Spatial Data Introduction

OGC simple features

mike johnson

Goal of these slides

- Introduce you to some core geospatial (vector) basics, jargon and standards
- Introduce point, line, and polygon geometry construction
- Know what WKT , WKB, GDAL, GEOS and OGC stand for

Not the Goal of these slides

- Introduce Java code
  - code snippets are in **R** (this is to get you you thinking about **concepts, not code**)
  - This is also to introduce existing bindings to spatial databases such as GDAL, GEOS, and POSTGIS including error handling and reporting

Spatial Data

Two flavors of spatial data … each has their own unique representation, operations, applications, and suitability

| RASTER	VECTOR
GRID (RESOLUTION, EXTENT, PROJECTION)	GEOMETRIES (Collection of points)
Continuous (although the data can be continuous or categorical)	Discrete
Single Value	Multi-attribute

2010 Ward Features
St. Louis, MO

Can you think of two variables – one that would best be represented by raster and one best represented as vector?

Coordinate Systems

think geog12, intro GIS, ect...
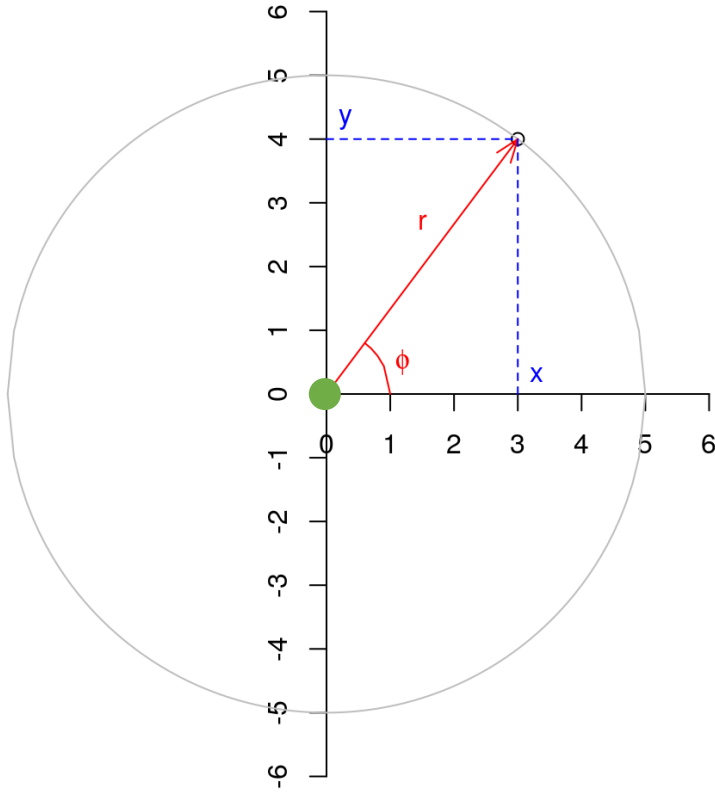
Conversion formulas

$x = r\cos\phi$
$y = r\sin\phi$

$\phi = \arctan(y/x)$
$r = \mathrm{sqrt}(x2+y2)$

** Be aware of quadrant for proper arctan conversions...

● : origin

Red : Geodetic coordinates (geographic coordinate systems)

Blue: Cartesian Coordinates (projected coordinate systems

The OGC

- The Open Geospatial Consortium (OGC) is an international non-profit organization committed to making quality open standards for the global geospatial community.

- Standards are made through a consensus process and are freely available for anyone to use to improve sharing of the world's geospatial data.

Vision:

> A world in which everyone benefits from the use of geospatial information and supporting technologies.

Mission:

> To advance the development and use of international standards and supporting services that promote geospatial interoperability. To accomplish this mission, OGC serves as the global forum for the collaboration of geospatial data / solution providers and users.

