Point distance: hypot — *Smooth CoffeeScript*

This literate program is *interactive* in its HTML form. Edit a CoffeeScript segment to try it. You can see the generated JavaScript as you modify a CoffeeScript function by typing 'show name' after its definition.

Point distance algorithm

Warning! This snippet is about calculating the distance between two points — not something you would normally worry about. It is a fundamental calculation that can be used in other algorithms such as 'nearest neighbor'.

A point is any object that implements `x` and `y` properties. Here defined as a class.

```
class Point
  constructor: (@x, @y) ->
  draw: (ctx) -> ctx.fillRect @x, @y, 1, 1
  toString: -> "(#{@x}, #{@y})"
```

Euclidean distance

The euclidean distance between two points is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Which can be written as $\sqrt{a^2 + b^2}$ where $a = x_1 - x_2$ and $b = y_1 - y_2$.

```
euclidean = (p1, p2) ->
  [a, b] = [p1?.x - p2?.x, p1?.y - p2?.y]
  Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2))
```

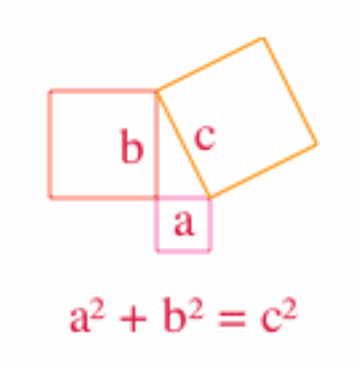
This is a classical algorithm based on the Pythagorean theorem.

```
draw = (ctx) ->
  # A hard-coded approximate figure
  ctx.beginPath()
  ctx.fillStyle = 'crimson'
  ctx.font = '14pt Times'
  ctx.fillText 'a² + b² = c²', 27, 110

  ctx.fillText 'a', 66, 74
  ctx.beginPath()
  ctx.strokeStyle = 'hotpink'
  ctx.strokeRect 60, 60, 20, 20

  ctx.fillText 'b', 46, 47
  ctx.beginPath()
  ctx.strokeStyle = 'tomato'
  ctx.strokeRect 20, 20, 40, 40

  ctx.fillText 'c', 74, 42
  ctx.beginPath()
  ctx.strokeStyle = 'darkorange'
  ctx.moveTo 60, 20
  ctx.lineTo 80, 60
  ctx.lineTo 120, 40
  ctx.lineTo 100, 0
  ctx.lineTo 60, 20
  ctx.stroke()
```



Improved precision calculation

Mathematically the `euclidean` calculation is correct and in normal use it will work. However for very large and very small numbers its results are imprecise — more imprecise than they have to be. This is a consequence of the difference between numbers in mathematics and computer floating point numbers. The precision of the results are limited due to the square of the differences, the square causes overflow or underflow to occur at the square root of the machine precision. Fortunately some clever person found that expressing the calculation in another way can improve upon this situation, see Wikipedia hypot.

$$\sqrt{a^2 + b^2} = \sqrt{a^2 \cdot \left(1 + \left(\frac{b}{a}\right)^2\right)} = |a| \sqrt{1 + \left(\frac{b}{a}\right)^2}$$

```
hypot = (a, b) ->
  if a is 0
    Math.abs(b)
  else
    Math.abs(a) * Math.sqrt(1 + Math.pow(b/a, 2))
hypotenuse = (p1, p2) ->
  [a, b] = [p1?.x - p2?.x, p1?.y - p2?.y]
  hypot a, b
```

Polar coordinates

As described in Wikipedia hypot, the `hypot` function can also be used to convert to polar coordinates.

```
polar = (p) ->
  [x, y] = [p.x, p.y]
  r = hypot(x, y)
   = Math.atan2(y, x)
  [r,  ]
show 'Distance from (0, 0), angle in  2 radians'
show polar new Point 1, 1
```

