WEB-programozás II (5. előadás)
NÉVTEREK

Névterek
- Az 5.3-as verziótól támogatottak.
- Hosszú függvénynevek leváltására és a névütközések elkerülésére lettek bevezetve.

Névterek definiálása
Minden PHP kód névterekbe szervezhető, viszont a névterek csak az alábbi 4 nyelvi elemet foglalják egységbe:
- Osztályok
- Interfészek
- Függvények
- Konstansok

Névtereket a **namespace** kulcsszó használatával lehet definiálni. Névteret csak a forrásfájl elején lehet meghatározni. Egy névteret több forrásfájl is leírhat, illetve egy fájlon belül több névtér is megadható.

Megjegyzés: A **php** és **PHP** nevekkel kezdődő névterek (pl. php/Classes) belső használatra vannak fenntartva. Ezen névterek (és alnévtereik) használata kerülendő!

WEB-programozás II (5. előadás)
NÉVTEREK

Példa:

```
<?php
        namespace MyProject;
        const CONNECT_OK = 1;
        class Connection { /* ... */ }
        function connect() { /* ... */ }
?>
```

Az alábbi kód hibás, mivel névtér csak a fájl elején adható meg:

```
<html>
<?php
        namespace MyProject;
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Több névtér definiálása egy fájlon belül

Lehetőség van több névtér deklarálására egy fájlon belül. Ezt két módon lehet megtenni:

1. Névterek egymás alatti definiálása (nem javasolt):

```
<?php
        namespace MyProject;
        const CONNECT_OK = 1;
        class Connection { /* ... */ }
        function connect() { /* ... */ }

        namespace AnotherProject;
        const CONNECT_OK = 1;
        class Connection { /* ... */ }
        function connect() { /* ... */ }
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

2. Névterek blokkokba zárása:

```
<?php
        namespace MyProject {
                const CONNECT_OK = 1;
                class Connection { /* ... */ }
                function connect() { /* ... */ }
        }

        namespace AnotherProject {
                const CONNECT_OK = 1;
                class Connection { /* ... */ }
                function connect() { /* ... */ }
        }
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Névterek blokkba zárt megadásánál a blokkokon kívül nem szerepelhet PHP kód (ezalól a declare utasítás kivételt képez):

```
<?php
        declare(encoding='UTF-8');
        namespace MyProject {
                const CONNECT_OK = 1;
                class Connection { /* ... */ }
                function connect() { /* ... */ }
        }
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Több névtér megadásánál a globális névtér is elérhető marad (bővítés céljából). Ezt egy név nélküli namespace blokk megadásával lehet bővíteni:

```
<?php
        namespace MyProject {
                const CONNECT_OK = 1;
                class Connection { /* ... */ }
                function connect() { /* ... */ }
        }

        namespace { // globális névtér
                session_start();
                $a = MyProject\connect();
                echo MyProject\Connection::start();
        }
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Alnévterek definiálása

A PHP-ban a névterek egymásba ágyazhatóak (ahogy a fájlrendszerben a könyvtárak), ezzel hierarchiát lehet kialakítani. Az alábbi példában a MyProject névtéren belül található a Sub névtér, amin belül definiálunk egy Level nevű alnévteret:

```
<?php
        namespace MyProject\Sub\Level;
        const CONNECT_OK = 1;
        class Connection { /* ... */ }
        function connect() { /* ... */ }
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

A PHP a következő módon választja ki a megfelelő névteret, és azon belül az osztályt (valamint egyéb elemeket):

Nem minősített névvel (unqualified name)
Ebben az esetben a névterek által befolyásolt elemek az aktuális névtérből lesznek kiválasztva, így ha az aktuális névtér a *\app\core*, akkor az *$a = new Foo();* osztályt a PHP *\app\core\Foo* -ként fogja keresni. Ha az aktuális névtér a globális névtér, akkor a *Foo* osztály lesz kiválasztva.

Ha egy névtér által befolyásolt elem nem található meg az aktuális névtérben, akkor a PHP a globális névtérben kezdi keresni. Az előző példát felhasználva így ha az *\app\core* névtéren belül nem található a *Foo* osztály, akkor a globális *\Foo* osztály kerül példányosításra (amennyiben ez létezik).

Minősített névvel (qualified name)
Minősített névvel ellátott elemnél az aktuális névtérhez képest lesz megállapítva az elem keresési helye. Ha az aktuális névtér a *\app\core*, és történik egy példányosítás, hogy *$a = new utils\Foo();*, akkor a PHP a *\app\core\utils\Foo* osztályt fogja megtalálni.

