Physics Programming in C#

Lecture 2 Tales of the Triangle

(a.k.a. Trigonometry)

Slides & lesson materials by Hans Wichman & Paul Bonsma

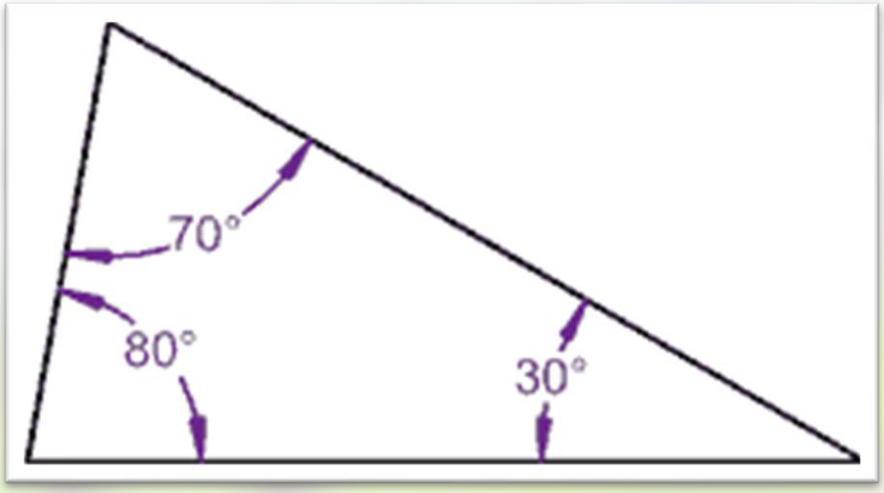
Image: http://communities.ptc.com/people/DanMarotta

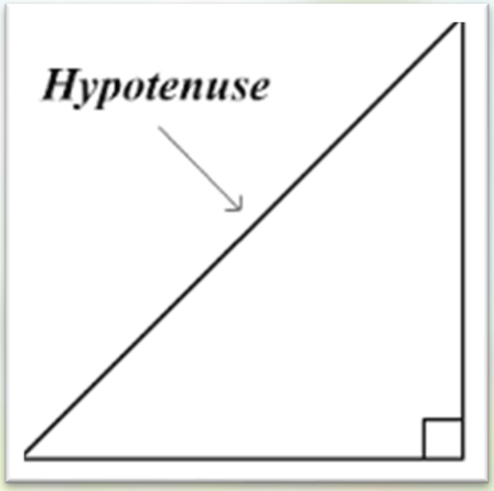
Lecture overview

- What is trigonometry?
- What do we use it for?
- Angles and polar coordinates
- Polar to Cartesian mapping and back
- Practical applications
- 2D rotation formula
- Acceleration & Euler Integration
- Assignment 2

Trigonometry

- Math that deals with:
  - triangles
  - ratios between angles and side lengths in a triangle
  - mostly right-angled triangles



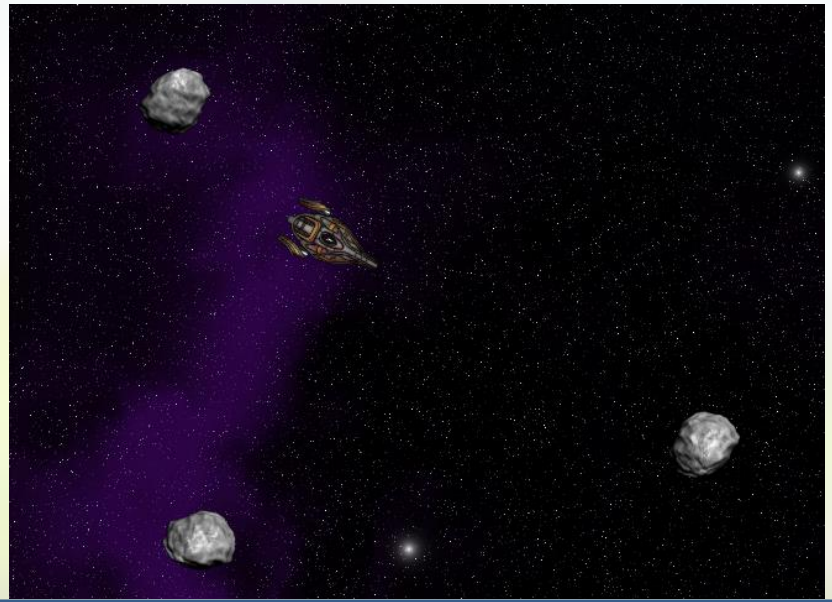


What can you use it for?

Examples:

- Move in a rotated direction
- Rotate towards a target
- Orbit around a target

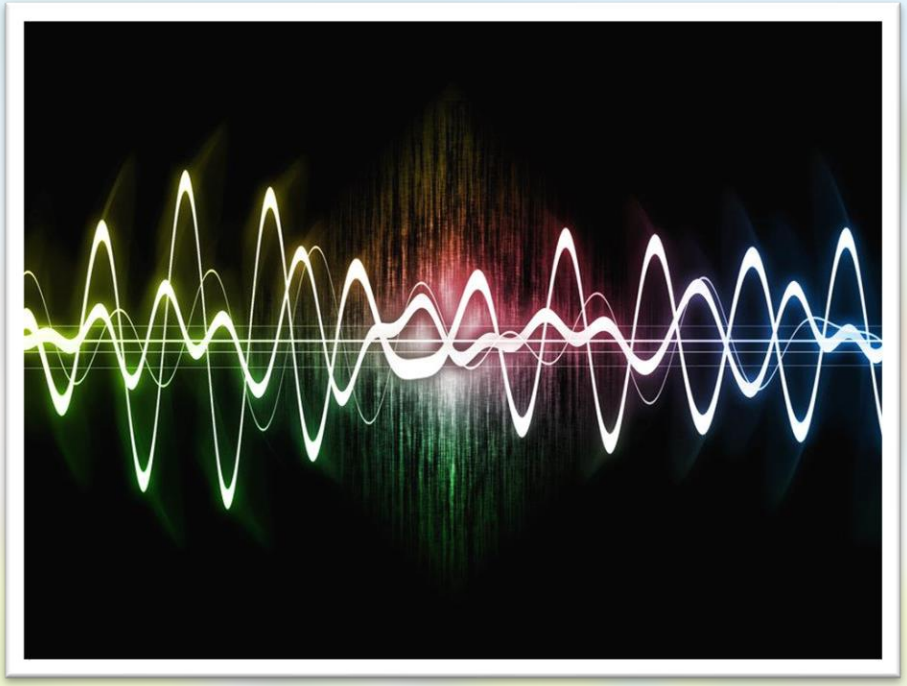
<demo>



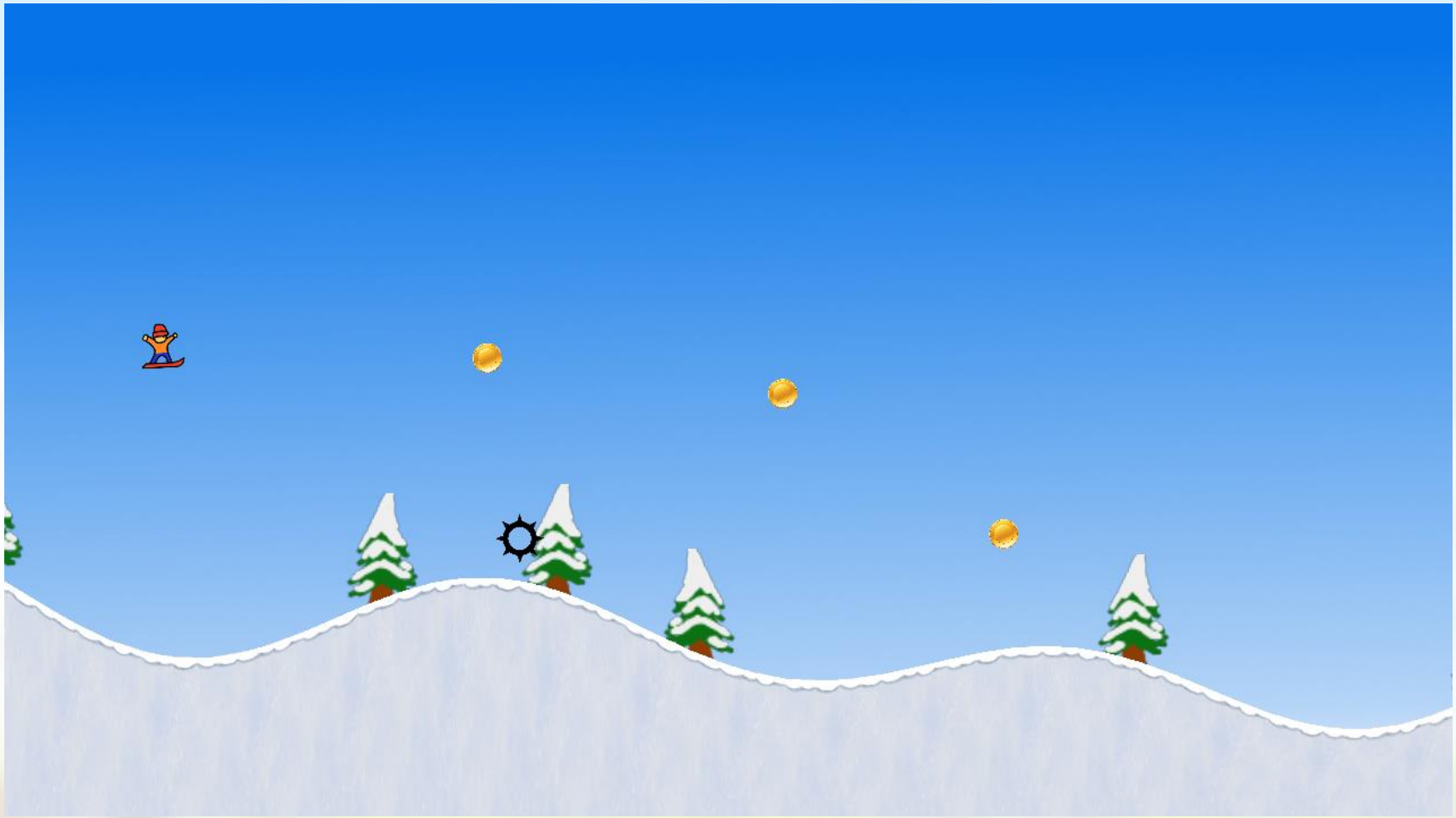
What can you use it for? (II)

Examples:

- Generating audio waves
- Realistic motion
- Screen shake effects
- Compressing (image, audio) data (?!)
- Etc…



What can you use it for? (III)



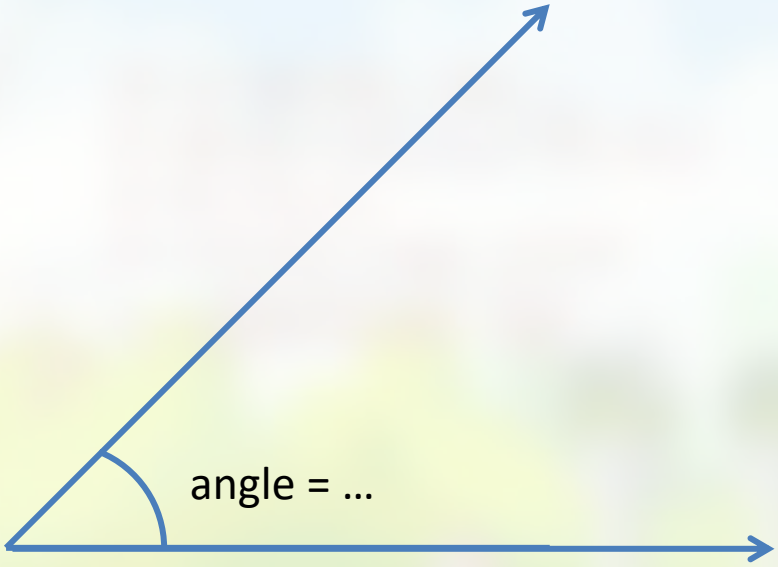
GXP Game by Soma Nyíro

Angles

The corner stones of tri-angles

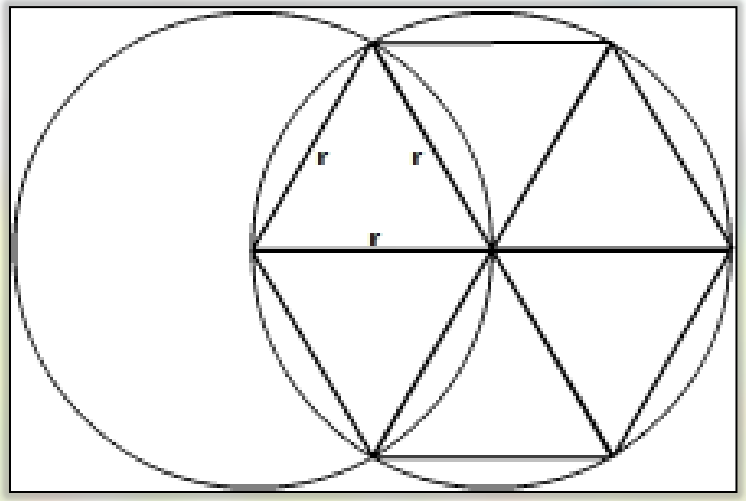
Trigonometry is about angles

- Angles are used to express directions
- Define amount of “turn” from **one** direction **to** another
- Two different measurements: degrees & radians



Angles in **degrees**

- Circles can be divided in **360 degrees**.
- 360 was “arbitrarily” chosen by man:
  - Babylonians counted to 60 on their fingers instead of 10
  - Circle divides nicely into 6 triangles with equal sides
  - 360 has a lot of factors (1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 72, 90, 120, 180, 360)



