנדרטי (STDIN) ועיבודן דרך פונקציית PHP (`base64_encode()`).
2. **Chat Server**: שרת צ'אט שלם המשתמש ב-`socket_select()` כדי לנהל משתמשים רבים בו-זמןית, תומך בחוצץ קלט/פלט ובפקודות כמו `/who` ו-`/quit`.

16.3 סיכום

בפרק זה למדתם להשתמש ב-PHP עבור תסריטי של (shell scripting) מעבר לסביבת הרשות:

- פונCTION _Getopt פקודה עם
- התנהגות וכונה של תסריטי של וניהול קלט/פלט סטנדרטי.
- בקרת תהליכי (fork, exec, signals).
- כתיבת שרת PHP עצמאיים.

נספח א

איןדקס חבילות PECL-ו-PEAR

תקסוט זה נוצר אוטומטית מקובצי ה-`PEAR.xml` של הזמינים דרך שרת CVS של PHP ומוצג בפורמט "כפי שהוא" (as-is).

A.1 אימות (Authentication)

A.1.1 Auth

מאג'ר: PEAR – רישיון: PHP License

יצירת מערכת אימות.

A.1.1.1 תיאור

חבילת `PEAR::Auth` מספקת שיטות לייצור מערכת אימות באמצעות PHP.
נכון לעכשו היא תומכת בכל האחסן הבאים לקריה/כתיבה של נתוניהתחברות:

- כל מסדי הנתונים הנתמכים על ידי שכבת מסד הנתונים של PEAR
- כל מסדי הנתונים הנתמכים על ידי שכבת מסד הנתונים MDB
- קובייצ טקסט פשוט (Plaintext)
- שרת LDAP
- שרת IMAP ו-POP3
- חשבון vpopmail
- RADIUS
- קובייצ סיסמות של SAMBA
- SOAP

A.1.2 Auth_Enterprise

שירות אימות והרשאות ברמת הארגון (Enterprise). החבילה כוללת שכבת שירות המטפלת בבקשות אימות ולköח PHP. תכונות עיקריות: מבוסס Web Service, מימוש מודול "ספק" (Provider) המטול להתמסח למאגרי נתונים שונים, ושימוש בסט הרשאות אחד.

A.1.3 Auth_HTTP

מספקת שיטות לייצור מערכת אימות HTTP בדומה לאימות מבוסס Apache. htaccess של

A.1.4 Auth_PrefManager

מחלקה לניהול העדפות משתמש באפליקציית ווב, המאחזרת ערכים לפי זיהוי משתמש ושם הصفה.

A.1.7 LiveUser

מסגרת עבודה (Framework) לניהול אימות משתמשים והרשאות. מרכיבת שלושה אלמנטים: מחלקת **LiveUser**, מכל' אימות (Auth) ומכל' הרשאות (Perm). המערכת מאפשרת לבדוק משתמשים מול מספר מכל' נתונים במקביל (למשל, מסד נתונים מקומי ושרת ארגוני).

A.2 בדיקות ביצועים (Benchmarking)

A.2.1 Benchmark

מסגרת עבודה למדידת זמן ריצה של סקורייפטים ב-PHP או קריאות לפונקציות.

A.3 מטמון (Caching)

(A.3.1 APC (Alternative PHP Cache

ספק מסגרת עבודה חופשית וחזקה לאופטימיזציה ואחסון במטמון של קוד הביניים (intermediate code) לתוצאות פונקציות או פלט של סקורייפט של PHP.

A.3.2 Cache

מסגרת עבודה לאחסון במטמון של נתונים שירוטיים, תוצאות פונקציות או פלט של סקורייפט שלם.

A.3.3 Cache_Lite

מערכת מטמון קטנה ומהירה המותאמת לעובדה עם קבצים, הכוללת מנגןוני נעלית קבצים למניעת שגיאות.

A.4 הגדרות (Configuration)

A.4.1 Config

"אולר שווייצרי" לניהול הגדרות. תומך ביצירת הגדרות מאפס, ניתוח (Parsing) ופלט בפורמטים שונים (XML, PHP, INI, Apache).

A.5 מסוף (Console)

A.5.1 Console_Color

מאפשרת שימוש קל בצבעי ANSI במסוף. למשל, הפיכת קודים כמו `%z` לצבע אדום.

A.5.2 Console_Getopt

מיושן PHP ל"-getopt", התומך באופציות קצרות וארוכות בשורת הפקודה.

A.5.3 Console_ProgressBar

ממשק להציג פסוי התקדמות במסוף (למשל עבור הורדות או משימות ממושכות).

A.5.5.2 System_Command

ממשק להרצה פקודות מערכת הפקודה בצורה בטוחה, עם טיפול בשגיאות PEAR והפרדה בין פלט סטנדרטי (stdout) לפלט שגיאות (stderr).

A.6 מסדי נתונים (Database)

A.6.1 DB

שכבה הפешטה למסדי נתונים (Abstraction Layer) המספקת API מונחה עצמים, נידות בין מערכות ניהול מסדי נתונים (DBMS), טיפול אחיד בשגיאות ותמייה בשאלות מוכנות (prepare/execute). תומכת ב有很多种 רחבות של מסדי נתונים כמו MySQL, PostgreSQL, SQLite MySQL ועוד.

A.6.6 DB_DataObject

מבצע שתי שימושות: בניית שאילות SQL מבוססות אובייקטים, ותפקיד כמארגן נתונים עבור שורות בטבלה.

A.6.7 DB_DataObject_FormBuilder

يוצר אוטומטית אובייקטים של HTML (טפסים) המבוססים על הגדרות מסד הנתונים של DB_DataObject, מה שמאיץ פיתוח אבות-טיפוף.

A.6.10 DB_NestedSet

וAPI לבניה ותשאול של מבני עץ בעלי עומק אינסופי בתוך מסד נתונים ייחודי.

A.6.16 MDB / MDB2

שכבה הפешטה מתקדמת המשלבת את PEAR DB Metabase-i. שמה דגש רב על נידות (Portability) ותומכת בתכונות מתקדמות כמו ניהול מבנה מסד הנתונים (Schema) באמצעות קובץ XML ללא תלות בסוג מסד הנתונים.

A.6.17 MDB2

מארגן: PEAR – רישיון: BSD – מأت: לוקיאס קהואה סמית' (móvel)

שכבה הפешטה למסדי נתונים.

A.6.17.1 תיאור

MDB2 הוא מיזוג של שכבות הפешטה PEAR DB Metabase-i. הוא מספק API אחיד לכל מערכות RDBMS הנתמכות. ההבדל העיקרי בין חבילות הפешטה האחרות הוא ש-MDB2 הולך רחוק יותר כדי להבטיח נידות (Portability).

בין התכונות של MDB2:

- API לשאלות בסגנון מונחה עצמים (OO).
- פורמט פונקציית (DSN) (Data Source Name) או מערך להגדרת שרת מסד נתונים.
- הפשתת סוג נתונים והמרתם לפי דרישת.
- קודי שגיאה נידים.
- שלילת שורות רציפה, לא רציפה או בקבוצות (Bulk).
- יכולת ביצוע שאילות מאוחסנות (buffered) או לא מאוחסנות.
- אמולציה של הינה/ביצוע (Prepare/Execute).
- ניהול מבנה מסד הנתונים (Schema) באמצעות XML ללא תלות ב-RDBMS.
- הנדסה לאחר (Reverse engineering) של סכימות XML מממד נתונים קיים.

מסדי נתונים נתמכים: MySQL, PostgreSQL, Oracle, Frontbase, Interbase/Firebird, MSSQL, SQLite ועוד.

A.6.19 MDB_QueryTool

ממשק מונחה עצמים לאחיזור ושינוי נתונים בקלות. זהה הפשטה של שפת SQL המספקת שיטות כמו `setWhere`, `setOrder`, `setJoin` וכו' לבנייה קלה של שאלות, ומתמחקת היטב עם טופס HTML באמצעות מערכים.

A.6.20 oci8

מאגר: PECL. מעתפת (Wrapper) לממשק הקריאות של אורקל (OCI), המאפשרת גישה למסדי נתונים מסוג Oracle 7/8/9.

A.6.23 SQLite

מאגר: PECL. קישורים למסד הנתונים SQLite – ספרית C המממשת מנוע מסד נתונים SQL משובץ (Embedded), שאינו דורש תיליך שרת נפרד.

A.7 תאריך ושעה (Date and Time)

A.7.1 Calendar

חברה לבניית מבני נתונים של לוח שנה (ללא תלות בפלט). מאפשרת בניית ממשק משתמש בקלות מעל מבנה הנתונים וחיבורו למאגרי מידע של "אירועים".

A.7.2 Date

מחלקות גנריות לייצוג וניהול תאריכים, זמנים ואזורי זמן מביי להסתמך על Timestamps (חותמות זמן), מה שמאפשר עבודה עם תאריכים שלפני 1970 ואחרי 2038.

A.8 הצפנה (Encryption)

- **Crypt_CHAP**: יצרת חבילות CHAP (פרוטוקול אימות). תומך ב-CHAPv1 (פרוטוקול אימות).
- **MS-CHAPv2**.
- **Crypt_HMAC**: מחלקה לחישוב חתימות (Hashes) תואמת RFC 2104.
- **Crypt_RC4**: מחלקה לביצוע הצפנה RC4.
- **mcrypt**: מימוש של אלגוריתם ההצפנה TEA (גרסה חדשה). אינו תליי ב-.
- **Crypt_Xtea**.

A.9 פורמטי קבצים (File Formats)

- **bz2** (PECL): הרחבה לניהול דחיסת2 Bzip2.
- **Contact_Vcard_Build/Parse**: יצרה ופענוח של קוביי vCard (אנשי קשר) בגרסאות 2.1-3.0.
- **File_DICOM**: קראיה ושינוי של קוביי DICOM (סטנדרט להדמיה רפואי כדוגמת צילומי רנטגן CT).
- **Spreadsheet_Excel_Writer**: יצרת גיליונות אקסל (פורמט BIFF5) ללא צורך באובייקט COM.
- **Zip** (PECL): הרחבה לקרואת קוביי Zip.

A.10 מערך קבצים (File System)

- **Archive_Tar**: ניהול קובצי Tar (יצירה, חילוץ והוספה), כולל תמיינה ב-Gzip ו-Bz2.
 - **File_Find**: חיפוש במערכת הקבצים, כולל חיפוש רקורסיבי ושימוש ב-Globbing.
 - **File_Passwd**: ניהול סוגים רבים של קובצי סיסמאות (Unix, SMB, .htpasswd, .htdigest ועוד).
 - **VFS**: ממשק API למערכת קבצים ירטואלית, עם תמיינה ב-SQL, FTP, NFS ועוד.

A.12 HTML

- **HTML_BBCodeParser**: מנוע לניטוח והחלפת תagi UBB (כמו בפורומים) בקוד HTML תקין.
 - **HTML_Crypt**: הצפנה טקסט (כמו כתובות מייל) המפוענה מצד הלוקה באמצעות JavaScript, למניעת איסוף נתונים על ידי רובוטי ספאם.
 - **HTML_CSS**: משקל מונחה עצמים לצירה, פענוח וניהול של הצהרות CSS וגילוונות סגנון.
 - **HTML_Menu**: יצירה וניהול של מבני ניווט ותפריטים באתרם מתוך מערך נתונים (Hashes).
 - **HTML_Page**: משקל ליצירת דפי XHTML תקינים, כולל תמיינה ב-Doctypes, תagi META, וקישור סקריפטים וגילוונות סגנון.

A.12.9 html parse

מاجر: PECL – רישיון: PHP License – מأت: הרטמות הולצגרפה (móvel)
הרחבה לניטוח (parsing) של HTML המבוססת על ספריית ekhtml.