Teljesen minősített névvel (fully qualified name)
Ekkor az osztályok, interfészek, konstansok és függvények a globális névtérhez képest lesznek megadva. Ha például a *$a = new \app\core\Foo();* utasítással példányosítunk, akkor a *Foo* osztályt a PHP az *\app\core* névtéren belül fogja keresni.

WEB-programozás II (5. előadás)
NÉVTEREK

Példák:

```
<?php
        namespace Foo\Bar;

        /* Nem minősített névvel */
        foo(); // visszatér a Foo\Bar\foo függvénnyel
        foo::staticmethod(); // visszatér a Foo\Bar\foo osztály statikus metódusával

        /* Minősített névvel */
        subnamespace\foo(); // visszatér a Foo\Bar\subnamespace\foo függvénnyel

        /* Teljesen minősített névvel */
        \Foo\Bar\foo(); // visszatér a Foo\Bar\foo függvénnyel
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Lehetőség van a névtereken belül a globális függvények, osztályok és konstansok elfedésére is. Ekkor az eredeti PHP által szolgáltatott függvényeket teljesen minősített névvel, a saját függvényeket nem minősített névvel lehet elérni:

```
<?php
        namespace Foo;
        function strlen() {…}
        const INI_ALL = 3;
        class Exception {…}
        $a = \strlen('hi'); // globális strlen függvény hívása
        $b = new \Exception('error'); // globális Exception osztály példányosítása
        $c = strlen('hi'); // saját strlen függvény hívása
        $d = new Exception('error'); // saját Exception osztály példányosítása
?>
```

WEB-programozás II (5. előadás)

NÉVTEREK

Aktuális névtér lekérdezése

Szükség lehet az aktuális névtér lekérdezésére is.
Ezt a *__NAMESPACE__* konstans segítségével tehetjük meg:

```
<?php
        namespace MyProject;
        echo "", __NAMESPACE__, ""; // outputs "MyProject"
?>
```

Illetve globális névtérben:

```
<?php
        echo "", __NAMESPACE__, ""; // outputs ""
?>
```

Az aktuális névtér egy osztályát a *namespace* kulcsszó segítségével érhetjük el explicit módon:

```
<?php
        namespace MyProject;

        blah\mine(); // MyProject\blah\mine()
        namespace\blah\mine(); // MyProject\blah\mine()
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Névtér álnevek (alias) és névtér importálás

A PHP lehetőséget nyújt a teljesen minősített névvel megadott névterek álnevesítésére. Az álneveken keresztül így könnyebben elérhetőek a kiválasztott névtér osztályai és interfészei.

Három fajta álnév áll rendelkezésre a PHP-ban:

- Osztályok álnévvel ellátása
- Interfészek álnévvel ellátása
- Névterek álnévvel ellátása

Függvények és konstansok esetében az álnevek nem használhatóak.

WEB-programozás II (5. előadás)

NÉVTEREK

Névteret álnévvel a ***use*** kulcsszóval lehet ellátni:

```
<?php
   namespace foo;
   use My\Full\Classname as Another;

   // ez a megadási mód azonos ezzel: use My\Full\NSname as NSname
   use My\Full\NSname;

   // globális osztály importálása
   use ArrayObject;

   $obj = new namespace\Another; // a foo\Another osztály példányosítása
   $obj = new Another; // a My\Full\Classname osztály példányosítása

   NSname\subns\func(); // a My\Full\NSname\subns\func függvény hívása

   $a = new ArrayObject(array(1)); // az ArrayObject osztály példányosítása
   /* a "use ArrayObject" importálás nélkül ez a kód a foo\ArrayObject osztályt
       példányosítaná */
?>
```

WEB-programozás II (5. előadás)
NÉVTEREK

Mivel névterek álnevesítésénél és importálásánál csak teljesen minősített névvel lehet hivatkozni az álnevesíteni vagy importálni kívánt névtérre, ezért ott a kezdő \ nem szükséges (és nem is ajánlott!).

Több névtér megadására is van lehetőség egyetlen *use* kulcsszó segítségével. Ez a következőképp tehető meg:

```
<?php
   use My\Full\Classname as Another, My\Full\NSname;

   $obj = new Another; // a My\Full\Classname osztály példányosítása
   NSname\subns\func(); // a My\Full\NSname\subns\func függvény hívása
?>
```

A névtér álnevesítése és importálása csak a nem minősített és minősített nevekre van érvényben. A teljesen minősített névvel ellátott osztályok és interfészek elérése abszolút módon történik, így ezeket nem befolyásolják.

