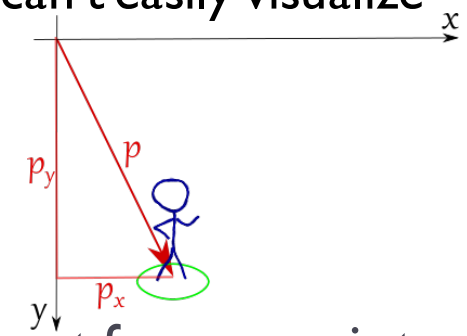
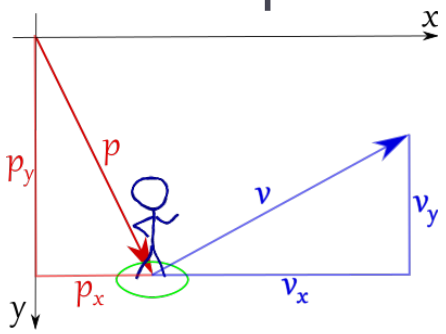
Back to the `point` class

- Point is a very useful class
  - More generally, *vector*
  - Vectors are the basis for all *spatial* simulations
    - 2d, 3d and even higher dimensions that we can't easily visualize
- For simplicity, we'll stick with 2d
  - x, y gives a direction and distance
  - Starting from (0,0), it specifies a point
  - But it could just as easily specify movement from a point
    - Adding vectors is just adding x and y
  - Some examples…





A Useful `vector` class

- Simple, general and flexible
  - 2 data members x and y
- Methods for all the basic arithmetic operations
  - These should all return a new vector (non-mutating)
  - add(other_vector)
  - subtract(other_vector)
  - multiply(scalar) # scalar is a number, not a vector
  - magnitude() # the length of the vector
  - normalized() # the same vector, with length one

A Useful `vector` class

- Starting class to work with:

```
class vector:
  def __init__(self):
    self.x = 0
    self.y = 0
```

A Useful `vector` class

- First, let's write a normal function that adds two vectors
  1. Pass in two vectors as parameters
  2. Create a new local variable vector inside
  3. Assign the new vector x to the sum of the other two x
  4. Assign the new vector y to the sum of the other two y
  5. Return the new vector

A Useful `vector` class

- First, let's write a normal function that adds two vectors

```
def add_vectors(v1, v2):
    sum = vector()
    sum.x = v1.x + v2.x
    sum.y = v1.y + v2.y
    return sum
```

- Called like:

```
add_vectors(position, movement)
```

A Useful `vector` class

- Methods are just functions that are part of a class
- The big difference!
  - Methods are *called on* an object of that class
  - Function: `add_vectors(position, movement)`
  - Method: `position.add_vectors(movement)`
- Think of it as telling the object to do something
  - E.g. "hey position, add movement to yourself"

A Useful `vector` class

- The method version:
  - Is defined inside the class
  - Replaces the first parameter (one of the vectors) with `self`
    - `self` is a reference to the object the method is called on

```
def add_vectors(self, v2):
    sum = vector()
    sum.x = self.x + v2.x
    Sum.y = self.y + v2.y
    return sum
```

- Called like:

```
position.add_vectors(movement)
```

  - Note that you only pass in the parameters *besides* self (v2 here)

A Useful `vector` class

- Could also do the self-mutating version
  - Changes itself rather than returning a new vector object

```
def add_vectors(self, v2):
  self.x = self.x + v2.x
  self.y = self.y + v2.y
  # no return necessary
```

That constructor

- We started our class with a *constructor*
  - A method that is called automatically whenever we create an object of the class
  - The special name `__init__` tells Python to call this automatically

```
def __init__(self):
    self.x = 0
    self.y = 0
```

  - This is called the *default constructor*

That constructor

- Alternatively, we could have a constructor that takes additional parameters

```
def __init__(self, x, y):
    self.x = x
    self.y = y
```

- This *parameterized constructor* allows us to specify the initial values, like:

```
v = vector(3, 4)
```

  - (note that you don't pass in self, it's already there!)

- The one we did before, with no additional parameters, is the *default constructor*

```
v = vector()
```