A.12.10 HTML Progress

מאג'ר: PEAR License 3.0 – רישיון:
דרך מהירה וקלה להטמעת פס טעינה (loading bar) במסמכים XHTML.
חבילה זו מאפשרת הוספת פס טעינה הנitin להתקנתה אישית מלאה. הדפדפן נדרש לתמוך ב-DHTML.

- יכולת ליצור פסים אופקיים, אנכיים וגם מעגלים, אליפסות ופוליגונים.
 - תמייכה בגילוונות סגנון (CSS) ו-JavaScript חייזרניים.
 - תאמימות מלאה לתקן CSS/XHTML ואינטגרציה קלה עם מנועי תבניות.
 - שימוש של תבנית העיצוב Observer (תצפיתן).
 - אפשרות לביטול התקדמות על ידי המשמש בכל עת.

A.12.11 HTML QuickForm

מאג'ר: PEAR – רישיון: PHP License –
חבילה המספקת שיטות לצירה, אימות ועיבוד של טופס HTML בצורה דינמית.
תזכורות:

- מעל 20 אלמנטים מוכנים לשימוש.
 - קוד תואם XHTML.
 - חוקי אימות ניתנים להרחבה וסינון אוטומטי בצד השרת.
 - יצרת קוד JavaScript לאימות בצד הלקוח לפי דרישתך.
 - תמיינה בהעלאת קבצים ובהתקarma אישית מלאה של תצוגת התוופס.

A.12.12 HTML_QuickForm_Controller

תוסס ל-HTML המאפשר בניית טפסים מרובי-דף (multipage forms), תוך שימוש בתבנית העיצוב PageController לניהול ערכי הטופס בין הבקשות השונות.

A.12.15 HTML_Table

מאג'ר: PEAR – רישיון:

הופכת את עיצוב טבלאות ה-HTML לקל, גמיש ויעיל. מאפשרת שינוי טבלאות בכל עת, טיפול ב-chosen colspan ו שימוש colspan rowspan/chosen בגדירות טבלה.

A.12.16 HTML_Table_Matrix

הרחבה ל-HTML_Table המאפשרת מילוי אוטומטי של טבלה בהתאם לדריכים שונים (למעלה, למטה, באלසון וכו') באמצעות מחלקות Filler.

A.12.18 HTML_Template_Flexy

מאג'ר: PEAR – רישיון:

מנוע תבניות עצמאי במילוי המבוסס על Tokenizer.

תכונות:

- API פשוט מאוד ומהיר ביותר (פחות פי 4 מרוב המנגעים האחרים).
- תמיכה בעריכה באמצעות עורך WYSIWYG מבלי שהקוד ישבר.
- אבטחה משופרת מפני התקפות XSS (סינון אוטומטי של משתמשים).
- תמיכה בריבוי שפות ובאלמנטים דינמיים.

A.12.24 Pager

מאג'ר: PEAR – רישיון:

חברילה לחולקת נתונים לדפים (Paging). היא מקבלת מערך נתונים ומבנה קישורים לדף בסגנון "חלון קופץ" או "חלון מחליק".

A.13 HTTP

A.13.2 HTTP_Client

דרך קלה לביצוע בקשות HTTP מרובות ועיבוד התוצאות שלהן. ניהול עוגיות (cookies) והפניות (redirections) באופן אוטומטי.

A.13.3 HTTP_Download

ממשק לשילוח הורדות קבצים ללקוח דרך HTTP, כולל תמיכה במתמונ (caching), דחיסה והורדות חלקיות (resuming).

A.13.5 HTTP_Request

ספקת דרך קלה לביצוע בקשות HTTP (תמיכה ב-GET, POST, HEAD, PUT, DELETE), אימות (authentication), פרוקס ו-SSL (SSL).

A.13.8 HTTP_Upload

ניהול קל ומאובטח של העלאת קבצים מטופס HTML. תומכת בהעלאת מספר קבצים, שינוי שמות בצורה בטוחה או אימות סימות קבצים.

A.14 תמונות (Images)

A.14.2 Image_Barcodes

יצירת ברקודים ממחוזות נתונות באמצעות פונקציות GD.

A.14.5 Image_Graph

שרטוט גרפים מנתונים מספריים (תנועה, כ سوف וכו') במגוון פורמטים (קווים, עמודות וכו'), כולל תמיכה בערוצי אלפא ורשתות.

A.14.6 Image_GraphViz

ממשק לכלי ה-GV של AT&T ליצירת גרפים מכונים ולא מכונים והציגם הוייזואלית.

A.14.11 Image_Transform

ספקת ממשק סטנדרטי לביצוע טרנספורמציות בתמונות (שינוי גודל, סיבוב, הוספה טקסט) תוך שימוש בספריות שונות כמו NetPBM, GD, ImageMagick.

A.15 בינלאומיות (Internationalization)

A.15.1 fribidi

מאגרי: PECL. ממשק לשפירות FriBidi המממשת את אלגוריתם BiDi Unicode, וספקת כלים לטיפול בטקסט הנקטב מימין לשמאל (RTL).

A.15.2 N18Nv1

מאגרי: PEAR – רישאון: PHP

חברילת בינהום (Internationalization). חברילה זו מסייעת בלוקלייזציה (התאמת לשפה ולתרבות) של אפליקציות. היא כוללת שיטות לתרגום, זיהוי שפת המשתמש (דרך הדפסן) ועיצוב תואם-תרבות של מספרים, תאריכים, שעות ומטבעות.

A.15.3 N18Nv2

גרסה מתקדמת של חברילת הבינהום. היא שואפת לספק דרך בלתי תליה במערכת ההפעלה לביצוע פקודות setlocale()) ומיצעה שמות של מדינות ושפות המתורגם לשפות רבות. המפתחים קוראים לציבור לתרום מידע על מוסכਮות עיצוב מקומיות מרחבי העולם.

A.15.5 Translation

מחלקה לייצרת אורותים רב-לשוניים המאפשרת אחסון ושליפת מחוזות טקסט מניסד נתונים (באמצעות PEAR::DB). היא טוענת את כל המחווזות הרלוונטיות לעמוד מסוים לערך משתנה כדי להאיץ את הגישה ולמנוע עומס על השרת.

A.15.6 Translation2

מחלקה לניהול אפליקציות רב-לשוניות. היא מספקת דרך קלה לשילוף מחרוזות ממוקורות נתונים שונים (DB, MDB, gettext) תוך שימוש במנגנון מטמון (Caching). קיימת מחלוקת ניהול (Admin) להוספה והסירה קלה של שפות ומחרוזות.

A.16 רישום (Logging)

A.16.1 Log

מאג'ר: PEAR – רישיון:

מערכת רישום (Logging) מופשטת התומכת בכתיבה יומני ריצה למסוף, קבצים, syslog, SQL, SQLite, DB. היא מימושה את תבנית העיצוב Subject-Observer. דואר אלקטרוני ועוד.

A.16.2 Log_Parser

מנתח (Parser) המיעוד לקרואת כמעט כל סוג שלקובץ יומן ריצה. ניתן להגדיר פורמט יומן אישי באמצעות קובץ הגדרות בסגנון XML ולסנן את הנתונים המתקבלים.

A.17 דואר אלקטרוני (Mail)

A.17.1 Mail

חברילה המגדירה ממשק אחד לשימוש שליחות דואר. היא תומכת בפונקציית mail() של PHP, ב-mail() ו-smtp. היא כוללת גם כלי לאימות רישומות כתובות דוא"ל לפי תקן RFC 822.

A.17.5 Mail_Mime

ספקת מחלקות ליצירה ופענוח של הודעות MIME (דואר עם HTML, קבצים מצורפים, תמונות מוטמעות וכו').

A.17.6 Mail_Queue

חברילה לניהול תור הודעות דואר. היא מאפשרת להכניס הודעות מכל זמני ושלוח אותן מאוחר יותר בركע (למשל באמצעות crontab), כדי לא לעכב את טיענת הדף).

A.18 מתמטיקה (Math)

A.18.2 Math_Complex

מחלקות המגדירות מספרים מרוכבים ופעולות עליהם (אריתמטיקה, פונקציות טריגונומטריות, לוגריתמים ועוד).

A.18.3 Math_Fibonacci

חברילה לחישוב מספרי פיבונאצ'י באמצעות נוסחת הנסיגה או נוסחת לוקאו (יחס הזהב).

A.18.6 Math_Matrix

מחלקה לייצוג מטריצות וביצוע פעולות עלייהן, כולל פתרון מערכות שלמשוואות ליניאריות.

A.18.9 Math_RPN

המרה של ביטויים מתמטיים לכתיבה פולני הפוך (Reverse Polish Notation) וחישובם.

(Networking) A.19

A.19.12 Net_DNS

ספריית Resolver לתקשורת עם שרת DNS לביצוע שאילתות, העברת אזוריים (zone transfers) ועדכונים דינמיים. היא עוקפת את ספריית המערכת ומתקשרת ישירות עם השרת.

A.19.14 Net_FTP

ממשק מנומנה עצמים לפונקציות ה-FTP של PHP. מוסיף תכונות כמו העלה והורדה רקורסיבית, ייצור ספריות ושימוש ב-Observer להציג פס' התקדמות.

A.19.22 Net_IRC

מחלקה ליישומי לקו או בוטים של IRC. תומכת בחיבור למספר שרתים במקביל, סוקטים שאינם חוסמים (non-blocking) ומערכת Callback לטיפול בהודעות.

A.19.23 Net_LDAP

ממשק מנומנה עצמים לחיפוש וניהול רשומות LDAP (mbased על הממשק של Perl). תומך ב-TLS, LDAP 3.7 וניהול סכימות.

A.19.31 Net_SmartIRC

חברילה מתקדמת במיוחד לתקשורת ברכשות IRC (תואמת RFC 2812). היא כוללת ניהול אירועים מבוסס זמן, הגנה מפני הצפה (Flood protection), חיבור חדש אוטומטי, סנכרון ערוצים ומשתמשים ומערכת רישום שגיאות מפורטת.

A.19.33 Net_Socket

ממשק לממשק סוקט של TCP. מספק פעולות חוסמות ושuinן חוסמות ומצבי קריאה/כתיבה שונים (לפי בית, לפי שורה וכו').

A.19.36 Net_UserAgent_Detect

מזהה את סוג דפדף האינטרנט, הגרסה ומערכת הפעלה מתוך מחזורת ה-HTTP User Agent.

A.19.37 Net_UserAgent_Mobile

מאגר: PEAR – רישיון: PHP – מأت: קובו אוטסוהירו (móvel)

מנתח (Parser) של מחזורות User Agent למכשירים ניידים (בעיקר יפנים). שימושי לניטוב דפים לפי סוג המכשיר הנידי.

A.19.38 Net_Whois

כלי לביצוע שאילתות בשירותי Whois לאייתור פרטי שמות מתחם (Domains) ומספרי רשת מול מאגרי NIC שונים.

A.19.39 opendirirectory

ממשק PHP ל-OpenDirectory Framework – ארכיטקטורת שירות ספריות המאפשרת לאפליקציות לשולף מידע מאוחסן בצורה מרכזית.

A.19.41 tcpwrap

קיישור (Binding) ל-`tcpwrappers` המאפשר ניהול של קובץ הרשאות `/etc/hosts.allow` ו-`etc/hosts.deny`.

A.19.42 `uuid`

פונקציות לייצירה וניתוח של מזהים ייחודיים אוניברסליים (UUIDs). תלוי בספרייה חיצונית (`p uuid`).

(Numbers) A.20 מספרים

A.20.1 Numbers_Roman

שיטות סטטיות להמרה בין מספרים רגילים לספרות רומיות (תומך במספרים עד 5,999,999).

A.20.2 Numbers_Words

מחלקה להמרת מספרים הכתובים בספרות למילים (איות המספר) בשפות רבות (כגון אנגלית, גרמנית, צרפתית, פולנית ועוד).