WEB-programozás II (5. előadás)
NÉVTEREK

Álneveket csak a fájl globális területén lehet megadni. Blokkokon belül fordítási hibát fog okozni, ahogy ez a példa is helytelen:

```
<?php
        namespace Languages;
        class Greenlandic {
                use Languages\Danish;
                ...
        }
?>
```

A névtér álnevek nem öröklődnek fájl include-olásánál!
Ezek használatához minden fájl elején meg kell adni a használni kívánt álneveket!

WEB-programozás II (5. előadás)
NÉVTEREK

Névtér feloldási szabályok

Definíciók

Nem minősített név (unqualified name):

azonosító névtér elválasztó nélkül, pl. *Foo*.

Minősített név (qualified name):

azonosító névtér elválasztóval, pl. *Foo**Bar*.

Teljesen minősített név (fully qualified name):

azonosító, amely névtér elválasztóval kezdődik, pl. *Foo**Bar*.
Teljesen minősített név továbbá a *namespace**Foo* megadási mód is.

WEB-programozás II (5. előadás)
NÉVTEREK

Feloldási szabályok

1. Teljesen minősített névvel ellátott függvények, osztályok és konstansok fordítási időben kerülnek kiértékelésre.
2. Minden minősített és nem minősített név szerkesztési időben kerül fordításra. Ezek megadásánál a névtér álnevek figyelembe lesznek véve. Példa:
   Az *A/B/C* névtér *C* álnéven van importálva, egy kódrészlet hivatkozik a *C\D\e()* függvényre, akkor a *A\B\C\D\e()* függvény lesz meghívva.
3. Egy névteren belül a minősített névvel ellátott névtér feloldás ha nem esik névtér importálás hatókörébe, akkor az aktuális névtér lesz a feloldás kiindulópontja. Példa:
   Ha az aktuális névtér a *A\B*, és a *C\D\e()* függvény kerül meghívásra, akkor
   a *A\B\C\D\e()* függvény lesz ami meghívódik.
4. Nem minősített névvel ellátott névtér feloldásnál ha a keresett osztály egy álnévre hivatkozik, akkor az álnév lesz feloldva. Példa:
   Ha az *A/B/C* névtér álneve *C*, akkor a *new C()* az *A\B\C()* osztályt fogja példányosítani.
5. Nem minősített névvel ellátott függvény esetében a névtér megállapítása a következő sorrend alapján megy végbe (az aktuális névtér legyen *A\B*):
   1. Ha a megadott nevű függvény definiálva van az aktuális névtérben, akkor az meghívásra kerül (ha a függvény neve *foo*, akkor a *A\B\foo* függvény kerül meghívásra)
   2. Ha a függvény nincs definiálva az aktuális névtérben, akkor a globális névtérből lesz kiválasztva.
6. Ha egy nem minősített névvel vagy minősített névvel ellátott függvény vagy osztály kerül meghívásra (például az *A\B* névteren belül a *C\D*osztály példányosítása), akkor
   1. A névtér feloldás az aktuális névtérből indul (a példában *A\B\C\D*)
   2. Betöltés megkísérlése a megtalált névtérből

   Egy névtéren belül a globális névtér egy függvényének eléréséhez teljesen minősített névvel kell hivatkozni.

WEB-programozás II (5. előadás)
PHPNG (PHP Next Generation) és PHP7

Tapogatás a sötétben:

- Kb. 2 évet "pazaroltak" egy JIT fordító fejlesztésére a PHP-5.5-re.
- 10%-kal gyorsul a bench.php.
- Elhanyagolható gyorsítás a valós alkalmazásokon (1% a Wordpress-en).

Következtetések:

- Nagyon gyors kódot generálhatunk, ha ismerünk minden érték futás közbeni típusát.
- A valós alkalmazásokban nem lehet megjósolni sok futás közbeni típusokat.
- Azokban az esetekben, amikor valamilyen típust várunk, de nem vagyunk biztosak 100%-ban, a belső reprezentációhoz PHP- kompatibilis adatstruktúrákat kell használnunk, ami nem hatékony kódot eredményez.