OGC Simple Feature Access

- **Simple Features** (officially **Simple Feature Access**) is both an OGC and International Organization for Standardization (ISO) standard that specifies a common storage and access model of mostly two-dimensional geometries used by geographic information systems.

- Part one of the standard defines a model for two-dimensional simple features, with linear interpolation between vertices. This part also defines representation using Well-Known Text (and Binary).
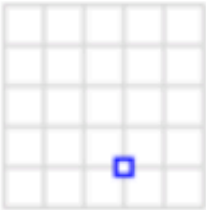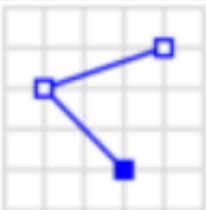
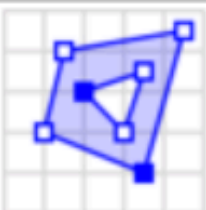# The `Big 7` Geometries

Type	Description
POINT	single point geometry
MULTIPOINT	set of points
LINESTRING	two or more points connected by straight lines
MULTILINESTRING	set of linestrings
POLYGON	Closed linestring ring with zero or more inner rings (holes)
MULTIPOLYGON	set of polygons
GEOMETRYCOLLECTION	set of the any above geometries

OGC **Simple** Feature Access
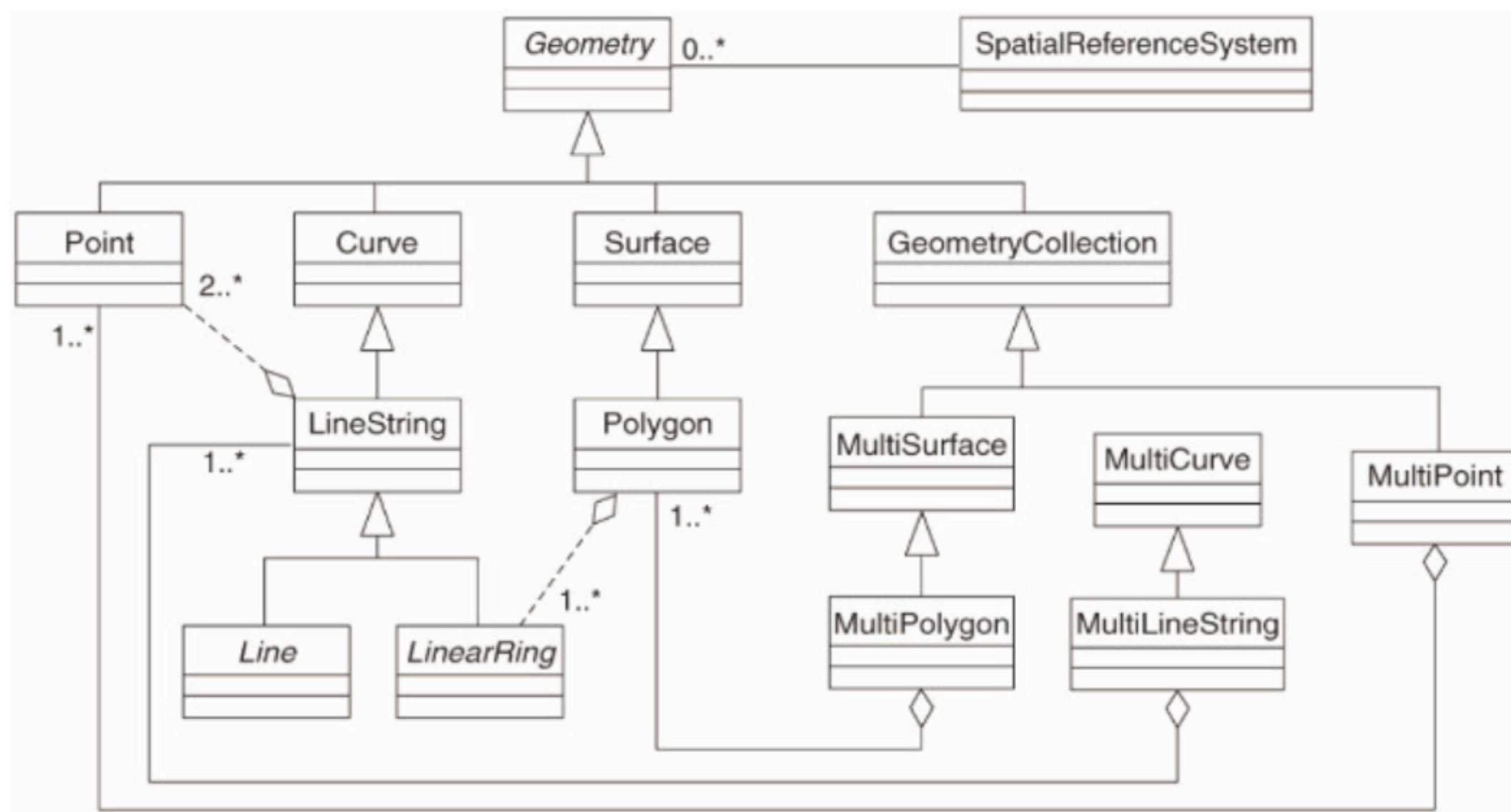
Geometry primitives (2D)

