Kifejezések és vezérlési folyamat PHP-ben

Az előző fejezet több témát is bemutatott, amelyekkel ez a fejezet részletesebben foglalkozik, mint például a választások meghozatala (elágazás) és összetett kifejezések létrehozása. Az előző fejezetben a legalapvetőbb szintaxisra és műveletekre akartam összpontosítani PHP-ben, de nem kerülhettem el a haladóbb témák érintését. Most már kitölthetem a háttérben, amelyre szüksége van ezeknek a hatékony PHP-funkcióknak a megfelelő használatához. Ebben a fejezetben alaposan megismerheti a PHP programozás működését a gyakorlatban és a program folyásának szabályozásában.

Kifejezések

Kezdjük minden programozási nyelv legalapvetőbb részével: a kifejezésekkel. A kifejezés értékek, változók, operátorok és függvények kombinációja, amely értéket eredményez. Mindenki számára ismerős, aki végzett középiskolai algebrával. Íme egy példa:

y = 3 (|2x| + 4)

PHP-ben ez lenne:

$y = 3 \* (abs(2 \* $x) + 4);

A visszaadott érték (y ebben a matematikai utasításban vagy $y a PHP-ben) lehet egy szám, egy karakterlánc vagy egy Boolean érték (a nevét George Boole-ról, egy 19. századi angol matematikusról és filozófusról kapta). Mostanra már ismernie kell az első két értéktípust, de a harmadikat elmagyarázom.

TRUE vagy FALSE?

Az alap logikai érték TRUE vagy FALSE lehet. Például a 20 > 9 (20 nagyobb, mint 9) kifejezés TRUE, az 5 == 6 (5 egyenlő 6) pedig FALSE. (Az ilyen műveleteket más klasszikus Booleanokkal is kombinálhatja, például AND, OR és XOR, amelyekről ebben a fejezetben később lesz szó.)

Vegye figyelembe, hogy nagybetűket használok az TRUE és FALSE nevekhez. Ez azért van, mert ezek előre definiált állandók a PHP-ben. Használhatja a kisbetűs változatokat, ha úgy tetszik, mivel ezek is előre meghatározottak. Valójában a kisbetűs verziók stabilabbak, mert a PHP nem teszi lehetővé ezek újradefiniálását; a nagybetűk újradefiniálhatók, amit érdemes szem előtt tartani, ha harmadik féltől származó kódot importál.

A PHP valójában nem nyomtatja ki az előre definiált állandókat, ha erre kéri, mint a 4-1. példában. A példa minden sorhoz kiír egy betűt, amelyet kettőspont és egy előre meghatározott állandó követ. A PHP tetszőlegesen 1-es számértéket rendel a TRUE-hoz, így az a: után 1 jelenik meg a példa futtatásakor. Még rejtélyesebben, mert a b: FALSE-ra értékeli, nem mutat semmilyen értéket. A PHP-ben a FALSE konstans NULL-ként van definiálva, egy másik előre definiált állandó, amely semmit sem jelöl.

Példa 4-1. A TRUE és FALSE értékek kiadása

<?php // test2.php  
echo "a: [" . TRUE . "]<br>";  
echo "b: [" . FALSE . "]<br>";  
?>

A <br> címkék sortörések létrehozására szolgálnak, és így a kimenetet két sorra osztják a HTML-ben. Íme a kimenet:

a: [1]  
b: []

A Boolean kifejezésekre térve, a 4-2. példa néhány egyszerű kifejezést mutat be: az előbb említett kettőt és még néhányat.

Példa 4-2. Négy egyszerű Boolean kifejezés

<?php  
echo "a: [" . (20 > 9) . "]<br>";  
echo "b: [" . (5 == 6) . "]<br>";  
echo "c: [" . (1 == 0) . "]<br>";  
echo "d: [" . (1 == 1) . "]<br>";  
?>

A kód kimenete a következő:

a: [1]  
b: []  
c: []  
d: [1]

Egyébként néhány nyelvben a FALSE 0 vagy akár –1 is lehet, ezért érdemes minden használt nyelven ellenőrizni a definícióját. Szerencsére a Boolean kifejezések általában más kódokban vannak eltemetve, így általában nem kell aggódnia amiatt, hogy az TRUE és a FALSE belül hogyan néz ki. Valójában még ezek a nevek is ritkán jelennek meg a kódban.

Pontosak és változók

Ezek a programozás legalapvetőbb elemei, és a kifejezések építőkövei. A pontos egyszerűen azt jelenti, hogy valami önmagát értékeli, mint például a 73-as szám vagy a "Hello" karakterlánc. Egy változó, amelynek már láttuk, dollárjellel kezdődő neve van, a hozzárendelt értékre értékelődik ki. A legegyszerűbb kifejezés csak egyetlen pontos vagy változó, mert mindkettő egy értéket ad vissza.

A 4-3. példa három pontosat és két változót mutat be, amelyek mindegyike értéket ad vissza, bár különböző típusúak.

Példa 4-3. Pontosak és változók

<?php  
$myname = "Brian";  
$myage = 37;  
echo "a: " . 73 . "<br>"; // Numeric literal  
echo "b: " . "Hello" . "<br>"; // String literal  
echo "c: " . FALSE . "<br>"; // Constant literal  
echo "d: " . $myname . "<br>"; // String variable  
echo "e: " . $myage . "<br>"; // Numeric variable  
?>

És ahogy az várható volt, ezekből mindegyikből visszatérési érték jelenik meg, kivéve a c:-t, amely FALSE-ra értékeli ki, és semmit sem ad vissza a következő kimenetben:

a: 73  
b: Hello  
c:  
d: Brian  
e: 37

Az operátorokkal együttműködve összetettebb kifejezések is létrehozhatók, amelyek hasznos eredményeket eredményeznek.