(Payment) A.21 תשלום

- **Payment_Process:** שילד (Framework) אחיד לעיבוד כרטיסי אשראי והמחאות אלקטטרוניות ללא תלות בשער התשלום (Gateway).
- **TCLink (PECL):** מאפשר עיבוד כרטיסי אשראי דרך שער התשלום TrustCommerce.

PEAR תשתיות A.22

- **PEAR Installer:** מערכת הבסיס הכלולת את מחלקת האם, מנגננו טיפול בשגיאות, מתקין החבילות ומחלקות עזר למערכת ההפעלה.
- **PEAR_Frontend_Web/Gtk:** ממשקי משתמש (גרפיים או מבוססי דפדפן) לניהול והתקנת PEAR.
- **PEAR_Info:** יוצר דף מידע מפורט על התקנת ה-PEAR שלך, בדומה ל-`phpinfo()`.

PHP שפת A.23

(A.23.2 bcompiler (PECL

מהדר Bytecode המאפשר לקוד תסריטי PHP כדי להגן על קוד המקור, ליצור ספריות קוד סגור או אפליקציות למערכות משובצות (Embedded).

(A.23.5 memcache (PECL

הרחבה לעובדת עם – שירות זיכרון מטמון שנועד להפחית עומס מסדי נתונים על ידי שמירת אובייקטים בזיכרון-`RAM`.

A.23.9 PHPUnit

שלד לבדיקות נסיגה (Unit tests) ובדיקות יחידה (Regression testing), מבוסס על `JUnit`.

A.23.12 PHP_Fork

מעטפת לפונקציית `fork_pcntl()` המאפשרת ניהול תהליכי (Multiprocessing) ב-PHP עם API דומה לשפת Java, כולל שיתוף זיכרון בין תהליכי.

A.23.15 Validate

חבילה לאימות נתונים שונים: מספרי כרטיס אשראי, כתובות אימייל, תאריכים, כתובות URL ונתונים פיננסיים (כמו IBAN).

A.23.16 Var_Dump

מעטפת לפונקציית `var_dump` המאפשרת להציג מידע מובנה על משתנים במגוון פורמטים ויזואליים (HTML, XML, טקסט פשוט ועוד).

(A.23.18 Xdebug (PECL

הרחבה רבת עצמה לניפוי שגיאות (Debugging). מספקת מעקב אחר פונקציות (Stack trace), ניתוח ביצועים (Profiling) ויכולת לעזור את הרצת הקוד באופן אינטראקטיבי.

A.24 עיבוד (Processing)

A.24.1 FSM

חבילה המממשת "מכונת מצבים סופית" (Finite State Machine).

A.25 מדע (Science)

A.25.1 Science_Chemistry

מחלקות לייצוג אובייקטים כימיים: אטומים, מולקולות ומרקדו-מולקולות. כולל מנגחי קבצים לפורמטים כימיים ומידע על הטבלה המוחזרית.

(A.26 Streams (זרמים

שימושי זרים וכלי עזר ל-PHP.

A.26.1 bz2_filter

מאגר: PECL – רישיון: PHP – מأت: שרה גולמן (mobilita)

מיומש מסנן זרם לכיווץ/פריסה בפורמט2 gzip. אפשר כיווץ "על המשתמש" (inline) בכל זרם קלט/פלט של PHP.

A.26.2 ogg vorbis

עטיפת fopen לקובץ OGG/Vorbis. מאפשר פריסה של נתונים OGG לאודיו PCM ולהיפך.

A.26.5 Stream_Var

מאגר: PEAR – רישיון: PHP License

אפשר גישה מבוססת זרם לכל משתנה. ניתן להתיחס למערכות כאלו ספריות, ובכך להחליף ספריות וקבצים זמינים במשתנים בתוך האפליקציה.

A.27 Structures (מבנה)

מבנה נתונים וטיפוס נתונים מתקדמים.

A.27.1 Games_Chess

מאג'ר: PEAR – רישיון: PHP License

בנייה איות של משחק שחמט לוגי. מטפל בלוח השחמט ובניתו פורמטים סטנדרטיים כמו FEN ו-SAN (תיאור מהלכים).

A.27.3 Structures_Grid

אפשרה ליצור מבנה דמי טבלה (Grid) המבוסס על סט נתונים. תומכת בהוצאה פלט במגוון פורמטים כגון טבלת HTML, מסמך XML, גליון Excel ועוד, כולל יכולות דפסוף ומיזן.

A.27.4 Structures_Graph

ספרייה ליצור וניהול של מבני נתונים מסווג גרף (Graph). מאפשרת בניית גרפים מכונים ולא מכונים ושליפת מאפיינים מהטופולוגיה שלהם.

A.27.6 Tree

ניהול עצים גנרי. תומך בסיס נתונים וב-XML כמקורות מידע. מאפשר עבודה עם עצים בזיכרון או קראת צמתים לפי צורך (שימוש לעצים ענקיים).

A.28 System (מערכת)

A.28.2 System_ProcWatch

ניתוח תהליכי ריצים על בסיס קובץ הגדרות (INI, XML או מערך) שבו מגדירים תנאים ופעולות לביצוע.

A.29 Text (טקסט)

A.29.1 enchant

מאג'ר: PECL. מקשר לספריית libenchant, התומכת כמעט בכל כלי איות השפה (כולל **hspell** לעברית ו-**uspell** ליהדות).

A.29.3 panda

ספרייה PDF חופשית לייצרת מסמכים PDF.

A.29.6 Text_Password

יצירת סיסמות הנינთנות להגיה (pronounceable) או כאלה שאין ניתנות להגיה.

A.29.8 xdiff

הרחבה ליצור והchèלה של טלאים (patches) על קובצי טקסט וקבצים ביןאריים.

A.30 Tools and Utilities (כלי ועזרים)

A.30.2 fann

מימוש של ספרית רשות עצביות מלאכותיות (Artificial Neural Networks).

A.30.4 PhpDocumentor

כלי ליצירת תיעוד אוטומטי ישירות מקוד המקור (דומה ל-JavaDoc). תומך בפלט HTML, PDF, CHM ו-XML.

(A.30.5 SPL (Standard PHP Library

הרחבה הממשת ממתקדים ומחלקות לעילות בגישה לנתחים, כגון Iterators (איטרטורים) מתקדמים ללולאות וטיפול בספריות.

A.31 Web Services

A.31.2 Services_Weather

ממשק לשירותי מזג אוויר מקוונים שונים. מאפשר חיפוש מיקומי תחזית ושליפה נתונים נוכחים.

A.31.3 SOAP

מימוש פרוקסוי ושירותי SOAP (פרוטוקול תקשורת מבוסס XML).

A.31.6 XML_RPC

מימוש PHP ל프וטוקול XML-RPC, כולל תמיכה בשירותי פרוקסוי ואיומות.

A.32 XML

A.32.1 XML_Beautifier

עיצוב מסמכים XML על ידי הוספת היחסות (indentation) ושבירת שורות להקלת הקריאה.

A.32.6 XML_HTMLSax

מנתוח SAX עבור מסמכים HTML או מסמכים XML שאיןם בנויים היטב (well formed).

A.32.9 XML_Parser

מנתוח XML המבוסס על הרחבה expat המובנית ב-PHP. תומך במצב עבודה מבוסס אירועים (callbacks).

A.32.11 XML_RSS

מנתוח עבור מסמכים RSS ו-RDF (סיכון אתרים).

A.32.12 XML_SaxFilters

תיאור: חבילה זו מספקת תשתיית לשימוש במסנני Sax ב-PHP. סינון Sax הוא גישה שהופכת את ניתוח מסמכים XML למודולרי וקל לתחזקה.

הדגם מבוסס על "הורה" (parent) ו"ילד" (child): הורה הוא המсон שנמצא "במעלה הזרם" ומקבל את אירועי-XML ראשוני, בעוד שילד הוא מסון "במוריד הזרם" שאליו מואצלים האירועים. הנתה Parser (עצמו נמצא תמיד בראש "אילן היוחסין") ומצביע את כל האירועים למסננים שתחתיו.

A.32.13 XML_Serializer

מאגר: PEAR – רישיון: PHP License – מأت: סטפן שמידט (móvel)

"אולר שוואץ'" לקריאה וכתיבה של קובצי XML. החבילה מסוגלת להפוך מבני נתונים מורכבים (כמו מערכים ואובייקטים) למסמכים XML ולהיפך. היא יכולה לשמש כחלופה לפונקציות `serialize()` ו-`unserialize()` המוגנות, ואף מסוגלת לנתח קובצי RSS ולהופכם למערך מובנה בקלהות.

A.32.14 XML_sql2xml

מחלקה המקבלת תוצאות משילת SQL (או בייקט DB::PEAR) ומחזירה ייצוג XML שלהן. היא מtabסת על הרחבה DOMXML של PHP.

A.32.15 XML_Statistics

חבילה לשילוף מידע סטטיסטי ממסמci XML, כגון כמות תגיוט, מאפיינים (attributes), ישיות (entities) והנחהות עיבוד.

A.32.16 XML_SVG

מספרקט ממשק תכונתי (API) מונחה עצמים לבניית מסמכי גרפייה וקטוריית מסווג SVG.

A.32.17 XML_svg2image

המרת קובצי SVG לתמונות PNG או JPEG בעזרת כלי חיצוני (apache-batik). דורשת תמיכה ב-Java-ב-PHP.

A.32.18 XML_Transformer

מאפשרת לקשר פונקציונליות של PHP לשירות לTAGIOT XML, ובכך להפוך עץ XML של קלט לעץ XML של פלט ללא צורך בשימוש ב-XSLT.

A.32.19 XML_Tree

ייצוג נתונים XML במבנה של עץ ללא צורך בהרחבות חיצונית כמו DOMXML.

A.32.20 XML_Util

אוסף שיטות עזר נפוצות לעובדה עם XML: יצירת רשימות מאפיינים מערכים, יצירת TAGIOT, אימות שמות TAGIOT ועוד.

A.32.21 XML_Wddx

כלי לסדרול (Serialization) של WDDX. משמש כחלופה להרחבת ה-XML המובנית ומיצר קבצים הנינתיים לעריכה נוחה (עם הzechot ושימוש ב-CDATA).

A.32.22 XML_XPath

מחלקה למניפולציה ושאליות על מסמכi XML באמצעות XPath ו-DOM. היא מאפשרת ניווט קל בעץ ה-DOM ומחזירה תוצאות כאובייקטי XPath_Result. דורשת את הרחבה domxml.

A.32.23 XML_XSLT_Wrapper

מספרקט ממשק אחיד לביצוע טרנספורמציות XSLT באמצעות ספריות ופקודות שונות (כמו, Sablotron, xsllproc, MSXML ועוד). היא מפשטת את העבודה מול מנועי XSLT שונים.

A.32.24 XML_XUL

תיאור: מחלוקת לבנית אפליקציות XUL (שפת ממשך המשתמש של Mozilla). היא מספקת API דומה ל-DOM ליצירת רכיבי ממשך מורכבים כמו טבלאות, עצים ותיבות לשוניות (tabs) כאובייקטי PHP.

נספח ב

סימוכין לפורתט **phpDocumentor**

"תיעוד הוא כמו סקו: כשהוא טוב מאוד, הוא טוב; וכשהוא רע, הוא עדיף על כלום." — דיק ברנדון

B.1 מבוא

מלבד תקני כתיבת קוד, לפרויקט PEAR יש שיטה סטנדרטית לתיעוד מחלקות וחבילות. שיטה זו משתמש בכללי **phpDocumentor** כדי ליצור תיעוד HTML הנitinן לדף מתוך הערות בקוד המקור. הכללי הרשמי לתיעוד מחלקות PEAR הוא **phpDocumentor** (כתובת: <http://phpdoc.org>), המסוגל ליצור לא רק HTML, HTML, אלא גם PDF ו-Docbook. הוא דומה מאוד ל-**JavaDoc** ומשתמש בשפת סימון דומה לתיעוד אלמנטים. ניתן להתקין אותו באמצעות הפקודה:

```
pear install phpDocumentor $
```

נספח זה יציג לכם את הכללי הרשמי, יחד עם דוגמאות לשימוש בו וכייזד לתעד את המחלקות שלכם.