Angles in **radians**

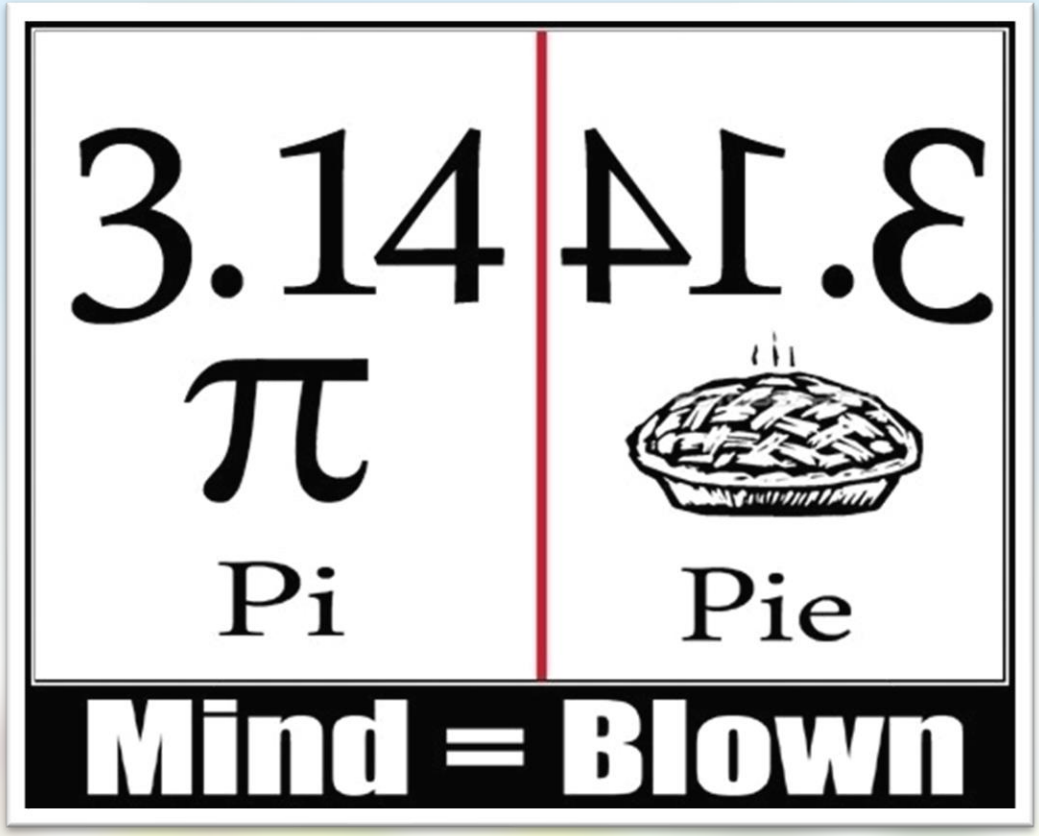
- Circles can be divided in **2π radians**
- Based on ratio between *diameter d* and *circumference C* of a circle:

  C/d = $\pi \approx 3.14159265\ldots$ (unending)
- In C#: Math.PI (or Mathf.PI in GXP / Unity)

$$\frac{C}{d} = \pi$$

$$C = d\pi$$
$$C = 2\pi r$$

Radians



Angles

- In conclusion: 2 units of measurement for angles, based on:
  - 360 slices: degrees
  - the 2π circumference of a unit circle: radians
- And of course these are related: two sides of same coin
- Game engines often use both types: conversion required!

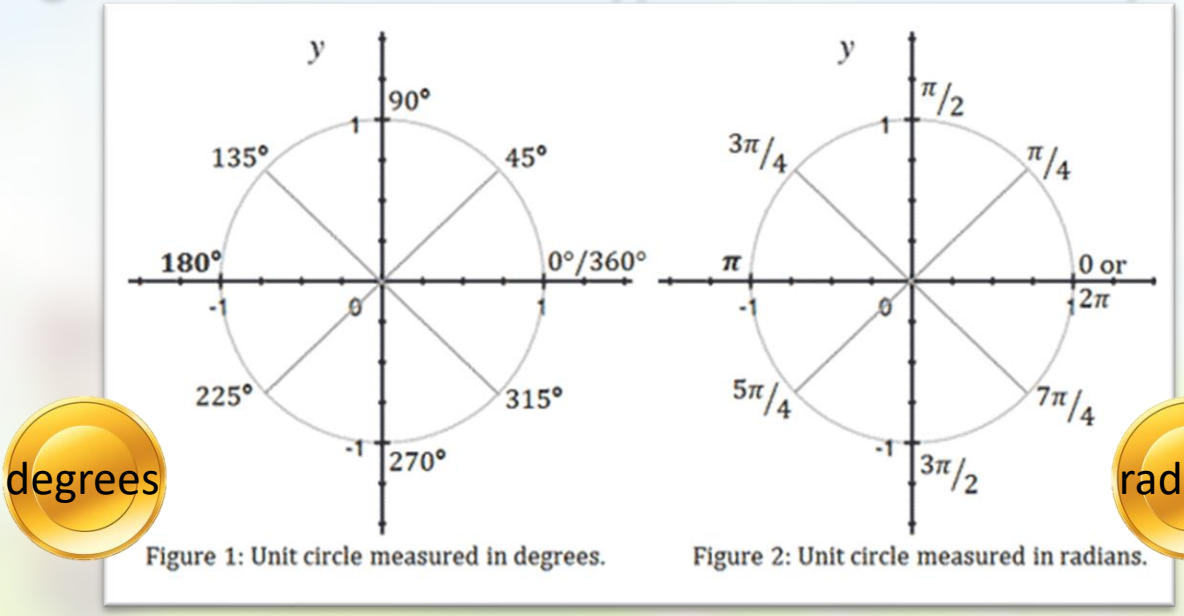


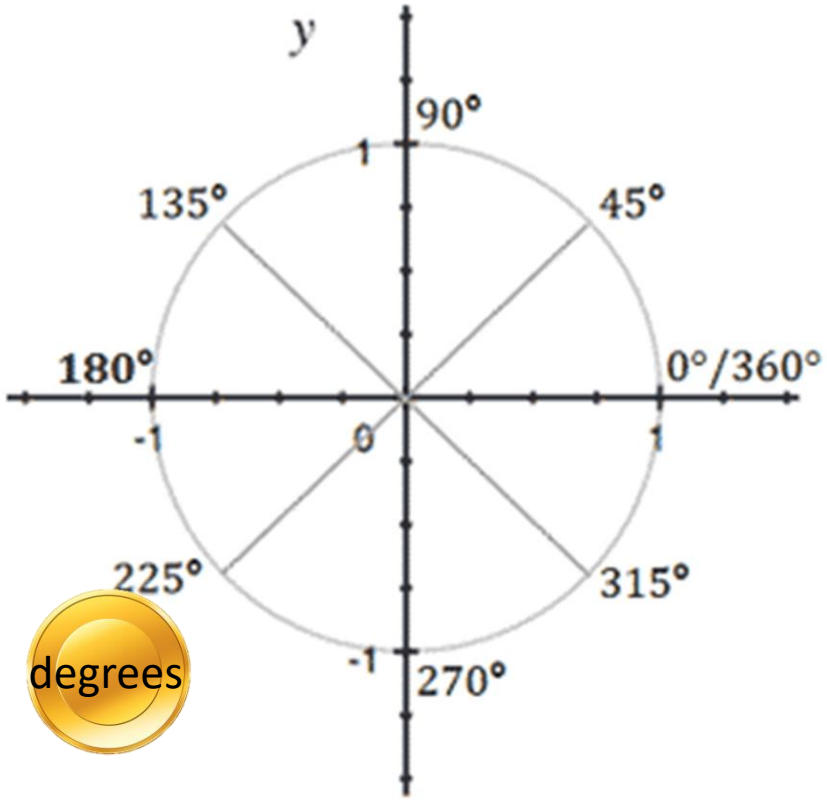
Figure 1: Unit circle measured in degrees.

Figure 2: Unit circle measured in radians.

degrees

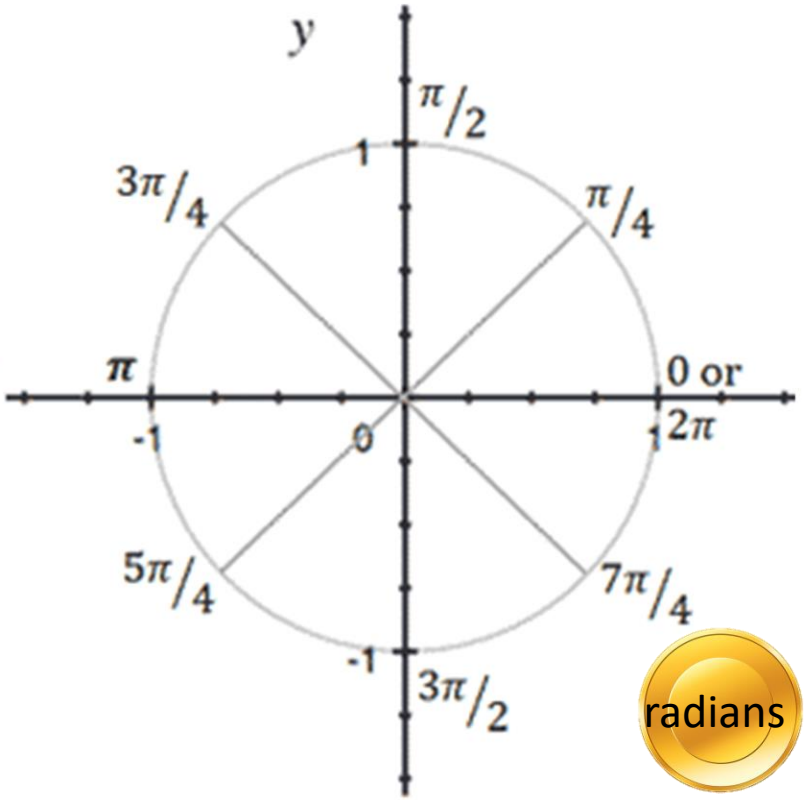
radians

Remember these figures!



degrees

Figure 1: Unit circle measured in degrees.



radians

Figure 2: Unit circle measured in radians.

Angle conversion

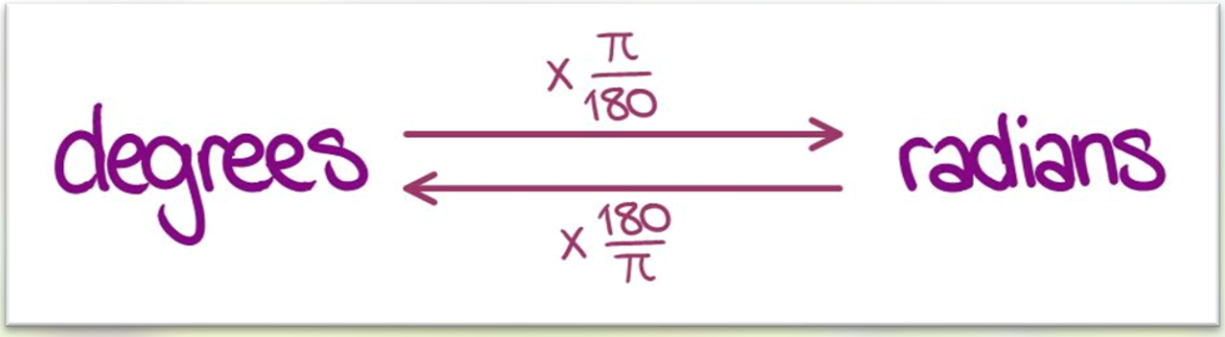
Angle Conversion ➔ $360° = 2\pi^{rad}$

- From degrees to radians:

$$1° = \frac{\pi}{180}\, rad$$

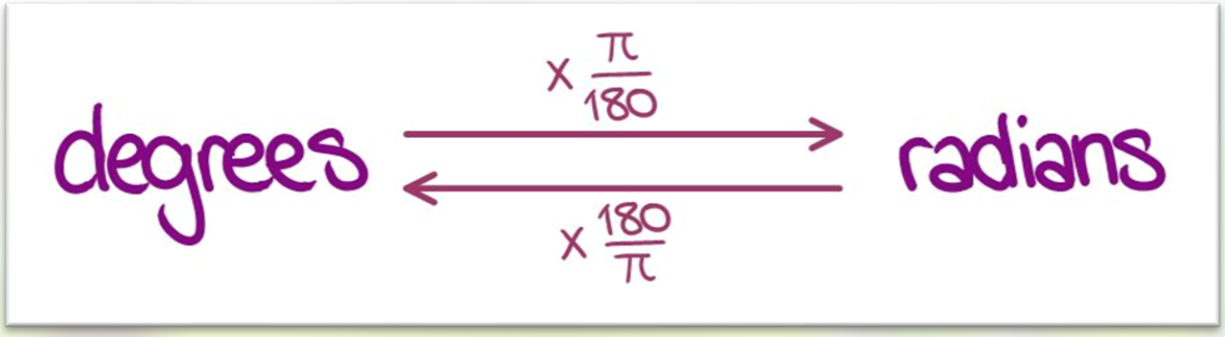
- From radians to degrees:

$$1^{rad} = \frac{180°}{\pi}$$



Exercises

- Calculate circumference of a circle with d = 4
- Calculate circumference of a circle with r = 4
- Convert 45 degrees to radians
- Convert 135 degrees to radians
- Convert 1.75 Pi to degrees



Angle conventions

Angle conventions: angle 0

- For example: mySprite.rotation = 0; (degrees)
- Example: +001_samples\MovingSpaceship
- If you want to keep the programming easy: only use/accept **correctly** aligned sprites!

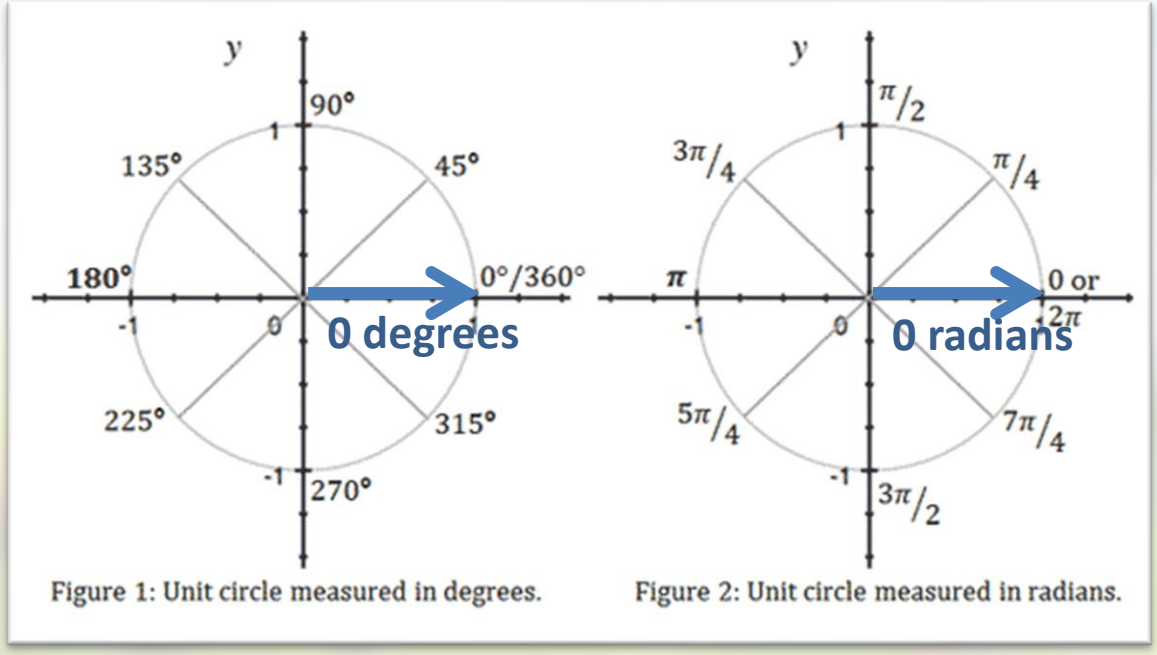


Figure 1: Unit circle measured in degrees.