A programozók a kifejezéseket más nyelvi konstrukciókkal kombinálják, például a korábban látott hozzárendelési operátorokkal, hogy utasításokat alkossanak. A 4-4. példa két állítást mutat be. Az első a 366 - $day\_number kifejezés eredményét hozzárendeli a $days\_to\_new\_year változóhoz, a második pedig csak akkor ad ki barátságos üzenetet, ha a $days\_to\_new\_year < 30 kifejezés TRUE értéket ad.

Példa 4-4. Egy kifejezés és egy kijelentés

<?php  
$days\_to\_new\_year = 366 - $day\_number; // Expression  
if ($days\_to\_new\_year < 30)  
{  
echo "Not long now till new year"; // Statement  
}  
?>

Operátorok

A PHP számos hatékony, különböző típusú operátort kínál – aritmetikai, karakterlánc, logikai, hozzárendelési, összehasonlító és sok más (lásd 4-1. táblázat).

4-1. táblázat. PHP operátortípusok

Operátor Leírás Példa

Aritmetikai Alapvető matematika $a + $b

Sor Sor egyesítés $a + $b

Feladat Értékek hozzárendelése $a = $b + 23

Bitenként Manipulálja a biteket bájton belül 12 ^ 9

Összehasonlítás Hasonlítson össze két értéket $a < $b

Végrehajtás Végezze el a backticks tartalmát `ls -al`

Növekedés/csökkentés Összeadás vagy kivonás 1-el $a++

Logikus Boolean $a and $b

Karakterlánc Összefűzés $a . $b

Minden operátor különböző számú operandust vesz fel:

• Az unáris operátorok, mint például a növelés ($a++) vagy a tagadás (! $a), egyetlen operandust vesznek fel.

• A bináris operátorok, amelyek a PHP operátorok nagy részét képviselik (beleértve az összeadást, kivonást, szorzást és osztást), két operandust foglalnak magukban.

• Az egy hármas operátor, amely expr ? x : y , három operandust igényel. Ez egy szűkszavú, egysoros IF utasítás, amely x if m expr TRUE, és y if expr FALSE-ot ad vissza.

Operátor elsőbbség

Ha minden operátornak azonos prioritása lenne, akkor a feldolgozás abban a sorrendben történne, amelyben találkoztak. Valójában sok operátornak ugyanaz az elsőbbsége. Vessen egy pillantást a 4-5. példára.

Példa 4-5. Három egyenértékű kifejezés

1 + 2 + 3 - 4 + 5  
2 - 4 + 5 + 3 + 1  
5 + 2 - 4 + 1 + 3

Itt látni fogja, hogy bár a számok (és az azt megelőző operátorok) el lettek mozgatva, minden kifejezés eredménye a 7, mivel a plusz és mínusz operátorok elsőbbsége megegyezik. Megpróbálhatjuk ugyanezt a szorzással és osztással (lásd a 4-6. példát).

Példa 4-6. Három kifejezés, amelyek szintén egyenértékűek

1 \* 2 \* 3 / 4 \* 5  
2 / 4 \* 5 \* 3 \* 1  
5 \* 2 / 4 \* 1 \* 3

Itt a kapott érték mindig 7,5. De a dolgok megváltoznak, ha különböző precedensű operátorokat keverünk egy kifejezésben, mint a 4-7. példában.

Példa 4-7. Három kifejezés vegyes prioritású operátorok használatával

1 + 2 \* 3 - 4 \* 5  
2 - 4 \* 5 \* 3 + 1  
5 + 2 - 4 + 1 \* 3

Ha nem lenne operátor elsőbbség, akkor ez a három kifejezés 25, –29 és 12 értékre számítana. De mivel a szorzás és az osztás elsőbbséget élvez az összeadásnál és a kivonásnál, a kifejezések kiértékelése úgy történik, mintha a kifejezések ezen részei körül zárójelek lennének, akárcsak a matematikai jelölések (lásd a 4-8. példát).

Példa 4-8. Három kifejezés, amelyek zárójeleket tartalmaznak

1 + (2 \* 3) - (4 \* 5)  
2 - (4 \* 5 \* 3) + 1  
5 + 2 - 4 + (1 \* 3)

A PHP először a zárójelben lévő részkifejezéseket értékeli ki, hogy levezesse a 4-9. példában szereplő félig kész kifejezéseket.

Példa 4-9. A zárójelben lévő részkifejezések kiértékelése után

1 + (6) - (20)  
2 - (60) + 1  
5 + 2 - 4 + (3)

Ezeknek a kifejezéseknek a végeredménye –13, –57 és 6 (meglehetősen különbözik a 25, –29 és 12 eredményétől, amelyet akkor láttunk volna, ha nem lett volna operátor elsőbbség).

Természetesen felülírhatja az alapértelmezett operátori prioritást saját zárójelek beszúrásával és a kívánt sorrend kikényszerítésével (lásd a 4-10. példát).

Példa 4-10. Balról jobbra történő értékelés kényszerítése

((1 + 2) \* 3 - 4) \* 5  
(2 - 4) \* 5 \* 3 + 1  
(5 + 2 - 4 + 1) \* 3

A helyesen beszúrt zárójelekkel most a 25, –29 és 12 értékeket látjuk.

A 4-2. táblázat felsorolja a PHP operátorait prioritási sorrendben a magastól az alacsonyig.