B.2 הערות תיעוד (Docblocks)

הכללי מייצר תיעוד לאלמנטים המוטמעים בקוד כהערות. הכללי מזזה תשעה שבעה סוגי שונים של מקטעים: משתנה גלובלי, include, קבוע, פונקציה, define, מחלוקת, משתנה, מתודה (method) ועמוד.

כל קובץ בפרויקט ה-PHP שלכם צריך להתחיל ב-**Docblock** בرمמת העמוד, המתעד היבטים כלליים (כמו מחבר, שם החבילה וכדומה). Docblock תמיד מתחילה ברכף התווים `/**`, בניגוד להערות "רגילים" שמתחילה בדרך כלל ב-`/*`:

```
PHP  
**/
```

package Examples@ *

/*

לאחר מכן, ניתן לתעד אלמנטים ספציפיים. הערת כוללת תיאור קצר (בשורה אחת), תיאור מפורט (שיכל לכנות תגיות HTML כמו `` או `<code>`) ולאחר מכן מקטע מילות מפתח (Tags) המתחילה בסימן `@`.

B.3 מילון תגיות (Tag Reference)

B.3.1 abstract

(זמן ב-4 PHP בלבד; ב-5 PHP המידע נשלף אוטומטית).

משמש לתיעוד מחלוקת מופשטת או פונקציה של מחלוקות ירושת למש.

B.3.2 access

(זמן ב-4 PHP בלבד).

קובע אם אלמנט הוא `public`, `protected` או `private`. אלמנטים פרטיים לא יופיעו בתיעוד המשתמש אלא אם תצביע פקודת מיוחדת. לפי תקני PEAR, פונקציות פרטיות צריכות להתחיל בקוו תחתון (`_`).

B.3.3 author

מצין את שם המחבר וכתובת האימייל שלו: `<author Name <email@example.com@>`

B.3.4 category

משير מחלוקת לקטגוריה רחבה (כמו Database או XML). שימושי מאוד עבור PEAR.

B.3.5 copyright

תיעוד זכויות יוצרים: `.copyright Copyright © 2026, Name@`

B.3.6 deprecated

מסמן פונקציות מישנות שאינן מומלצות לשימוש, עם הסבר מדוע גרסה הנו goede.

B.3.7 example

מאפשר להוסיף דוגמאות קוד. ניתן להוסיף קוד בתוך הערה או לחברו לקובץ דוגמה חיצוני באמצעות: `.example example_file.php@`

B.3.8 filesource

גורם ל-`phpDocumentor` ליצור גרסה של הקוד עם הדגשת תחביר (Syntax Highlighting) ולקשר אליה מהתיעוד.

B.3.9 final

PHP בלבד). מצין שלא ניתן לדדרו (overload) את המחלוקת או התוכנה הזאת.

B.3.10 global

מתעד שימוש המשתנה גלובלי בתחום פונקציה, או מגדר המשתנה כגלובלי ברמת התסריט כולו.

B.3.11 ignore

משמש להחarget אלמנטים מהתיעוד הסופי (למשל כדי למנוע כפליות במקרה של הגדרות מותניות).

B.3.13 internal

תיעוד המיועד לשימוש פנימי בלבד (בתוך החברה) ולא יופיע בתיעוד הציבורי.

B.3.14 licence

קישור לרישיון הקוד (למשל License PHP).

(B.3.16 link) (inline)

יצירת קישור בתוך טקסט התיעוד לכתובת URL חיצונית או לאלמנט אחר בקוד:

link http://www.example.com@{}

B.3.17 name

תחביר:

<name@>

מילת מפתח זו מעניקה שם "ידיוטי" למשתנה גלובלי. בדוגמה הבאה, בתיעוד שיווצר יעשה שימוש ב-\$foo במקומם ב-\$GLOBALS['foo']:

PHP

**/

* דוגמה לתיעוד משתנה גלובלי

name \$foo@ *

['global' string \$GLOBALS['foo']@ *

/*

;'"\$GLOBALS['foo'] = "Foobar\$

B.3.18 package

תחביר:

<package@>

הtagית package@ משמשת לקיבוץ אלמנטים (וחבילות-משנה ב-*phpDocumentor*). זהו פריט הקיבוץ ברמה העילונה ובדרך כלל הוא משוייך לחבילת PEAR. ראו את הדוגמה באIOR B.1, המשתמשת בתגיות package ו-subpackage כדי לטעוד פונקציות במבנה בעל שתי רמות.

אIOR B.1: מבנה חבילות.

PHP

**/

* ניהול מתומן (Cache)

```
package Cache@ *
```

```
/*
```

```
} ()function Cache
```

```
{
```

```
**/
```

```
* שמירה במתמון בבסיס נתונים
```

```
package Cache@ *
```

```
subpackage Cache_DB@ *
```

```
/*
```

```
} ()function Cache_DB
```

```
{
```

```
**/
```

```
* שמירה במתמון בbasis נתונים MySQL
```

```
package Cache@ *
```

```
subpackage Cache_DB@ *
```

```
/*
```

```
} ()function Cache_DB_MySQL
```

```
{
```

B.3.19 param

תחביר:

```
<param (type | object_definition) <$variable@
```

פרמטרים של פונקציות מתועדים באמצעות התגית `param@`. להלן מספר דוגמאות:

PHP

```
**/
```

* פונקציה לחיבור מספרים והכפלתם בשתיים

algo float \$a@ *
param float \$a@ *

algo int \$b@ *
param int \$b@ *

return mixed@ *
/*

(function addNumbersAndMultiplyByTwo (\$a, \$b
})

;return (\$a + \$b) * 2

{

mhaha את ערך בירית המודול של משתנה מתוך קוד המקור, כולל אותו באופן אוטומטי בתיעוד שנוצר. להלן דוגמה מורכבת יותר:

PHP

**/

* החזרת שורות

*

* הרצת שאלתה על חיבור בסיס הנתונים והחזרת מספר השורות שצווין, אם צוין

private@ *

משאב החיבור לבסיס הנתונים param resource \$conn@ *

השאילתה param string \$query@ *

הגבלה למספר זה של שורות מוחזרות param int \$limit@ *

return array@ *

/*

(function _runQuery (\$conn, \$query, \$limit = 0
})

// ... קוד הפונקציה ...

{

B.3.20 return

תחביר:

`<תיאור> (return (type | object_definition@`

השתמשו בתגית `return@` כדי לטעות את סוג הערך המוחזר מהפונקציה שלכם:

PHP

`**/`

`param string $filename@ *
שם הקובץ של התמונה`

`return resource@ *
משאב תמונה מסווג GD`

`/*`

`(function returnNiceGif ($filename`

`) {`

`; (return imagecreatefromgif ($filename`

`{`

B.3.21 see

תחביר:

`see@ <אלמנט>`

באמצעות התגית `@see`, ניתן להוסיף קישורים לאלמנטים אחרים בתיעוד. כל סוג אלמנט של `see@` נתמך כפרמטר לתגית `phpDocumentor`:

PHP

`**/`

`* מחבר מספרים`

`()see string::add@ *
(()see string::add@ *`

`/*`

`(function addNumbers ($number1, $number2`

`{`

```
;return $number1 + $number2
```

```
{
```

```
**/
```

```
* מחלקת לManipולציה של מחרוזות
```

```
/*
```

```
} class string
```

```
**/
```

```
* לחבר מחרוזות
```

```
see addNumbers@ *
```

```
/*
```

```
(function add ($string1, $string2
```

```
)
```

```
;return $string1 . $string2
```

```
{
```

```
{
```

B.3.22 since

תחביר:

```
<גרסה>since@
```

תגית זו מתעדת מתי אלמנט נוסף ל-API. פורמט מחרוזת התיאור הוא חופשי. להלן דוגמה מתוך מחלוקת
PEAR בשם HTML_Common:

```
PHP
```

```
**/
```

```
* מחזירה את ה tabOffset-
```

```
*
```

```
since 1.5@ *
```

```
return void@ *  
/*  
()function getTabOffset  
}  
;return $this->_tabOffset  
{
```

B.3.23 static

זמן עבור 4 PHP בלבד.

תחביר:

```
static@
```

תגית זו מתעדת שניתן לקרוא לMETHODS באופן סטטי (לדוגמא:

PHP

```
**/
```

* מחלקת foo המבצעת פעולה bar סטטית

```
/*
```

```
} class foo
```

```
**/
```

* ניתן לקרוא לפונקציה זו באופן סטטי.

```
static@ *
```

```
/*
```

```
} () function bar
```

```
{
```

```
{
```

```
;()foo::bar
```

B.3.24 staticvar

זמן עבור 4 PHP בלבד.

תחביר:

```
<staticvar (type | object_definition) <$variable@
```

התגית staticvar@ מתעדת משתנה סטטי בתוך פונקציה. משתנים סטטיים אינם נהרסים כאשר הפונקציה מסתיימת. הדוגמה הבאה תדפיס:123

```
PHP  
**/
```

* דוגמה למשנה סטטי בתוך פונקציה

. staticvar integer \$count@ * סופר את מספר הפעמים שנקרה הפונקציה.

```
/*
```

```
} ()function foo
```

```
;static $count
```

```
;++$count$
```

```
;"echo $count. "\n
```

```
{
```

```
foo(); // 1
```

```
foo(); // 2
```

```
foo(); // 3
```

B.3.25 subpackage

תחביר:

```
<subpackage >שם_תת_חבילה@
```

ניתן להשתמש בsubpackage כשכבה קיבוץ נוספת עבור אלמנטים בחבילת שלים. ראו את התיאור של תגית ה- package@ לקבלת דוגמה.

B.3.26 todo

תחביר:

```
<todo >תיאור@
```

באמצעות תגית todo@, ניתן לתעד שינויים שעדיין יש לבצע אלמנט ספציפי. לדוגמה:

PHP

**/

לתעד פרמטרים todo@ *

/*

```
} (function todo_example($a, $b  
{
```

B.3.27 uses

תחביר:

uses@<אלמנט>

תגית זו מבצעת פעולה דומה לתגית `see`, פרט לכך שהיא יוצרת קישור דו-כיווני בין האלמנט ה"בשימוש" לבין האלמנט שבו נעשה שימוש בתגית `uses`. כל ה-`phpDocumentor` מבצע זאת על ידי הוספה תגית `uses` מודומה בשם `usedby` לאלמנט שאליו מצביעה תגית `uses`.

B.3.28 var

תחביר:

<תיאור>var (type | object_definition) <\$variable@

התגית `var` מתעדת את הסוג של משתני מחלקה. הסוג צריך להיות סוג נתונים תקני של PHP, או `"mixed"` אם המשתנה יכול להכיל סוגי שונים:

PHP

**/

* מחלוקת ה"מחלוקת" מבנה (struct) כמו בשפת C

/*

} class person

**/

שםו של האדם var string \$name@ *

/*

;var \$name

**/

```
var int $age@ * גיל האדם
```

/*

```
;var $age
```

{

B.3.29 version

תחברה:

<גרסת> version@

ניתן לטעד את גרסה האלמנט באמצעות תגיות זו. אם אתם משתמשים ב-CVS, ניתן להשתמש בתגיות CVS כגון `Id` ו/`$Revision$`, המוחלפות אוטומטית בגרסתה הנכונה בעת שמיירת הקוד (check-in) בעץ CVS.