Type	Examples	
Point		`POINT (30 10)`
LineString		`LINESTRING (30 10, 10 30, 40 40)`
Polygon		`POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))`
		`POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10),` `(20 30, 35 35, 30 20, 20 30))`

Well-Known Text (WKT)

- Text markup language representing vector geometry

- Originally defined by the OGC (Open Geospatial Consortium) and described in the Simple Features Access and Coordinate Transformation Services

- Fully described here
  - In total, there are 18 distinct geometry objects that can be represented

- Meant to be **human readable**

Well Known Text Binary (WKB)

- Binary conversion is used to communicate geometries to external libraries (GDAL, GEOS, liblwgeom) and spatial databases
  - GDAL = geospatial data abstraction library
  - GEOS = Geometry Engine Open Source
  - Liblwgeom = "Light Weight Geom"

- Meant to be **machine readable**

```
> st_point(c(1/4, 1/3))
POINT (0.25 0.3333333)
```

```
> (wkb = st_as_binary(st_point(c(1/4, 1/3))))
 [1] 01 01 00 00 00 00 00 00 00 d0 3f 55 55 55 55 55 55 d5 3f
```

POINT()

- A point is the base vector unit
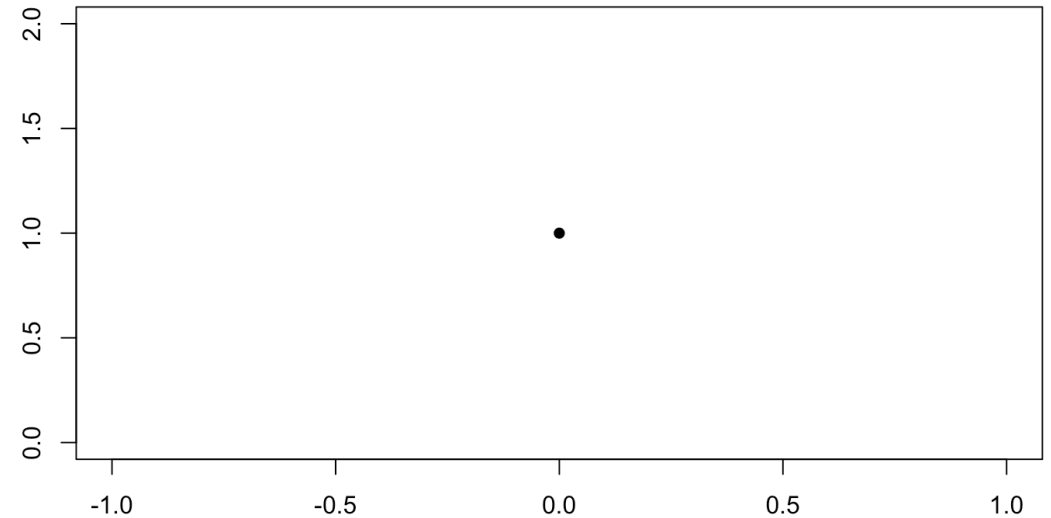- Comprised of an **ordered** `XY` coordinate (can include Z, M – eg `XYZ`)