4-2. táblázat. A PHP operátorok elsőbbsége (magastól alacsonyig)

Operátor(ok) Típus

() Zárójelek

++ -- Növekedés/csökkentés

! Logikus

\* / % Aritmetikus

+ - . Aritmetikus és karakterlánc

<< >> Bitenként

< <= > >= <> Összehasonlítás

== != === !== Összehasonlítás

& Bitenként (és hivatkozások)

^ Bitenként

| Bitenként

&& Logikus

|| Logikus

? : Hármas

= += -= \*= /= .= %= &= != ^= <<= >>= Feladat

and Logikus

xor Logikus

or Logikus

Ebben a táblázatban a sorrend nem önkényes, hanem gondosan megtervezett, hogy a legáltalánosabb és legintuitívabb elsőbbségek azok legyenek, amelyeket zárójelek nélkül kaphat meg. Például szétválaszthat két összehasonlítást egy AND vagy OR jellel, és azt kapja, amit vár.

Az asszociativitás

A kifejezések feldolgozását balról jobbra néztük, kivéve, ha az operátor elsőbbsége van érvényben. Néhány operátor azonban megköveteli a jobbról balra haladó feldolgozást, és ezt a feldolgozási irányt az operátor asszociativitásának nevezik. Egyes operátorok esetében nincs asszociativitás. Az asszociativitás (amint azt a 4-3. táblázatban részletezi) olyan esetekben válik fontossá, amikor nem kényszerítjük kifejezetten az elsőbbséget, ezért tisztában kell lennie az operátorok alapértelmezett műveleteivel.

Például vessünk egy pillantást a hozzárendelési operátorra a 4-11. példában, ahol három változó mindegyike 0-ra van állítva.

Táblázat 4-3. Operátor asszociativitás

Operátor Leírás Asszociativitás

< <= >= == != === !== <> Összehasonlítás Egyik sem

! Logikus NOT Jobb

~ Bitenként NOT Jobb

++ -- Növekedés és csökkentés Jobb

(int) Átküldés egész számra Jobb

(double) (float) (real) Átküldés lebegőpontos számra Jobb

(string) Átküldés egy karakterláncra Jobb

(array) Átküldés egy sorra Jobb

(object) Átküldés egy objektumra Jobb

@ A hibajelentés letiltása Jobb

= += -= \*= /= Feladat Jobb

.= %= &= |= ^= <<= >>= Feladat Jobb

+ Kiegészítés és unáris plusz Bal

- Kivonás és tagadás Bal

\* Szorzás Bal

/ Osztás Bal

% Abszolút érték Bal

. Karakterlánc összefűzése Bal

<< >> & ^ | Bitenként Bal

?: Hármas Bal

|| && and or xor Logikus Bal

, Szétválasztó Bal

Példa 4-11. Többszörös hozzárendelést tartalmazó nyilatkozat

<?php  
$level = $score = $time = 0;  
?>

Ez a többszörös hozzárendelés csak akkor lehetséges, ha először a kifejezés jobb szélső részét értékeljük ki, majd a feldolgozás jobbról balra halad tovább.

Újoncként a PHP-ben érdemes elkerülni az operátor-asszociativitás lehetséges buktatóit, ha a részkifejezéseket mindig zárójelek közé helyezi, hogy kikényszerítse az értékelési sorrendet. Ez segít más programozóknak is, akiknek karban kell tartaniuk az Ön kódját, hogy megértsék, mi történik.

Relációs operátorok

A relációs operátorok olyan kérdésekre válaszolnak, mint például: „Nulla értéke van ennek a változónak?” és „Melyik változónak nagyobb az értéke?” Ezek az operátorok két operandust tesztelnek, és TRUE vagy FALSE logikai eredményt adnak vissza. A relációs operátoroknak három típusa van: egyenlőség, összehasonlítás és logikai.

Egyenlőség

Ahogy ebben a fejezetben már néhányszor láthattuk, az egyenlőség operátor == (két egyenlőségjel). Fontos, hogy ne keverjük össze az = (egyszeres egyenlőségjel) hozzárendelési operátorral. A 4-12. példában az első utasítás értéket ad, a második pedig az egyenlőséget vizsgálja.

Példa 4-12. Érték hozzárendelése és az egyenlőség tesztelése

<?php  
$month = "March";  
if ($month == "March") echo "It's springtime";  
?>

Amint látja, az TRUE vagy FALSE érték visszaadásával az egyenlőség operátor lehetővé teszi a feltételek tesztelését, például egy IF utasítás használatával. De ez nem az egész történet, mert a PHP egy lazán begépelt nyelv. Ha egy egyenlőségkifejezés két operandusa különböző típusú, a PHP olyan típusra konvertálja őket, amelyik a legjobban értelmezhető. Egy ritkán használt azonosító operátor, amely három egyenlőségjelből áll egymás után, használható az elemek összehasonlítására átalakítás nélkül. Például a teljes egészében számokból álló karakterláncok számokká alakulnak, ha egy számmal hasonlítják össze. A 4-13. példában $a és $b két különböző karakterlánc, ezért nem várjuk el, hogy az if utasítások egyike se adjon eredményt.

Példa 4-13. Az egyenlőség és identitás operátorai

<?php  
$a = "1000";  
$b = "+1000";  
if ($a == $b) echo "1";  
if ($a === $b) echo "2";  
?>

Ha azonban futtatja a példát, látni fogja, hogy az 1-es számot adja ki, ami azt jelenti, hogy az első IF utasítás kiértékelése TRUE. Ennek az az oka, hogy mindkét karakterláncot először számokká alakították, és az 1000 ugyanaz, mint a +1000. Ezzel szemben a második if utasítás az identitás operátort használja, így összehasonlítja az $a és $b as-t karakterláncokat, látja, hogy különböznek, és így nem ad ki semmit.

