Hack

Hack is a superset of PHP that runs under a virtual machine called HHVM. Hack is almost completely interoperable with existing PHP code and adds a bunch of useful features from statically typed languages.

Only Hack-specific features are covered here. Details about PHP's syntax are available in the PHP article on this site.

```
<?hh

// Hack syntax is only enabled for files starting with an <?hh marker
// <?hh markers cannot be interspersed with HTML the way <?php can be.
// Using the marker "<?hh //strict" puts the type checker in strict mode.


// Scalar parameter type hints
function repeat(string $word, int $count)
{
    $word = trim($word);
    return str_repeat($word . ' ', $count);
}

// Type hints for return values
function add(...$numbers) : int
{
    return array_sum($numbers);
}

// Functions that return nothing are hinted as "void"
function truncate(resource $handle) : void
{
    // ...
}

// Type hints must explicitly allow being nullable
function identity(?string $stringOrNull) : ?string
{
    return $stringOrNull;
}

// Type hints can be specified on class properties
class TypeHintedProperties
{
    public ?string $name;

    protected int $id;

    private float $score = 100.0;

    // Hack's type checker enforces that typed properties either have a
    // default value or are set in the constructor.
    public function __construct(int $id)
    {
        $this->id = $id;
    }
```

```
}


// Concise anonymous functions (lambdas)
$multiplier = 5;
array_map($y ==> $y * $multiplier, [1, 2, 3]);


// Generics
class Box<T>
{
    protected T $data;

    public function __construct(T $data) {
        $this->data = $data;
    }

    public function getData(): T {
        return $this->data;
    }
}

function openBox(Box<int> $box) : int
{
    return $box->getData();
}


// Shapes
//
// Hack adds the concept of shapes for defining struct-like arrays with a
// guaranteed, type-checked set of keys
type Point2D = shape('x' => int, 'y' => int);

function distance(Point2D $a, Point2D $b) : float
{
    return sqrt(pow($b['x'] - $a['x'], 2) + pow($b['y'] - $a['y'], 2));
}

distance(
    shape('x' => -1, 'y' => 5),
    shape('x' => 2, 'y' => 50)
);


// Type aliasing
//
// Hack adds a bunch of type aliasing features for making complex types readable
newtype VectorArray = array<int, Vector<int>>;

// A tuple containing two integers
newtype Point = (int, int);

function addPoints(Point $p1, Point $p2) : Point
```

```
{
    return tuple($p1[0] + $p2[0], $p1[1] + $p2[1]);
}

addPoints(
    tuple(1, 2),
    tuple(5, 6)
);


// First-class enums
enum RoadType : int
{
    Road = 0;
    Street = 1;
    Avenue = 2;
    Boulevard = 3;
}

function getRoadType() : RoadType
{
    return RoadType::Avenue;
}


// Constructor argument promotion
//
// To avoid boilerplate property and constructor definitions that only set
// properties, Hack adds a concise syntax for defining properties and a
// constructor at the same time.
class ArgumentPromotion
{
    public function __construct(public string $name,
                                protected int $age,
                                private bool $isAwesome) {}
}

class WithoutArgumentPromotion
{
    public string $name;

    protected int $age;

    private bool $isAwesome;

    public function __construct(string $name, int $age, bool $isAwesome)
    {
        $this->name = $name;
        $this->age = $age;
        $this->isAwesome = $isAwesome;
    }
}
```

```
// Co-operative multi-tasking
//
// Two new keywords "async" and "await" can be used to perform multi-tasking
// Note that this does not involve threads - it just allows transfer of control
async function cooperativePrint(int $start, int $end) : Awaitable<void>
{
    for ($i = $start; $i <= $end; $i++) {
        echo "$i ";

        // Give other tasks a chance to do something
        await RescheduleWaitHandle::create(RescheduleWaitHandle::QUEUE_DEFAULT, 0);
    }
}

// This prints "1 4 7 2 5 8 3 6 9"
AwaitAllWaitHandle::fromArray([
    cooperativePrint(1, 3),
    cooperativePrint(4, 6),
    cooperativePrint(7, 9)
])->getWaitHandle()->join();


// Attributes
//
// Attributes are a form of metadata for functions. Hack provides some
// special built-in attributes that introduce useful behaviour.

// The __Memoize special attribute causes the result of a function to be cached
<<__Memoize>>
function doExpensiveTask() : ?string
{
    return file_get_contents('http://example.com');
}

// The function's body is only executed once here:
doExpensiveTask();
doExpensiveTask();


// The __ConsistentConstruct special attribute signals the Hack type checker to
// ensure that the signature of __construct is the same for all subclasses.
<<__ConsistentConstruct>>
class ConsistentFoo
{
    public function __construct(int $x, float $y)
    {
        // ...
    }

    public function someMethod()
    {
        // ...
    }
}
```

```
class ConsistentBar extends ConsistentFoo
{
    public function __construct(int $x, float $y)
    {
        // Hack's type checker enforces that parent constructors are called
        parent::__construct($x, $y);

        // ...
    }

    // The __Override annotation is an optional signal for the Hack type
    // checker to enforce that this method is overriding a method in a parent
    // or trait. If not, this will error.
    <<__Override>>
    public function someMethod()
    {
        // ...
    }
}

class InvalidFooSubclass extends ConsistentFoo
{
    // Not matching the parent constructor will cause a type checker error:
    //
    //  "This object is of type ConsistentBaz. It is incompatible with this object
    //   of type ConsistentFoo because some of their methods are incompatible"
    //
    public function __construct(float $x)
    {
        // ...
    }

    // Using the __Override annotation on a non-overriden method will cause a
    // type checker error:
    //
    //  "InvalidFooSubclass::otherMethod() is marked as override; no non-private
    //   parent definition found or overridden parent is defined in non-<?hh code"
    //
    <<__Override>>
    public function otherMethod()
    {
        // ...
    }
}


// Traits can implement interfaces (standard PHP does not support this)
interface KittenInterface
{
    public function play() : void;
}

trait CatTrait implements KittenInterface
```

```
{
    public function play() : void
    {
        // ...
    }
}

class Samuel
{
    use CatTrait;
}


$cat = new Samuel();
$cat instanceof KittenInterface === true; // True
```

More Information

Visit the Hack language reference for detailed explainations of the features Hack adds to PHP, or the official Hack website for more general information.

Visit the official HHVM website for HHVM installation instructions.

Visit Hack's unsupported PHP features article for details on the backwards incompatibility between Hack and PHP.