**X, Y Coordinate
(Cartesian)**

Create a point (st is an R, POSTGIS an
and GDAL convention for 'spatio-temporal')

```
> ( st_point(c(0,1)) )
POINT (0 1)
```

WKT String

Single Point

MULTIPOINT()

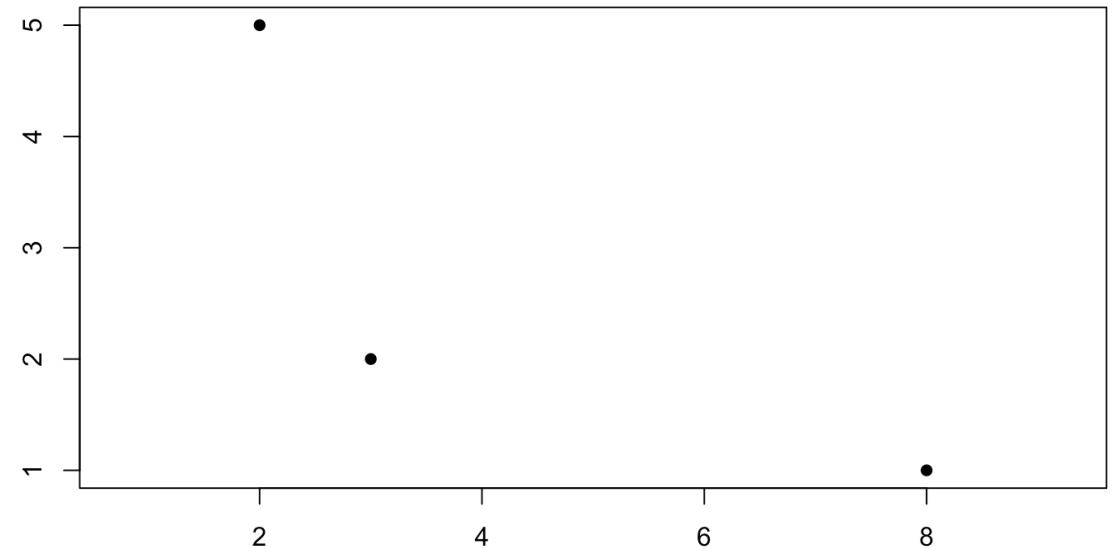**X, Y Coordinate (Cartesian)**

```
> (m1 = rbind(c(8, 1), c(2, 5), c(3, 2)))
     [,1] [,2]
[1,]    8    1
[2,]    2    5
[3,]    3    2
```

