PHP PROGRAMOZÁS

7. forduló

A kategória támogatója: Human Priority Kft.

Ismertető a feladathoz

A feladatok értelmezéséhez a PHP 7.4.32 verziót javasoljuk!

Felhasznált idő: 39:59/40:00 **Elért pontszám:** 3.67/16

1. feladat 0/1 pont

Az a feladatunk, hogy egy dátumhoz hozzáadjunk 1 hónapot. Az alábbi megoldások közül melyek valósítják ezt meg?

Válaszok

- [ ]
```
time() + 30 * 24 * 3600
```

- [x]
```
strtotime("+1 month")
```
Ez a válasz helyes, és meg is jelölted.

- [x]
```
strtotime("+1 months")
```

Ez a válasz helyes, de nem jelölted meg.

Ez a válasz helytelen, de megjelölted.

Magyarázat

A helyes megoldások a ***strtotime("+1 month") és a strtotime("+1 months")***, mivel ezek figyelembe veszik, hogy egy hónap hány napos (28, 29, 30 vagy 31). A másik két megoldás mindig fixen 30 nappal számol, így nem minden hónapban ad helyes eredményt.

2. feladat 1/1 pont

A következő kódrészletet futtatva mi kerül a kimenetre?

```php
class Food
{
    private $price;
    private $name;

    public function getName() {
        return $this->name;
    }
    public function setName($name) {
        $this->name = $name;
    }
    public function getPrice() {
        return $this->price;
    }
    public function setPrice($price) {
        $this->price = $price;
    }
}

$pizza = new Food();
$pizza->setName('Pizza');
$pizza->setPrice(3);

$pasta = new Food();
$pasta->setName('Pasta');
$pasta->price = 4;

$items = [$pizza, $pasta];

foreach ($items as $item) {
    echo $item->getName() . ' is ' . $item->getPrice() . ' EUR, ';
}
```

Válasz

- ○ Pizza is 3 EUR, Pasta is 4 EUR,
- ○ Pizza is 3 EUR, Pasta is EUR,
- ◉ Hibát kapunk
  Ez a válasz helyes, és meg is jelölted.

Magyarázat

A "hibát kapunk" válasz helyes, hiszen a Food osztály egyik private változóját (***price***) próbáltuk meg direktbe írni az osztályon kívülről, amit nem tehetünk meg (***$pasta->price = 4***).

3. feladat 2.67/4 pont

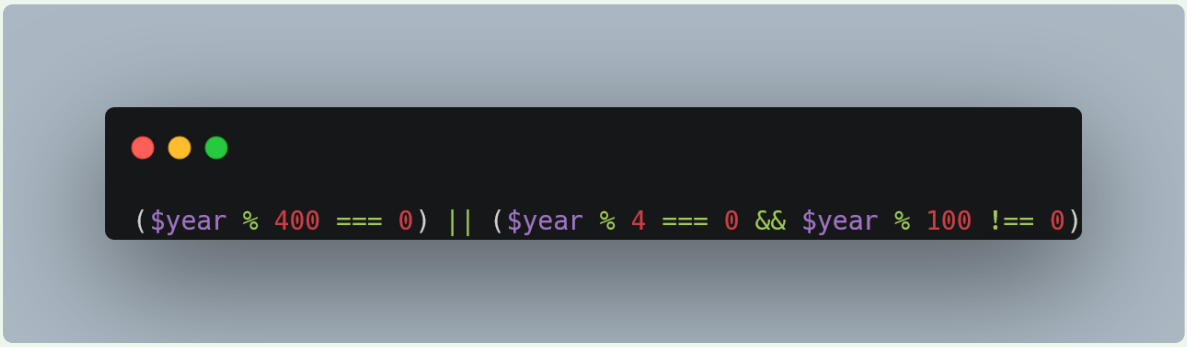
Az alábbiak közül mely módszerekkel tudjuk megállapítani, hogy egy adott év szökőév-e vagy sem, amennyiben a visszatérési értéknek logikai értéknek kell lennie, ahol az IGAZ érték jelenti a szökőévet?

Válaszok

- [x] A válasz:

Ez a válasz helyes, és meg is jelölted.

- [x] B válasz:



Ez a válasz helyes, és meg is jelölted.

- [ ] C válasz:

- [x] D válasz:

Ez a válasz helyes, de nem jelölted meg.

E válasz:

Magyarázat

A C és E kivételével mindegyik megoldás helyes. A ***checkdate()*** metódus egy konkrét dátumot vizsgál és visszatérít egy BOOLEAN értéket attól függően, hogy a dátum valós-e vagy sem. Mivel nap és hónap értékeknek fixen február 29-ét adunk meg, így bármilyen évszámnál vizsgálhatjuk vele a szökőévet.