(Tag Table) טבלת תגיות B.4

טבלה B.1 מציגה סקירה של המקומות שבהם ניתן להשתמש בתגיות המתווארות בסוף זה. הסימן **X** מצביע שניtin להשתמש בתגית זו לтиיעוד האלמנט, והסימן **M** מצביע **שחובה** (Mandatory) להשתמש בתגית זו כדי לטעד את האלמנט.

טבלה 1.B: טבלת תגיות (Tags)

טבלה זו מסכמת את השימוש האפשרים עבור התגיות השונות ב-[Documentor.php](#).

$X = \text{נתן לשימוש}, M = \text{חובה}$.

עמו ד	מתוד ה	משתנה	מחלקה	Defi ne	פונקטיה	קבוע	Inclu de	משתנה גלובלי	תגית
X	X	X	X	X	X	X	X	X	Access
X	X	X	X	X	X	X	X	X	Author
X	X	X	X	X	X	X	X	X	Copyright

X	X	X	X	X	X	X	X	X	X	Deprecated
X	X	X	X	X	X	X	X	X	X	Example
X	X	X	X	X	X	X	X	X	X	Ignore
X	X	X	X	X	X	X	X	X	X	internal
X	X	X	X	X	X	X	X	X	X	Link
X	X	X	X	X	X	X	X	X	X	see
X	X	X	X	X	X	X	X	X	X	since
X	X	X	X	X	X	X	X	X	X	version
X									X	name
	X				X				M	global
	X				X					param
	X				X					return
X			X							package

X	X								static
---	---	--	--	--	--	--	--	--	--------

B.5 שימוש בכלי **phpDocumentor**

כדי ליצר את התיעוד מקוד המקוור, תזדקקו לכלי **phpDocumentor** המותקן דרך PEAR. הכלי כולל מספר פרמטרים חשובים:

טבלה B.2: פרמטרים של כלי **phpDocumentor**

דוגמה	הערות	אפשרות
index.php,lib.php -f	רשימת קבצים לניטוח (מופרדים בפסיקים).	-f-, --filename
core,libs -d	רשימת ספריות לניטוח.	-d-, --directory
docs/ -t	ספריית היעד עבור התיעוד שיוצר.	-t-, --target
"My Project" -ti"	כותרת התיעוד שיוצר.	-ti-, --title
on -pp	הכללת אלמנטים פרטיים ופנימיים בתיעוד.	-pp-, --parseprivate
HTML:frames:default -o	פורמט הפלט והתבנית לשימוש.	-o-, --output
-s	הכללת קוד מקור עם הדגשת תחביר.	-s-, --sourcecode

פקודת הרצה לדוגמה:

```
$ phpdoc -d directory -pp on -s on -o HTML:frames:default -t outputdir
```

דוגמה לתיעוד שנוצר

מתוך קובצי המקור של הפרויקט, `Documentor.php` מיציר מסך דף דין המציג את היררכיה המחלקות והמתודות.

- **איור B.2 ו-B.3:** מציגים את התיעוד עבור המחלקה `SumNumberElements`. ניתן לראות בבירור את עץ הירושה (`Inheritance`) המראה שהיא ירושת מהמחלקה `Sum`.
- **איור B.4:** מציג את היחסים בין כל המחלקות בחבילה עצם, ומראה אילו מחלקות הן תתי-מחלקות של `Sum`.
- **איור B.5:** מציג אינדקס של כל האלמנטים הזמינים בחבילה: מודולים, מחלקות, פונקציות, משתנים וקבועים.

נספח C: מדריך התחלת מהירה ל-Zend Studio

C.1 גרסה 3.x

Zend Studio היא סביבת פיתוח מושלבת (IDE) המיועדת למפתחי PHP מתקני. זהו הכלי היחיד המאפשר את כל רכיבי הפיתוח החדשניים למחזר החיים המלא של אפליקציית PHP.

C.3 אודוט Zend

Zend היא "חברת ה-PHP". מייסדה, **אנדי גוטמןס וזאב סורסקי**, הם היוצרים והמצדדים של שפת PHP ומנווה Zend Engine (קוד פתוח). המשימה של Zend היא להביא את הדור הבא של המוציאים והשירותים לפיתוח, פרישה וניהול של אפליקציות PHP ברמה ארגונית (Enterprise).

C.4 Zend Studio Client

Zend Studio מפשט את MERCHANTABILITYS הפיתוח הכרוכות ביצירת אפליקציות PHP:

- **פיתוח:** השלמת קוד מתקדם, ניהול פרויקטים והdagשת תחביר.
- **ניפוי שגיאות (Debugging):** מנפה שגיאות מרחב המאפשר העבודה ישירות מול השרת, ומנפה שגיאות פנימי למחשב המקומי.
- **ניהול:** כלי אבחון מתקדמים לניהול מחזר חי' התוכנה.

נספח ג

נספח C: מדריך התחלת מהירה ל-Zend Studio

C.1 גרסה 3.x

המידע במסמך זה עשוי להשתנות ללא הودעה מוקדמת ואינו מהווה התחייבות מצד חברת Zend Technologies, Ltd. אין לשכפל או להעביר שם חלק ממדריך זה בכל צורה או אמצעי לכל מטרה שאינה לשימושו האישי של הרוכש, ללא אישור בכתב מהחברה. כל הסימנים המסחריים שייכים לבנייהם.

C.2 אודוט המדריך

מדריך זה נועד לסייע לך להתחיל לעבוד עם התוכנה באופן מיידי. למידע מלא על התוכנות הנთמכות, יש לעיין בעזרה המקוונת (Online Help) המצורפת לאפליקציה.

C.3 אודוט Zend

בפשתות, Zend היא חברת ה-PHP. מייסדה—אנדי גוטמןס וזאב סורסקי—הם היוצרים והמצדדים של שפת PHP ומנווה Zend Engine (קוד פתוח). החברה שמה לה למטרה להביא את מוצרי הפיתוח והניהול

של PHP לרמה ארגונית (Enterprise). כיום, PHP היא שפת הסקרייפט הפופולרית ביותר ברשת, ומונע ה-eCommerce ממעלה מ-15 מיליון אתרי אינטרנט.

C.4 Zend Studio Client

סביבת Zend Studio עוצבה עבור מפתח ה-PHP המקצועי. זהה סביבת הפיתוח (IDE) היחידה המקופה את כל הרכיבים הדרושים למחזור החיים המלא של אפליקציית PHP. היא מסייעת בהאצת תהליך הפיתוח ומוניבה קוד חסין ובטוח.

התוכנה מפשטת שימושות כגון:

- **פיתוח:** השלמת קוד (Code Completion) מתקדמת, בוחני קוד (Inspectors) בرمת הקובץ והפרויקט, וחיפוש רוחבי בין קבצים.
- **ניפוי שגיאות (Debugging):** מנפה שגיאות מרוחק (Remote Debugger) לעבודה מול שרת, ומונפה פנימי לעבודה מקומית.
- **ניהול:** כלי אבחון מתקדמים כגון Profiler-Code Analyzer.
- **פרישה (Deployment):** פרטום האפליקציה לשרת אירוח באמצעות FTP/SFTP או אינטגרטיב CVS עצמאית.

C.4.1 רכיבי המערכת

המערכת מורכבת משני רכיבים עיקריים:

1. **Zend Studio Client:** ממשק המשתמש המותקן במחשב המקומי. כולל את סרגל הכלים לדפסר, מדריך ה-PHP ורכיבי ניפוי שגיאות ל-4.5 PHP.
2. **Zend Studio Server:** מוסף יכולות ניפוי שגיאות ו-Profiling לשרתים קיימים, ומאפשר הקמת שרת אינטרנט מבוסס Apache ו-PHP במידת הצורך.

C.5 עriticת קובץ

כדי לעורך קובץ, פשוט פתח את התוכנה והתחל לכתוב. התוכנה כוללת תוכנות חוסכות זמן כמו **השלמת קוד (Code Completion)** המאפשרת אפליקציות ל-PHP או HTML באופן אוטומטי. למשל, הקלדת התו < תפתח רשימת תגיות HTML לבחירה.

C.6 עבודה עם פרויקטים

ניהול פרויקטים מאפשר לך לארגן את הקבצים שלך בקלות:

1. עבור אל Project > New Project.
2. הzin שם לפרויקט.
3. הוסף נתיבים (Paths) לקבצים ולספירות שירכיבו את הפרויקט.
4. הגדר את הגדרות ניפוי השגיאות (Debug Mode) – מקומי או מרוחק.

C.7 הרצת מנפה השגיאות (Debugger)

התוכנה תומכת בשני סוגי ניפוי:

- **Internal Debugger (פנימי):** לניפוי אפליקציות PHP עצמאיות (דורש התקנת לkom בלבד).
- **Remote Debugger (מרוחק):** לניפוי קבצים הרכבים על שרת אינטרנט מרוחק.

C.7.1 ניפוי שגיאות פנימי - שלבי עבודה:

1. פתח את הקובץ המבוקש (למשל DebugDemo.php).

2. לחץ על כפתור **Debug** ברגל הכלים. המערכת תעצור בנקודות העצירה (Breakpoint).
3. השתמש ב-**Step Over** כדי להתקדם עוד משלבים אחד אחר שורה.
4. עמוד עם הסמן מעל משתנים כדי לראות את ערכם (ToolTip).
5. השתמש ב-**Step Into** כדי להיכנס לתוך פונקציות, וב-**Step Out** כדי לצאת מהן חזרה לקוד הראשי.
6. התוצאות יופיעו בחלון **Output** והודעות מערכת יופיעו בחלון **Debug Messages**.

C.7.1 מנפה שגיאות פנימי (Internal Debugger)

השתמש בתיבת הדו-שיח "Tip of the Day" (טיפ היום) כדי לגשת לקוד לדוגמה ולהסביר קצר על ניפוי שגיאות:

1. הפעיל את Zend Studio Client; לחופין, בחר בתפריט **Help > Tip of the Day**.
2. מתוך תיבת הדו-שיח "Tip of the Day", Tip of the Day, לחץ על הכפתור המתאים. הקובץ `DebugDemo.php` יפתח בחלון העריכה.
3. ברגל הכלים של הלקוח, לחץ להפעלת מנפה השגיאות. האיקון יופיע בזמן שירות הניפוי של Zend רץ, וישאר על המסך עד שמנפה השגיאות יזהה נקודות עצירה (breakpoint) בשורה 46.
4. לחץ על **Step Over** (דילג מעל) מספר פעמים עד שהסמן יגיע לשורה 51.
5. הנה והזק את הסמן מעל המשתנים `$worker_name`, `$worker_address` ו-`$worker_phone`. יופיע ToolTip (חלונית צפה) המציג את ערכי המשתנים.
6. לחץ על כפתור **Step Into** (כנס פנימה). מנפה השגיאות יתקדם לשורה 26.
7. בחלון הניפוי (Debug), לחץ על לשונית **Stack** (מחסנית) ולאחר מכן ל-`row_color`. עץ מחסנית הקרייאוט יתרחב ויציג את המשטנה.
8. לחץ על כפתור **Step Out** (צא החוצה). הסמן יחזור לשורה 51.
9. לחץ על כפתור החץ ימינה. פלט יופיע בחלון **Output**; הודעה **Notice** תופיע בחלון הודעות הניפוי (**Debug Messages**).
10. בחלון הודעות הניפוי, לחץ פעמיים על ה-**Notice**. הסמן יקופץ לשורה 61 בחלון העריכה.
11. הנה את הסמן בחלון **Output**, לחץ על הלחוץ הימני ובחר **Show in Browser** מהתפריט. יפתח חלון דףHTML המציג את תוכן חלון הפלט.

C.7.2 מנפה שגיאות מרוחק (Remote Debugger)

מנפה השגיאות המרוחק דומה מאוד בתכונותיו למנפה הפנימי, פרט לכך שהקוד מבוצע על שירות אינטרנט מרוחק. אם ברצונך לנפות אפליקציית רשת טיפוסית מבוססת דףHTML, עיין בסעיף "Debug URL" בהמשך הנוסף.

