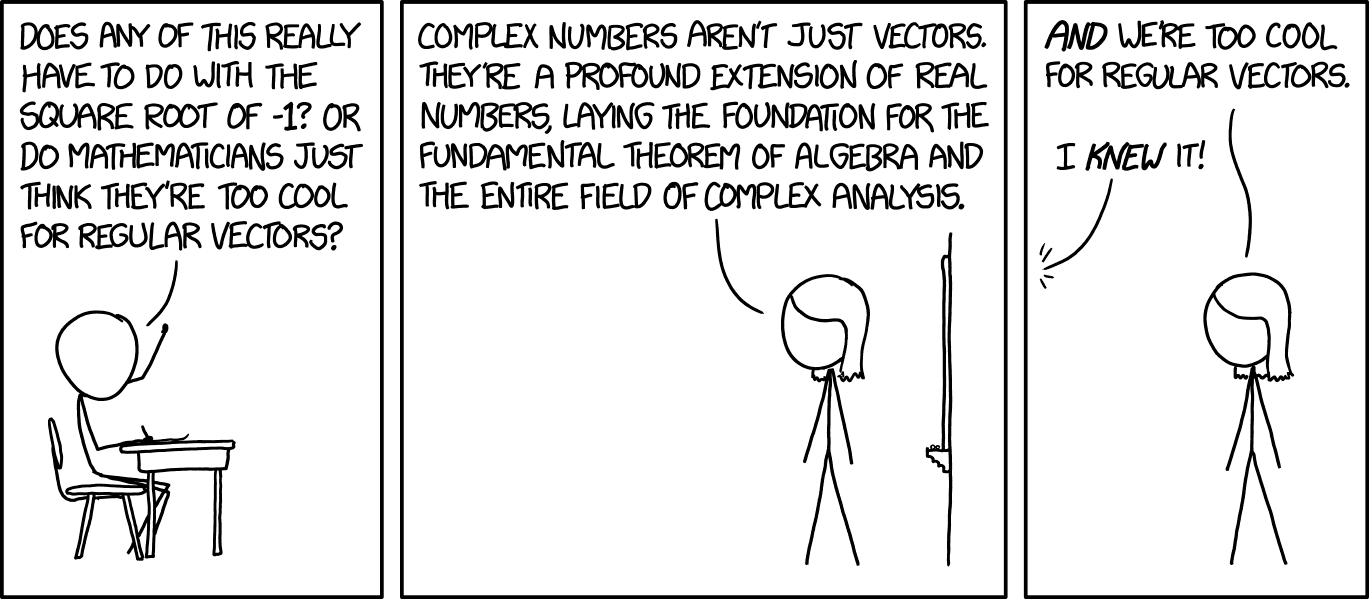
# PHPComplex

PHP Class Library for working with Complex numbers

[Build Status](https://github.com/MarkBaker/PHPComplex/actions) [Total Downloads](https://packagist.org/packages/markbaker/complex) [Latest Stable Version](https://packagist.org/packages/markbaker/complex) [License](https://packagist.org/packages/markbaker/complex)

[](https://xkcd.com/2028/)

The library currently provides the following operations:

* addition
* subtraction
* multiplication
* division
  + division by
  + division into

together with functions for

* theta (polar theta angle)
* rho (polar distance/radius)
* conjugate
* negative
* inverse (1 / complex)
* cos (cosine)
* acos (inverse cosine)
* cosh (hyperbolic cosine)
* acosh (inverse hyperbolic cosine)
* sin (sine)
* asin (inverse sine)
* sinh (hyperbolic sine)
* asinh (inverse hyperbolic sine)
* sec (secant)
* asec (inverse secant)
* sech (hyperbolic secant)
* asech (inverse hyperbolic secant)
* csc (cosecant)
* acsc (inverse cosecant)
* csch (hyperbolic secant)
* acsch (inverse hyperbolic secant)
* tan (tangent)
* atan (inverse tangent)
* tanh (hyperbolic tangent)
* atanh (inverse hyperbolic tangent)
* cot (cotangent)
* acot (inverse cotangent)
* coth (hyperbolic cotangent)
* acoth (inverse hyperbolic cotangent)
* sqrt (square root)
* exp (exponential)
* ln (natural log)
* log10 (base-10 log)
* log2 (base-2 log)
* pow (raised to the power of a real number)

# Installation

composer require markbaker/complex:^1.0

# Important BC Note

If you've previously been using procedural calls to functions and operations using this library, then from version 3.0 you should use [MarkBaker/PHPComplexFunctions](https://github.com/MarkBaker/PHPComplexFunctions) instead (available on packagist as [markbaker/complex-functions](https://packagist.org/packages/markbaker/complex-functions)).

You'll need to replace markbaker/complexin your composer.json file with the new library, but otherwise there should be no difference in the namespacing, or in the way that you have called the Complex functions in the past, so no actual code changes are required.

composer require markbaker/complex-functions:^3.0

You should not reference this library (markbaker/complex) in your composer.json, composer wil take care of that for you.

# Usage

To create a new complex object, you can provide either the real, imaginary and suffix parts as individual values, or as an array of values passed passed to the constructor; or a string representing the value. e.g

$real = 1.23;

$imaginary = -4.56;

$suffix = 'i';

$complexObject = new Complex\Complex($real, $imaginary, $suffix);

or as an array

$real = 1.23;

$imaginary = -4.56;

$suffix = 'i';

$arguments = [$real, $imaginary, $suffix];

$complexObject = new Complex\Complex($arguments);

or as a string

$complexString = '1.23-4.56i';

$complexObject = new Complex\Complex($complexString);

Complex objects are immutable: whenever you call a method or pass a complex value to a function that returns a complex value, a new Complex object will be returned, and the original will remain unchanged. This also allows you to chain multiple methods as you would for a fluent interface (as long as they are methods that will return a Complex result).

## Performing Mathematical Operations

To perform mathematical operations with Complex values, you can call the appropriate method against a complex value, passing other values as arguments

$complexString1 = '1.23-4.56i';

$complexString2 = '2.34+5.67i';

$complexObject = new Complex\Complex($complexString1);

echo $complexObject->add($complexString2);

or use the static Operation methods

$complexString1 = '1.23-4.56i';

$complexString2 = '2.34+5.67i';

echo Complex\Operations::add($complexString1, $complexString2);

If you want to perform the same operation against multiple values (e.g. to add three or more complex numbers), then you can pass multiple arguments to any of the operations.

You can pass these arguments as Complex objects, or as an array, or string that will parse to a complex object.

## Using functions

When calling any of the available functions for a complex value, you can either call the relevant method for the Complex object

$complexString = '1.23-4.56i';

$complexObject = new Complex\Complex($complexString);

echo $complexObject->sinh();

or use the static Functions methods

$complexString = '1.23-4.56i';

echo Complex\Functions::sinh($complexString);

As with operations, you can pass these arguments as Complex objects, or as an array or string that will parse to a complex object.

In the case of the pow() function (the only implemented function that requires an additional argument) you need to pass both arguments when calling the function

$complexString = '1.23-4.56i';

$complexObject = new Complex\Complex($complexString);

echo Complex\Functions::pow($complexObject, 2);

or pass the additional argument when calling the method

$complexString = '1.23-4.56i';

$complexObject = new Complex\Complex($complexString);

echo $complexObject->pow(2);