Edge case tests

Testing with some exceptional values is a good way to check that the function behave as intended.

```
show "euclidean vs hypotenuse"
p1 = p2 = undefined
show "#{euclidean p1, p2} vs #{hypotenuse p1, p2}"

p1 = new Point 0, 0
p2 = new Point 0, 0
show "#{euclidean p1, p2} vs #{hypotenuse p1, p2}"
```

```
p1 = new Point 1e-200, 1e-200
p2 = new Point 2e-200, 2e-200
show "#{euclidean p1, p2} vs #{hypotenuse p1, p2}"

p1 = new Point 1e200, 1e200
p2 = new Point 2e200, 2e200
show "#{euclidean p1, p2} vs #{hypotenuse p1, p2}"
```

QuickCheck

More comprehensive testing can be performed with QuickCheck. It is a test method where the properties of a function are described and data is generated to see if the properties hold. It is suitable for testing algorithms (in game logic and graphics) and is introduced in Smooth CoffeeScript Functions. There is also an interface reference.

How to

You can find some support code in the solution below. It can be used with `qc.js` to run QuickCheck with the standalone CoffeeScript compiler.

```
unless exports?
  _ = window._ # Workaround for interactive environment quirk.
else
  show = console.log
  _ = require 'underscore'
  qc = require 'qc'
  # Import functions into the global namespace with globalize,
  # so that they do not need to be qualified each time.
  globalize = (ns, target = global) ->
    target[name] = ns[name] for name of ns
  # qc is only used for testing so ignore namespace pollution.
  globalize qc

if exports?
  # Set to `no` to get monochrome output
  useColors = no

  # Node colored output for QuickCheck.
  class NodeListener extends ConsoleListener
    constructor: (@maxCollected = 10) ->
    log: (str) -> show str
    passed: (str) -> # print message in green
      console.log if useColors then "\033[32m#{str}\033[0m" else "#{str}"
    invalid: (str) -> # print message in yellow
      console.warn if useColors then "\033[33m#{str}\033[0m" else "#{str}"
    failure: (str) -> # print message in red
      console.error if useColors then "\033[31m#{str}\033[0m" else "#{str}"
    done: ->
      show 'Completed test.'
      resetProps() # Chain here if needed

  # Enhanced noteArg returning its argument so it can be used inline.
  Case::note = (a) -> @noteArg a; a
  # Same as Case::note but also logs the noted args.
  Case::noteVerbose = (a) -> @noteArg a; show @args; a

  # Helper to declare a named test property for
  # a function func taking types as arguments.
  # Property is passed the testcase, the arguments
  # and the result of calling func, it must return
```

```
  # a boolean indicating success or failure.
  testPure = (func, types, name, property) ->
    declare name, types, (c, a...) ->
      c.assert property c, a..., c.note func a...

  # Default qc configuration with 100 pass and 1000 invalid tests
  qcConfig = new Config 100, 1000

  # Test all known properties
  test = (msg, func) ->
    _.each [msg, func, runAllProps qcConfig, new NodeListener],
      (o) -> unless _.isUndefined o then show o
```

Test cases

This test says that the `hypot` function is expected to return the same result as the `euclidean`. The `euclidean` is assumed to be correct for the usually generated arbInt numbers.

```
declare 'same results for normal range numbers',
  [arbInt, arbInt, arbInt, arbInt],
  (c, x1, y1, x2, y2) ->
    p1 = new Point x1, y1
    p2 = new Point x2, y2
    d1 = euclidean p1, p2
    d2 = hypotenuse p1, p2
    diff = (d1 - d2)
    epsilon = 1e-10
    c.assert -epsilon < diff < epsilon
```

The next tests say that the `hypot` function is expected to return different results for large numbers. There are different ways of doing so.

```
arbBig = arbRange(1e155, 1e165)
declare 'different results for big range numbers',
  [arbBig, arbBig, arbBig, arbBig],
  (c, x1, y1, x2, y2) ->
    p1 = new Point x1, y1
    p2 = new Point x2, y2
    d1 = euclidean p1, p2
    d2 = hypotenuse p1, p2
    diff = Math.abs d1 - d2
    epsilon = 1e-10
    c.assert diff > epsilon
```

```
declare 'different results for large numbers',
  [arbInt, arbInt, arbInt, arbInt, arbInt, arbInt],
  (c, x1, y1, e1, x2, y2, e2) ->
    p1 = new Point x1*Math.pow(10, e1), y1*Math.pow(10, e1)
    p2 = new Point x2*Math.pow(10, e2), y2*Math.pow(10, e2)
    d1 = euclidean p1, p2
    d2 = hypotenuse p1, p2
    diff = Math.abs d1 - d2
    c.guard diff > 1
    exp = 9
    c.assert e1 < -exp or e1 > exp or e2 < -exp or e2 > exp
```

