Curry

Traditionally partial evaluation of functions is handled with the `partial` higher order function from `functools`. Currying provides syntactic sugar.

```
>>> double = partial(mul, 2)    # Partial evaluation
>>> double = mul(2)             # Currying
```

This syntactic sugar is valuable when developers chain several higher order functions together.

Partial Evaluation

Often when composing smaller functions to form big ones we need partial evaluation. We do this in the word counting example:

```
>>> def stem(word):
...     """ Stem word to primitive form """
...     return word.lower().rstrip(",.!:;'-\"").lstrip("'\"")

>>> wordcount = compose(frequencies, partial(map, stem), str.split)
```

Here we want to map the `stem` function onto each of the words produced by `str.split`. We want a `stem_many` function that takes a list of words, stems them, and returns a list back. In full form this would look like the following:

```
>>> def stem_many(words):
...     return map(stem, words)
```

The `partial` function lets us create this function more naturally.

```
>>> stem_many = partial(map, stem)
```

In general

```
>>> def f(x, y, z):
...     # Do stuff with x, y, and z

>>> # partially evaluate f with known values a and b
>>> def g(z):
...     return f(a, b, z)

>>> # partially evaluate f with known values a and b
>>> g = partial(f, a, b)
```

Curry

In this context currying is just syntactic sugar for partial evaluation. A curried function partially evaluates if it does not receive enough arguments to compute a result.

```
>>> from toolz import curry

>>> @curry                # We can use curry as a decorator
... def mul(x, y):
...     return x * y

>>> double = mul(2)       # mul didn't receive enough arguments to evaluate
...                       # so it holds onto the 2 and waits, returning a
...                       # partially evaluated function, double

>>> double(5)
10
```

So if `map` was curried...

```
>>> map = curry(map)
```

Then we could replace the `partial` with a function evaluation

```
>>> # wordcount = compose(frequencies, partial(map, stem), str.split)
>>> wordcount = compose(frequencies, map(stem), str.split)
```

In this particular example it's probably simpler to stick with `partial`. Once `partial` starts occurring several times in your code it may be time to switch to the `curried` namespace.

The Curried Namespace

All functions present in the `toolz` namespace are curried in the `toolz.curried` namespace.

So you can exchange an import line like the following

```
>>> from toolz import *
```

For the following

```
>>> from toolz.curried import *
```

And all of your favorite `toolz` functions will curry automatically. We've also included curried versions of the standard Python higher order functions like `map`, `filter`, `reduce` so you'll get them too (whether you like it or not.)