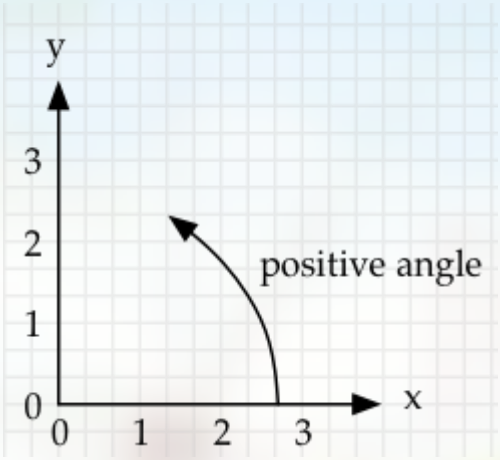
Figure 2: Unit circle measured in radians.

Angle conventions: positive angle

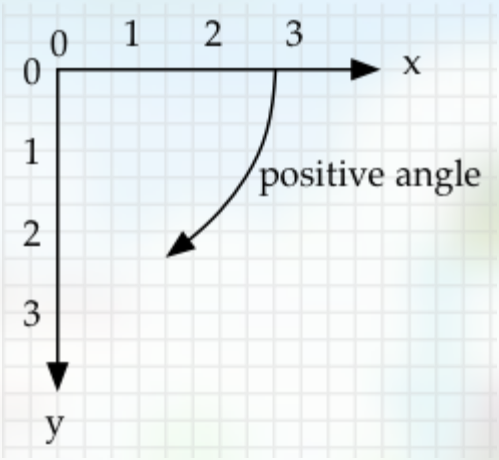
- *A positive rotation angle rotates from +x to +y*

  (More precisely: rotating (1,0) 90 degrees gives (0,1).)

+ angle when +y is up = CCW:



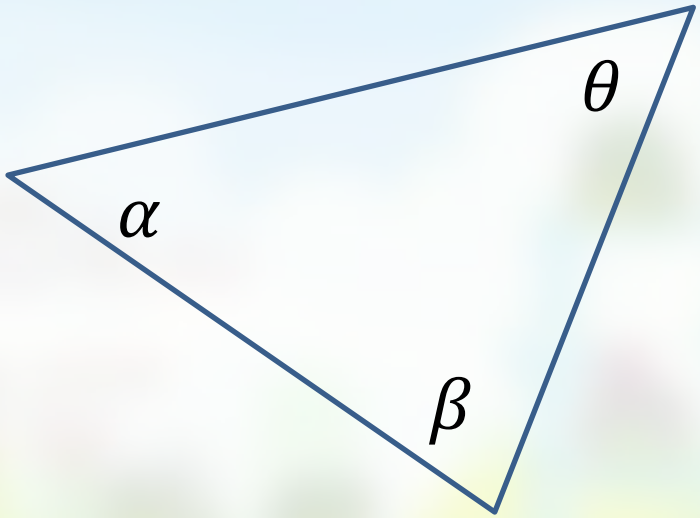
+ angle when +y is down = CW:



- GXPEngine? +001_samples\RotatingSpaceship

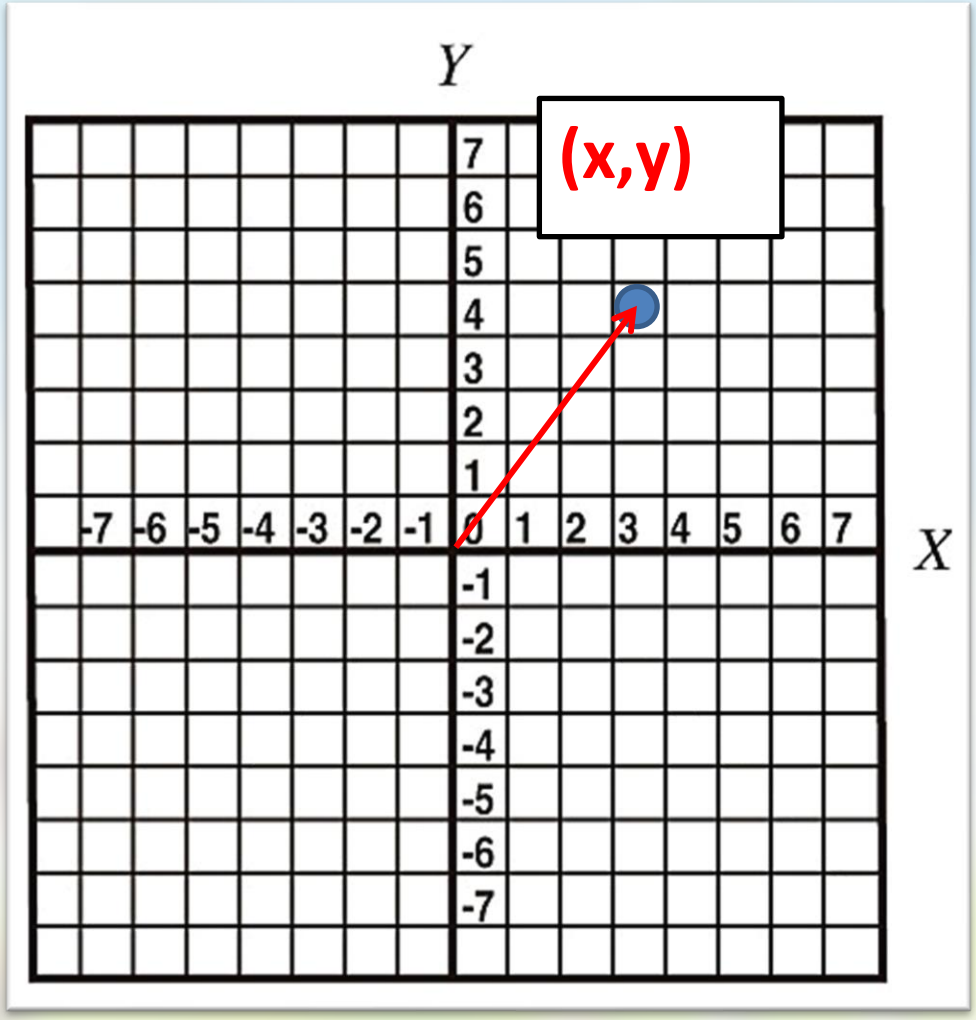
Angle conventions: angle names

- Angles are denoted by using Greek letters:
  - α alpha
  - β beta
  - θ theta
  - … etcetera

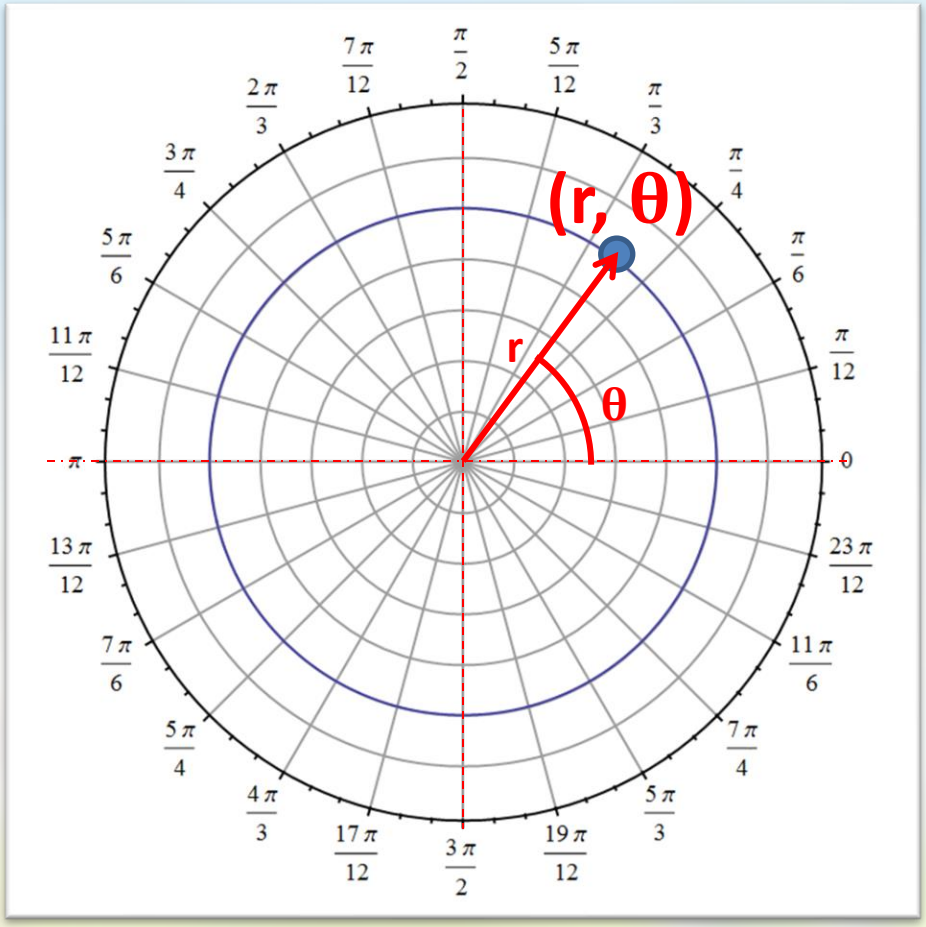


Polar Coordinate System

Cartesian Coordinates

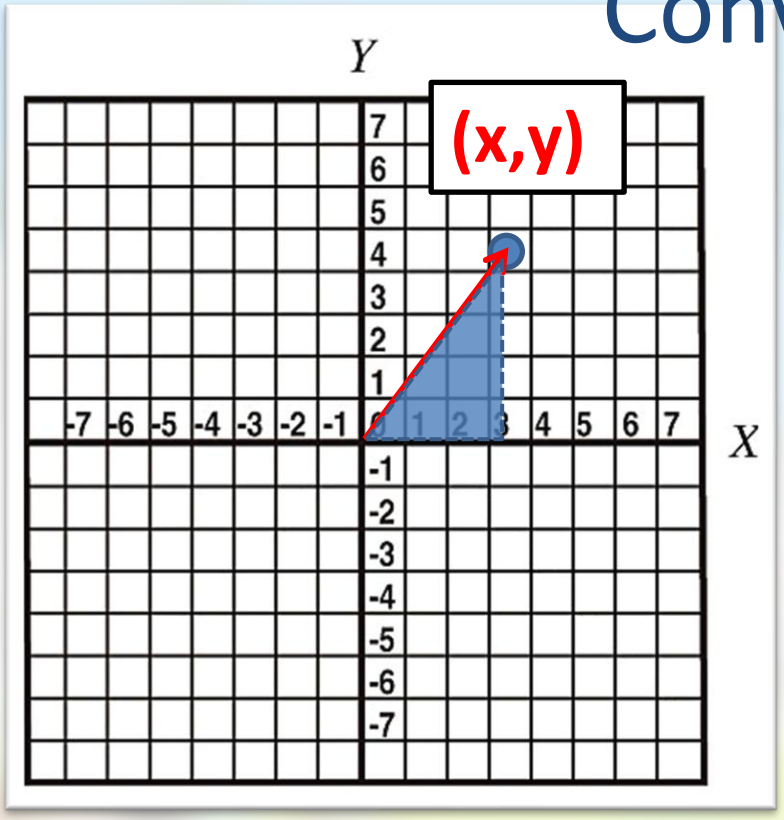


Polar Coordinate System

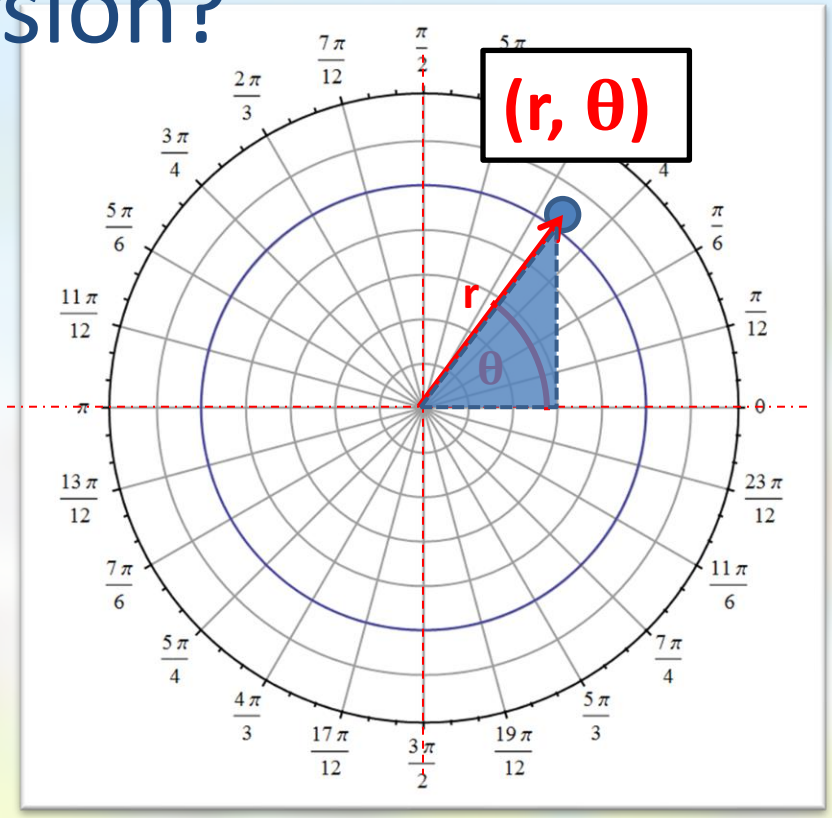


Coordinate System Conversion

Conversion?