Akárcsak az operátor elsőbbségének kikényszerítésekor, ha kétségei vannak azzal kapcsolatban, hogy a PHP hogyan konvertálja az operandustípusokat, az identitásoperátorral kikapcsolhatja ezt a viselkedést.

Ugyanúgy, ahogy az egyenlőség operátort használhatja az operandusok egyenlőségének tesztelésére, az egyenlőtlenség operátor != használatával ellenőrizheti, hogy nem egyenlőek-e. Vessen egy pillantást a 4-14. példára, amely a 4-13. példa átírása, amelyben az egyenlőség és az azonosság operátorait az inverzeikre cseréltük.

Példa 4-14. Az egyenlőtlenség és a nem azonos operátorok

<?php  
$a = "1000";  
$b = "+1000";  
if ($a != $b) echo "1";  
if ($a !== $b) echo "2";  
?>

És ahogy várható is, az első IF utasítás nem adja ki az 1-es számot, mert a kód azt kérdezi, hogy $a és $b nem egyenlő-e egymással számszerűen. Ehelyett ez a kód a 2-es számot adja ki, mivel a második IF utasítás azt kérdezi, hogy $a és $b nem azonos-e egymással a tényleges karakterlánc típusában, és a válaszTRUE; nem egyformák.

Összehasonlító operátorok

Összehasonlító operátorok használatával nemcsak egyenlőséget és egyenlőtlenséget tesztelhet. A PHP a következőt is megadja: > (nagyobb, mint), < (kisebb, mint), >= (nagyobb vagy egyenlő), és <= (kisebb vagy egyenlő), amellyel játszani lehet. A 4-15. példák ezeket a használat során mutatják be.

Példa 4-15. A négy összehasonlító operátor

<?php  
$a = 2; $b = 3;  
if ($a > $b) echo "$a is greater than $b<br>";  
if ($a < $b) echo "$a is less than $b<br>";  
if ($a >= $b) echo "$a is greater than or equal to $b<br>";  
if ($a <= $b) echo "$a is less than or equal to $b<br>";  
?>

Ebben a példában, ahol $a 2 és $b 3, a következő kimenet jelenik meg:

A 2 kisebb, mint 3

2 kisebb vagy egyenlő, mint 3

Az eredmények megtekintéséhez próbálja ki ezt a példát saját maga, és módosítsa az $a és $b értékét. Próbáld meg ugyanazt az értéket beállítani, és nézd meg, mi történik.

Logikai operátorok

A logikai operátorok TRUE vagy FALSE eredményeket adnak, ezért Boolean operátoroknak is nevezik őket. Négy van belőlük (lásd 4-4. táblázat).

Táblázat 4-4. A logikai operátorok

Logikai operátor Leírás

AND TRUE, ha mindkét operandus TRUE

OR TRUE, ha bármelyik operandus TRUE

XOR TRUE, ha a két operandus egyike TRUE

! (NOT) TRUE, ha az operandus FALSE, vagy FALSE, ha az operandus TRUE

Ezeket az operátorokat a 4-16. példában láthatja. Vegye figyelembe, hogy a ! szimbólumra van szükség a PHP-ben a NOT helyett. Ezenkívül az operátorok lehetnek kis- vagy nagybetűk.

Példa 4-16. A használatban lévő logikai operátorok

<?php  
$a = 1; $b = 0;  
echo ($a AND $b) . "<br>";  
echo ($a or $b) . "<br>";  
echo ($a XOR $b) . "<br>";  
echo !$a . "<br>";  
?>

Ez a példa soronként ad ki semmit, 1-et, 1-et és semmit, ami azt jelenti, hogy csak a második és harmadik visszhang-utasítás értékeli ki TRUE-ként. (Ne feledje, hogy a NULL – vagy a semmi – FALSE értéket jelent.) Ennek az az oka, hogy az AND utasításnak mindkét operandusnak TRUE-nak kell lennie, ha TRUE értéket ad vissza, míg a negyedik utasítás NOT parancsot hajt végre a $ értékre. a, TRUE-ról (1-es érték) FALSE-ra változtatva. Ha ezzel szeretne kísérletezni, próbálja ki a kódot, és adjon meg $a és $b változó értéket 1 és 0 között.

Kódoláskor ne feledje, hogy az ÉS és a VAGY kisebb prioritású mint az && és || operátorok többi verziója.

Példa 4-17. Az OR operátort használó utasítás

<?php  
if ($finished == 1 OR getnext() == 1) exit;  
?>

Ha minden IF utasításnál meg kell hívnia a getnext-et, akkor átírhatja a kódot a 4-18. példában leírtak szerint.

Példa 4-18. Az IF...OR utasítás módosítva, hogy biztosítsa a getnext meghívását

<?php  
$gn = getnext();  
if ($finished == 1 OR $gn == 1) exit;  
?>

Ebben az esetben a kód végrehajtja a getnext függvényt, és eltárolja a visszaadott értéket a $gn-ben, mielőtt végrehajtaná az IF utasítást.

Egy másik megoldás az, hogy felcseréljük a két tagmondatot, hogy megbizonyosodjunk arról, hogy a getnext végrehajtásra kerül, mivel ekkor először jelenik meg a kifejezésben.

A 4-5. táblázat bemutatja a logikai operátorok használatának összes lehetséges változatát. Azt is meg kell jegyezni, hogy a !TRUE értéke FALSE, a !FALSE pedig IGAZ.