כדי להשתמש במנפה המרוחק, יש להגדיר תחילת את הלקוח (Client) ואת השירות.

להגדרת הלקוח:

1. מהתפריט הראשי, בחר **Tools > Preferences**. חלון ההעדפות יופיע.
2. בחר בלשונית **Debug**.
3. באפשרות **Server/Internal**, בחר במצב **ניפוי (Project Properties)**.
4. לחץ על **OK**.

כעת ניתן לנפות את הקובץ הנוכחי באמצעות המנפה המרוחק.

הערה: ניתן להפעיל/להשבית ניפוי מרוחק גם מחלון **מאפייני הפרויקט (Project Properties)**.

C.7.3 ניפוי כתובת אתר (Debug URL)

אפשרות זו מאפשרת להריץ הליך ניפוי על דפים המותקנים כתע באתר האינטרנט. ניתן לאותחל את סשן הניפוי מהלך על ידי בחירה בתפריט 'Debug URL' או דרך סרגל הכלים של Zend בדף.

שרת Zend Studio נותן עדיפות עליונה לקבצים שעליהם אתה עובד. לשם כך, השרת פועל לפי ההיררכיה הבאה בעט בקשת קבצים:

1. בודק אם הקובץ שנקרא פתוח כתע בלקוח; אם נמצא, הוא משתמש בו.
2. מhapus את הקובץ בנתיב של הפרויקט הפתוח.
3. מhapus את הקובץ בנתיב השרת.

היררכיה זו מאפשרת לך להימנע מהעלאת (Upload) הגרסאות האחרונות שלך בכל פעם. לדוגמה, אם אתה גולש באתר ומגלה דף פגום, תוכל להתחיל ניפוי שירות מהדף. לאחר התיקון, תוכל להריץ ניפוי חדש ולראות את התוצאה ללא צורך בהעלאת הקבצים ששונו תחיליה.

C.8 הגדרת שרת Studio עבור ניפוי ופרופילינג מרוחקים

משמעותי אבטחה, על המשתמש להיות מוגדר כמשתמש מורשה ב-IP Zend Server Center. רק כתובות IP מורשות יכולות לגשת לשירותים אלו.

להגדרת משתמש מורשה:

1. התחבר ל-Zend Server Center כמנהל (Administrator) מכותבת IP מורשתית.
2. פתח את מסך הגדרות האבטחה (Security Settings).
3. בלשונית **Manage IP Permissions**, הוסף את כתובת ה-IP המבווקשת לרישימת המארחים (Allowed Host List).
4. וודא שכותובת ה-IP אינה מופיעה ברשימה המארחים החסומים (Denied Host List).
5. לחץ על **OK**.
6. אתחול מחדש את שרת האינטרנט.

C.9 הרצת פרופילר (Profiler)

הפרופילר המשלב עוזר לאופטימיזציה של ביצוע האפליקציה על ידי איתור "צוארי בקבוק" (bottlenecks) – קטעי קוד הצורכים זמן טעינה מופרז.

הפרופילר מבצע את הפעולות הבאות:

- ניתוח קריאות לפונקציות ומספר הפעמים שקטע קוד בוצע.
- חישוב זמן הביצוע הכלול והפקת דוחות מפורטים.
- תצוגה גרפית של חלוקת הזמן והשוואה סטטיסטית בין פונקציות.
- הציג המבנה היררכי של הפונקציות שהשתתפו בהרצת הסקריפט.

להרצת פרופילר:

1. מתפריט **Tools**, בחר **Profile URL**, אשר את כתובת האתר המוגדרת כברירת מחדל או שנה אותה ולחץ על **OK**. הדף יציג את הדף,
2. ולאחר מספר שניות יופיעחלון מידע של פרופילר עם שלוש לשוניות: **Profiler Information** (מידע כללי ותרשים עוגה), **Function Statistics** (רשימת קבצים וסטטיסטיקת פונקציות), ו-**Call Trace** (תצוגה היררכית של סדר הפעולות).

C.10 תמייה ב מוצר

Zend מחויבת לספק שדרוגים ותמיכה. ברכישת מוצר, מקבלים 60 ימי תמיכה חינם להתקנה ולהגדרה. תמיכה משודרגת (Enhanced Product Support) זמינים במינוי שנתי וכוללת שדרוגים משמעותיים ומענה בעדיות גבוהה. שעות הפעילות הן שני עד שישי (GMT+2).

ניתן לקבל תמיכה דרך ה-[Helpdesk](#) באתר החברה, דרך תפריט העזרה בתוכנה, או דרך הפורומים של Zend.

C.11 תוכנות עיקריות

Zend Studio משלב את כל הכלים הדרושים בממתק מאוחד:

- ניפוי שגיאות (Debug) ו Oprfiling.
- אינטגרציית CVS להשוואת קבצים וניהול גרסאות.
- ניהול פרויקטים לחיפוש ונימוס קל.
- השלמת קוד (Code Completion) ובנייה קוד (Code Templates) מתקדמות ל-5 PHP.
- הדגשת תחביר (Syntax highlighting) ל- JavaScript ו-PHP, HTML וכו' זמינה.
- עריכה ופריסה חלקה מול שרטתי FTP.
- **ניתוח קוד סטטי:** איתור בעיות באפליקציה עוד לפני הרצה!
- ניפוי ו Oprfiling שירות מהדפן, גם עבור טפסים מורכבים או אפליקציות מבוססות Session.

להלן תרגום האינדקס לעברית:

סמלים

!#	(hash-bangs), סקריפטים של PHP בממתק שורה (CLI), 511–512	•
? (סימן שאלה),	39	•
& (סימן reference - התיחסות),	8	•
@ (אופרטור השתקה),	39	•
' (מרקאות יחידות),	20	•
configure,	486/	•
COOKIE,	115_	•
GET,	115_	•
POST,	115_	•
key,	26\$	•
this\$		•
properties),	62–59	○
gettodot(),	59–61	○
method(),	65–64	○
attribute(),	64–62	○
PHP 5,	437–440	○
type,	130\$	•
value,	26\$	•
autoload(),	7, 80, 82, 197	•
call(),	87, 109	•

- `()construct`, 3, 57 (בנאי), 3, 57
 - `()destructor`, 3 (מפרק), 3
 - `toString()`, 76–77, 76–77
 - `I18N (ビナオム)`, 567, 567
 - `I18Nv2`, 567, 567
-

A

- **a**, 366–366
- **ab** (כלי Apache Benchmarking), 457–458
- **abstract** (מודול מופשט), 616–615
- **abstract classes**, 5, 72–73
- **מחלקה מופשטת (abstract classes)**, 5, 72–73
- **מתודות מופשטות**, 6, 73–72
- **abstracted errors**, PEAR DB, 186
- ○ קודי שגיאה, 186–187
- ○ טיפול בשגיאות, 187
- **Acceleration Mode (ZPS accessing)**, 616–617
- ○ אלמנטים במערך, 24
- ○ קבצים, 261
- ○ פונקציות, 264–262
- ○ `this`, 59, 61–62
- ○ `this`, 59–61
- ○ `this`, 64–65
- ○ `this`, 62–64
- ○ ערכי סטטיות, 500–501
- ○ **הוספת הנחיותINI להרחבות**, 504–503
- **APEX** (מערכת חשיפה לצילום), 327
- **addslashes()**, 128
- **APC** (Advanced PHP Cache), 470, 530
- **APD** (Advanced PHP Debugger), 461, 588
- ○ ביתוח נתונים מעקב trace data), 462–465
- ○ התקינה, 462–461
- ○ **API**, 422
- ○ `extension API` (הרחבה), 490
- ○ `reflection API` (שיקוף), 105–103
- ○ עטיפת הרחבות צד-שלישי, 493
- ○ **ארQUITטורה (architecture)**, 493
- ○ סקריפט אחד לכל פונקציה, 144
- ○ סקריפט אחד משרת את כלם, 143
- ○ הפרדת לוגיקה מעיצוב layout), 144–146
- ○ **Archive_Tar**, 548
- ○ **Archive_Zip**, 548
- ○ **arguments**, 487
- ○ **מערכות arrays**, 23

- שינוי/יצירת אלמנטים, 25
 - אינדקסים במערך, 23
 - קריית ערכים, 25
 - מעבר על מערך (traversing), 30
 - מערכים אוטומטיים, 88
 - **ArrayAccess** (ממשק), 88
 - **הקצתה \$this** ב-**PHP 5**, 437, 439–440
 - **אופרטורים של הקצאה**, 33–32
 - **atomic**, 278
 - **מאפיינים** (בנושא **attributes**), 221
 - **Auth** (אימות), 392, 398
 - DB ונתוני משתמשים, 394–396
 - קבצי סיסמות, 394–393
 - אבטחה, 396
 - **authentication**, 527–530
 - **author** (מחבר), 617
 - **אופטימיזציה אוטומטית**, 472–470
-

B

- **B**, 365–366
 - **תאינות לאחר** (backward compatibility), 408
 - **תרשיימי עמודות** (bar charts), 320–325
 - **Benchmark** (מבחן ביצועים), 457, 530
 - **ApacheBench**, 457–458
 - **binaries**, 353
 - **BLOB** (אובייקטBINARIE גודול), 158
 - **օպרטורים ביטוויזיים** (bitwise), 35
 - **blocks**, 384–387
 - **Booleans** (בוליאנים), 22
 - **טפסי הגשה חסיני בוטים**, 320–315
 - **bottlenecks**, 459–460
 - **צוארי בקבוק** (break), 459–460
 - **break**, 43
 - **בנייה חבילות** (building packages), 411–415
-

C

- **C / -c**, 356–357
- **C (שפה תכנות)**
 - הכללה בחבילות PEAR, 428
 - כתיבת הערות, 14
 - כתיבת הערות, 14
- **++C**
- ירושא, 70
- כתיבת הערות, 15
- **Cache** (מטמון), 451, 453–453
- **APC**, 470, 530

- Cache_Lite, 399–401, 531 ○
- מיטמן שאילותות מסד נתונים, 455–453 ○
- מיטמן פלט (output caching), 456 ○
- regnisot לאותיות רישיות (Calendar (לוח שנה)), 542 ●
- אופרטורים של המרה (case sensitivity), 30 ●
- תפיסת שגיאות (catching errors), 207, 216–218 ●
- המרה בין סטימ של תווים (character set conversions), 330–335 ●
- מחלקות (classes), 56, 422 ●
 - הצורה, 57 ○
 - מחלקות סופיות (final), 4, 76 ○
 - חברים סטטיים, 5 ○
- CLI (ממשק שורת פקודה), 507 ●
- ניתוח אפשרויות (parsing), 512–515 ○
- סביבת סקריפטים, 512–510 ○
- שיבוט אובייקטים (cloning), 4, 66–67 ●
- הערות (comments), 14, 613–615 ●
- תאיימות (compatibility), 433–445 ●
 - מצב תאימות (compatibility mode), 435 ○
 - קבועים (constants), 30–31 ●
 - constructors), 57–58, 442 ●
 - עוגיות (cookies), 131–133 ●
 - XML), 239–243 XSLT ●

D

- טיפול נתונים (data handling), 301 ●
- תאריכים (dates), 301–305, 542 ●
- DB (שכבות הפשטה מסד נתונים), 534–533 ,176 ●
- ניפוי שגיאות (debugging) ○
 - Zend Studio Debugger, 648 ○
 - הצורה (declaring) על מחלקות, 57 ●
- תלות (dependencies) בPEAR, 423–426 ●
- tabniot Uizob (design patterns), 94–109 ●
- מפרקם (destructors), 58–59 ●
- מדרייכים/ספריות (directories), 353, 359–360 ●
- DOM (מודול אובייקטי של מסגר), 231–222 ,9 ●
- DSN (שם מקור נתונים), 178 ●