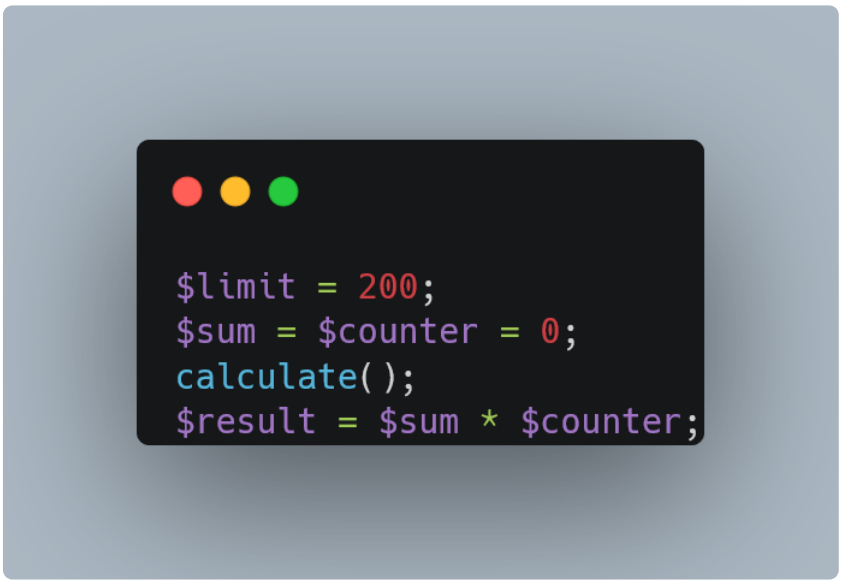
A második megoldás osztások maradékát vizsgálja, hogy meghatározza a szökőévet a Gergely naptár szabályai alapján. A harmadik megoldás ugyan megadja, hogy szökőév van-e vagy sem, de nem logikai értéket térít vissza, míg a negyedik esetben az '1'-el való (laza) összehasonlítás miatt már logikai értéket kapunk. Az E válaszban ugyanazt csináljuk, mint a D-nél, de itt típusösszehasonlítás is történik, így ez a megoldás nem lesz jó, mert a ***date()*** függvény minden esetben stringgel tér vissza.

4. feladat 0/5 pont

A felsorolt válaszokban található **calculate()** metódusok do-while ciklusában egy 1 és 50 közötti véletlenszámot vizsgálunk, hogy az osztható-e hárommal. Ha igen, hozzáadjuk ezt a számot az ***$i*** változóhoz, majd megvizsgáljuk, hogy az így kapott szám 9-re végződik-e, valamint hogy ***$i*** értéke nem lépte-e túl a globális ***$limit*** változó értékét. Amennyiben ez a feltétel igaz, csökkentjük ***$i*** értékét egyel, valamint növeljük ***$j*** változó értékét eggyel. A ciklus addig fut, amíg ***$i*** értéke kisebb a globális ***$limit*** változó értékénél.

A ciklusból való kilépés után a globális ***$sum*** változó értékét ***$j*** értékével megnöveljük. Végül, ha ***$j*** értéke kisebb 2-nél, vagy a metódus paraméterében kapott ***$cnt*** érték kisebb kettőnél, úgy ***$cnt*** értékét eggyel megnöveljük, majd újra meghívjuk a metódust ezen értékkel (rekurzív hívás).

A metódus elején a paraméterül kapott ***$cnt*** értéket mindig letároljuk a globális ***$counter*** változóba is. A célunk, hogy a metódus kívülről történő egyszeri meghívása után a globális ***$sum*** és ***$counter*** változók értékét összeszorozva egy új számot kapjunk eredményül. Az alábbi kódot fogjuk futtatni:



Válasz

A válasz:

```php
function calculate($cnt = 1) {
    $i = $j = 0;
    global $limit, $sum, $counter;
    $counter = $cnt;

    do {
        $random = rand(1, 50);
        if ($random % 3 === 0) {
            $i += $random;
        }
        if (substr($i, -1) == 9 && $i <= $limit) {
            $i--;
            $j++;
        }
    } while ($i < $limit);
    $sum += $j;

    if ($j < 2 || $cnt < 2) {
        calculate($cnt++);
    }
}
```

B válasz:

```php
function calculate($cnt = 1) {
    $i = $j = 0;
    global $limit, $sum, $counter;
    $counter = $cnt;

    do {
        $random = rand(1, 50);
        if ($random % 3 === 0) $i += $random;
        if (substr($i, -1) == 9 && $i <= $limit) {
            $i--;
            $j++;
        }
    } while ($i < $limit);
    $sum += $j;

    if ($j < 2 || $cnt < 2) {
        calculate(++$cnt);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

C válasz:

```php
function calculate($cnt = 1) {
    $i = $j = 0;
    global $limit, $sum, $counter;
    $counter = $cnt;

    do {
        $random = rand(1, 50);
        if ($random % 3 === 0) {
            $i += $random;
        }
        if (substr($i, -1) === 9 && $i <= $limit) {
            $i--;
            $j++;
        }
    } while ($i < $limit);
    $sum += $j;

    if ($j < 2 || $cnt < 2) {
        calculate(++$cnt);
    }
}
```

D válasz:

```php
function calculate($cnt = 1) {
    $i = $j = 0;
    global $limit, $sum, $counter;
    $counter = $cnt;

    do {
        $random = rand(1, 50);
        if ($random % 3 === 0) $i += $random;
        if ($i[end($i)] == 9 && $i <= $limit) {
            $i--;
            $j++;
        }
    } while ($i < $limit);
    $sum += $j;

    if ($j < 2 || $cnt < 2) {
        calculate($cnt++);
    }
}
```

Magyarázat

A helyes megoldás a B válasz.

Az A válasz végtelen ciklust okoz azáltal, hogy a ***calculate()*** metódus rekurzív hívásakor előbb átadja a ***$cnt*** változót, majd csak utána növeli az értékét, így annak értéke végig 1 marad.

A C válasz esetében a 9-re végződő számok vizsgálatánál ún. strict comparison történik (három egyenlőségjel), emiatt ***$j*** változó értékét sosem növeljük, s így ismét végtelen ciklust idézünk elő (a ***substr()*** metódus mindig stringgel tér vissza, akkor is, ha bemenetként integert kapott).

A D válaszban szintén a 9-re végződés vizsgálata hibás, mert tömbként próbálja vizsgálni az utolsó karaktert, ráadásul az ***end()*** függvénynek integert adunk át tömb helyett. PHP 7.4 esetén ez ugyan nem okoz végzetes hibát (PHP 8-nál már igen), viszont a C válaszhoz hasonló okokból itt is végtelen ciklust okozunk (ráadásul az A válaszhoz hasonlóan itt is rosszul hívjuk a ***calculate()*** metódust).

5. feladat 0/5 pont

Az alábbi Cart osztály egy webáruház egyszerűsített kosár-kezelését demonstrálja. Az ***$items*** változó a kosárban lévő termékeket tartalmazza, ahol az 'sku' a termék egyedi cikkszáma, a 'name' a neve, 'price' az egységára kedvezmények nélkül, a 'quantity' pedig a termék darabszáma a kosárban.

A ***$coupons*** tömb kuponkódok listáját tartalmazza, ahol a kuponkódok százalékos (pl. 10%-kal csökkentett) vagy fix áras (pl. -100 Ft) kedvezményeket adhatnak. A kuponok a cikkszámuk alapján az azzal megegyező cikkszámú termékekre alkalmazhatók, illetve amennyiben a kupon cikkszámában az 'all' kulcsszó áll, az a kupon az összes termékre érvényes.

Ezen kívül a ***$tierPrice*** tömb ún. mennyiségi kedvezményeket tartalmaz, ahol a termékek egységára a kosárban lévő darabszámuktól függ (a kuponokhoz hasonlóan a cikkszámok egyezése alapján). Pl. az első mennyiségi kedvezmény elemnél a 002-es cikkszámú termék egységára az eredeti 1649 Ft-ról 1599 Ft-ra csökken, ha legalább 2 db van belőle a kosárban. Amennyiben legalább 4 db van, akkor a termék egységára 1529 Ft-ra változik.

Fontos kitétel, hogy kuponból és mennyiségi kedvezményből is bármennyit létrehozhatunk, de ez utóbbiból egyszerre csak egy érvényesülhet egy adott termékre (a deklarálásuk szerinti első alkalmazható érvényesül), míg kupokonból bármennyi alkalmazható. A kuponok a deklarálásuk sorrendjében egymás után alkalmazódnak.

Elsőként mindig a mennyiségi kedvezményt alkalmazzuk, majd ezután a kuponokat.

A ***getUnitPrice()*** metódus feladata, hogy kiszámolja egy termék végleges egységárát a fenti szabályokat alkalmazva.

Az alábbiak közül melyik megoldás valósítja meg ezt a feladatot? Válaszd ki, hogy a hiányzó részhez melyik kódrészletet kell beilleszteni!