Táblázat 4-5. Minden lehetséges PHP logikai kifejezés

Bemenetek Operátorok és eredmények

**a b AND OR XOR**

TRUE TRUE TRUE TRUE FALSE

TRUE FALSE FALSE TRUE TRUE

FALSE TRUE FALSE TRUE TRUE

FALSE FALSE FALSE FALSE FALSE

Feltételek

A feltételek megváltoztatják a programfolyamatot. Lehetővé teszik, hogy kérdéseket tegyen fel bizonyos dolgokról, és különböző módon válaszoljon a kapott válaszokra. A feltételes feltételek központi szerepet játszanak a dinamikus weboldalak létrehozásában – elsősorban a PHP használatában –, mivel megkönnyítik az oldal minden egyes megtekintésénél más-más kimenet megjelenítését. Ebben a részben három alapvető feltételt mutatok be: az if utasítást, a switch utasítást és a ? operátort. Ezen túlmenően a hurkolt feltételes feltételek (amivel hamarosan kitérünk) újra és újra végrehajtják a kódot, amíg egy feltétel nem teljesül.

Az IF utasítás

A programfolyamatról való gondolkodás egyik módja az, ha egysávos autópályaként képzeljük el, amelyen haladunk. Ez nagyjából egy egyenes vonal, de időnként különféle táblákkal találkozik, amelyek jelzik, merre menjen. Az IF állítás esetén elképzelhető, hogy egy kitérő táblával találkozik, amelyet követnie kell, ha egy bizonyos feltétel IGAZ. Ha igen, akkor hajtson le, és kövesse a kerülőutat, amíg vissza nem tér a főútra, majd folytatja az utat az eredeti irányba. Vagy ha a feltétel nem TRUE, figyelmen kívül hagyja a kitérőt, és továbbhalad (lásd 4-1. ábra).

Az IF feltétel tartalma lehet bármilyen érvényes PHP-kifejezés, beleértve az egyenlőségi teszteket, az összehasonlító kifejezéseket, a 0 és a NULL teszteket, és még függvényeket is (akár beépített, akár Ön által írt függvényeket). Azok a műveletek, amelyeket akkor kell végrehajtani, ha az IF feltétel IGAZ, általában a kapcsos zárójelek közé ({ }) kerülnek. Figyelmen kívül hagyhatja a kapcsos zárójeleket, ha csak egyetlen utasítást kell végrehajtania, de ha mindig kapcsos zárójelet használ, elkerülheti a nehezen nyomon követhető hibák felkutatását, például amikor egy feltételhez egy extra sort ad, és zárójelek hiánya miatt nem kerül értékelésre.

A „goto fail” hibaként ismert hírhedt biztonsági rés sok éven át kísértette az Apple termékeinek Secure Sockets Layer (SSL) kódját, mivel egy programozó elforgatta a kapcsos zárójeleket egy IF utasítás körül, ami miatt egy funkció néha sikeres kapcsolódást jelentett. ez nem biztos, hogy mindig így volt. Ez lehetővé tette a rosszindulatú támadó számára, hogy megkapja a biztonságos tanúsítvány elfogadását, amikor azt el kellett volna utasítani. Ha kétségei vannak, tegyen kapcsos zárójelet az IF állítások köré.

Vegye figyelembe, hogy a rövidség és az érthetőség kedvéért azonban a könyvben szereplő példák közül sok figyelmen kívül hagyja ezt a javaslatot, és elhagyja a kapcsos zárójeleket az egyes kijelentéseknél. A 4-19. példában képzelje el, hogy a hónap vége van, és az összes számláját kifizették, tehát bankszámla karbantartást végez.

Példa 4-19. Egy IF kifejezés kapcsos zárójelekkel

<?php  
if ($bank\_balance < 100)  
{  
$money = 1000;  
$bank\_balance += $money;  
}  
?>

Ebben a példában egyenlegét ellenőrzi, hogy az kevesebb-e 100 USD-nál (vagy bármilyen pénznemnél). Ha igen, fizet magának 1000 dollárt, majd hozzáadja az egyenleghez. (Ha csak a pénzkeresés lenne ilyen egyszerű!) Ha a banki egyenleg 100 USD vagy nagyobb, a feltételes kivonatok figyelmen kívül maradnak, és a programfolyamat a következő sorra ugrik (nem mutatott). Ebben a könyvben a kapcsos zárójelek nyitása általában új sorban kezdődik. Vannak, akik szeretik az első kapcsos zárójelet a feltételes kifejezés jobb oldalára tenni; mások új sort kezdenek vele. Ezek bármelyike megfelelő, mert a PHP lehetővé teszi a szóközök (szóközök, újsorok és tabulátorok) tetszőleges módon történő elhelyezését. Azonban a kód könnyebben olvasható és hibakereshető, ha a feltételes feltételek minden szintjét egy tabulátorral behúzza.

Az ELSE utasítás

Néha, amikor egy feltétel nem TRUE, előfordulhat, hogy nem akar azonnal továbblépni a fő programkódra, hanem valami mást szeretne tenni helyette. Itt jön be az ELSE utasítás. Ezzel beállíthat egy második kerülőutat az autópályán, mint a 4-2. ábrán.

Az IF...ELSE utasítással az első feltételes utasítás végrehajtásra kerül, ha a feltétel TRUE. De ha FALSE, akkor a második végrehajtásra kerül. A két lehetőség közül az egyiket végre kell hajtani. Semmilyen körülmények között sem hajtható végre mindkettő (vagy egyik sem). A 4-20. példa az IF...ELSE szerkezet használatát mutatja be.