The results will vary on each run as the data is generated. The floating point precision is implementation dependent.

```
do test
```

Output

```
Distance from (0, 0), angle in 2  radians
[ 1.4142135623730951, 0.7853981633974483 ]
euclidean vs hypotenuse
NaN vs NaN
0 vs 0
0 vs 1.414213562373095e-200
Infinity vs 1.414213562373095e+200
Pass: same results for normal range numbers (pass=100, invalid=0)
Pass: different results for big range numbers (pass=100, invalid=0)
Pass: different results for large numbers (pass=100, invalid=220)
Completed test.
```

JavaScript

```
(function() {
  var NodeListener, Point, arbBig, draw, euclidean, globalize, hypot, hypotenuse, p1, p2, polar, qc, qcConfig, show, showDocument, te
  var __hasProp = Object.prototype.hasOwnProperty, __extends = function(child, parent) { for (var key in parent) { if (__hasProp.call

  show = console.log;

  showDocument = function(doc, width, height) {
    return show(doc);
  };

  Point = (function() {

    function Point(x, y) {
      this.x = x;
      this.y = y;
    }

    Point.prototype.draw = function(ctx) {
      return ctx.fillRect(this.x, this.y, 1, 1);
    };

    Point.prototype.toString = function() {
      return "(" + this.x + ", " + this.y + ")";
    };

    return Point;

  })();

  euclidean = function(p1, p2) {
    var a, b, _ref;
    _ref = [(p1 != null ? p1.x : void 0) - (p2 != null ? p2.x : void 0), (p1 != null ? p1.y : void 0) - (p2 != null ? p2.y : void 0)]
    return Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
  };

  draw = function(ctx) {
    ctx.beginPath();
    ctx.fillStyle = 'crimson';
    ctx.font = '14pt Times';
    ctx.fillText('a² + b² = c²', 27, 110);
    ctx.fillText('a', 66, 74);
    ctx.beginPath();
    ctx.strokeStyle = 'hotpink';
    ctx.strokeRect(60, 60, 20, 20);
    ctx.fillText('b', 46, 47);
    ctx.beginPath();
    ctx.strokeStyle = 'tomato';
    ctx.strokeRect(20, 20, 40, 40);
    ctx.fillText('c', 74, 42);
    ctx.beginPath();
    ctx.strokeStyle = 'darkorange';
    ctx.moveTo(60, 20);
    ctx.lineTo(80, 60);
    ctx.lineTo(120, 40);
```

```
    ctx.lineTo(100, 0);
    ctx.lineTo(60, 20);
    return ctx.stroke();
  };

  hypot = function(a, b) {
    if (a === 0) {
      return Math.abs(b);
    } else {
      return Math.abs(a) * Math.sqrt(1 + Math.pow(b / a, 2));
    }
  };

  hypotenuse = function(p1, p2) {
    var a, b, _ref;
    _ref = [(p1 != null ? p1.x : void 0) - (p2 != null ? p2.x : void 0), (p1 != null ? p1.y : void 0) - (p2 != null ? p2.y : void 0)
    return hypot(a, b);
  };

  polar = function(p) {
    var r, x, y,  , _ref;
    _ref = [p.x, p.y], x = _ref[0], y = _ref[1];
    r = hypot(x, y);
      = Math.atan2(y, x);
    return [r,  ];
  };

  show('Distance from (0, 0), angle in 2  radians');

  show(polar(new Point(1, 1)));

  show("euclidean vs hypotenuse");

  p1 = p2 = void 0;

  show("" + (euclidean(p1, p2)) + " vs " + (hypotenuse(p1, p2)));

  p1 = new Point(0, 0);

  p2 = new Point(0, 0);

  show("" + (euclidean(p1, p2)) + " vs " + (hypotenuse(p1, p2)));

  p1 = new Point(1e-200, 1e-200);

  p2 = new Point(2e-200, 2e-200);

  show("" + (euclidean(p1, p2)) + " vs " + (hypotenuse(p1, p2)));

  p1 = new Point(1e200, 1e200);

  p2 = new Point(2e200, 2e200);

  show("" + (euclidean(p1, p2)) + " vs " + (hypotenuse(p1, p2)));

  if (typeof exports === "undefined" || exports === null) {
    _ = window._;
  } else {
    show = console.log;
    _ = require('underscore');