WEB-programozás II (5. előadás)
PHPNG (PHP Next Generation) és PHP7

PHPNG (PHP Next Generation)

Célok:

- Refactoring.
- Fő cél - új teljesítményszint elérése, amely alapja a jövőbeli fejlesztéseknek.
- Elválik a „main-stream” PHP-től 2014 januárjában.
- Nincs új funkció a felhasználók számára (csak belsőleg).
- Tartsa a 100%-os kompatibilitást a PHP viselkedésében.

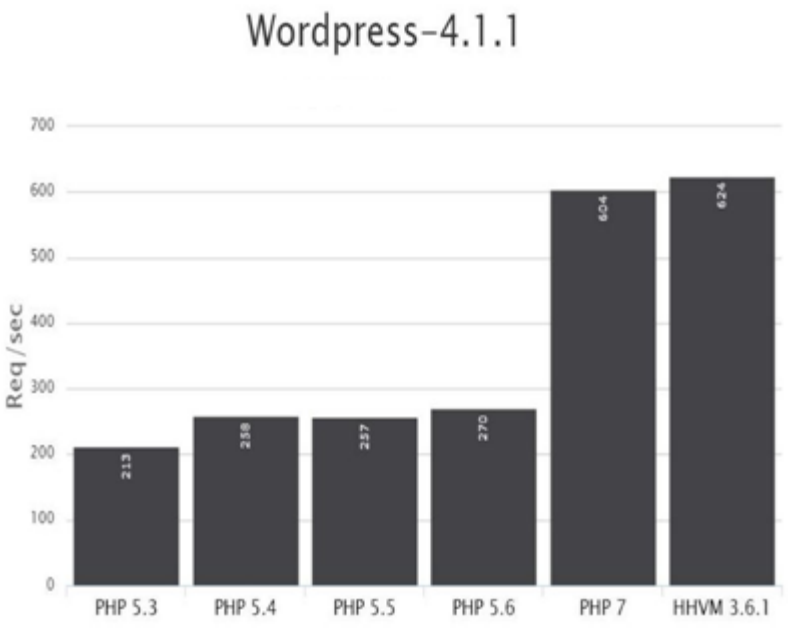
Fejlesztés:

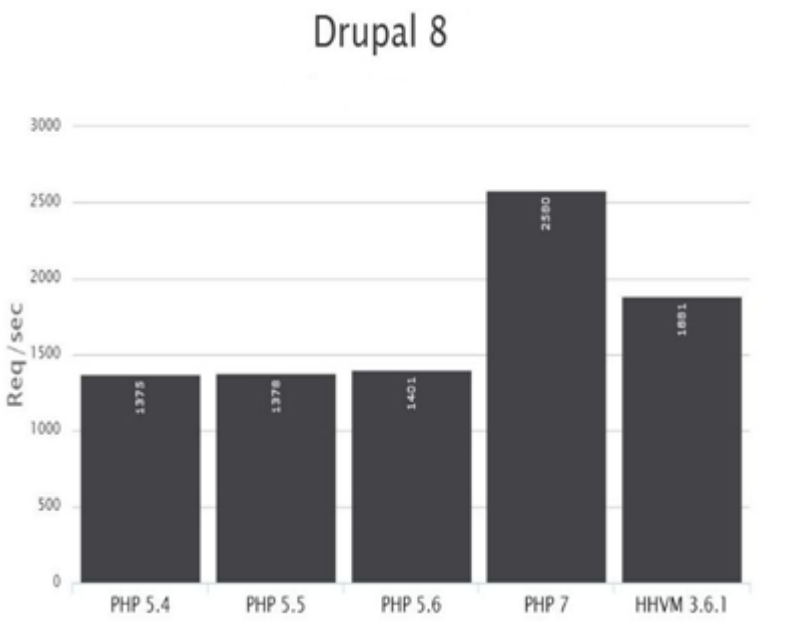
- Két hét alatt megírták a magot.
- A következő két hétben elindult a bench.php futtatása.
- A következő 1,5 hónap elteltével kompatibilis lett a Wordpress programmal.
- Egy hónappal később (május elején) megindították a projektet.
- Visszacsatolása a „main-stream”-het, mint a PHP7 alapja 2014 augusztusában.

WEB-programozás II (5. előadás)

PHP7

A PHP hatékonyság:





HHVM (HipHop Virtual Machine) – A Facebook által fejlesztett, JIT fordítón alapuló, PHP kódot végrehajtó virtuális gép. Először a PHP forrást fordítja le HipHop bájtkódra (HHBC), majd dinamikusan a bájtkódot gépi kódra.