Cartesian Coordinate System



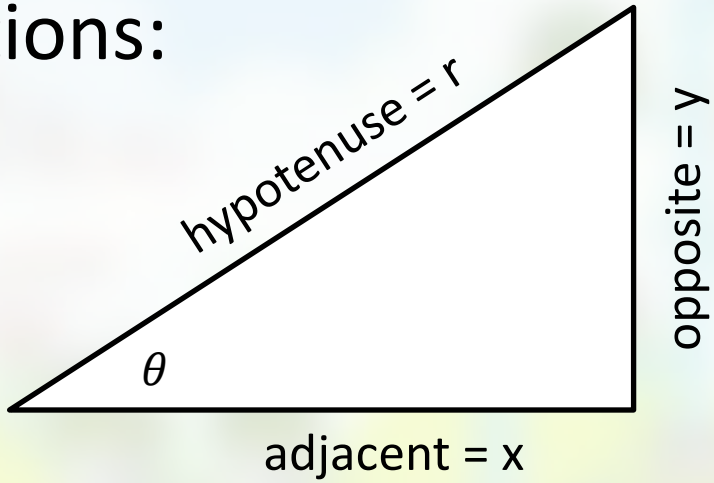
Polar Coordinate System

Trigonometric functions

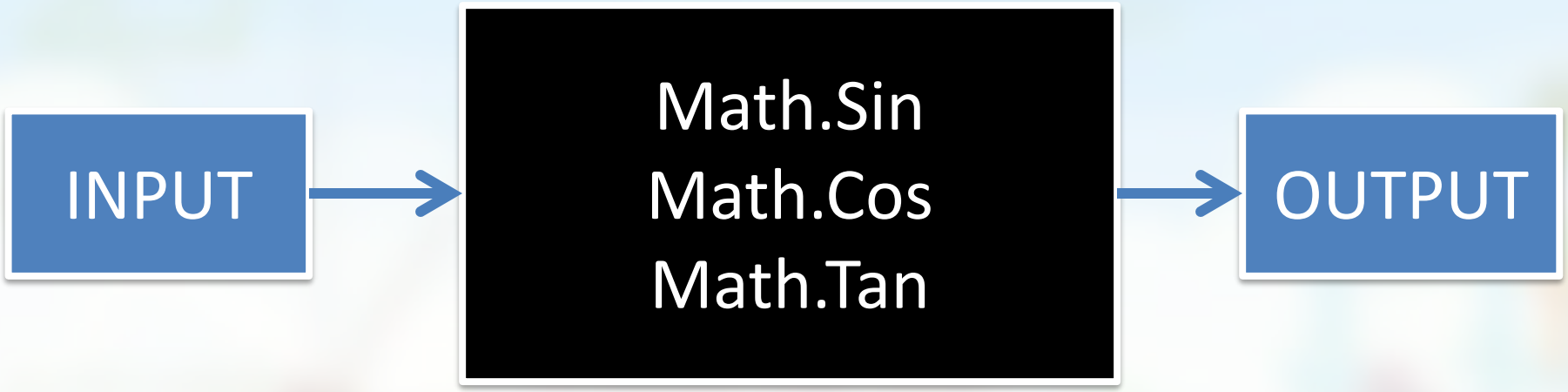
Trigonometric functions

- Describe relations between **the angles** in a right sided triangle and **the length** of that triangle's sides

- Most well known trig functions:
  - $\boldsymbol{sin\,\theta}$ = **opp/hyp** = $\boldsymbol{y/r}$
  - $\boldsymbol{cos\,\theta}$ = **adj /hyp** = $\boldsymbol{x/r}$
  - $tan\,\theta$ = opp/adj = y/x
  - θ is always in **radians !**

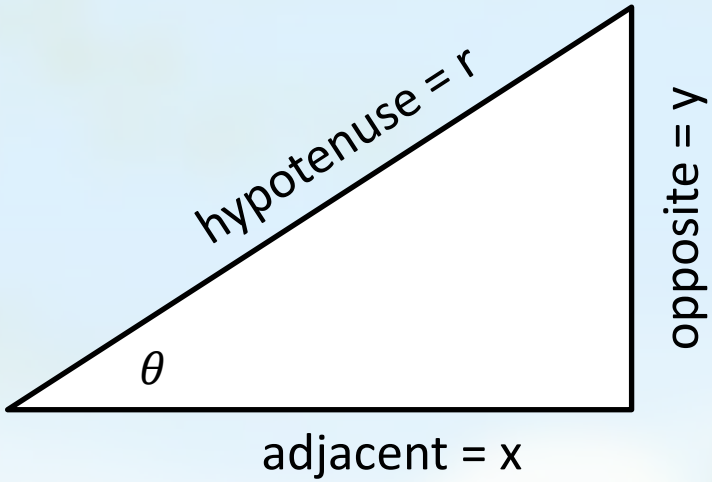


Trig functions are black box functions



Memorizing trig functions

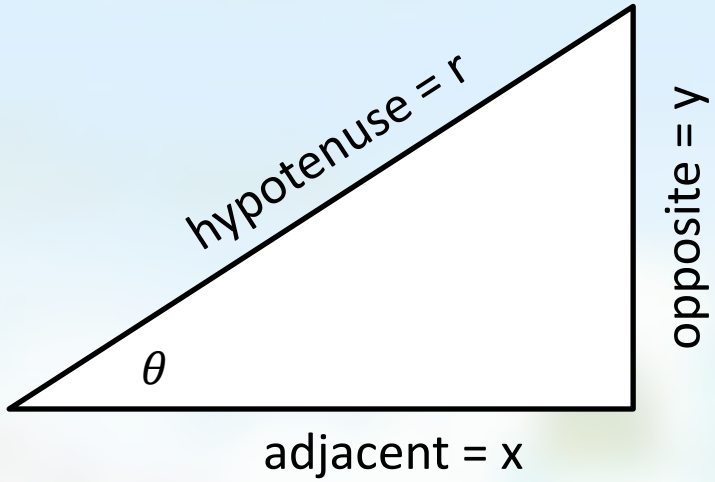
- $\boldsymbol{sin}\,\theta = \textbf{o}\text{pp} / \textbf{h}\text{yp} = y/r$
- $\boldsymbol{cos}\,\theta = \textbf{a}\text{dj} / \textbf{h}\text{yp} = x/r$
- $\boldsymbol{tan}\,\theta = \textbf{o}\text{pp}/ \textbf{a}\text{dj} = y/x$



From Polar to Cartesian

If we know that:

$sin\,\theta$ = opp/hyp = y/r
$cos\,\theta$ = adj/hyp = x/r
$tan\,\theta$ = opp/adj = y/x

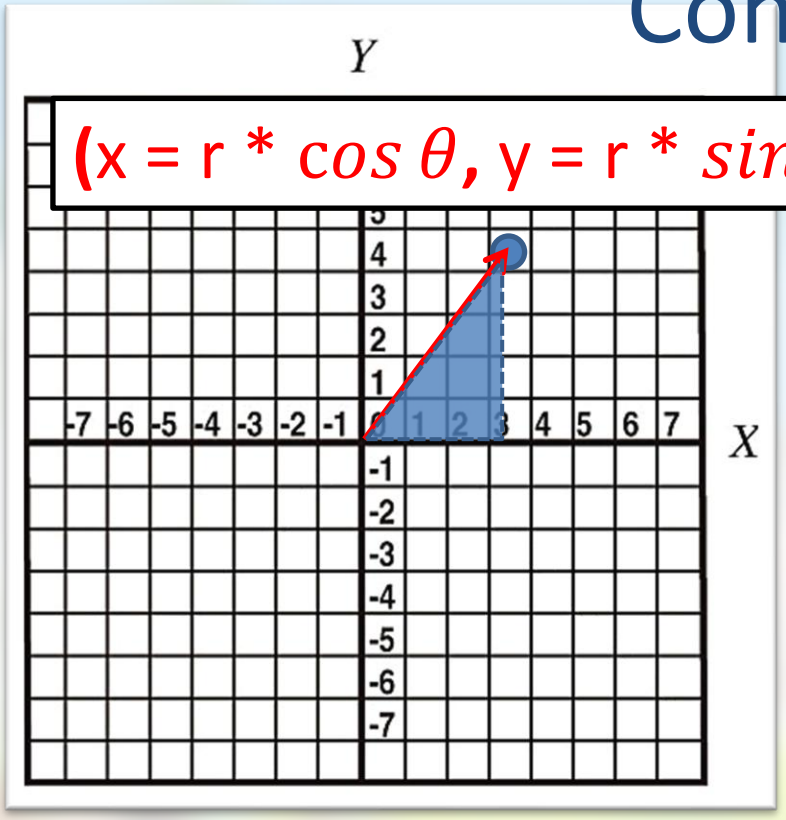


then, given **(r, θ)** what is **(x, y)** ?

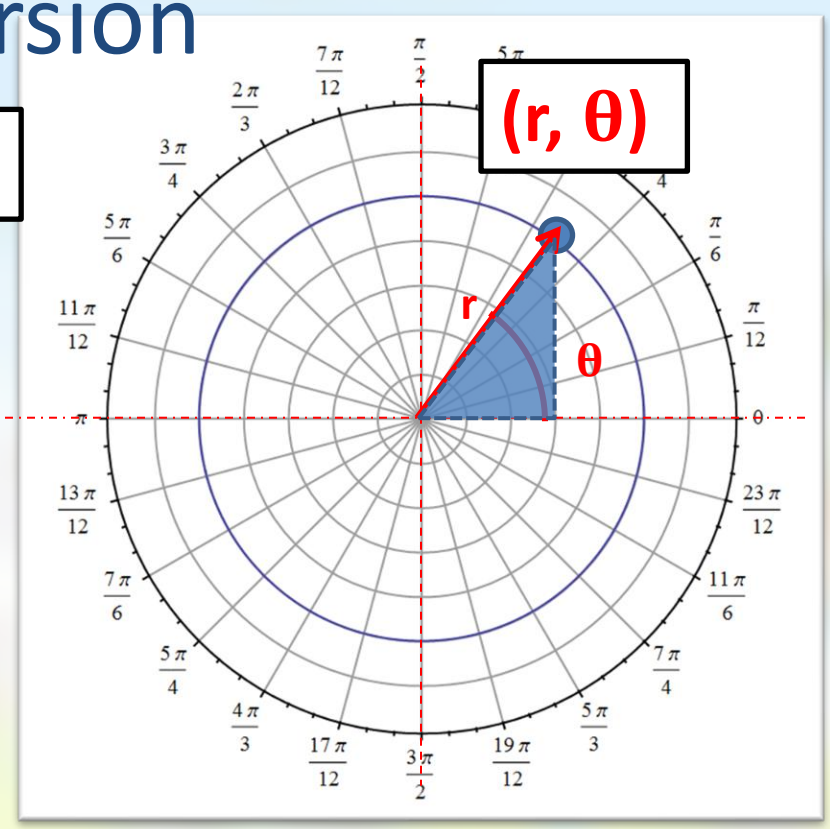
(r, θ) → x = r * $cos\,\theta$
(r, θ) → y = r * $sin\,\theta$

Coordinate System Conversion

Conversion



Cartesian Coordinate System



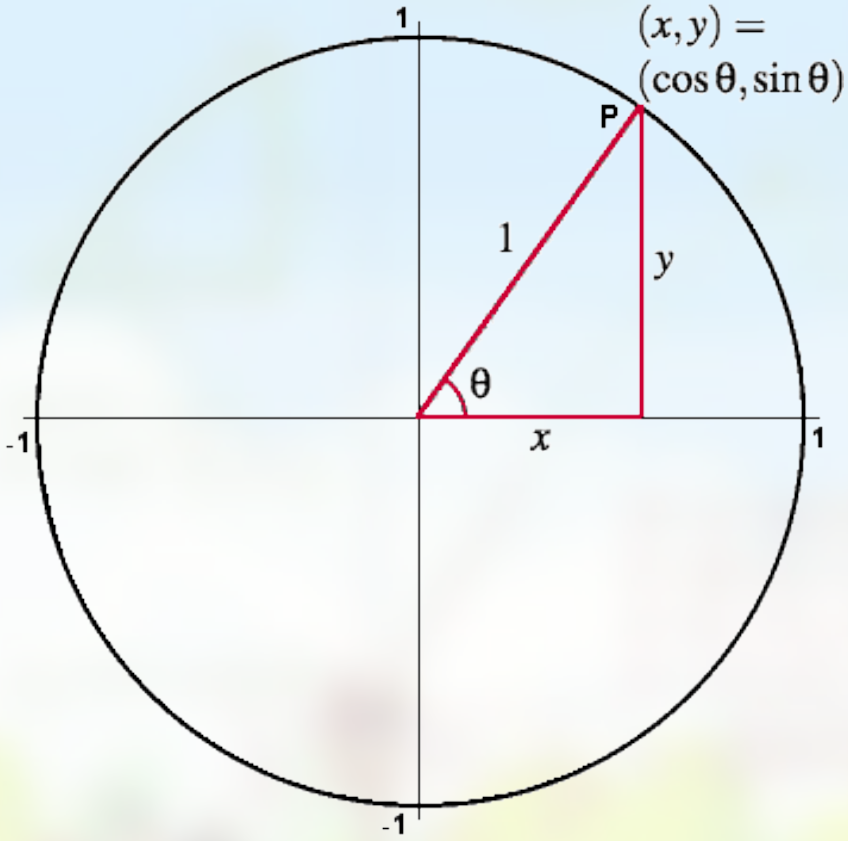
Polar Coordinate System

Other angles?

- Our current definition of sin, cos, tan only works for angles between 0 and 90 degrees.
- What about other angles?

- There is a *more general definition*, based on the *unit circle*. (Next slide)

Unit circle, r = 1



NEW AND IMPROVED DEFINITION!

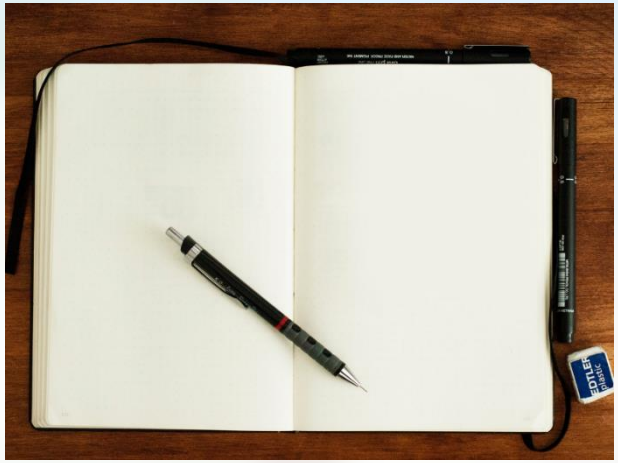
If (x,y) is a *unit length* vector at *angle* θ, then:

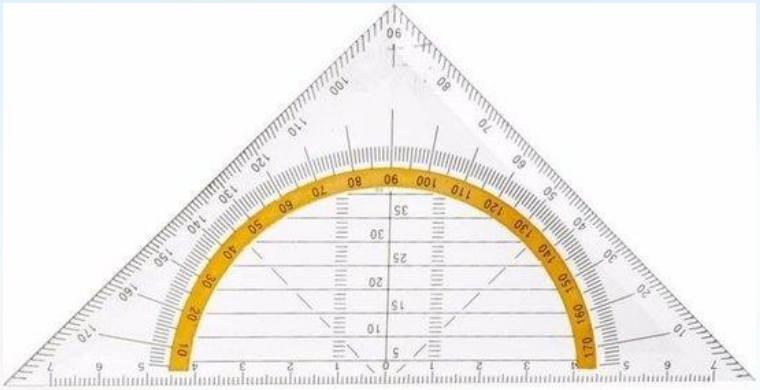
- $\cos(\theta)$ = x
- $\sin(\theta)$ = y
- $\tan(\theta)$ = y/x

Note: this works for *any* angle, including negative / very large

Math Survival

- So: if you find yourself on a deserted island without a calculator or computer, but with these things…:

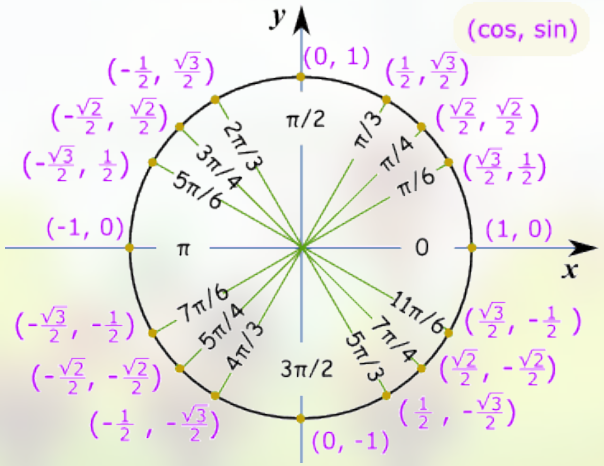


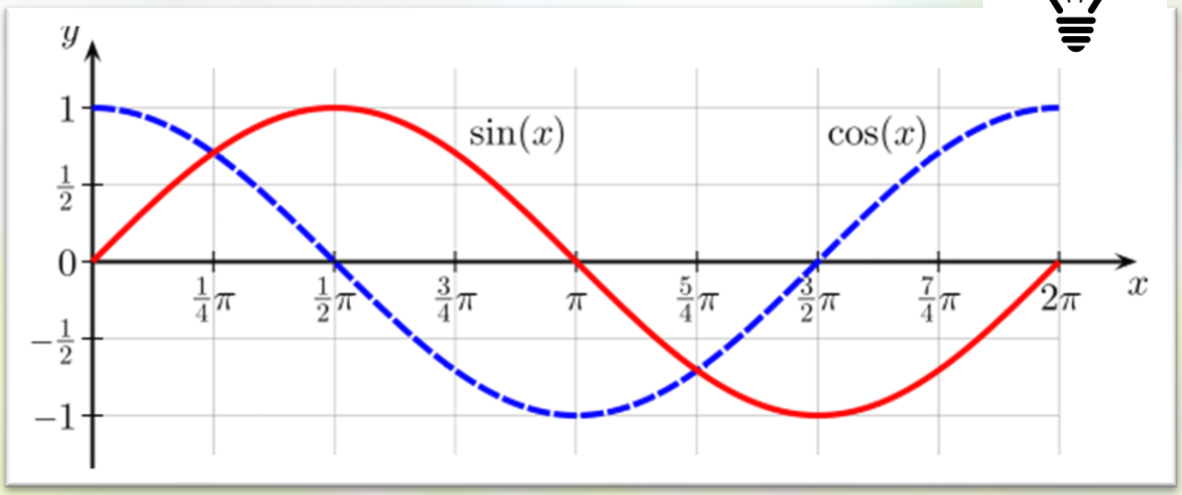


- …you can still approximate sin and cos! ☺
- (Might be useful for building a fancy treehouse, or determining time of day / date)

sin & cos graphs

- Armed with this improved definition, we can (let our computer) draw the *graph* for the *sin* and *cos* functions
  - Input: any number (interpreted as radians, so 2π is "full circle")
  - Output between -1 and 1
- *Memorize these graphs!*

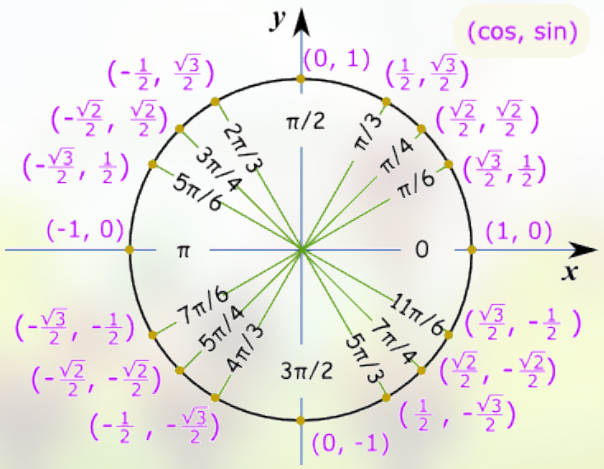


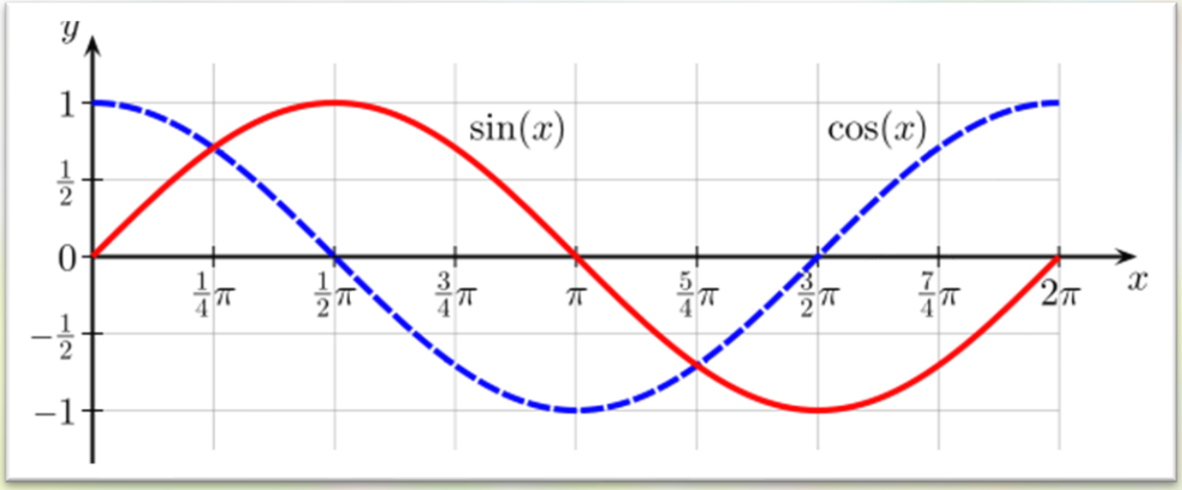


Symmetry

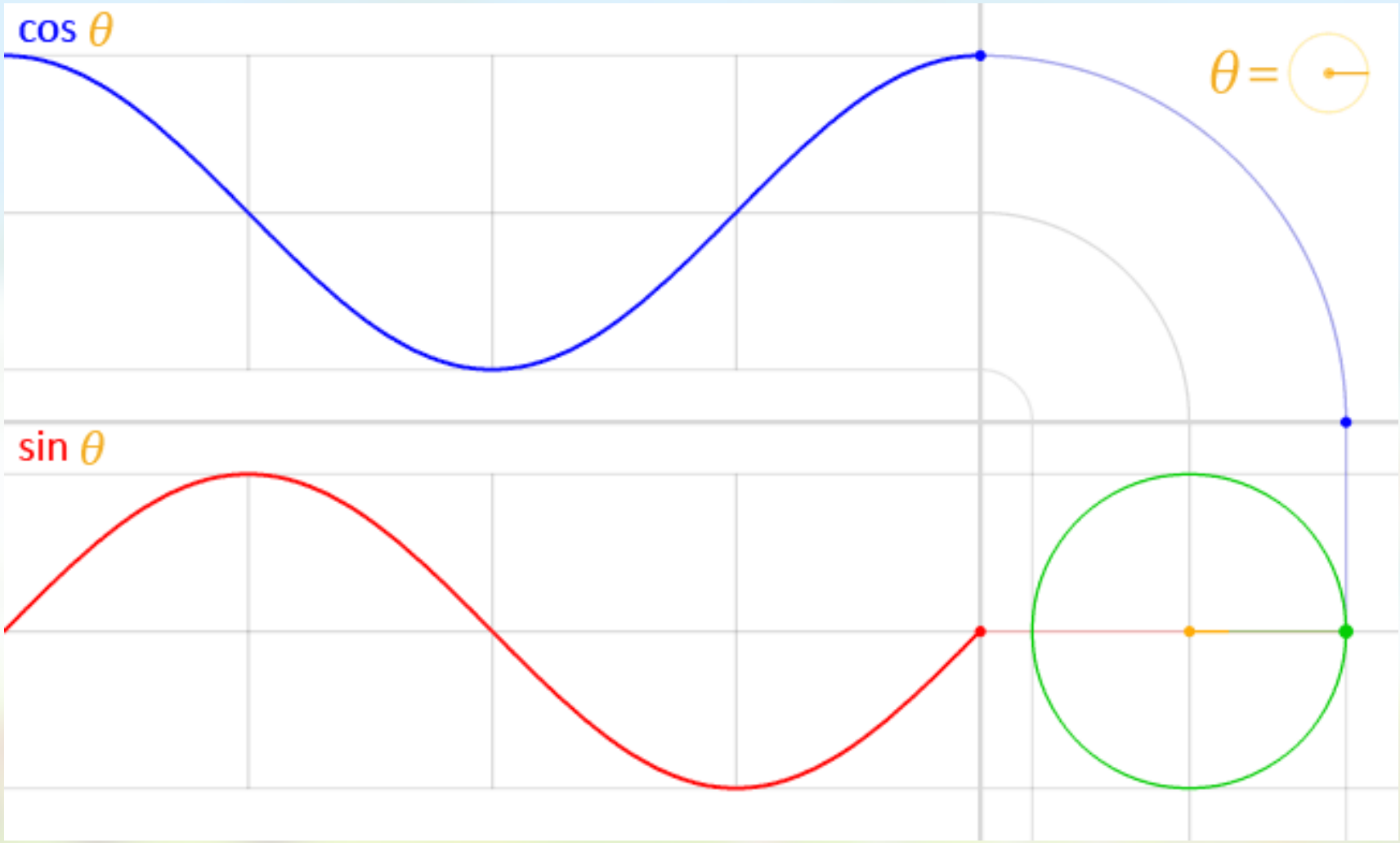
Just looking at these graphs, we can deduce a lot of things, e.g.:

- Cos/sin → same movement, with a "phase shift" of $\frac{1}{2}\pi$:
  - $\cos(\alpha) = \sin(\alpha + \frac{1}{2}\pi)$ & $\sin(\alpha) = \cos(\alpha - \frac{1}{2}\pi)$
  - $\sin(-\alpha) = -\sin(\alpha)$ & $\cos(\alpha) = \cos(-\alpha)$
  - etc…



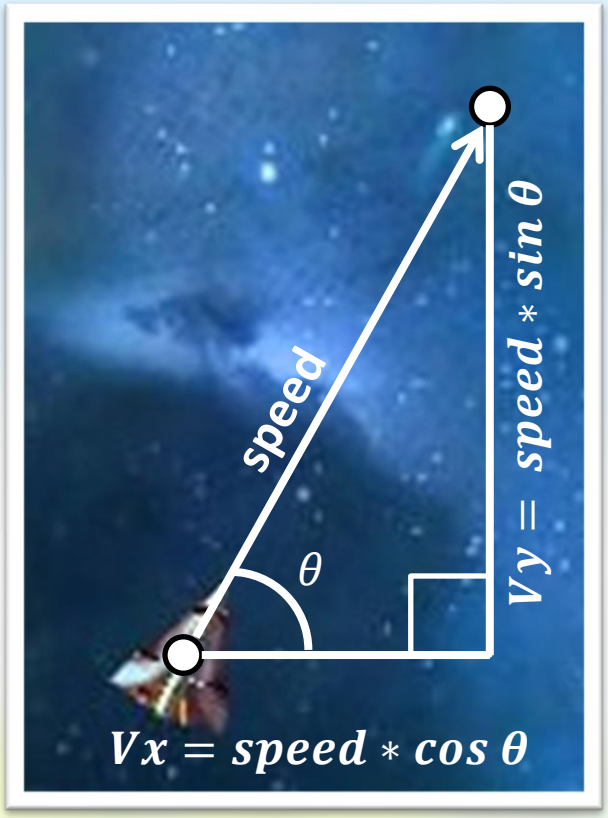


“Realtime” demo of cos/sin



Applications of Co(sine)

Move in a direction



Move in a direction

- Different ways (direct conversion):
  - velocity.x = speed * cos (angle)
  - velocity.y = speed * sin (angle)
- Or (generate unit vector and scale):
  - velocity.SetXY (cos(angle), sin(angle))
  - Velocity *= speed;
- Or (implement helper method to encapsulate cos/sin):
  - Velocity = UnitVector(angle) * speed;

Can't we just use the Sprite Move(x,y) method for this ??

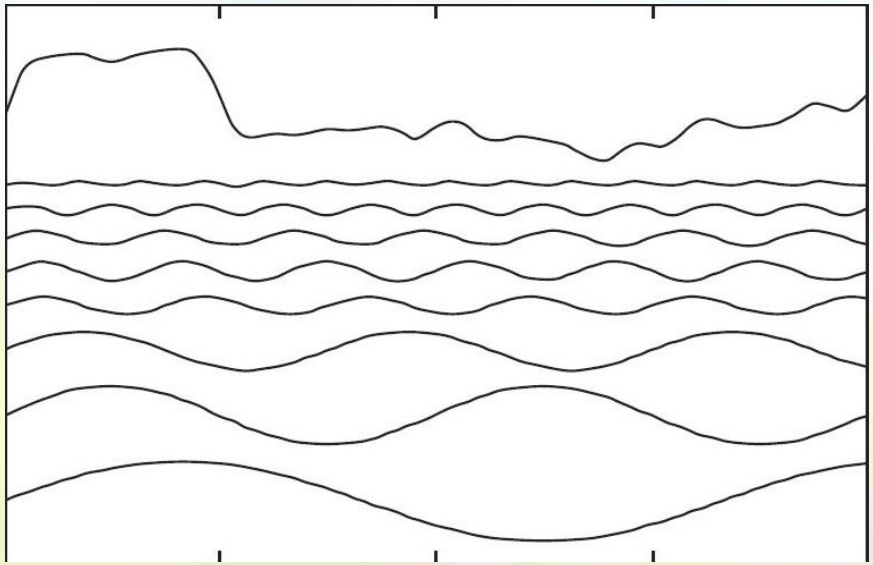
Best way! You should do it like this for Assignment 2!

