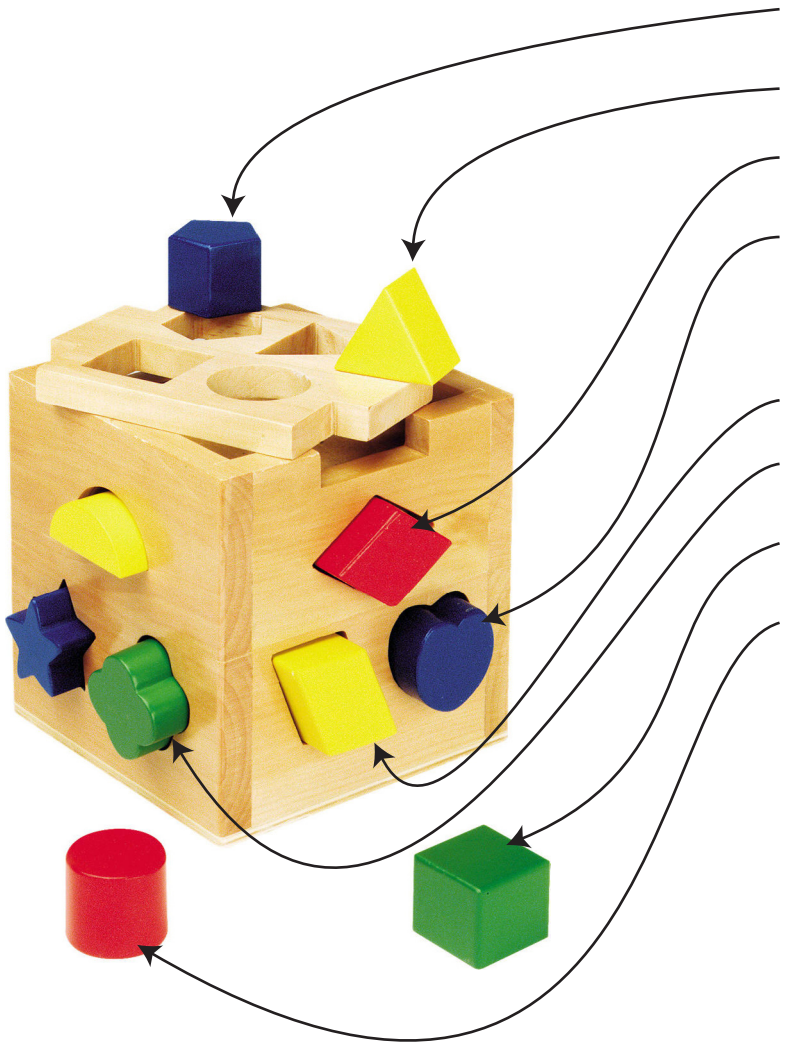


int	Counting numbers (0, 5, -4, 1000, -500)
double	Real numbers(0.5, -7.8 ,15.0, 1598.5434)
boolean	Answer to a logical question (true, false)
IPoint	GC's special point
IDirection	GC's special part of a vector
ICurve	CG's own curve, includes lines, arcs, bsplines
ISurface	CG's own surface
ISolid	CG's own Solids
User defined	You can define your own types

Data exists as an abstract thing totally separate to the function that created it.

You can imagine data as a shape block with smethign written on it, certain types of block for certain types of thing.

so data is of a type, if you get 45.5 from an angle query it is a double and is no different to 45.5 from a length query. It is not an angle!

You can covert from type to type, but if a feature asks for a **double** and you give it a **point** it will be upset.

myLine	is an ICurve
myLine.Endpoint	is an IPoint
myLine.Endpoint.X	is a double
myLine[5]	is an int
myPoint.Success	is a boolean

The whole statement is typed the same as the last thing in the statement.