WEB-programozás II (5. előadás)
PHP 7 ÚJDONSÁGAI

- Jobb teljesítmény: A PHPNG kód összevonásával a PHP7 kétszer olyan gyors mint PHP 5.
- Alacsonyabb memória felhasználás: Az optimalizált PHP 7 kevesebb erőforrást használ.
- Konzisztens 64 bites támogatás: Egységes támogatás a 64 bites architektúrákhoz.
- Javított kivétel hierarchia - A kivétel hierarchia javul.
- Számos végzetes hiba átalakul kivétellé: A kivételek tartománya nő, amely számos végzetes kimenetelű hibát lefed.
- Biztonságos véletlenszám-generátor: Új biztonságos véletlenszám-generátor API bevezetése.
- Az elavult SAPI-k és kiterjesztések eltávolítása: Különböző régi és nem támogatott SAPI-k és bővítmények eltávolításra kerültek.
- A null coalescing operátor (??): Új null coalescing operátor bevezetésre került.
- Scalar típusok deklarációja: Támogatás a visszatérési értékek és a paraméterek scalar típusainak a megadásához.
- Névtelen osztályok: Anonim osztályok támogatásának a bevezetése.
- Zero cost asserts: Assert (ellenőrzések) használhatók fejlesztésre és hibakeresésre, de nulla költséggel produkciós (éles) környezetben.

A PHP 7 új Zend Engine 3.0 technológiájával a PHP 5.6-hoz képest csaknem duplájára növeli az alkalmazások teljesítményét és 50%-kal csökkenti a memória felhasználást. A PHP 7 lehetővé teszi több konkurens felhasználó kiszolgálását további hardver szükséglete nélkül. A PHP 7-et a mai munkaterhelést figyelembe véve tervezték és fejlesztették.

WEB-programozás II (5. előadás)

PHP 7 - Scalar Típusok Deklarációja

A PHP 7-ben bevezetésbe került a Scalar típusok deklarációját a paraméterekben.

A deklaráció lehet:

- kényszerítő (coercive) – a kényszerítő az alapértelmezett mód, nem kell megadni.
- szigorú (strict) – a szigorú mód kifejezetten kell megadni.

A következő paraméter típusok alkalmazhatók:

int, **float**, **bool**, **string**, **interfaces**, **array**, **callable**

Példák:

Kényszerítő (coercive) mód:

```
<?php

function sum(int ...$ints) {
        return array_sum($ints);
}
print(sum(2, '3', 4.1));
?>
```

Output:

```
9
```

Szigorú (strict) mód:

```
<?php
declare(strict_types=1);
function sum(int ...$ints) {
        return array_sum($ints);
}
print(sum(2, '3', 4.1));
?>
```

Output:

```
Fatal error: Uncaught TypeError: Argument 2 passed
to sum() must be of the type integer, string given, ...
```

WEB-programozás II (5. előadás)

PHP 7 - Visszatérési Típus Megadása

A PHP 7-ben bevezetésbe került a visszatérési (return) típus deklarációja. A deklaráció meghatározza azt az értéket, amelyet a függvénynek vissza kell adnia.

A következő visszatérési típusok alkalmazhatók:

int, **float**, **bool**, **string**, **interfaces**, **array**, **callable**

Példák:

Érvényes visszatérési típus:

```
<?php
declare(strict_types=1);
function returnIntValue(int $value): int
{
        return $value;
}
print(returnIntValue(5));
?>
```

Output:
5

Érvénytelen visszatérési típus:

```
<?php
declare(strict_types=1);
function returnIntValue(int $value): int
{
        return $value + 1.0;
}
print(returnIntValue(5));
?>
```

Output:
Fatal error: Uncaught TypeError: Return value of returnIntValue() must be of the type integer, float returned...

WEB-programozás II (5. előadás)

PHP 7 - Null Coalescing Operator

A PHP 7-ben bevezették a null coalescing operátort (**??**). Az új operátor felváltja a három operandusú operátort (**?:**) és az **isset()** függvény kombinációját.

A null coalescing operátor egy két operandusú operátor, amely visszaadja az első operandust ha létezik és nem null, különben a második operandust adja vissza.