E

- 441 ,202–201 (רמות שגיאה), ...E_ERROR, E_NOTICE, E_STRICT ●
- עריכת קבצים, 647 ●
- דואר אלקטרוני (email), 418 ●
- הטמעה (embedding) של PHP ב-HTML, 14, 112–114 ●
- הצפנה (encryption), 124–127, 543–544 ●

-
- **שגיאות (errors)**, 191
 - **שגיאות תחביר**, 192
 - **שגיאות זמן ריצה**, 201
 - **שגיאות לוגיות**, 197
 - **חריגות (exceptions)**, 216–218

F

- **תבנית פקטורי (factory pattern)**, 97–101
- **מערכת קבצים (file systems)**, 548–549
- **קבצים (files)**, 261
 - **נעילת קבצים**, 277–276
 - **העלאת קבצים**, 142–137
- **פליטרים (filters)**, 127
- **פונקציות (functions)**, 48, 293
 - **פונקציות מובנות**, 456
 - **פונקציות מוגדרות משתמש**, 49–48

G

- **GD (ספריות גרפית)**, 325–314
- **משתנים גלובליים**, 502–501, 118–117
- **גרפיקה**, 329–314

H

- **HTML**, 550–560
 - **ניהול מאפיינים**, 390
 - **ניהול אלמנטים**, 391
- **HTTP**, 561–563
- **HTTPS**, 397

I

- **iconv (הרחבה)**, 334
- **תמונות (images)**, 563–566
- **ירושא (inheritance)**, 68–75
- **התקינה (installing)**
 - **PEAR**, 350–354
 - **Zend Studio**, 645–647
- **ממשקים (interfaces)**, 4, 55, 74–75
- **בינאום (internationalization)**, 566–568
- **迭代器 (iterators)**, 89–94, 170–172

להלן תרגום האינדקס לעברית:

J–K ##

-JPEG**, בהשוואה ל-
PNG**, 320**
kadm5**, 574**
, (KCachegrind** (Zdebug**
470–468
HMAC לאימות הודעות. ראה **Keyed-Hashing**
**מילות מפתח (keywords)
abstract**, 615–616**
access**, 616–617**
author**, 617**
category**, 618**
copyright**, 618**
deprecated**, 618**
example**, 619**
filesource**, 620**
final**, 4, 620–621**
global**, 621**
ignore**, 622**
inheritdoc** (inline), 622**
internal** (inline), 622–623**
licence**, 623**
link**, 623**
link** (inline), 623**
name**, 624**
new**, 57–58**
package**, 624, 626**
param**, 626–627**
return**, 627**
see**, 627**
since**, 628**
static**, 628**
staticvar**, 629**
subpackage**, 629**
todo**, 630**
uses**, 630–631**
var**, 631**
version**, 631**
Knowles, Alan**, 613**

L ##

תכונות שפה (language features)

ערכות תווים, 329
 מודל מונחה עצמים, 5–1
 שכבות (layers), 360
 פריסה (layout), הפרדה מה-
 לוגיקה, 146–144
 התאמה עצלה (lazy matching), 288–289
 lead**, 418**
 ספריות (libraries)
 ספריות GD**. ראה ספריות GD
 libxslt**, 239**
 libsxlt**, 239**
 ספרית licence**, 623**
 licence**, 623**
 אלמנט license**, 418–419**
 מגבילות של PHP 3 ו-4, 2, 239**
 limitQuery** (PEAR DB), 180–181**
 link**, 623**
 link** (inline), 623**
 PEAR**, 350**
 הפקת Linux PHP, התקנת (list**, 28–30)**
 LiveUser**, 529**
 איזון עומסים (load-balancing), 623**
 מזהה שון (session id), 398
 נעילת קבצים, 277–276
 Log**, 568**
 log_errors**, 130**
 logging**, 568**
 203 (בוליאני), 203
 203 (שם), log_errors_max_len**
 Log_Parser**, 568**
 רישום לוגים (logging)**
 לוגיקה**, הפרדה מה-
 פריסה (layout), 144–146
 שגיאות לוגיות, 197
 34–35 אופרטורים לוגיים**
 טפסי התחברות**, 399
 מבנה בקרת לולאה**
 לולאות do.. while**, 43–44
 לולאות for**, 44–45
 לולאות while**, 42**
 לולאות, בקרה**, 43
 lowercasing**) (PEAR DB), 186
 הפיכה לאותיות קטנות (lowercasing)** (PEAR DB), 186

M ##

**מакראים (macros)
 מאקראים להצרת פונקציות, 495

פרמטרים של מאקרו^{**}, 503^{**}
TSRM^{**}, 504–505^{**}
VCWD^{**}, 496^{**}

ארגומנטים של מאקרו^{**}, 498^{**}
ZEND_INIT_MODULE_GLOBALS^{**}, 502^{**}

מакראים לגישה ל-
magic quotes^{**}, 198^{**}
magic_quotes_gpc^{**}, 198^{**}
magic_quotes_runtime^{**}, 198^{**}

Mail^{**}, 569^{**}

Mail_IMAP^{**}, 569^{**}

Mail_Mbox^{**}, 569^{**}

Mail_Mime^{**}, 570^{**}

Mail_Queue^{**}, 570^{**}

mailparse^{**}, 569^{**}

POP3^{**}, 570^{**}

vpopmail^{**}, 571^{**}

maintainers^{**}, 418^{**}

ארסה ראשית לעומת משנית לעומת רמת תיקון^{}, 349

ניהול משתנים^{}

empty^{**}, 17^{**}

isset^{**}, 16–17^{**}

unset^{**}, 17^{**}

מניפולציה של גרפייה^{}

תרשיים עמודות, 325–320

טפסי שליחה חסיני בוטים, 320–315

exif^{**}, 326–329^{**}

, **(markup format)

HTML_Template_Flexy^{**}, 390^{**}

פונקציות התאמה^{}, 293, 295

מתמטיקה^{}

Math_Base^{**}, 571^{**}

Math_Complex^{**}, 571^{**}

Math_Fibonacci^{**}, 571^{**}

Math_Histogram^{**}, 571^{**}

Math_Integer^{**}, 572^{**}

Math_Matrix^{**}, 572^{**}

Math_Numerical_RootFinding^{**}, 572^{**}

Math_Quaternion^{**}, 573^{**}

Math_RPN^{**}, 573^{**}

Math_Stats^{**}, 573^{**}

Math_TrigOp^{**}, 573^{**}

Math_Vector^{**}, 574^{**}

mb_convert_encoding^{**}, 333()^{**}

mb_strlen^{**}, 335–337^{**}

mb_strpos^{**}, 340()^{**}

mb_substitute_character^{**}, 332()^{**}

mb_substr^{**}, 340()^{**}

mbstring**, 330 הרחבות
mbstring**, 331 פלט **phpinfo() של mbstring**, 333(**)
PEAR**, 430 אוצר החבילות של MD5 checksum**
md5sum**, 421 תכונת
MDB**, 538–539 MDB**
MDB_QueryTool**, 541**
MDB2**, 539–540**
mdbtools**, 541**
משתני מחלקה (member variables), **(member variables)
שים סמלים (PCS**), 406**
memcache**, 589**
זיכרון (** זיכרון
מנהל זיכרון חדש, 11
 כתיבת הרחבות, 489
Message**, 544**
XML RPC)**, 244–245
metacharacters)**, 280, 283, 285)**
תווים מיוחדים (methods)
מתודות (**
toString**, 76–77 __()
abstract**), 6, 72–73**
מתודות מופשטות (abstract methods), 6, 72–73**
גישה באמצעות this\$\$, 59, 61–62**
call**, 253()**
connect**, 62**
createAggregate**, 174()**
createDbConnection**, 62()**
draw**, 72()**
מתודות סופיות (final**), 75–76**
getAll**, 189()**
getAssoc**, 188–189()**
getAttribute**, 228()**
getCol**, 188()**
getName**, 56()**
getOne**, 188()**
getRow**, 188()**
hash**, 125()**
E_STRICT**, 442–443**
outputObject**, 388()**
overloading)**, 85–87**
arity מетодות (arity methods), 85–87**
METHOD POST**, 115**
raiseError**, 207**
service**, 255()**
setDbConnection**, 62()**
setName**, 56()**
singleQuery**, 168()**
arity מетодות סטטיות, 5
גישה באמצעות this\$\$, 64–65**
שים סמלים (PCS**), 405**

מיקרו- מבחני ביצועים (micro-benchmarks), 477–479
 microtime**, 302()
 טיעויות בהגנה על קלט משתמש (XSS), 117–119
 סקורייטים חוצי-אטררים (**SQL**), 119–120
 משתמשים גלובליים, 118–119
 הזרקת (**mktimes**), 303()
 modifiers)**, 289, 293
 **משנים (shiny), 25
 module_number**, 494
 mono**, 590
 move_uploaded_file**, 142()
 rename**, 278()(**הברחת קבצים** באתmüות פונקציית rename)
 MP3_ID**, 547
 mqseries**, 574
 ריבוי הצהרות (MySQL), 155–156
 multi-byte strings)**, 330
 מחוזות מרובות-בתים (MySQL), 149
 מערכים רב-מדים**. ראה מערכים מקוונים (**BLOB**
 הכנסת נתונים (**BLOB**), 159
 שליפת נתונים (**BLOB**), 159–160
 שאלות מאוחסנות (buffered), 153
 תאמות, 445
 חיבורים, 153–151
 נתונים דוגמה, 151
 מצב שיליפה (fetching modes), 156
 ריבוי הצהרות, 156–155
 משק (**PHP**), 150–151
 הצהרות מוכנות (prepared statements), 156–158
 שאלות, 155–154
 סקלריות, 150
 מהירות, 150
 נקודות חזק וחולשה, 150
 שאלות לא מאוחסנות (unbuffered), 154
 MySQLi**, 10
 פונקציות חיבור (**mysql**), 152
 mysqli**, 156
 פונקציות שיליפה (**mysqli**), 154
 mysqli_options**, 153
 קבועי (**name**), 624
 תכונת (name), 421–423, 428
 אלמנט (name), 417

N ##

name**, 624
 תכונת (name), 421–423, 428
 אלמנט (name), 417

מוסכמות שיקום, סמלים ב-403**, PCS**, 403**
מחלקות, 404
קבועים, 403
פונקציות, 404
משתנים גלובליים, 404
משתני מחלקה, 406
מתודות, 405
אופרטורים של שלילה, 36
מערכות מוקונים, גישה, 26
 Net_CheckIP**, 575**
 Net_Curl**, 575**
 Net_Cyrus**, 575**
 Net_Dict**, 575**
 Net_Dig**, 575**
 Net_DIME**, 576**
 Net_DNS**, 576**
 Net_Finger**, 576**
 Net_FTP**, 576**
 Net_GameServerQuery**, 576**
 Net_Geo**, 577**
 Net_Gopher**, 577**
 Net_Ident**, 577**
 Net_IMAP**, 577**
 Net_IPv4**, 577**
 Net_IPv6**, 578**
 Net_IRC**, 578**
 Net_LDAP**, 578**
 Net_LMTP**, 579**
 Net_NNTP**, 579**
 Net_Ping**, 579**
 Net_POP3**, 579**
 Net_Portscan**, 579**
 Net_Server**, 580**
 Net_Sieve**, 580**
 Net_SmartIRC**, 580–581**
 Net_SMTP**, 582**
 Net_Socket**, 582**
 Net_Traceroute**, 582**
 Net_URL**, 582**
 Net_UserAgent_Detect**, 582**
 Net_UserAgent_Mobile**, 583**
 Net_Whois**, 583**
 netools**, 574**
 רשות
 cvsclient**, 574**
 cyrus**, 574**
 kadm5**, 574**
 mqseries**, 574**