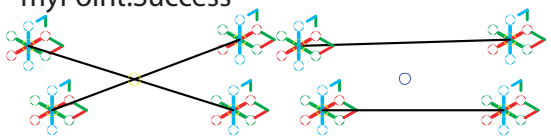
data types

notion parallax
ben doherty

property mining

notion parallax
ben doherty

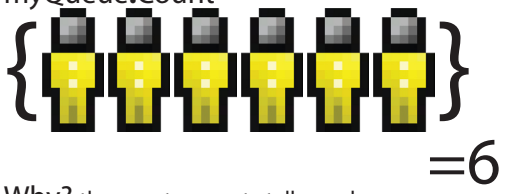
myPoint.Success



Why? you can query something has been successfully created. If it has then it will return 'true' if not, it will return false and will give a red ring on the symbolic graph.
above is a point by intersection, if they intersect then it is 'true' otherwise, 'false'

Returns a **boolean**

myQueue.Count

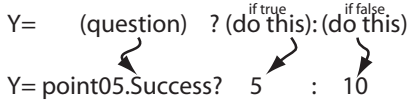


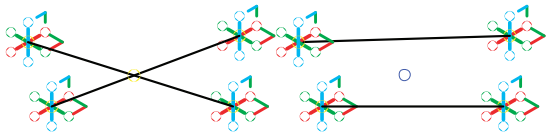
Why? the count property tells you how many items there are in a 1D list.
you can plug it back into the indexing to give you the last item in an array i.e.
myReplicatedPoint[myReplicatedPoint.Count]

Returns an **int**

Inline Conditional Statements

If *question* is *true* **then** do *this* **otherwise** do *this*





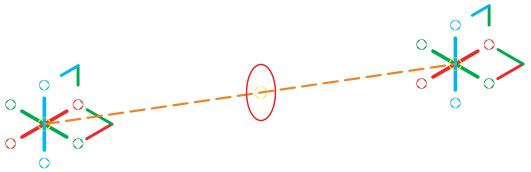
Why? inline conditionals are a very quick and easy way to get logical operations into your models. It all hinges on a well formed question, i.e. one that returns a true or false answer e.g.

point.Y==5	Equal to 5
point.Y!=5	Not equal to 5
point.Y<=5	Lesser than or equal to 5
point.Y>=5	Greater than or equal to 5
point.Y<5	Lesser than 5
point.Y>5	Greater than 5

these can be nested so that instead of a single value you have another inline conditional statement and so on forever!

Returns a **whatever it's told to**

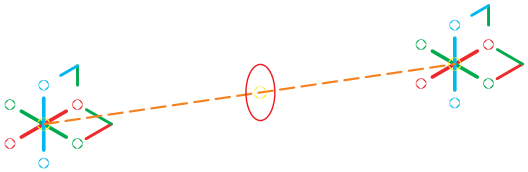
myPointOnACurve.DistanceAlongCurve



Why? if you want to know how far along a curve a point is then this will tell you in real terms, as if you laid a piece of string along the curve and then took it off, straightened it out and measured it.

Returns a **double**

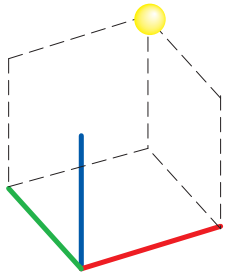
myPointOnACurve.T



Why? the T value of a point on a curve tells you how far along that point is in parameter space. parameter space goes between 0 & 1 and does some funny things in BSplines, but simply put, T=0.5 is half way along the curve

Returns a **double**

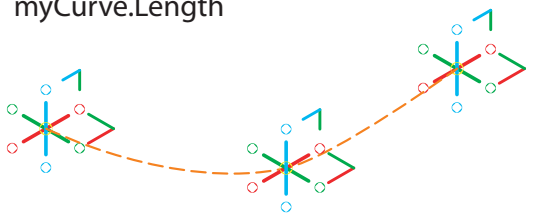
myPoint.X
myPoint.Y
myPoint.Z



Why? these functions give the x,y&z values of a point relative to the baseCS.
useful if you want to match a point in just one axis

Returns a **double**

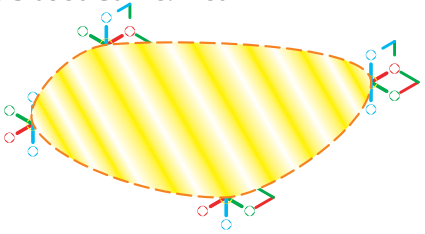
myCurve.Length



Why? length will tell you how long your curve is. (remember lines and circles are curves too)

Returns a **Double**

myClosedCurve.Area



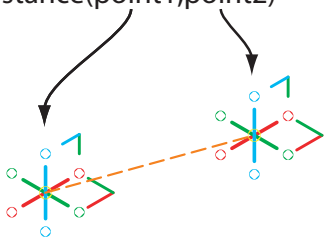
Why? returns the area enclosed by a curve or a shape. good for doing proper architectural things like calculating floor space or volumes of things.