Példák:

```
<?php
// fetch the value of $_GET['user'] and returns 'not passed' if username is not passed
$username = $_GET['username'] ?? 'not passed';
print($username); print("<br/>");
// Equivalent code using ternary operator
$username = isset($_GET['username']) ? $_GET['username'] : 'not passed';
print($username); print("<br/>");
// Chaining ?? Operation (a ?? operátor jobbra asszociál)
$username = $_GET['username'] ?? $_POST['username'] ?? 'not passed';
print($username);
?>
```

Output:

```
not passed
not passed
not passed
```

WEB-programozás II (5. előadás)

PHP 7 – Spaceship Operator

A PHP 7-ben bevezetésre került az „űrhajó” (spaceship) operátor, amely két kifejezésre összehasonlításra alkalmazható, és -1, 0 vagy 1 értéket ad vissza, ha az első kifejezés kisebb, egyenlő vagy nagyobb, mint a második kifejezés.

Példák:

```
<?php
//integer comparison
print( 1 <=> 1);print("<br/>");
print( 1 <=> 2);print("<br/>");
print( 2 <=> 1);print("<br/>");
print("<br/>");
//float comparison
print( 1.5 <=> 1.5);print("<br/>");
print( 1.5 <=> 2.5);print("<br/>");
print( 2.5 <=> 1.5);print("<br/>");
print("<br/>");
//string comparison
print( "a" <=> "a");print("<br/>");
print( "a" <=> "b");print("<br/>");
print( "b" <=> "a");print("<br/>");
?>
```

Output:

```
0
-1
1

0
-1
1

0
-1
1
```

WEB-programozás II (5. előadás)

PHP 7 – Konstans Tömbök

Az array-konstansok most definiálhatók **define ()** függvénnyel. A PHP 5.6-ban csak a **const** kulcsszó segítségével lehetett megadni konstans tömböt.

Példa:

```
<?php
        //define a array using define function
        define( 'animals', [
                'dog',
                'cat',
                'bird'
        ]);
        print( animals[1] );
?>
```

Output:

cat

WEB-programozás II (5. előadás)

PHP 7 – Névtelen osztályok

Az anonim osztályok mostantól **new class** használatával definiálhatók. Névtelen osztály használható egy teljes osztály definíciója helyett.

Példa:

```
<?php
interface Logger {
    public function log(string $msg);
}
class Application {
    private $logger;
    public function getLogger(): Logger {
        return $this->logger;
    }
    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }
}
```

```
$app = new Application;
$app->setLogger(new class implements Logger
{
    public function log(string $msg) {
        print($msg);
    }
});
$app->getLogger()->log("My first Log
Message");
?>
```

Output:

My first Log Message

WEB-programozás II (5. előadás)

PHP 7 – Closure::call()

A **Closure::call()** metódus került bevezetésre röviden hozzákötni egy objektum hatókörét egy closure-hez (lezáráshoz). Ez jóval gyorsabb a teljesítményén, mint a PHP 5.6 **bintTo()** függvénye.

Példa:

PHP 7 előtt:

```
<?php
    class A {
        private $x = 1;
    }
    // Define a closure
    $getValue = function() {
        return $this->x;
    };
    // Bind a clousure
    $value = $getValue->bindTo(new A, 'A');
    print($value());
?>
```

Output:
1

PHP 7:

```
<?php
    class A {
        private $x = 1;
    }
    // Define a closure
    $value = function() {
        return $this->x;
    };
    // Bind a closure and invoke
    print($value->call(new A));
?>
```

Output:
1

WEB-programozás II (5. előadás)

PHP 7 – Filtered unserialize()

A PHP 7 bevezeti a filtered **unserialize**() függvényt a nagyobb biztonság érdekében nem megbízható objektumok deszerializálásában. Használatával elkerülhető a kód injekció.

Példa:

```
<?php
class MyClass1 {
    public $obj1prop;
}
class MyClass2 {
    public $obj2prop;
}

$obj1 = new MyClass1();
$obj1->obj1prop = 1;

$obj2 = new MyClass2();
$obj2->obj2prop = 2;

$serializedObj1 = serialize($obj1);
$serializedObj2 = serialize($obj2);
```

```
// default behaviour that accepts all classes,
// second argument can be ommited.
// if allowed_classes is passed as false, unserialize
// converts all objects into __PHP_Incomplete_Class object
$data = unserialize($serializedObj1 , ["allowed_classes" => true]);

// converts all objects into __PHP_Incomplete_Class object
// except those of MyClass1 and MyClass2
$data2 = unserialize($serializedObj2 , ["allowed_classes" =>
                                        ["MyClass1", MyClass2"]]);
print($data->obj1prop);
print("<br/>");
print($data2->obj2prop);
?>

Output:
1
2
```

WEB-programozás II (5. előadás)

PHP 7 – CSPRNG

A PHP 7-ben két új funkciót vezettek be egész számok és stringek kriptográfiailag biztonságos generálására keresztplatform módon.

random_bytes() generál egy tetszőleges hosszúságú kriptográfiailag biztos véletlen bájtsorozatot (felhsználási példák: sók, kulcsok, inicializáló vektorok.