```php
class Cart
{
    private $items = [
        ['sku' => '001', 'name' => 'Termék #1', 'price' => 2999, 'quantity'
=> 2],
        ['sku' => '002', 'name' => 'Termék #2', 'price' => 1649, 'quantity'
=> 5],
        ['sku' => '003', 'name' => 'Termék #3', 'price' => 7819, 'quantity'
=> 1],
    ];

    private $coupons = [
        [
            'sku' => '001',
            'name' => '10% kedvezmény az "Termék #1"-re',
            'code' => 'SALE10',
            'discount' => '10',
            'type' => 'percentage',
        ],
        [
            'sku' => 'all',
            'name' => '500 Ft kedvezmény minden termékre',
            'code' => '500ALL',
            'discount' => '500',
            'type' => 'fixed',
        ],
    ];

    private $tierPrices = [
        [
            'sku' => '002',
            'name' => 'Mennyiségi kedvezmény "Termék #2"-re',
            'prices' => [
                ['quantity' => 2, 'price' => 1599],
                ['quantity' => 4, 'price' => 1529],
            ],
        ],
        [
            'sku' => '002',
            'name' => 'Mennyiségi kedvezmény "Termék #2"-re',
            'prices' => [
                ['quantity' => 2, 'price' => 1559],
                ['quantity' => 5, 'price' => 1499],
            ],
        ],
    ];

    public function getUnitPrice($item) {
        $price = $item['price'];

          /****************/
         /* HIÁNYZÓ RÉSZ */
        /****************/

        return max(0, $price);
    }
}
```

Válasz

A válasz:

```
foreach ($this->tierPrices as $tierPrice) {
    if ($tierPrice['sku'] === $item['sku']) {
        foreach ($tierPrice['prices'] as $tier) {
            if ($item['quantity'] >= $tier['quantity']) {
                $price = $tier['price'];
                break;
            }
        }
    }
}
foreach ($this->coupons as $coupon) {
    if (in_array($coupon['sku'], ['all', $item['sku']])) {
        switch ($coupon['type']) {
            case 'fixed':
                $price -= $coupon['discount'];
                break;
            case 'percentage':
                $price = round($price * (100 - $coupon['discount']) / 100);
                break;
        }
    }
}
```

B válasz:

```
foreach ($this->tierPrices as $tierPrice) {
    if ($tierPrice['sku'] === $item['sku']) {
        foreach ($tierPrice['prices'] as $tier) {
            if ($item['quantity'] >= $tier['quantity']) {
                $price = $tier['price'];
            }
        }
        break;
    }
}
foreach ($this->coupons as $coupon) {
    if (in_array($coupon['sku'], ['all', $item['sku']])) {
        switch ($coupon['type']) {
            case 'fixed':
                $price -= $coupon['discount'];
                break;
            case 'percentage':
                $price = round($price * (100 - $coupon['discount']) / 100);
                break;
        }
        break;
    }
}
```

C válasz:

```php
foreach ($this->tierPrices as $tierPrice) {
    if ($tierPrice['sku'] === $item['sku']) {
        foreach ($tierPrice['prices'] as $tier) {
            if ($item['quantity'] >= $tier['quantity']) {
                $price = $tier['price'];
            }
        }
        break;
    }
}
foreach ($this->coupons as $coupon) {
    if (in_array($coupon['sku'], ['all', $item['sku']])) {
        switch ($coupon['type']) {
            case 'fixed':
                $price -= $coupon['discount'];
                break;
            case 'percentage':
                $price = round($price * (100 - $coupon['discount']) / 100);
                break;
        }
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

D válasz:

```php
foreach ($this->tierPrices as $tierPrice) {
    if ($tierPrice['sku'] === $item['sku']) {
        foreach ($tierPrice['prices'] as $tier) {
            if ($item['quantity'] >= $tier['quantity']) {
                $price = $tier['price'];
            }
        }
        break;
    }
}
foreach ($this->coupons as $coupon) {
    if (in_array($coupon['sku'], ['all', $item['sku']])) {
        switch ($coupon['type']) {
            case 'fixed':
                $price = $item['price'] - $coupon['discount'];
                break;
            case 'percentage':
                $discount = 100 - $coupon['discount'];
                $price = round($item['price'] * $discount / 100);
                break;
        }
    }
}
```

Magyarázat

A C az egyetlen helyes megoldás.

Az A megoldásban az a hiba, hogy egy tier price elemnél csak az első lépcsőt alkalmazza (jelen esetben a 002-es termékre a 2-es darabszámot), utána viszont a további tier price elemeket is alkalmazza (hasonló módon csak az első lépcsőig).

A B megoldás a kupon kódok közül csak az elsőt alkalmazza (jelen esetben tehát az “all” azonosítójú kupon nem érvényesül a 001-es termékre).

Végül a D megoldás a kuponok alkalmazásakor nem a már addig kalkulált egységárral számol, hanem mindig fixen az eredeti egységárral, így a tier price alkalmazása teljesen kimarad (felülíródik), a kuponokból pedig csak a legutolsó érvényesül (jelen esetben tier price-ok nélkül csak az “all” azonosítójú kupon alkalmazódik minden termékre).