Returns a **double**

functions

notion parallax
ben doherty

Distance(point1,point2)



Why? Distance() measures how far it is between two **points** without having to draw a line and then mine it to get line.Length

Returns a **double**

Round(doubleToRound, decimalPlaces)

Round(point1.X, 2)

Why? GC will usually tell you things to loads of decimal places (e.g. 1.22756496133486) so Round() will round off to a sensible amount.
One thing to note is that if you leave the second argument empty it will give you a double not an int, i.e.14.0 not 14 (to get an int you'd need to ToInt() it)

Returns a **double**

Series(from, to, inStepsOf)

Series(0, 15.5, 1)

Why? This gets 2 boxes because you'll use it so much!
The Series() function is the simplest way of making an item replicate.
In the basic Series you need to give it a start value, an end value and the step size you want it to increment by. if your step doesn't fit exactly into your total then you'll get a bit left over at the end.
There are other types of series for when you need to control it from the other end.

SeriesByCount(start, final, count)
SeriesByCount(0, 15.5, 12)

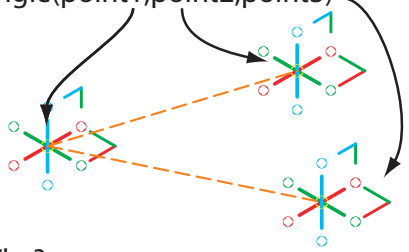
Tells GC how many items you want and it does the maths for you to sontrol the spacing

SeriesUniformly(start, final, increment)
SeriesUniformly(0, 15.5, 1)

controls the increment, but adjusts it slightly so that there is no left over bit.

Returns a **list of doubles**

Angle(point1,point2,point3)



Why? Angle() measures the angle formed between three **points**. As three points define a plane it will always work, but it will never give a value above 180°

Returns a **double**

ToString(anything)

ToString(point1.X)

Why? often you'll want to put a label somewhere so you can see quickly what it's values are, but as the text feature requires a string these functions convert anything, be it an int, a double, a boolean or even a feature to a string.
to add more to this you just type what you want to add in inverted commas and put a plus sign between it and the value i.e.

"point1's X value is " + ToString(point1.X)

Returns a **string**

Random(low,high)

Returns a **int** between low and high

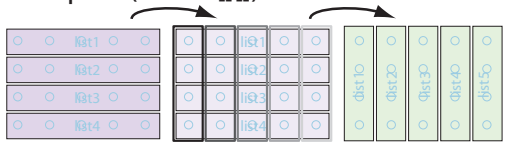
Random()

Returns a **double** between 0 & 1

Why? Random() gives a way of getting unpredictable results so you can just keep hitting Update All Graphs ( ) and seeing what happens
With a bit of clever maths you can multiply a value between 0 & 1 to be whatever you want so you aren't stuck with an int

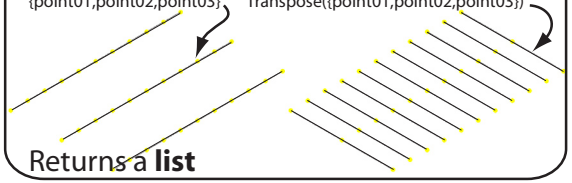
(high - low) * Random() + low

Transpose(2D list[][])



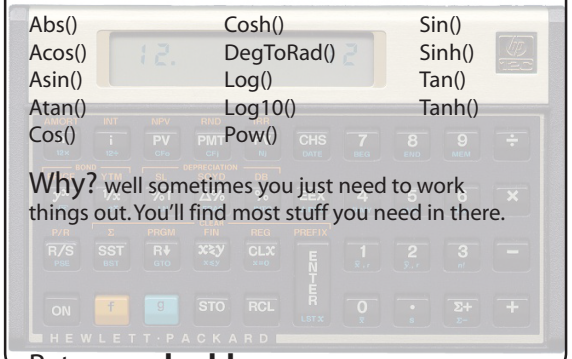
Why? sometimes you'll need to join up some points in the other way around.

{point01,point02,point03} Transpose({point01,point02,point03})



Returns a **list**

Math functions



Abs()
Acos()
Asin()
Atan()
Cos()
Cosh()
DegToRad()
Log()
Log10()
Pow()
Sin()
Sinh()
Tan()
Tanh()

Why? well sometimes you just need to work things out. You'll find most stuff you need in there.

Returns a **double**