Syntax
string random_bytes (int $length)

Parameters
length – a generált karaktersorozat hossza bájtokban.

Return Values
Egy a kért hosszúságú kriptográfiailag biztos bájtsorozatot tartalmazó string.

Errors/Exceptions
- Dob egy kivételt, ha nem talál megfelelő forrást a véletlen generálásra.
- Dob egy TypeError érvénytelen paraméter esetén.
- Dob egy Error érvénytelen hossz megadása esetén.

Example

```
<?php
$bytes = random_bytes(5);
print(bin2hex($bytes));
?>
```

Output:
54cc305593

random_int() generál kriptográfiai véletlen számokat (alkalmazhatók „elfogulatlan” eredmények elérésére).

Syntax
int random_int (int $min , int $max)

Parameters
min – legkisebb visszaadható érték (≥ **PHP_INT_MIN**).
max – legnagyobb visszaadható érték (≤ **PHP_INT_MAX**).

Return Values
Egy kriptográfiailag biztos **min** és **max** közötti egész számot.

Errors/Exceptions
- Dob egy kivételt, ha nem talál megfelelő forrást a véletlen generálásra.
- Dob egy TypeError érvénytelen paraméter esetén.
- Dob egy Error, ha **max** < **min**

Example

```
<?php
print(random_int(100, 999));
print(random_int(-1000, 0));
?>
```

Output:
614
-882

WEB-programozás II (5. előadás)

PHP 7 – use utasítás

A PHP7-ben egyetlen **use** utasítás használható osztályok, függvények és konstansok importálására azonos névtérből.

Példa:

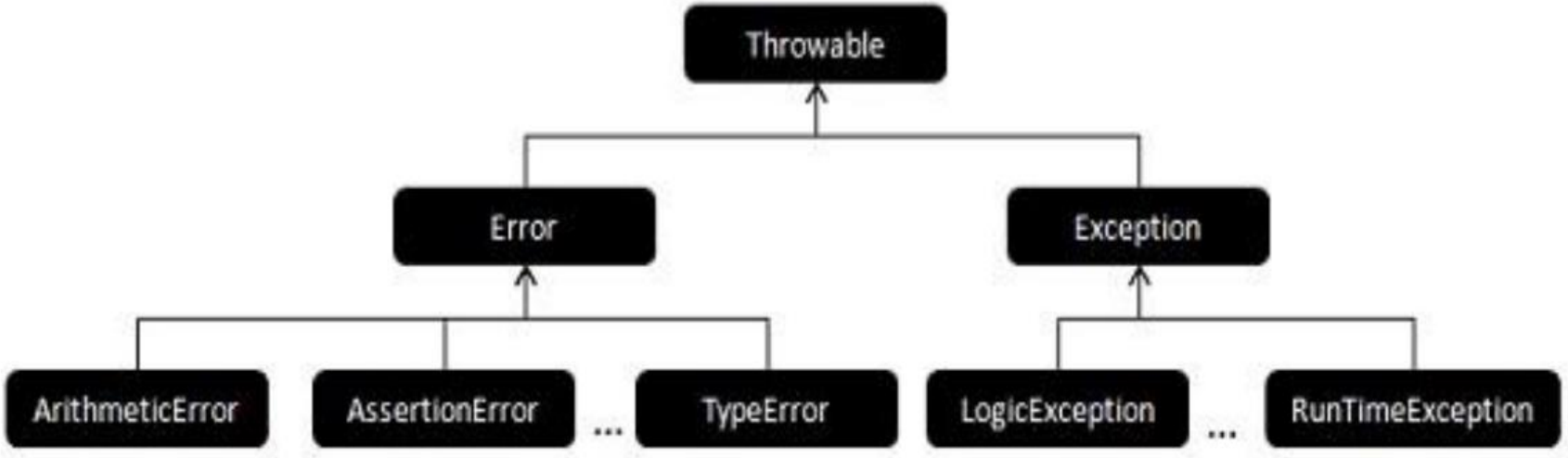
```
<?php
// PHP 7 előtt
use com\tutorialspoint\ClassA;
use com\tutorialspoint\ClassB;
use com\tutorialspoint\ClassC as C;
use function com\tutorialspoint\fn_a;
use function com\tutorialspoint\fn_b;
use function com\tutorialspoint\fn_c;
use const com\tutorialspoint\ConstA;
use const com\tutorialspoint\ConstB;
use const com\tutorialspoint\ConstC;
// PHP 7-ben
use com\tutorialspoint\{ClassA, ClassB, ClassC as C};
use function com\tutorialspoint\{fn_a, fn_b, fn_c};
use const com\tutorialspoint\{ConstA, ConstB, ConstC};
?>
```