    qc = require('qc');
    globalize = function(ns, target) {
      var name, _results;
      if (target == null) target = global;
      _results = [];
      for (name in ns) {
        _results.push(target[name] = ns[name]);
      }
      return _results;
    };
    globalize(qc);
  }
```

```
    if (typeof exports !== ”undefined” && exports !== null) {
      useColors = false;
      NodeListener = (function() {

        __extends(NodeListener, ConsoleListener);

        function NodeListener(maxCollected) {
          this.maxCollected = maxCollected != null ? maxCollected : 10;
        }

        NodeListener.prototype.log = function(str) {
          return show(str);
        };

        NodeListener.prototype.passed = function(str) {
          return console.log(useColors ? ”\033[32m” + str + ”\033[0m” : ”” + str);
        };

        NodeListener.prototype.invalid = function(str) {
          return console.warn(useColors ? ”\033[33m” + str + ”\033[0m” : ”” + str);
        };

        NodeListener.prototype.failure = function(str) {
          return console.error(useColors ? ”\033[31m” + str + ”\033[0m” : ”” + str);
        };

        NodeListener.prototype.done = function() {
          show('Completed test.');
          return resetProps();
        };

        return NodeListener;

      })();
      Case.prototype.note = function(a) {
        this.noteArg(a);
        return a;
      };
      Case.prototype.noteVerbose = function(a) {
        this.noteArg(a);
        show(this.args);
        return a;
      };
      testPure = function(func, types, name, property) {
        return declare(name, types, function() {
          var a, c;
          c = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
          return c.assert(property.apply(null, [c].concat(__slice.call(a), [c.note(func.apply(null, a))])));
        });
      };
      qcConfig = new Config(100, 1000);
      test = function(msg, func) {
        return _.each([msg, func, runAllProps(qcConfig, new NodeListener)], function(o) {
          if (!_.isUndefined(o)) return show(o);
        });
      };
    }

    declare('same results for normal range numbers', [arbInt, arbInt, arbInt, arbInt], function(c, x1, y1, x2, y2) {
      var d1, d2, diff, epsilon;
      p1 = new Point(x1, y1);
      p2 = new Point(x2, y2);
      d1 = euclidean(p1, p2);
      d2 = hypotenuse(p1, p2);
      diff = d1 - d2;
      epsilon = 1e-10;
      return c.assert((-epsilon < diff && diff < epsilon));
    });

    arbBig = arbRange(1e155, 1e165);
```

```
  declare('different results for big range numbers', [arbBig, arbBig, arbBig, arbBig], function(c, x1, y1, x2, y2) {
    var d1, d2, diff, epsilon;
    p1 = new Point(x1, y1);
    p2 = new Point(x2, y2);
    d1 = euclidean(p1, p2);
    d2 = hypotenuse(p1, p2);
    diff = Math.abs(d1 - d2);
    epsilon = 1e-10;
    return c.assert(diff > epsilon);
  });

  declare('different results for large numbers', [arbInt, arbInt, arbInt, arbInt, arbInt, arbInt], function(c, x1, y1, e1, x2, y2, e
    var d1, d2, diff, exp;
    p1 = new Point(x1 * Math.pow(10, e1), y1 * Math.pow(10, e1));
    p2 = new Point(x2 * Math.pow(10, e2), y2 * Math.pow(10, e2));
    d1 = euclidean(p1, p2);
    d2 = hypotenuse(p1, p2);
    diff = Math.abs(d1 - d2);
    c.guard(diff > 1);
    exp = 9;
    return c.assert(e1 < -exp || e1 > exp || e2 < -exp || e2 > exp);
  });

  test();

}).call(this);
```

Formats CoffeeScript Markdown PDF HTML