Scalar type hints in PHP

Ilia recently brought up the topic of scalar type hints again. I would love scalar type hints, but a sensible implementation is not easy. I summarize some approaches in this post and talk about the problems they raise.

Requirements for scalar type hints

Different users of PHP have different requirements regarding type hints. In following I will try to categorize these needs and ideas.

Strongly typed languages

People coming from strongly typed languages (such as Java and friends) would want that a type hint really ensures a given value is of a certain type:

```
<?php
function foo( int $bar )
{
    return $bar + 1;
}
// Working, result 24
foo( 23 );
// Error
foo( "23" );
// Error
foo( "bar" );
?>
```

This is the most obvious behaviour, if you come from a a strongly typed language. It is clear, straight forward and intuitive if you are used to strong typing. The drawback here is, that PHP is not strongly typed, by intention. It is completly valid to juggle with types like this:

```
<?php
$foo = "23" + 42 + 23.42;
?>
```

The result is of type **float** and has the value **88.42**.

The PHP way

The original PHP developer would expect a different behavior. He knows that scalar types can all be cast around and that PHP has somewhat intelligent knowledge about type casting:

```
<?php
function foo( int $bar )
{
    return $bar + 1;
}
// Working, result 24
foo( 23 );
// Working, result 24
foo( "23" );
// Working, result 1
foo( "bar" );
?>
```

While this would be feasible for PHP, since it reflects the typical behavior of variables, its still against the idea of type hints. You could pass any scalar type to the function above, no matter if the resulting cast is sensible or not.

This basically makes the type hint above useless, you could also just introduce a **scalar** type hint to ensure that the received value is not an object/array/resource. Also I nice addition, IMO.

Potential solutions

Ilia suggested the first solutions presented in this section, the others are just dumps of my brain about this issue.

The numeric way

Ilia now suggests an addition to the strong way of type hinting, by introducing additional **numeric** and **scalar** type hints. As I understood his post, the behavior would then be as follows:

```
<?php
function foo( int $bar )
{
    return $bar + 1;
}
function bar( numeric $baz )
```

```
{
    return $baz + 1;
}
// Working, result 24
foo( 23 );
// Working, result 24
bar( 23 );
// Error
foo( "23" );
// Working, result 24
bar( "23" );
// Error
foo( "bar" );
// Error
bar( "bar" );
// Error
foo( 42.23 );
// Working, result 43.23
bar( 42.23 );
?>
```

Well, this obviously solves the problems stated in the previous section. However, you still get a problem, if you want to ensure working with an integer value in the **bar()** function. This one also accepts floats and strings containing floats, because of:

```
<?php
var_dump( is_numeric( 23.42 ), is_numeric( "23.42" ) );
// bool(true)
// bool(true)
?>
```

The casting way

The basic problem with Ilias approach is, that you can not determine receiving an integer value (in whatever format):

```
<?php
function foo ( numeric $bar )
{
    return $bar + 1;
}
// Working, result 24
```

```
foo( 23 );
// Working, result 24
foo( "23" );
// Working, result 43.23
foo( 42.23 );
// Error
foo( "bar" );
?>
```

The first calls bevavior is intentionally, as well as the third one. However, the second one breaks the idea of working in a somewhat type safe environment, since **foo()** may now also return **float** instead of **integer**.

One idea here could be to enable casting in type hints:

```
<?php
function foo (scalar (int) $bar )
{
    return $bar + 1;
}
// Working, result 24
foo( 23 );
// Working, result 24
foo( "23" );
// Working, result 43
foo( 42.23 );
// Working, result 1
foo( "bar" );
?>
```

Instead of returning a **float**, this method will always return an **int**. However, this variant has almost the same problem as just requiring **scalar**, since you can still pass **string** values to it, that do not contain numbers at all:

```
<?php
var_dump( (int) 42.23, (int) "bar" );
// int(42)
// int(0)
?>
```

The checking way

Another idea could be to add information to the **scalar** type hint, which type is actually expected. Doing this, PHP could become a bit more intelligent in casting types, to ensure a correct type is received by a funtion:

```
<?php
function foo( scalar(int) $bar )
{
    return $bar + 1;
}
// Working, result 24
foo( 23 );
// Working, result 24
foo( "23" );
// Error
foo( 42.23 );
// Error
foo( "bar" );
?>
```

In this case I would expect the type hint to check if the given scalar can safely be interpreted as an **int**. **42.23** and **"bar"** are obviously not interpretable as **int**, while **"23"** is. The actual value of **$bar** should be an int in this case. Although keeping the actually passed value would make no difference here, since PHP would cast it on the first use as an **int**.

Conclusion

PHP really has a paradigm conflict here: On the one hand, we are usually open for most things users want to do (look at **goto**) and add them where needed. This would mean to basically add all variants above to let people do things the way they want. On the other hand, adding strong type hints and **scalar** (with and without additional info) and **numeric** and casts results in endless confusion and violates the KISS principle PHP has always been based on.

This basically means that there is no optimal solution that satisfies everyones needs, as so often.

My personal preference would be to go with Ilias solution. I generally favor that people check if values are correct before they submit them to my code, so I can pretty much live with strong type hints. Although I want **int** to be valid is **float** and any scalar type to be a valid **string**. ;) Having **numeric** is a nice addition and I have been looking for **scalar** for a long time now, to just ensure neither an object nor an array is passed.

One big problem I still have with type hints is the error handling. Type hints currently raise an **E_RECOVERABLE_ERROR** which is almost a fatal. There is no "nice" solution to catch these in your code. This should not make any problems if you only deal with your own code, but when using library code, its hard to catch such errors coming from it. The hackish way of switching from error to exception looks like this:

```
<?php
set_error_handler(
    function ( $errno, $errstr /* ... */ )
    {
        throw new Exception( $errstr );
    },
    E_RECOVERABLE_ERROR
);
function foo( array $bar )
{
    return $bar[] = true;
}
foo( "bar" );
?>
```

Note that this is just a basic sketch and a real implementation should be worked out some more. This basically works, but is not nice. So, my wish for PHP 6: Please make type hints throw an exception. Thanks.