* (רשימת ספריות Net דומה למופיע לעיל)
 opendirectory**, 583**
 tcpwrap**, 584**
 uuid**, 584**
 yaz**, 584**
 new**, 57–58
 nextId**, 185**
 nobuild**, 365--**
 nocompress**, 365--**
 notes**, 420**
 אלמנט null**, 23**
 מספרים
 מספרים נקודה צפה, 19
 Number_Roman**, 584**
 Number_Words**, 585**
 אופרטורים נומריים, 32

O

** o-**, 366**
** שכפול עצמים (Object Cloning)**, 4
** מודל אובייקטים (PHP 5**), 433**
** תכונות מונחות עצמים**, 7–3
** אובייקטים (objects)**, 55
המרה (casting) (מצב תאימות), 436–435
 מחלקות, 56
 שכפול, 4, 67–66
 השוואה (מצב תאימות), 437–436
 יצירה אוטומטית, 441
 איטרטורים, 94–89
 עברית לפונקציות, 435–433
 SimpleXML
 דף, 234–233
 יצירה, 233–232
 אחסון, 234
** תבנית צופה (observer pattern)**, 101–103
 PEAR DB**, 176
 השגת oci8**, 541**
 odbtp**, 541**
 oggvorbis**, 595**
 OLE**, 596**
** סקרייפט אחד לכל פונקציה**, 144
 סקרייפט אחד לכלם**, 143
 onlyreqdeps**, 366--**
** מודל OO (מנחה עצמים)**, 2–1
 tabנויות עיצוב, 94

tabaniot_mapul (**), 97–101**
tabaniot_zofha (**), 101–103**
tabaniot_singlton (**), 97–98**
tabaniot_astratgia (**), 95–96**
leumot_kod_frozdorli, 481–480
**yshomi, 70, **OO, 456
**muftot_OO_lfonkzit_movenot, **
**tcanot_moncha_uzchim, 55
polimorfizm, 69–67
opcode_cache (**), 470
matmon_aopkod (, 595**
openal (**, 583**
**tagiot_ftihah, 221
OpenSSL, 272**
talot_bmercat_hafula (, 425**
**habdli_murcat_hafula, **, shagiot_niyot, 197
**aoptrorim, 31
aoptrorim_shl_shema, 33–32
aoptrorim_binarim, 32–31
aoptrorim_ull_sibiot (**bitwise (**), 35**
aoptrorim_shl_harma (**cast (**), 38**
aoptrorim_shl_shwoah, 34–33
aoptrorim_shl_kidom_pichot, 38–37
instanceof (**, 4, 71**
aoptrorim_logim, 35–34
aoptrorim_shl_shilah, 36
aoptrorim_shl_shatka, 39
aoptrorim_trneri, 39
aoptrorim_onarim, 36
**aoptimizha, **, 459
kod, 477
mikro-mbchni_bitzuim, 479–477
OO_leumot_kod_frozdorli, 481–480
crtiba_mhdash_b-C, 479–480
**tcanot, 424
**tagiot_aopcionalit, **, 366
htknt_chbilot_itn, 366
PEAR, 426**
**afshrioit, **
aoptimizha, 509
INI, 14**
**short_open_tags, **-b-
**output_caching, 456
matmon_plat (Cache_Lite (**, 400–401**
**matoda, ()
outputObject (, 388)
overloading (, 85)
tchbir_gisha_lmurk, 89–88
aitriza, 90
matodot, 87–85

P ##

- **צינורות (**pipes**), 266
- package**, 624, 626**
- package**, 417**
- אלמנט (**element**), 417
- מידע על חבילה (**package.xml**), 417–419
 - package.xml**, 416, 431**
 - מידע על חבילה, 419–417
 - מידע על גרסה הפעה (**release**), 419–422**
 - חבילות (**packages**)
- בנייה
 - EAR** (**HelloWorld**), 411–414**
 - בדיקות רגרסיה (**PEAR**), 416**
 - ארכינוי (**PEAR**), 414**
 - אימות (**PEAR**), 414–415**
 - Cache_Lite**, 399–401**
 - התקינה, 354
 - עם תלויות אופציונליות, 366
 - עם פקודות (**pear**), 354–357**
 - PEAR**, 346**
 - שחרור חבילות, 428
 - תהליך השחרור של (**PEAR**), 430**
 - אריזה (**packaging**), 430**
 - MD5 checksum**, 430**
 - package.xml**, 431**
 - ניתוח מקור, 430
 - ארכינוי (**tar**), 431**
 - Pager**, 560**
 - Pager_Sliding**, 560**
 - panda**, 599**
 - Paradox**, 542**
 - התקנות PEAR מקבילות**, 362
 - param**, 626–627**
 - פרמטרים**
- رمזים טיפוס מחלוקת (**type hints**)** בפרמטרי פונקציה, 82–83
- הצהרה על פרמטרי פונקציה, 52
 - parent**, 70**
- שגיאות ניתוח (**parse errors**), 192–194
 - ניתוח (**parsing**)
- אפשריות שורת פקודה, 515–512
 - פורמטי תאריך, 314–313
 - XML**, 222**
- DOM**, 226–229**
- SAX**, 222–226**

מטען חלק, 475
עברית

אובייקטים לפונקציות, 435–433
לפי ייחוי (**), 52**
לפי ערך, 52

קבצי סיסמה, 394–394**
סיסמות, הגנה על סקריפטים, 129–127
תחבר תבנית, 280

tabnios (patterns), 279. ראה גם *tabnios* עיצוב
**תשלומים (payment)
cybercash**, 585**
cybermut**, 585**
Payment_Clieop**, 586**
Payment_DTA**, 586**
Payment_Process**, 586**
spplus**, 586**
TCLink**, 586**
פלט (phpinfo() של PCRE, 279
תקן הקידוד של PEAR (ר"ת PCS), 403
indentation), 406–407
שים סמלים, 403

PEAR (PHP Extension and Application Repository)**, 200–201, 345, 587)**
Auth**, 392, 398**
נתוני משתמש (DB)
קבצי סיסמה, 394–394
394–393
שיקול סקלביות, 398–397
שיקול אבטחה, 397–396
בנייה חבילות, 414–411
матмо, 399
סביבת**, 408–410**
Crypt_HMAC**, 124–127**
תלוויות, 424–423
טיפול בשגיאות, 212, 207–206, 411
תגיית פתיחה, 113
מנהל חבילות (Installer), 354
pear**, 354–357, 363
PEAR DB**, 176**
שגיאות מופשיות, 186
חיבורים, 179–178
מתודות נוחות, 188
ביצוע שאלות, 180
שליפה תוצאות, 182
ニידות, 185
רצפים (sequences**), 184–185**
PECL (PHP Extension Community Library)**, 345**
PECL_Gen**, 601**
ביצועים (performance), 449, 456
APC**, 470**

מבחן ביצועים**), 457**)
מטען caching**), 451–453**
profiling**) עם APD**, 461** (**profiling**) עם Zend Studio**, 459–461** Profiler**) של מצב (state**), 450**
Zdebug**, 465**
profiling עם Zend Studio**, 459–461** Profiler**) של מצב (state**), 450**
ZPS**, 470**
PHP
apd**, 588**
bcompiler**, 589**
הטמעת קוד ב-HTML**, 112–114**
ffi**, 589**
Inline_C**, 589**
memcache**, 589**
mono**, 590**
perl**, 590**
PHP_CompatInfo**, 591**
PHP_Fork**, 591**
PHP_Parser**, 592**
PHPDoc**, 590**
PHPUnit**, 590**
python**, 592**
כלים
crack**, 600**
fann**, 601**
PECL_Gen**, 601**
PhpDocumentor**, 601**
SPL**, 602**
Valkyrie**, 602**
PHP 3, מגבלות, 2
PHP 4, מעבר ל-5**, PHP 5**, מצב תאימות, 2, 433
PHP 5**, 433**
שגיאות 201
מטפל שגיאות מותאמים אישית, 204
רמות שגיאה, 201–201
דיווח שגיאות, 202, 202
השתקת שגיאות, 206–206
MySQL
MySQL**, 150–151**
SQLite**, 162**
Sessions (sessions)
כל, 601, 613, 633–640
הפולימורפיזם polymorphism), 67–69
ניידות portability), 200
שגיאות ניידות, 200–197
כלי ניידות, 201–200
שאילתות מוכנות prepared statements), 156
קוד פרוצדורלי לעומת OO, 480–481

`proc_open`, 265
`profiling`, 459
הגנה על סקריפטים, 120
`provides`, 422
`public`**, `E_STRICT`**, 441**

Q–R

שאלות (queries)
MySQL)**, 153–155**
PEAR DB)**, 180–182**
SQLite)**, 162–165**
סימן שאלה (?)
מראות
כפולות (""), מחרוזות, 19
בודדות (''), מחרוזות, 20
`raiseError`**, 207, 209**
קריאת ערכי מערך, 25
`reflection`**, 103)
דוגמאות, 106–107
**API של רפלקציה, 105–103
`regular expressions`**, 279
ביטויים רגולריים (regular expressions), 279
פונקציות, 301–293
תחביר, 280–279
שחרור חבילות (PEAR), 428
נוקי/הסרת קבצים, 278–277
שימום חדש (rename), 278
דיווח שגיאות, 202, 204
בקשות (XML RPC), 245–246
משאבים (resources), 493–499
תגובה (XML RPC), 246
`return`**, 627**
הזרת ערכים מפונקציות, 490
לפי ייחוס, 51
לפי ערך, 50
`RSS`**, 235–236**

S

SAPI (Server API)**, 507**
SAX**, 9, 222–226**
קלbilità (scalability)
Auth**, 397–398**
MySQL**, 150**

AKERIPIM, הגנה, 120
 **בטיחה (security)
 Auth**, 396–397**
 הגנה על קלט משתמש, 120–117
 see**, 627**
 self**, 70**
 הפרדת לוגיקה מפרישה, 146–144
 **רצפים (sequences)
 PEAR DB**, 184–185**
 שירותים (XML RPC), 250–252
 sessions)**, 134–137, 450
 Sessions (sessions), 134–137
 תגיות קצרות (short tags), 113
 אופרטור השתקה, 39
 SimpleXML**, 9–10, 222, 231–234**
 SOAP**, 10, 252–259**
 SQLite**, 160–176**
 state)**, 450
 ** מצב (state)
 ** סטטי (static)
 חברים סטטיים, 62 ,5
 מתודות סטטיות, 5, 65–64
 מאפיינים סטטיים, 64–62
 משתנים סטטיים, 53
 תזרים (streams), 261–276
 מחרוזות (strings), 19–22, 38
 ** מבנים (structures)**, 596–597
 overloading)**, 85–87
 **דרישת מתודות/מאפיינים (overloading)
 syntax)**, 88, 279–280

T-Z

תגיות (tags), 113, 221, 615–632
 מערכות תבניות (template systems), 383–392
 בדיקות (testing), 416, 430, 459, 501
 טקסט (text), 325, 599–600
 זמן (time), 301–314
 אזור זמן, 312
 פורמט ISO 8601**, 309**
 כלי (tools). ראה גם כל שורת פקודה
 טרנץקיות (SQLite), 164–165
 עצים (trees), **DOM**, 229–231
 טריגרים (triggers), **SQLite**, 165
 טיפול בשגיאות (try/catch), 216–218
 TSRM**, 504–505**
 SQLite**, 165–168**
 פונקציות מוגדרות משתמש (UDFs)
 ** משתנים (variables)**
 גלובליים, 117 ,18

סוטרים, 53
ניהול, 17–16
גרסאות (versioning), 347–349
ווב (Web), 243, **(Web), שירותי ווב,
XML**, 219–241**
Zend**, 643–653**
Zend Engine**, 1**
Zend Studio**, 644–653**
zval**, 499–501**