- Tipos de datos
- Estructuras de control de flujo
- Funciones

Desarrollo Web con PHP

Tipos de datos

- **Versiones anteriores a la 7:**
  - `callable`
  - `array`
  - `object`
- **En la versión 7:**
  - `string`
  - `int`
  - `float`
  - `bool`

Tipos de datos. Arreglos

Hay dos tipos de arreglos:

1. **Arreglos con índices**

```
$a = [‘uno’, ‘dos’,’tres’,’cuatro’];
echo $a[3]; // cuatro
```

2. **Arreglos con llaves (asociativos)**

```
$b = [‘1’ => ‘Lunes’, ‘2’ => ‘Martes’, ‘3’=>Miércoles];
echo $b[‘3’]; // Miércoles
```

Las llaves son cadenas y los valores de cualquier tipo.

Tipos de datos. Cadenas

Hay 4 maneras de definir una cadena, nos enfocaremos en 2:

1. **Comillas simples ‘ ‘;**

2. **Comillas dobles “ ”;**
   - Los nombre de variables serán extendidos.
   Ejemplo:

```
$var = "uno";
echo "cero $var"; // cero uno
echo ‘cero $var’; // cero $var
```

Tipos de datos. Booleanos

Los siguientes valores se pueden interpretarse como falso:

- **FALSE**
- **int 0**
- **float 0.0**
- **string “” ‘’ ó ‘0’**
- **array []**
- **NULL**

Osea FALSE == valor regresa 1(TRUE),
dónde valor es cualquiera de los descritos anteriormente.

Operadores de comparación

- **Equivalencia (==): devuelve verdadero si dos variables tienen valores iguales pero para compararlas fue necesario hacer algún cast.**

- **Identidad (===): devuelve verdadero si las variables son del mismo tipo y además uno a unos sus valores son iguales. Devuelve falso en otro caso.**

TIPOS DE DATOS

Las validaciones de los tipos se pueden dar de las siguientes dos maneras:

1. **Coercitiva (por defecto)**
2. **Estricta**

Y sólo aplica para los parámetros y valores de retorno de las funciones/métodos.

TIPOS DE DATOS

Las validaciones de los tipos se pueden dar de las siguientes dos maneras:

1. **Coercitiva (por defecto)**
    a. **Hace castings implícitos. Se forza el valor, en caso de ser posible.**
    b. **Aplica para parámetros y valores de retorno.**

2. **Estricta**

TIPOS DE DATOS

Las validaciones de los tipos se pueden dar de las siguientes dos maneras:

1. **Coercitiva (por defecto)**
2. **Estricta**
    a. **Debe ser especificada una cláusula de declare dentro del archivo.**

```
declare(strict_types=1);
```

    b. **Afecta tanto a los parámetros como a los valores de retorno.**
    c. **Espera un tipo igual al de la declaración. (Excepción int->float)**
    d. **El efecto estricto de las funciones es dado por el archivo dónde fueron declaradas.**
    e. **Si la función fue declarada en ambiente estricto y usada en ambiente no estricto, se respetará el ambiente llamador.**
    f. **Deben cachar la excepción de TypeError**

tipos de DAtos

Las validaciones de los tipos se pueden dar de las siguientes dos maneras:

1. **Coercitiva (por defecto)**
2. **Estricta**

Corolario 1: La manera en que se controla la validación es especificada en cada archivo.

Corolario 2: De nuevo, no siempre entre más, mejor.

Declaración por omisión

En los parámetros de las funciones se puede escribir un valor por omisión

```
function bla1(string $cadena = ‘pops’){



}
```

Si el valor por omisión es null, no es recomendable que haya más de uno y se encuentre al final.

Declaración por omisión

```
function bla2($cadena=null, $dos, $tres = null,
$cuatro=null, $cinco){

    return ['uno' => $cadena

            , 'dos' => $dos

            , 'tres' => $tres

            , 'cuatro' => $cuatro

            , 'cinco' => $cinco]; }

var_dump(bla2(7,3));
```

Declaración por omisión

```
array(5) {
  ["uno"]=> int(7)
  ["dos"]=> int(3)
  ["tres"]=>  NULL
  ["cuatro"]=>  NULL
  ["cinco"]=>  NULL
}
```

Y Warning de parámetro 5 faltante.