- There is a catch:
  - angle is stored in sprite.rotation, which is in degrees
  - all trig functions only take radians => *conversion required!*

- Example +002_moving_and_aiming ➔ *Move* method

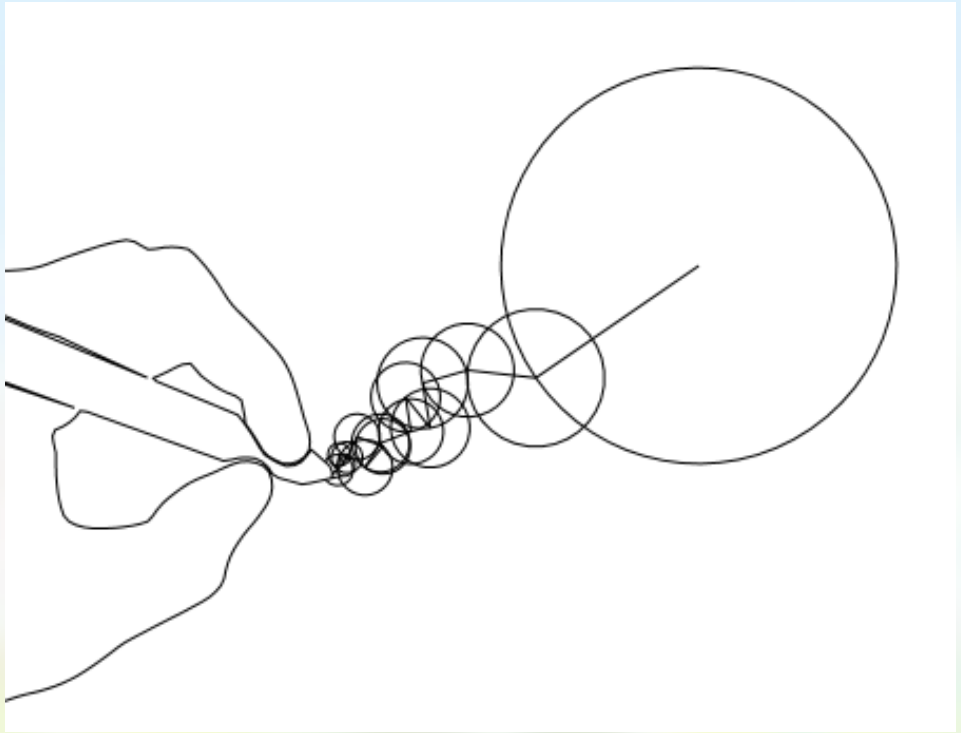
Other using of trig methods

- Cos/sin are just functions, mapping values
- Check out 004_various_applications, which demonstrates:
  - Positions, orbiting, scaling, colors, slashing blades, wave generation, etc



Other using of trig methods

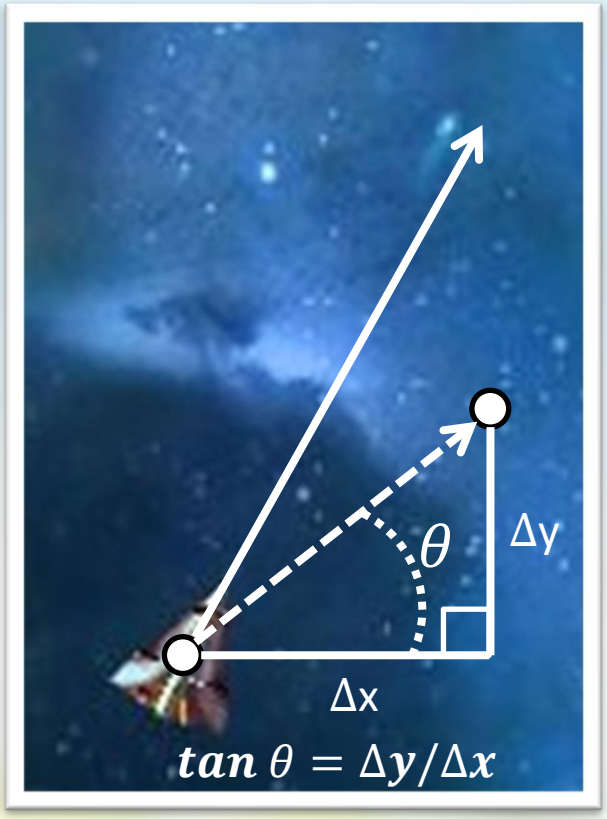
- By adding sine values you can also do… pretty much anything really!

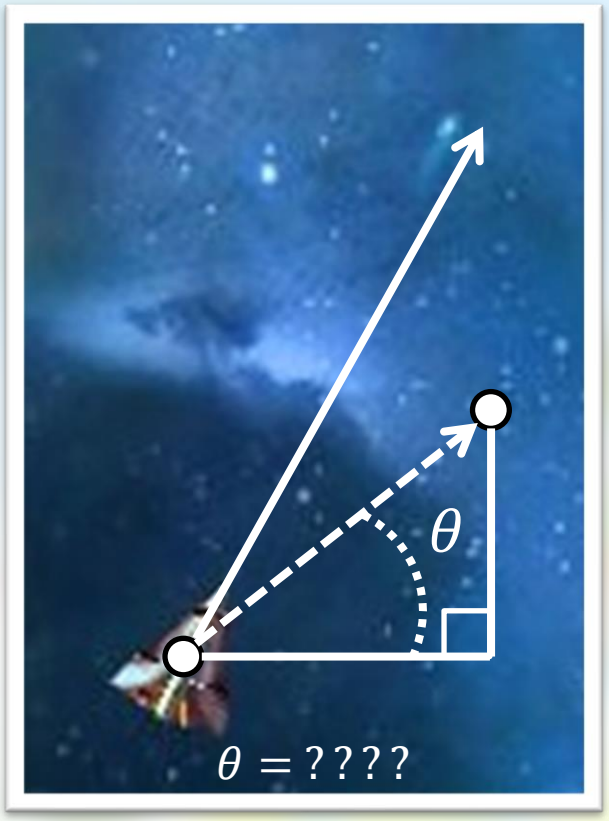


Check it out, it's really cool!
But of course you don't need to know this for the assessment.

- http://www.jezzamon.com/fourier/

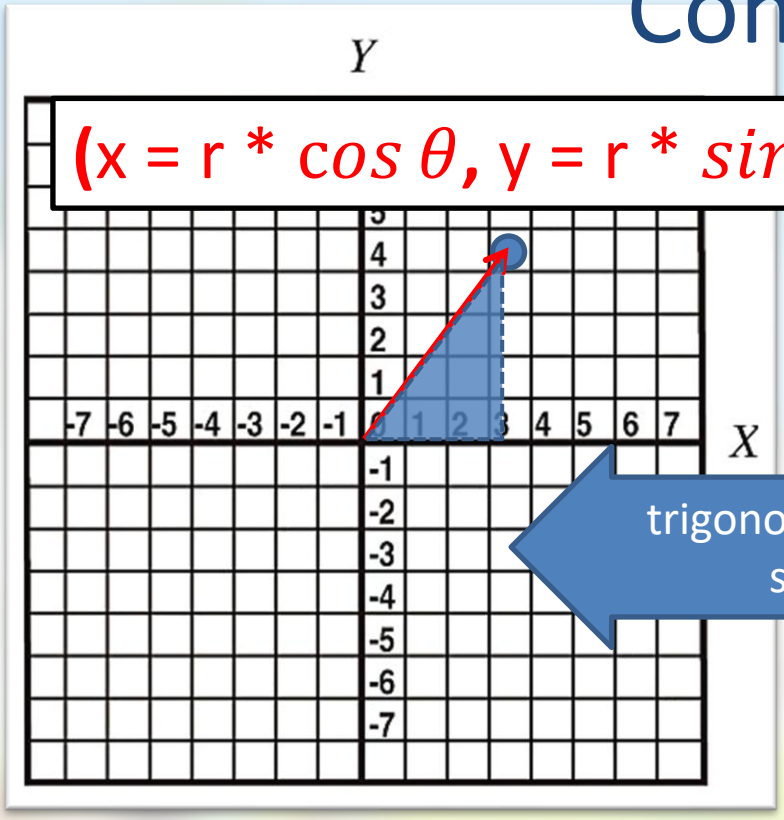
Rotate to a target: Δx Δy to angle?

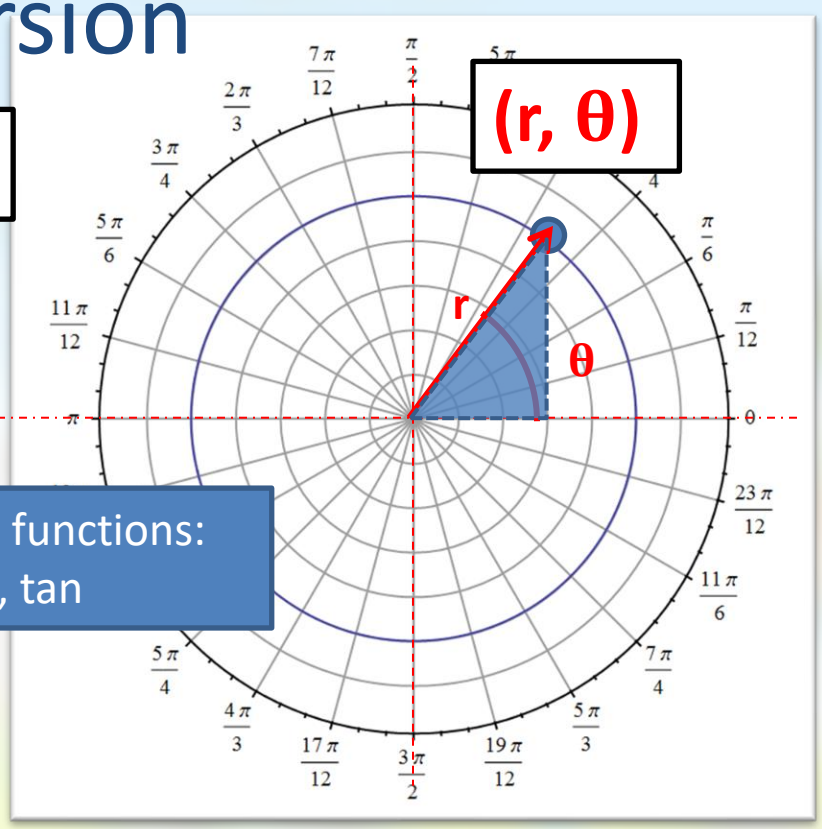




Coordinate System Conversion

Conversion





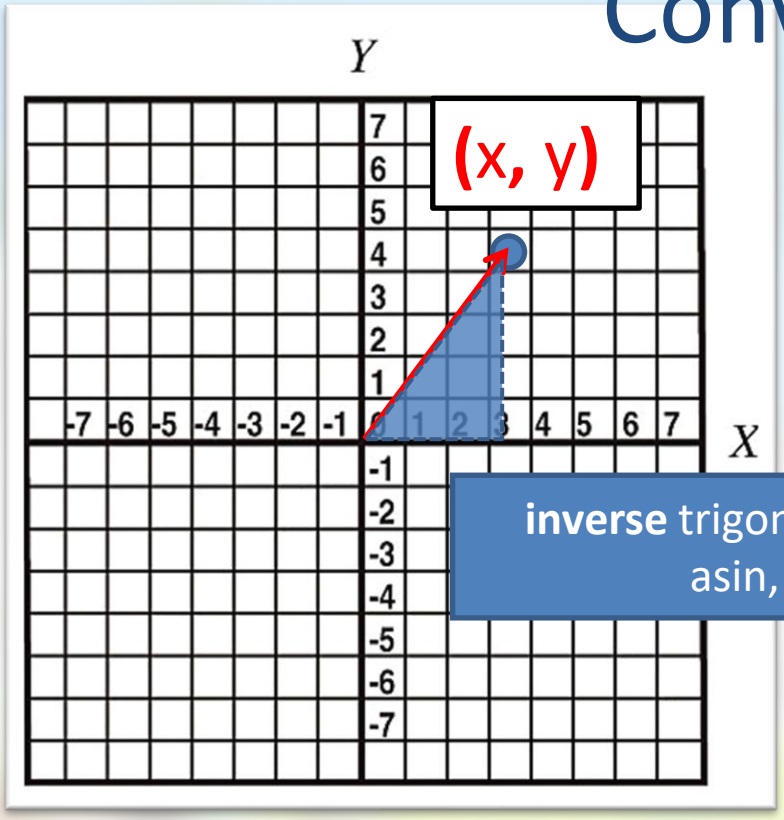
trigonometric functions:
sin, cos, tan

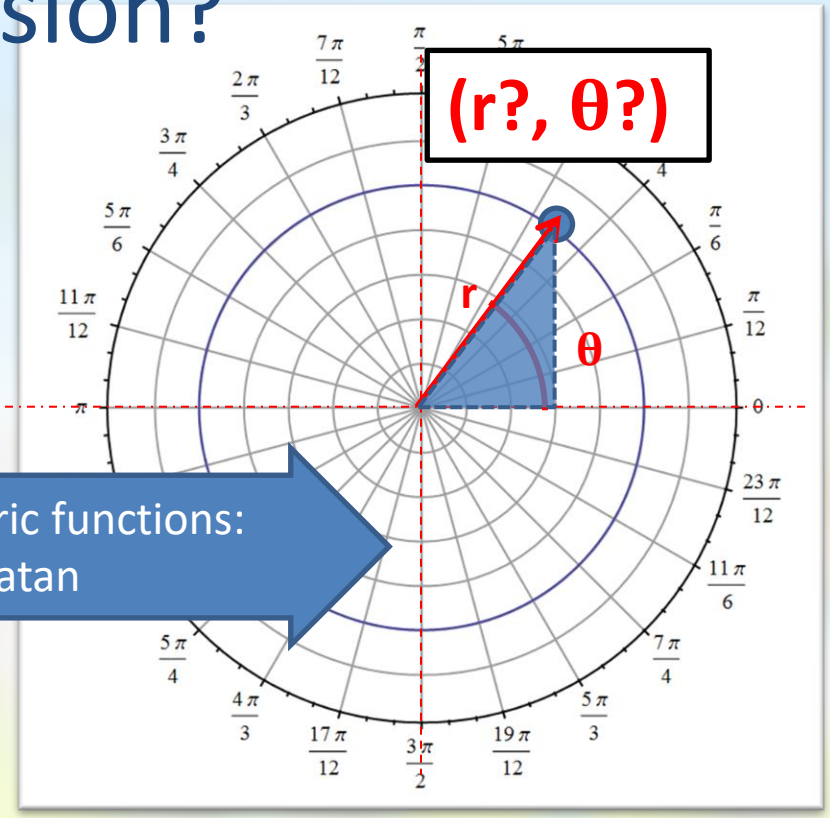
Cartesian Coordinate System

Polar Coordinate System

Coordinate System Conversion

Conversion?





inverse trigonometric functions:
asin, acos, atan

Cartesian Coordinate System

Polar Coordinate System

Inverse Trigonometric functions

Inverse trigonometric functions

- Inverse "undoes" the result of the non-inverse:
  - asin (sin θ) = θ, acos (cos θ) = θ, atan (tan θ) = θ

    (actually, this is only true for θ around 0… See later.)