Példa 4-20. Egy IF...ELSE kijelentés kapcsos zárójelekkel

<?php  
if ($bank\_balance < 100)  
{  
$money = 1000;  
$bank\_balance += $money;  
}  
else  
{  
$savings += 50;  
$bank\_balance -= 50;

}  
?>

Ebben a példában, ha megbizonyosodott arról, hogy 100 dollár vagy több van a bankban, akkor az ELSE utasítás kerül végrehajtásra, és ennek a pénznek egy részét a megtakarítási számlájára helyezi. Akárcsak az állításoknál, ha a másiknak csak egy feltételes állítása van, választhatja a kapcsos zárójelek elhagyását. (A kapcsos zárójelek azonban mindig ajánlottak. Először is könnyebben érthetővé teszik a kódot. Másodszor pedig lehetővé teszik, hogy később könnyen hozzáadhasson további utasításokat az ághoz.)

Az ELSEIF utasítás

Vannak olyan esetek is, amikor azt szeretné, hogy számos különböző lehetőség forduljon elő, a feltételek sorozata alapján. Ezt az ELSEIF utasítással érheti el. Elképzelhető, hogy olyan, mint egy ELSE utasítás, kivéve, hogy egy további feltételes kifejezést helyez el a feltételes kód elé.

A 4-21. példában egy teljes IF...ELSEIF...ELSE konstrukciót láthatunk.

Példa 4-21. Egy IF...ELSEIF...ELSE kijelentés kapcsos zárójelekkel

<?php  
if ($bank\_balance < 100)  
{  
$money = 1000;  
$bank\_balance += $money;  
}  
elseif ($bank\_balance > 200)  
{  
$savings += 100;  
$bank\_balance -= 100;  
}  
else  
{  
$savings += 50;  
$bank\_balance -= 50;  
}  
?>

A példában egy ELSEIF utasítás került beillesztésre az IF és ELSE utasítások közé. Ellenőrzi, hogy banki egyenlege meghaladja-e a 200 USD-t, és ha igen, úgy dönt, hogy ebben a hónapban megengedheti magának, hogy 100 USD-t megtakarítson. Bár kezdem kissé túlfeszíteni a metaforát, elképzelhető ez többirányú kerülőútként is (lásd 4-3. ábra).

Az ELSE utasítás vagy egy IF...ELSE vagy egy IF...ELSEIF...ELSE utasítást zár le. Kihagyhatsz egy final ELSE-t, ha nem kötelező, de nem lehet egy ELSEIF előtt; az IF utasítás előtt sem lehet ELSEIF.

Annyi ELSEIF állításod lehet, amennyit csak akarsz. De ahogy az ELSEIF utasítások száma növekszik, valószínűleg jobb lenne egy SWITCH utasítás megfontolása, ha az megfelel az Ön igényeinek. Ezt nézzük meg legközelebb.

A SWITCH utasítás

A SWITCH utasítás akkor hasznos, ha egy változónak vagy egy kifejezés eredményének több értéke is lehet, amelyek mindegyike más-más tevékenységet válthat ki. Vegyünk például egy PHP-vezérelt menürendszert, amely egyetlen karakterláncot ad át a főmenü kódjának a felhasználó kérésének megfelelően. Tegyük fel, hogy a lehetőségek a következők: Kezdőlap, Névjegy, Hírek, Bejelentkezés és Linkek, és ezek valamelyikére állítjuk be a $page változót, a felhasználó bevitelének megfelelően. Ha ehhez írjuk a kódot az IF...ELSEIF...ELSE használatával, akkor a 4-22. példához hasonlóan nézhet ki.

Példa 4-22. Többsoros IF...ELSEIF...ELSE utasítás

<?php  
if ($page == "Home") echo "You selected Home";  
elseif ($page == "About") echo "You selected About";  
elseif ($page == "News") echo "You selected News";  
elseif ($page == "Login") echo "You selected Login";  
elseif ($page == "Links") echo "You selected Links";  
else echo "Unrecognized selection";  
?>

Ha SWITCH utasítást használunk, a kód úgy nézhet ki, mint a 4-23. példa.

Példa 4-23. Egy SWITCH utasítás

<?php  
switch ($page)  
{  
case "Home":  
echo "You selected Home";  
break;  
case "About":  
echo "You selected About";  
break;  
case "News":  
echo "You selected News";  
break;  
case "Login":  
echo "You selected Login";  
break;  
case "Links":  
echo "You selected Links";  
break;  
}  
?>

Mint látható, a $page csak egyszer kerül említésre a SWITCH utasítás elején. Ezt követően a CASE parancs ellenőrzi az egyezéseket. Az egyik előfordulásakor a megfelelő feltételes utasítás végrehajtásra kerül. Természetesen egy valódi programban itt lenne a kód, amely megjeleníti vagy egy oldalra ugorhat, ahelyett, hogy egyszerűen elmondaná a felhasználónak, hogy mit választottak.

A SWITCH utasításokkal nem használ kapcsos zárójeleket a kis- és nagybetűs parancsokon belül. Ehelyett kettősponttal kezdődnek, és a BREAK utasítással fejeződnek be. A SWITCH utasításban szereplő esetek teljes listája azonban kapcsos zárójelek között van.

Kitörés

Ha egy feltétel teljesülése miatt szeretne kitörni a SWITCH utasításból, használja a BREAK parancsot. Ez a parancs arra utasítja a PHP-t, hogy lépjen ki a SWITCH-ből és ugorjon a következő utasításra. Ha elhagyná a 4-23. példában szereplő BREAK parancsokat, és a Home esetet TRUE-nak értékelné, akkor mind az öt eset végrehajtásra kerülne. Vagy ha a $page értéke News lenne, akkor onnantól kezdve az összes esetparancs végrehajtásra kerülne. Ez szándékos, és némi haladó programozást tesz lehetővé, de általában mindig ne feledje, hogy ki kell adnia egy BREAK parancsot minden alkalommal, amikor a feltételes kis- és nagybetűk végrehajtása befejeződött. Valójában a BREAK utasítás elhagyása gyakori hiba.