Número de parámetros variables

Las funciones pueden recibir un número de parámetros variables. Todos ellos se almacenan en un arreglo.
Se utiliza el token (partícula del lenguaje)

```
function bla3(...$parametros){

}
```

Valor de regreso Void

Valor de regreso Void

Estructuras de control de flujo

- **¿Qué se entiende por flujo?**
- **¿Qué estructuras de control de flujo conocen?**

Estructuras de control de flujo

El flujo se puede definir la secuencia de instrucciones que se ejecutan.

Veremos:

1. **Condicionales.**
    a. `if then else`
    b. `switch case`
2. **Ciclos.**
    a. `for`
    b. `for each`
    c. `while`

CONDICIONALES. IF, ELSE IF Ó ELSEIF

Sintaxis:

```
if(cond1){

}else if(cond2){

}else if(cond3){

}else{

}
```

Sintaxis:

```
if(cond1){

}elseif(cond2){

}elseif(cond3){

}else{

}
```

CONDICIONALES. IF, ELSE IF Ó ELSEIF

```
Sintaxis:

if(cond1):

elseif(cond2):

elseif(cond3):

else:

endif;
```

CONDICIONALES. SWITCH CASE

Sintaxis

```
switch(expresion_simple){

    case Caso1:
        break;

    case Caso2:
        break;

    default:
        break;

}
```

```
switch(expresion_simple):

    case Caso1:
        break;

    case Caso2:
        break;

    default:
        break;

endswitch;
```

CONDICIONALES. SWITCH CASE

expresion_simple es de tipo cadena, entero y flotante.

Ciclos. For

Sintaxis

```
for(exp1; exp2; exp3){



}
```

```
for(exp1; exp2; exp3):



endfor;
```

exp1: valor inicial del iterador.
exp2: condición que se tiene que cumplir para cada iteración.
exp3: incremento del iterador.

Ciclos. WHILE

Sintaxis

```
while(cond){



}
```

```
while(cond):



endwhile;
```

Ciclos. For EACH

Sintaxis

```
foreach(array as $val){

}

//Para arreglos asociativos
foreach(array as $key => $val){

}
```

También aplica la versión de los dos puntos.

Ciclos. For EACH

Si los valores del arreglo fueron modificados dentro del ciclo, al terminar el ciclo no se verán reflejado esos cambios.

Para modificar el arreglo dentro del ciclo hay que usar referencias:

```
foreach(array as &$val){

}

unset($val); //borramos la referencia.
```

Funciones: Manejo de Excepciones

- **¿Qué es una excepción?**
- **¿A qué se refiere el evento de lanzar una excepción?**
- **¿Qué procedimiento tiene que cachar una excepción?**

Funciones: Manejo de Excepciones

- **¿Qué es una excepción?**
  Una excepción es la combinación de valores de entrada que producen un comportamiento anómalo en la función.

- **¿A qué se refiere el evento de lanzar una excepción?**
  Una función avisa que, bajo ciertos valores de entrada, tendrá un comportamiento anómalo.

- **¿Quién tiene que cachar una excepción?**
  Cuando una segmento de código hace uso de un función que lanzará una excepción.

Funciones: Manejo de Excepciones

- **set_exception_handler($funcion):** sirve para poner un manejador de excepciones no cachadas personalizado.

- **Lanzar excepciones:**

```
if(división entre 0):
    throws new Exception('Division entre 0.');
```

- **Cachar excepciones:**

```
try{
    //código que puede lanzar excepciones
}catch(Excepcion_tipo1){ //Manejo de excepcion tipo1
}catch(Excepcion_tipo2){ //Manejo de excepcion tipo2
}finally { //Siempre se hará}
```

Funciones: Manejo de Excepciones

- **Pueden crearse Excepciones personalizadas.**

```
class Div extends Exception{

    public function __construct($message,$code = 0,
                                Exception $previous = null) {

        parent::__construct($message, $code, $previous);

    }

}
```

Ejercicios:

- **Abre el archivo funciones.php.**
- **Lee el comentario de las funciones.**
- **Impleméntalas con tipificación estricta.**