WEB-programozás II (5. előadás)

PHP 7 – Hibakezelés

A PHP 7-ben a hibakezelés és a hibajelzés megváltozott. A legtöbb hiba **Error** kivételt dob. A kivételekhez hasonlóan ezek az **Error** kivételeket felfelé továbbítja, amíg el nem érik az első megfelelő **catch** blokkot. Ha nincs olyan blokk, akkor a **set_exception_handler ()** függvényben beállított alapértelmezett kivételkezelő kerül hívásra. Ha nincs alapértelmezett kivételkezelő, akkor a kivételből **fatal error lesz**, és hagyományos hibáként lesz kezelve.

Mivel az **Error** kivétel nem az **Exception** kivételből származik, a PHP 5-ben minden kivétel kezelésére használt **catch (Exception $ e) {...}** blokk nem fogja kezelni az ilyen hibákat. A **fatal error** kezeléséhez egy **catch (Error $ e) {...}** blokk vagy egy **set_exception_handler ()** kezelő szükséges.



WEB-programozás II (5. előadás)

PHP 7 – Hibakezelés

Példa:

```
<?php
    class MathOperations
    {
        protected $n = 10;
        // Try to get the Division by Zero error object and display as Exception
        public function doOperation(): string
        {
            try {
                $value = $this->n % 0;
                return $value;
            }
            catch (DivisionByZeroError $e) {
                return $e->getMessage();
            }
        }
    }

    $mathOperationsObj = new MathOperations();
    print($mathOperationsObj->doOperation());
?>
```

Output:
Modulo by zero

WEB-programozás II (5. előadás)

PHP 7 – Integer osztás

A PHP 7-ben bevezetésre kerül az **intdiv()** függvény, amely végrehajtja operanduszainak egész osztását és egész eredményt ad vissza.

Example

```
<?php
    $value = intdiv(10,3);
    var_dump($value);
    echo "<br>";
    print($value);
?>
```

Output:

```
int(3)
3
```

WEB-programozás II (5. előadás)

PHP 7 – Munkamenet opciói

A PHP 7-ben a **session_start ()** függvény egy sor opciót fogad el, amelyek felülbírálják a munkamenetekre a **php.ini** –ben beállított konfigurációt.

Egy ilyen opció a PHP 7-ben bevezetett, alapértelmezés szerint beállított **session.lazy_write** opció, amely a munkamenet fájl azonnali módosítását írja elő, ha megváltoznak a munkamenet adatai.

Egy másik hozzáadott opció **read_and_close**, amely azt jelzi, hogy a munkamenet adatainak olvasása után azonnal le kell zárni a munkamenetet.

Példa:

A **session.cache_limiter** privátra állítása (a kliensnek szabad tárolni gyorsító tárban az oldal tartalmát, de a közbülső proxy szervereknek tilos), és a munkamenet azonnali bezárása a beolvasása után.

```
<?php
    session_start([
        'cache_limiter' => 'private',
        'read_and_close' => true,
    ]);
?>
```

WEB-programozás II (5. előadás)

PHP 7 – Elavult funkciók, eltávolított bővítmények és SAPI-k

Elavult funkciók:

- PHP 4 típusú osztály konstruktorok.
- Nem statikus metódus statikus hívása.
- A **password_hash()** só (salt) opciója. Automatikusan generálja a sót.
- **capture_session_meta** SSL környezet opciója.

Eltávolított bővítmények:

- ereg
- mssql
- mysql
- sybase_ct

Eltávolított SAPI-k:

- aolserver
- apache
- apache_hooks
- apache2filter
- caudium
- continuity
- isapi
- milter
- nsapi
- phttpd
- pi3web
- roxen
- thttpd
- tux
- webjames