Alapértelmezett művelet

A SWITCH utasítások tipikus követelménye, hogy vissza kell állítani az alapértelmezett műveletet, ha egyik esetfeltétel sem teljesül. Például a 4-23. példa menükódja esetén hozzáadhatja a 4-24. példában szereplő kódot közvetlenül az utolsó kapcsos zárójel előtt.

Példa 4-24. A 4-23. példához hozzáadandó alapértelmezett utasítás

default:  
echo "Unrecognized selection";  
break;

Ez megismétli a 4-22. példában szereplő ELSE utasítás hatását. Bár itt nincs szükség BREAK parancsra, mert az alapértelmezett az utolsó részutasítás, és a programfolyamat automatikusan a záró kapcsos zárójelig folytatódik, de ha úgy dönt, hogy az alapértelmezett utasítást feljebb helyezi, feltétlenül szükség lesz egy BREAK parancsra. Általánosságban elmondható, hogy a legbiztonságosabb gyakorlat az, hogy mindig tartalmazza a BREAK parancsot.

Alternatív szintaxis

Ha szeretné, a SWITCH utasításban az első kapcsos zárójelet lecserélheti egyetlen kettőspontra, az utolsó kapcsos zárójelet pedig egy ENDSWITCH paranccsal, a 4-25. példában leírtak szerint. Ezt a megközelítést azonban nem használják általánosan, és itt csak abban az esetben említik, ha harmadik féltől származó kódban találkozik vele.

Példa 4-25. Alternatív SWITCH utasítás szintaxis

<?php  
switch ($page):  
case "Home":  
echo "You selected Home";  
break;  
// etc  
case "Links":  
echo "You selected Links";  
break;  
endswitch;  
?>

A ? Operátor

Az IF és ELSE utasítások bőbeszédűségének elkerülésének egyik módja a tömörebb hármas operátor, a ? használata, ami szokatlan, mivel három operandust igényel a tipikus kettő helyett. Ezzel röviden találkoztunk a 3. fejezetben a PRINT és az ECHO utasítások közötti különbségről szóló vitában, mint egy olyan operátortípus példáját, amely jól működik PRINT-tel, de nem ECHO-val. A ? operátor kap egy kifejezést, amelyet ki kell értékelnie, valamint két végrehajtandó utasítást: az egyiket arra az esetre, amikor a kifejezés kiértékelése TRUE, a másikat pedig arra, hogy ha FALSE. A 4-26. példa azt a kódot mutatja be, amelyet arra használhatunk, hogy egy autó üzemanyagszintjére vonatkozó figyelmeztetést írjunk a digitális műszerfalra.

Példa 4-26. A ? operátor használata

<?php  
echo $fuel <= 1 ? "Fill tank now" : "There's enough fuel";  
?>

Ebben az utasításban, ha egy gallon vagy kevesebb üzemanyag van (más szóval, a $fuel értéke 1 vagy kevesebb), a „Tölts meg a tankot” karakterlánc visszakerül az előző ECHO utasításhoz. Ellenkező esetben az „Van elég üzemanyag” üzenet visszakerül. Hozzárendelheti A ? utasításba visszakapott értéket egy változóba (lásd a 4-27. példát).

Példa 4-27. Egy ? feltételes eredményt hozzárendelünk egy változóhoz

<?php  
$enough = $fuel <= 1 ? FALSE : TRUE;  
?>

Itt az $elég érték csak akkor lesz hozzárendelve TRUE értékhez, ha egy gallonnál több üzemanyag van; ellenkező esetben a FALSE értéket kapja. Ha a ? operátort zavarónak találja, szabadon ragaszkodhat az IF utasításokhoz, de ismernie kell az operátort, mert látni fogja mások kódjában. Nehéz lehet olvasni, mert gyakran keveri ugyanazon változó többszörös előfordulását. Például az alábbi kódok meglehetősen népszerűek:

$saved = $saved >= $new ? $saved : $new;  
If you take it apart carefully, you can figure out what this code does:  
$saved = // Set the value of $saved to...  
$saved >= $new // Check $saved against $new  
? // Yes, comparison is true...  
$saved // ... so assign the current value of $saved  
: // No, comparison is false...  
$new; // ... so assign the value of $new

Ez egy tömör módja annak, hogy nyomon kövesse a legnagyobb értéket, amelyet a program előrehaladása során tapasztalt. Elmenti a legnagyobb értéket $saved-ben, és összehasonlítja a $new értékkel minden alkalommal, amikor új értéket kap. Azok a programozók akik ismerik a ? operátort kényelmesebbnek találják az IF utasításoknál az ilyen rövid összehasonlításokhoz. Ha nem használják kompakt kód írására, akkor általában valamilyen soron belüli döntés meghozatalára használják, például amikor azt tesztelik, hogy egy változó be van-e állítva, mielőtt átadná egy függvénynek.