$$\sin\theta = y/r \quad \cos\theta = x/r \quad \boldsymbol{\tan\theta = y/x}$$

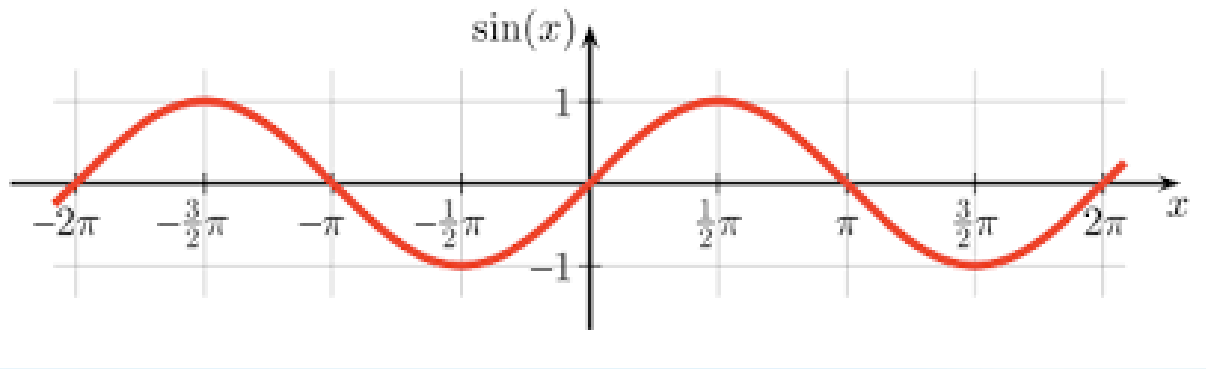
$$\Rightarrow \quad \text{asin}(\sin\theta) = \text{asin}(y/r) \quad \text{acos}(\cos\theta) = \text{acos}(x/r) \quad \boldsymbol{\text{atan}(\tan\theta) = \text{atan}(y/x)}$$

$$\Rightarrow \quad \theta = \text{asin}(y/r) \quad \theta = \text{acos}(x/r) \quad \boldsymbol{\theta = \text{atan}(y/x)}$$

  - Another way to write $asin$ is: $arcsin$, sin^{-1}
  - It's called arc because it results in an arc (angle)
  - Math.Asin, Math.Acos, Math.Atan, Math.Atan2 (**?!**)

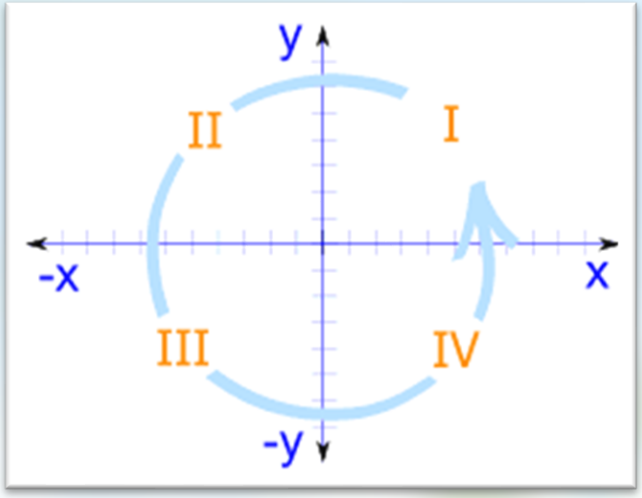
Test

- sin(0) = ?
- asin(0) = ?
- sin(π/2) = ?
- asin(1) = ?
- sin(- π/2) = ?
- asin(-1) = ?
- sin(π) = ?
- asin(0) = ?



Using atan (y/x) to calculate the angle

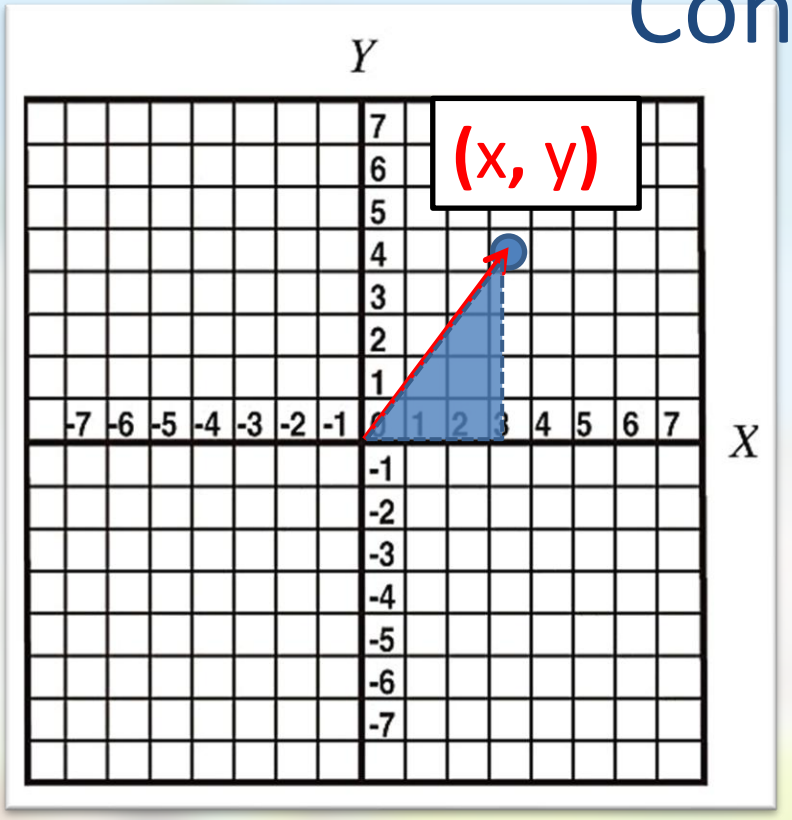
- y/x means:
  - hole in the universe when x ➔ 0
  - y/x == -y/-x and -y/x == y/-x ➔ which quadrant??



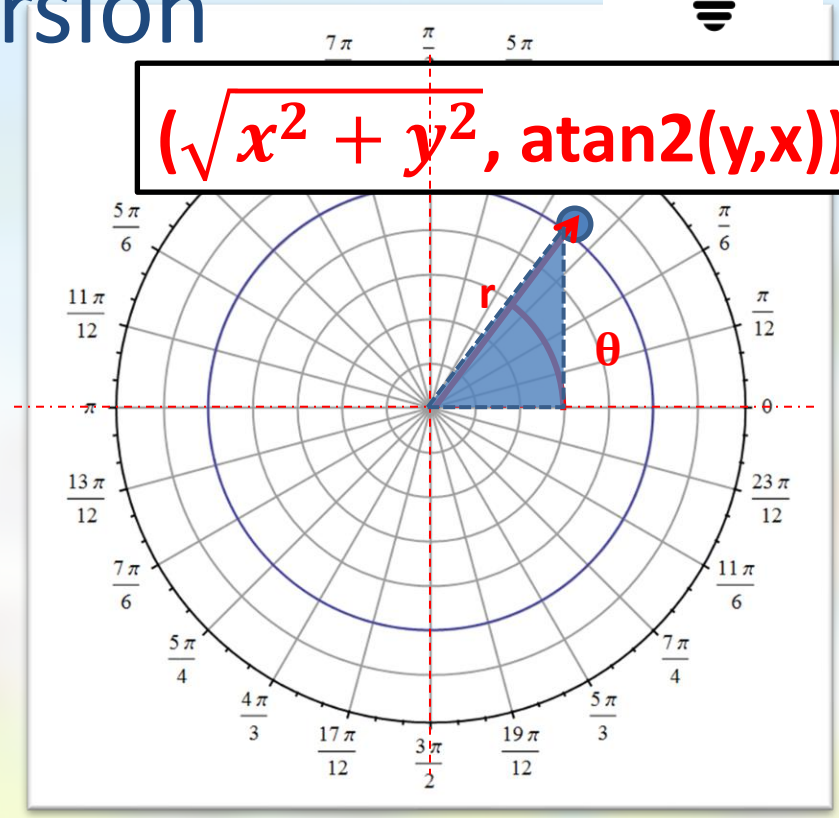
- Conclusion: converting y/x back to an angle is doable but cumbersome with Atan
- **Math.Atan2 (y,x)** takes care of these details
- **Warning**: result is in **radians**
- **Note:** (y,x), not (x,y)!

