Named Tuples

DocStrings and Default Values

Default Docs for Named Tuples

When we create a named tuple class, default docstrings are created

```
Point2D = namedtuple('Point2D', 'x y')
        Point2D.__doc__       →  Point2D(x, y)
        Point2D.x.__doc__     →  Alias for field number 0
        Point2D.y.__doc__     →  Alias for field number 1
```

```
help(Point2D)    →    class Point2D(builtins.tuple)
                          Point2D(x, y)

                      x
                          Alias for field number 0

                      y
                          Alias for field number 1
```

Overriding DocStrings

We can override the docstrings simply by specifying values for the `__doc__` properties

(this is not unique to named tuples!)

```
Point2D.__doc__ = 'Represents a 2D Cartesian coordinate.'

Point2D.x.__doc__ = 'x coordinate'

Point2D.y.__doc__ = 'y coordinate'
```

`help(Point2D)` →

```
class Point2D(builtins.tuple)
   Represents a 2D Cartesian coordinate.

   x
       x coordinate

   y
       y coordinate
```

Default Values

The `namedtuple` function does not provide us a way to define default values for each field

Two approaches to this:

Using a Prototype

Create an instance of the named tuple with default values - the `prototype`

Create any additional instances of the named tuple using the `prototype._replace` method

You will need to supply a default for every field (can be `None`)

Using the `__defaults__` property

Directly set the defaults of the named tuple constructor (the `__new__` method)

You do not need to specify a default for every field

Remember that you cannot have non-defaulted parameters
after the first defaulted parameter

```
def func(a, b=10, c=20)
```
✓

```
def func(a, b=10, c)
```
✗

Using a Prototype

```
Vector2D = namedtuple('Vector2D', 'x1 y1 x2 y2 origin_x origin_y')

vector_zero = Vector2D(x1=0, y1=0, x2=0, y2=0, origin_x=0, origin_y=0)
```

or

```
vector_zero = Vector2D(0, 0, 0, 0, 0, 0)

    vector_zero → Vector2D(x1=0, y1=0, x2=0, y2=0, origin_x=0, origin_y=0)
```

To construct a new instance of `Vector2D` we now use `vector_zero._replace` instead:

```
v1 = vector_zero._replace(x1=10, y1=10, x2=20, y2=20)

    v1 → Vector2D(x1=10, y1=10, x2=20, y2=20, origin_x=0, origin_y=0)
```

Using `__defaults__`

```
def func(a, b=10, c=20):
        pass
```

```
func.__defaults__ → (10, 20)
```

```
 a   b   c
    10  20
```

↑ no default

The `__defaults__` property is writable

So we can set it to a tuple of our choice

Just don't provide more defaults than parameters!   (extras are ignored)

Using `__defaults__`

We need to provide defaults to the constructor of our named tuple class `__new__`

```
Vector2D = namedtuple('Vector2D', 'x1 y1 x2 y2 origin_x origin_y')

Vector2D.__new__.__defaults__ = (0, 0)      x1 y1 x2 y2 origin_x  origin_y
                                                         0          0

v1 = Vector2D(10, 10, 20, 20)
```

v1 → Vector2D(x1=10, y1=10, x2=20, y2=20, origin_x=0, origin_y=0)

Isn't this cleaner than the prototype approach?!!

```
v1 = vector_zero._replace(x1=10, y1=10, x2=20, y2=20)
```

Code