Create a point multipoint

```
> (mp = st_multipoint(m1))
MULTIPOINT (8 1, 2 5, 3 2)
```
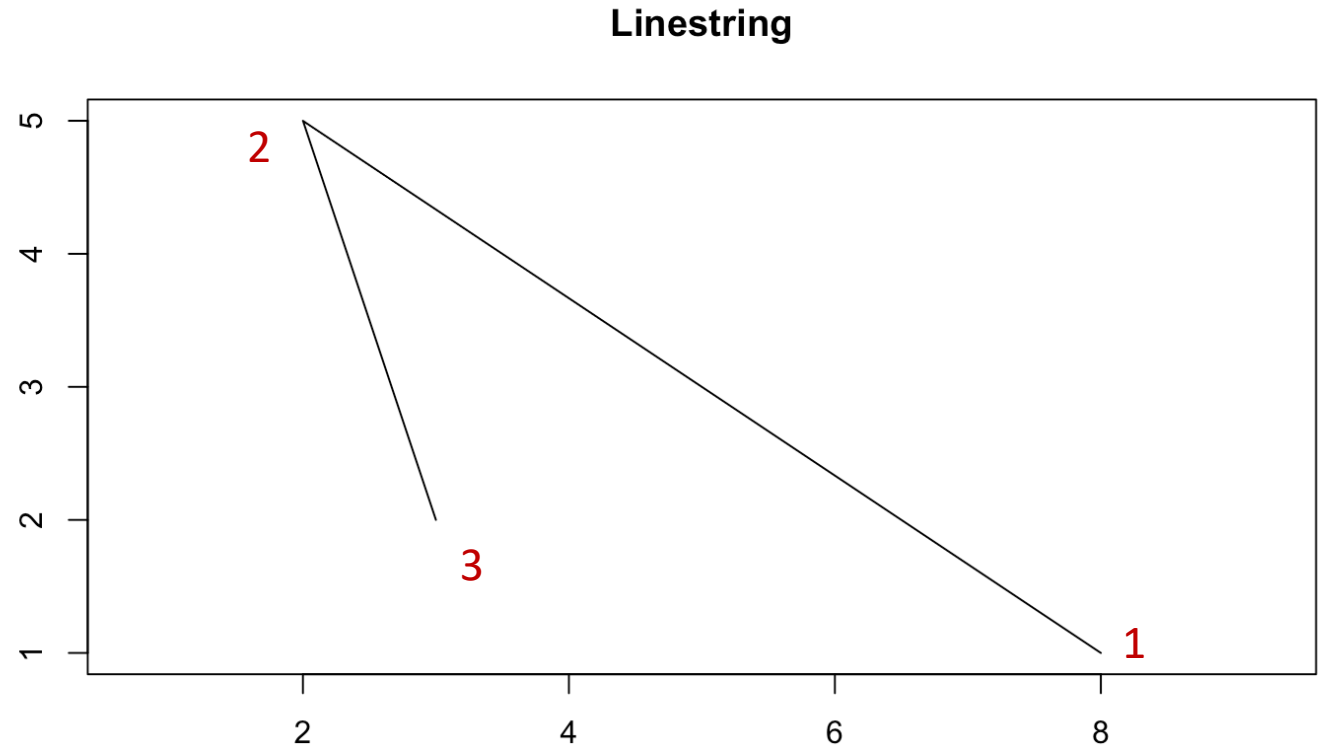
WKT String

Mulitpoint Plot

LINESTRING()

Single linestrings are made up of an **ordered** collection of POINTs() (two or more connected by straight lines)
Lines CANNOT self-intersect.

Create a point linestring

```
> (ls = st_linestring(m1))
LINESTRING (8 1, 2 5, 3 2)
```

WKT String

Linestring

POLYGON()

- Polygon rings must be closed (the last point equals the first)
- Polygon holes (inner rings) must be inside their exterior ring
- Inner rings shall maximally touch the exterior ring at a single point, not over a line
- Polygon ring cannot repeat their own path

```
> (m1 = rbind(c(8, 1), c(2, 5), c(3, 2)))
     [,1] [,2]
[1,]    8    1
[2,]    2    5
[3,]    3    2
```

```
> pg = st_polygon(list(m1))
```

Is this a valid polygon?