Hurkolás

A számítógépek egyik nagyszerű tulajdonsága, hogy gyorsan és fáradhatatlanul meg tudják ismételni a számítási feladatokat. Gyakran előfordulhat, hogy egy program újra és újra megismétli ugyanazt a kódsorozatot, amíg valami nem történik, például a felhasználó beír egy értéket vagy el nem éri a természetes végét. A PHP ciklusstruktúrái tökéletes módot biztosítanak erre. Ennek működésének bemutatásához nézze meg a 4-4. ábrát. Ez nagyjából megegyezik a kijelentések illusztrálására használt autópálya-metaforával, azzal a különbséggel, hogy a kitérőnek van egy hurokszakasza is, amelyből – ha egy jármű belépett – csak a megfelelő programkörülmények között lehet kiszállni.

WHILE hurkok

A 4-26. példában szereplő digitális autó műszerfalát alakítsuk hurokká, amely folyamatosan ellenőrzi az üzemanyagszintet vezetés közben, egy WHILE hurok segítségével (4-28. példa).

Példa 4-28. Egy WHILE hurok

<?php  
$fuel = 10;  
while ($fuel > 1)  
{  
// Keep driving...  
echo "There's enough fuel";  
}  
?>

Valójában szívesebben hagyhatja a zöld lámpát, mint a kimeneti szöveget, de a lényeg az, hogy bármilyen pozitív jelzést szeretne adni az üzemanyag szintjéről, az a WHILE hurokba kerüljön. Egyébként, ha kipróbálja ezt a példát, vegye figyelembe, hogy addig nyomtatja a karakterláncot, amíg a böngészőben a Leállítás gombra nem kattint.

Az IF utasításokhoz hasonlóan észre fogja venni, hogy kapcsos zárójelekre van szükség ahhoz, hogy az utasításokat a WHILE utasításokon belül rögzítsék, kivéve, ha csak egy van.

Egy másik példa a 12-szeres táblázatot megjelenítő WHILE hurokhoz, lásd a 4-29. példát.

Példa 4-29. A WHILE hurok a 12-szeres táblázat nyomtatásához

<?php  
$count = 1;  
while ($count <= 12)  
{  
echo "$count times 12 is " . $count \* 12 . "<br>";  
++$count;  
}  
?>

Itt a $count változó 1-es értékre inicializálódik, majd egy WHILE hurok indul a $count <= 12 összehasonlító kifejezéssel. Ez a hurok addig fut, amíg a változó nem nagyobb, mint 12. A kód kimenete a következő:

1-szer 12 az 12

2-szer 12 az 24

3-szor 12 az 36

stb...

A hurkon belül egy karakterlánc kerül kinyomtatásra a $count 12-vel szorzott értékével együtt. A rendezettség érdekében ezt egy <br> címkével követi, amely új sort kényszerít.

Ezután a $count növekszik, készen áll az utolsó kapcsos zárójelre, amely arra utasítja a PHP-t, hogy térjen vissza a hurok elejére. Ezen a ponton a $count ismét tesztelésre kerül, hogy megbizonyosodjon arról, hogy nagyobb-e 12-nél. Nem az, de most 2-es az értéke, és további 11-szeres hurok után 13-as lesz. Amikor ez megtörténik, a WHILE hurkon belüli kód kimarad, és a végrehajtás átmegy a ciklust követő kódra, ami ebben az esetben a program vége.

Ha a ++$count utasítás (ami akár $count++ is lehetett volna) nem lett volna ott, akkor ez a ciklus olyan lenne, mint az első ebben a szakaszban. Soha nem érne véget, és csak az 1 \* 12 eredményt nyomtatnák ki újra és újra.

De van egy sokkal ügyesebb módja ennek a huroknak, ami szerintem tetszeni fog. Tekintse meg a 4-30. példát.

Példa 4-30. A 4-29. példa rövidített változata

<?php  
$count = 0;  
while (++$count <= 12)  
echo "$count times 12 is " . $count \* 12 . "<br>";  
?>

Ebben a példában lehetséges volt a ++$count utasítás áthelyezése a WHILE hurkon belüli utasításokból a hurok feltételes kifejezésébe. Most az történik, hogy a PHP a hurok minden iterációjának elején találkozik a $count változóval, és miután észreveszi, hogy az INCREMENT operátor előtagja van, először növeli a változót, és csak azután hasonlítja össze a 12-es értékkel. hogy a $count-ot most 0-ra kell inicializálni, nem 1-re, mert a hurokba lépéskor azonnal növekszik. Ha az inicializálást 1-en tartja, csak a 2 és 12 közötti eredmények kerülnek kiadásra.

DO…WHILE hurkok

A WHILE ciklus egy kis változata a DO...WHILE hurok, amelyet akkor használunk, ha azt szeretnénk, hogy egy kódblokk legalább egyszer lefusson, és csak ezután legyen feltételes. A 4-31. példa az ilyen hurkot használó 12-szeres tábla kódjának módosított változatát mutatja be.

Példa 4-31. DO...WHILE hurok a 12-szeres táblázat nyomtatásához

<?php  
$count = 1;  
do  
echo "$count times 12 is " . $count \* 12 . "<br>";  
while (++$count <= 12);  
?>

Figyeljük meg, hogyan tértünk vissza a $count inicializálásához 1-re (nem 0-ra), mivel a hurok ECHO utasítása végrehajtásra kerül, mielőtt lehetőségünk lenne a változó növelésére. Ettől eltekintve azonban a kód nagyon hasonlónak tűnik. Természetesen, ha egy DO...WHILE cikluson belül egynél több utasítás van, ne felejtse el használni a kapcsos zárójelet, mint a 4-32. példában.

Példa 4-32. A 4-31. példa kiterjesztése kapcsos zárójelek használatára

<?php  
$count = 1;  
do {  
echo "$count times 12 is " . $count \* 12;  
echo "<br>";  
} while (++$count <= 12);  
?>