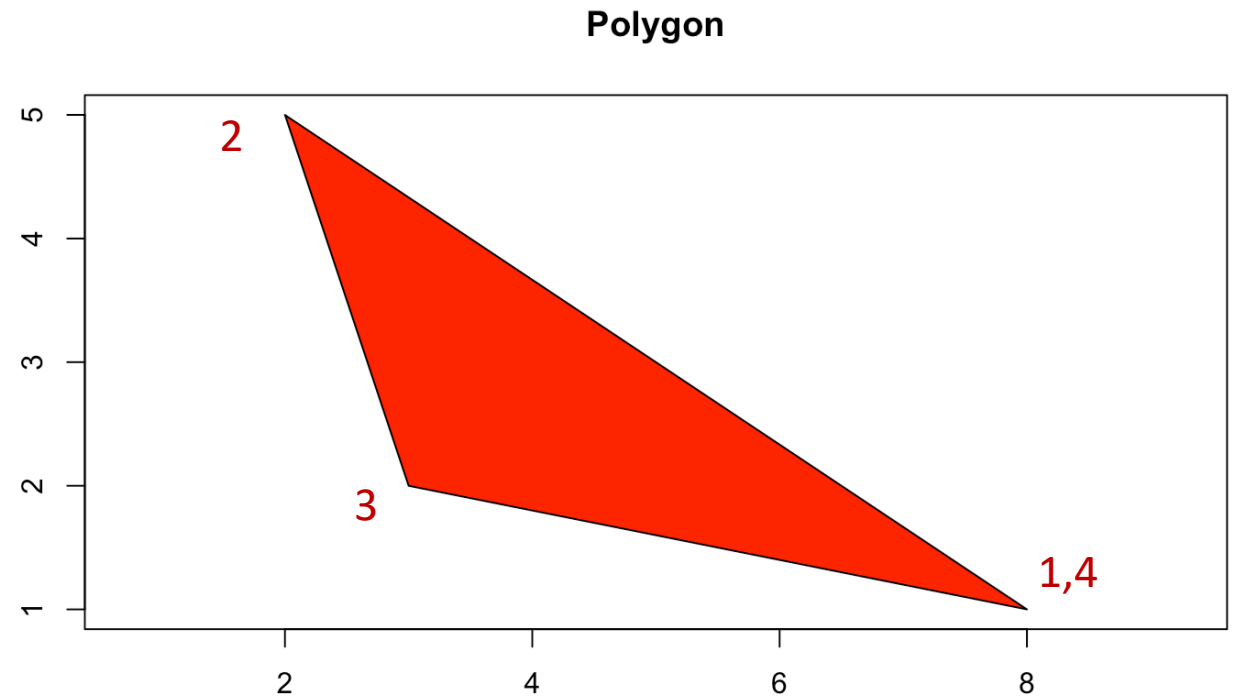**If so, what shape does it make?**
If not, why?

POLYGON()

```
> pg = st_polygon(list(m1))
```
```
Error in MtrxSet(x, dim, type = "POLYGON", needClosed = TRUE) :
  polygons not (all) closed
```

Built in error handling...
You will not have this is your own Java implementations !

```
> (m2 = rbind(c(8, 1), c(2, 5), c(3, 2), c(8,1)))
     [,1] [,2]
[1,]    8    1
[2,]    2    5
[3,]    3    2
[4,]    8    1
> (pg2 = st_polygon(list(m2)))
POLYGON ((8 1, 2 5, 3 2, 8 1))
```

Polygon

Homework hints …

- **Part 2: Define POINT()s and determine the distance between them?**
    - How is a POINT() defined?
        - What components are needed?
    - How many POINT()s do you need to compute a distance operation?
    - How is distance in Cartesian space determined?

Homework hints …

- **Part 3a: Define a LINESTRING()…**
  - What are the minimum number of POINTS()s needed to define a LINESTRING() ?

- **Part 3b: Define a POLYGON()…**
  - What is the minimum number of unique POINT()s needed to define a POLYGON() ?
  - What is the minimum number of POINT()s needed to initialize a POLYGON() ?

How many unique POINT()s are needed to define a valid LINESTRING() and POLYGON() ?

Homework hints …

- **Part 4: Is the POLYGON() closed … aka … is it a valid geometry??**

  - Think of this is building your own error handling
  - What conditional statement is needed to generate a warning like:

```
Error in MtrxSet(x, dim, type = "POLYGON", needClosed = TRUE) :
  polygons not (all) closed
```