Coordinate System Conversion

Conversion

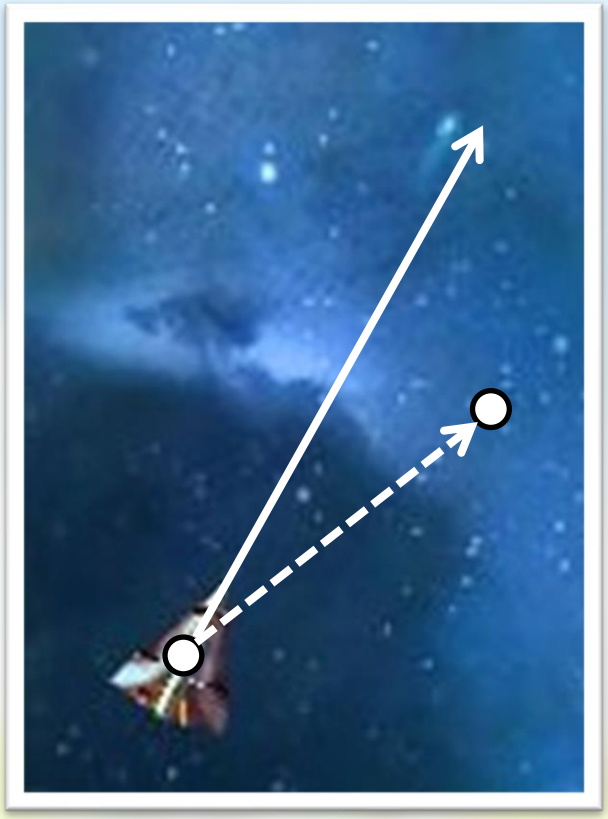


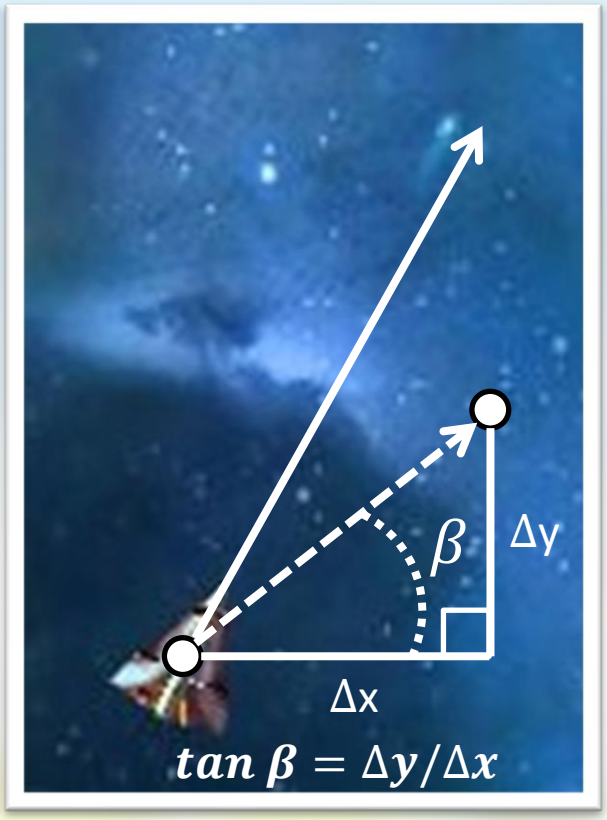
Cartesian Coordinate System

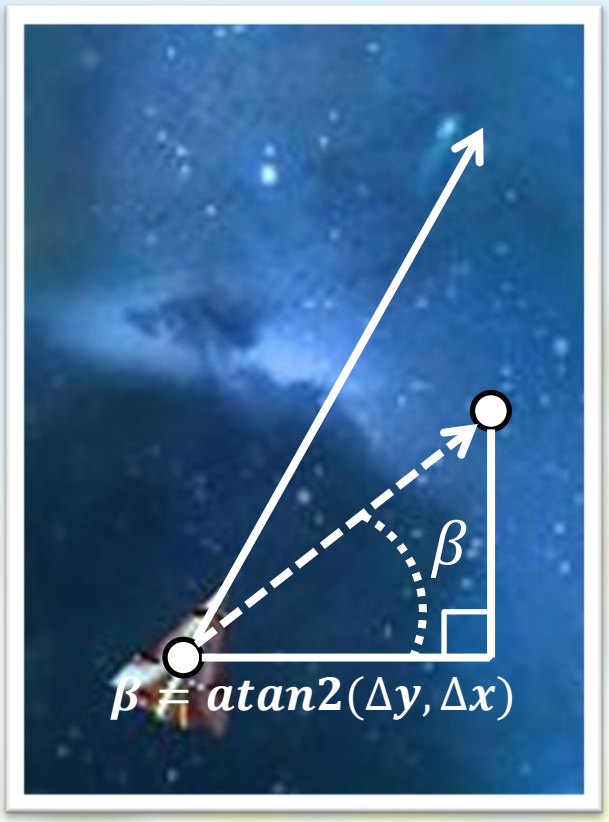


Polar Coordinate System

Rotate to a target: Δx Δy to angle

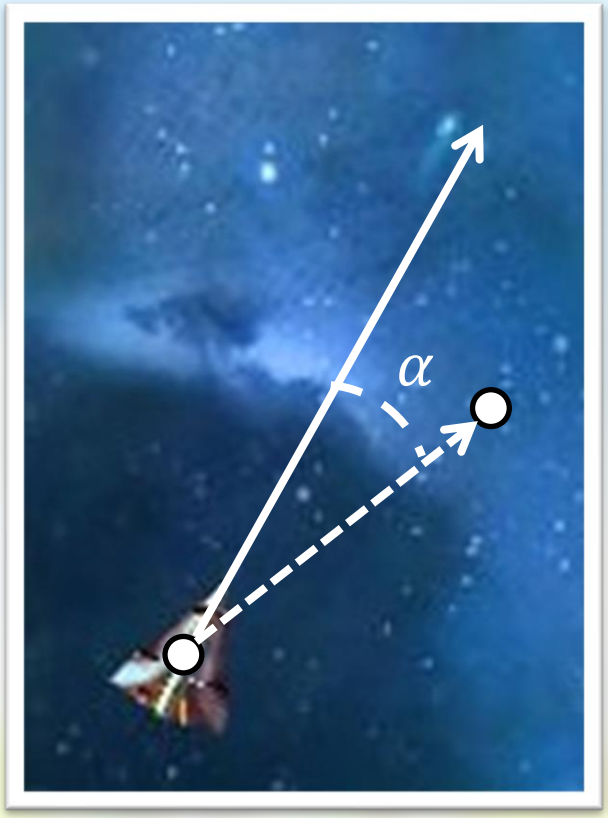


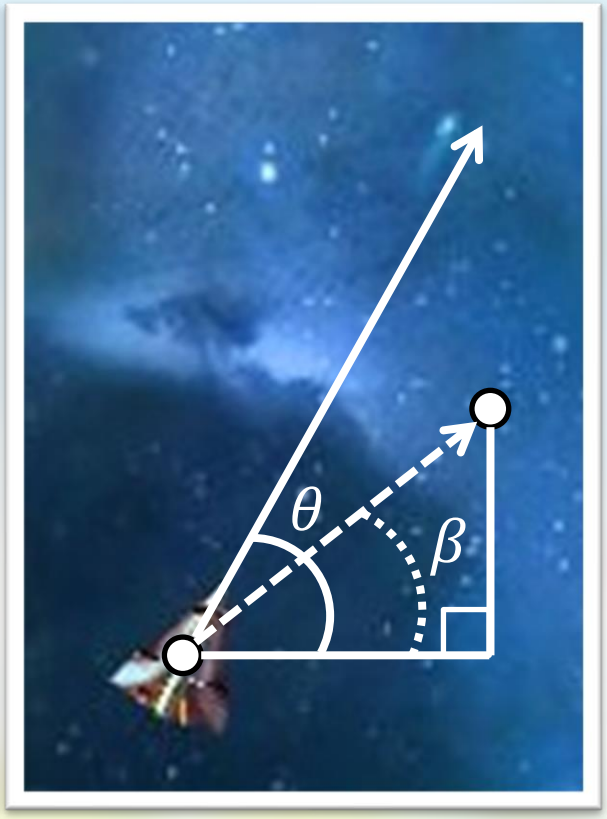


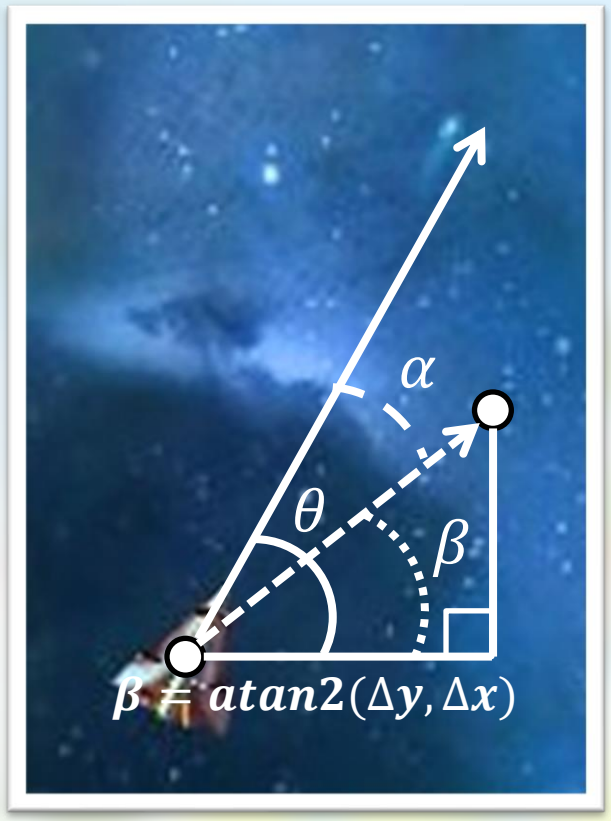


+002_moving_and_aiming ➔ *Aim* method

Rotation **towards** a target?





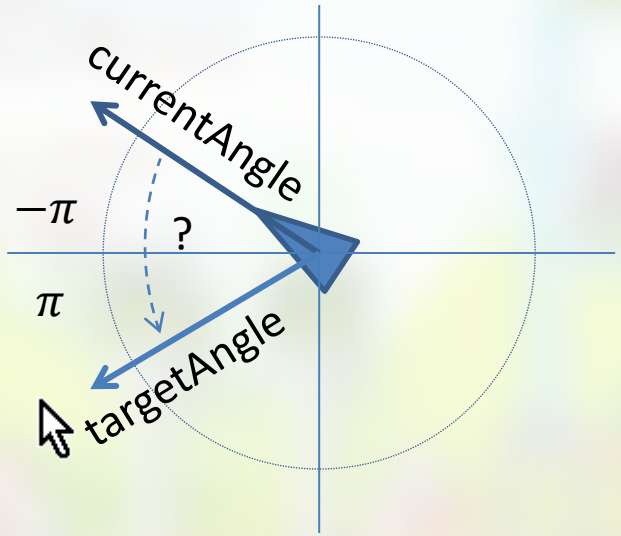


Harder than it seems…

+002_moving_and_aiming ➔ *Aim* method, *press shift*

Rotating **towards** vs **to**

- Rotating towards "eases" to target rotation instead of just setting it, in pseudo:
  - if (targetAngle > currentAngle) currentAngle++;
  - if (targetAngle < currentAngle) currentAngle--;
- Issue is that atan output range is from $(-\pi, \pi)$
- Fix is part of Assignment 2

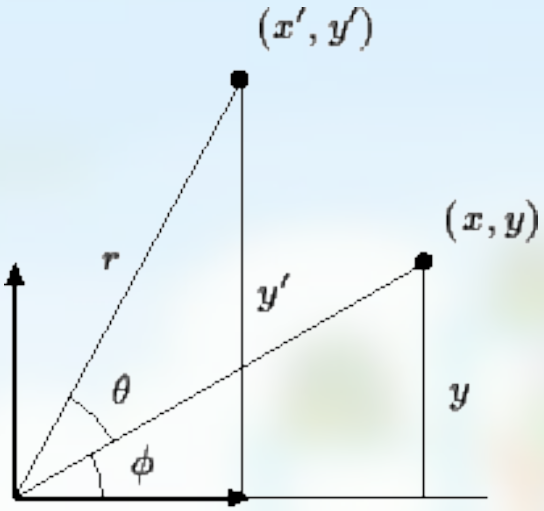


2D rotation

Using trigonometric functions to derive a 2d rotation formula

2D Rotation, naive "wrong" approach

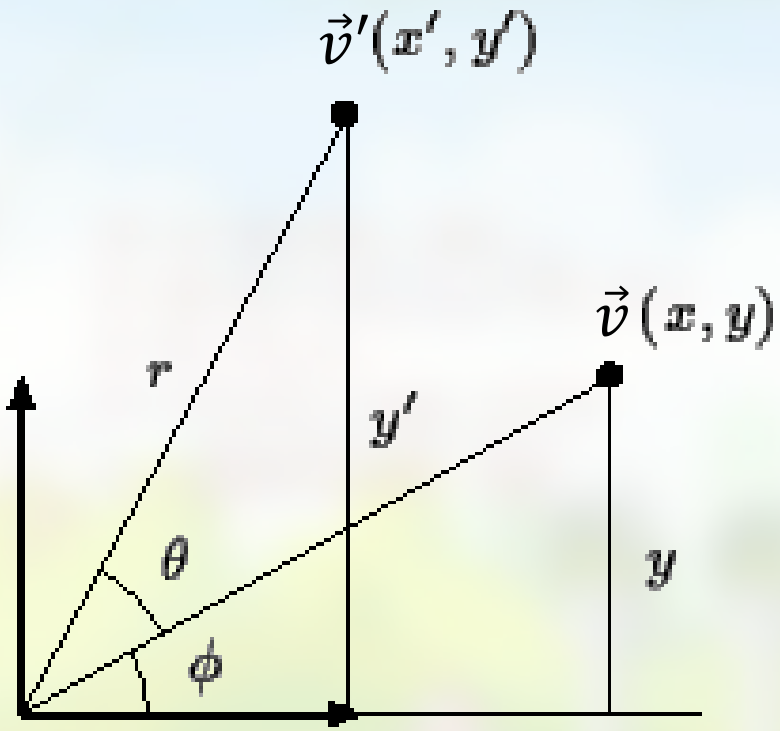
- The slow computation-intensitive option:
  - $currentAngle = Atan2(y, x)$
  - $r = \sqrt{x^2 + y^2}$
  - $newAngle = currentAngle + \theta$
  - $x' = r * \cos(newAngle)$
  - $y' = r * \sin(newAngle)$



3 trigonometric functions, two squares and one square root.
Has to be **repeated** for each point that you want to rotate.
This is **not** the way to do it… although it works (try it for practice)

A better 2D Rotation formula?

- Given $\vec{v}$ and angle of rotation θ, can we derive a better/faster 2d rotation formula for $\vec{v}'$?

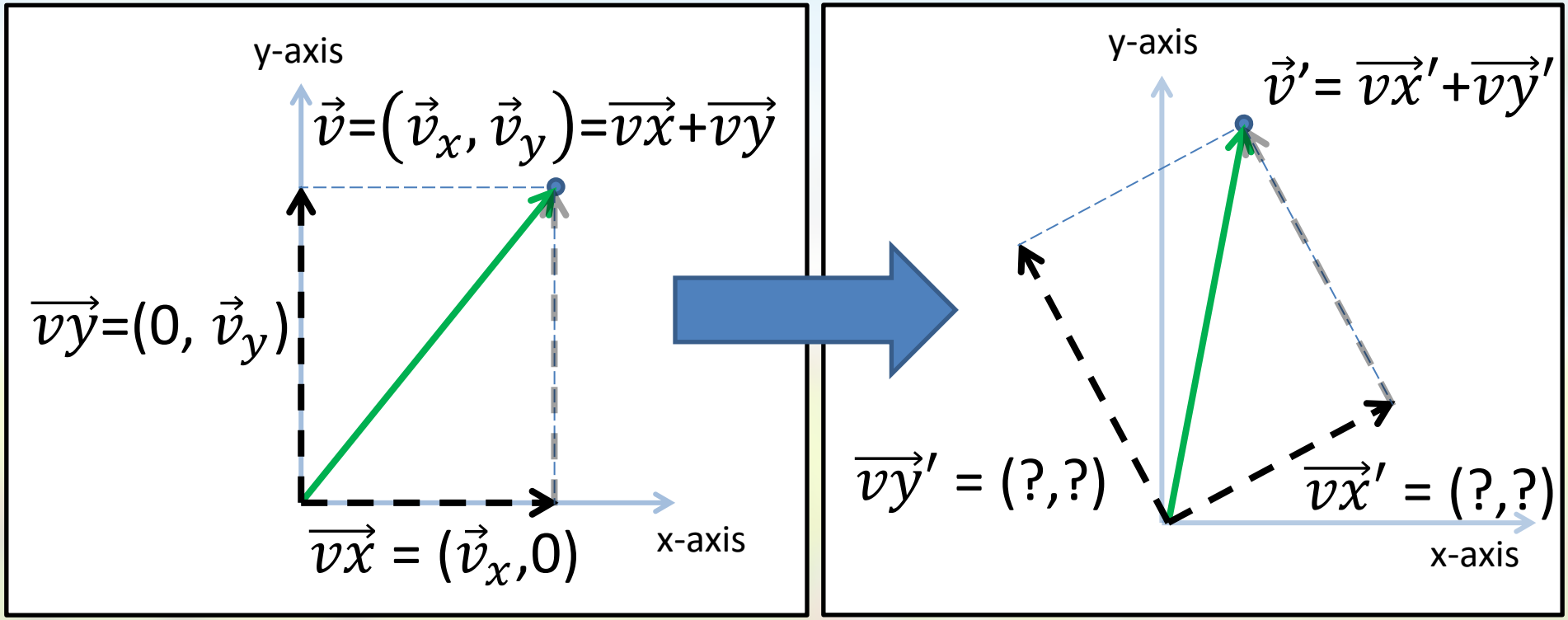


The idea

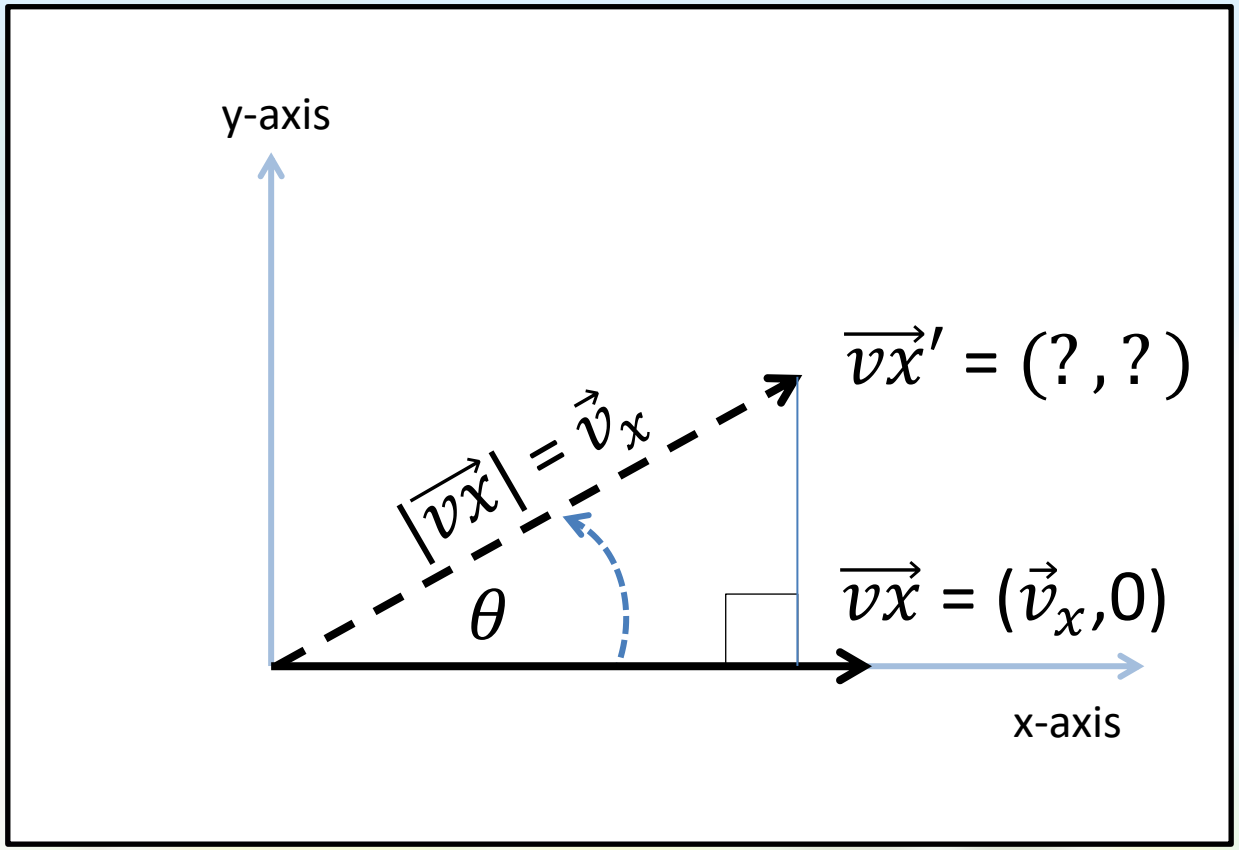
Decompose $\vec{v}$ into $\overrightarrow{vx}$ and $\overrightarrow{vy}$

Rotate $\overrightarrow{vx}$ and $\overrightarrow{vy}$ to $\overrightarrow{vx}'$ and $\overrightarrow{vy}'$

Add $\overrightarrow{vx}'$ and $\overrightarrow{vy}'$ back together to get $\vec{v}'$

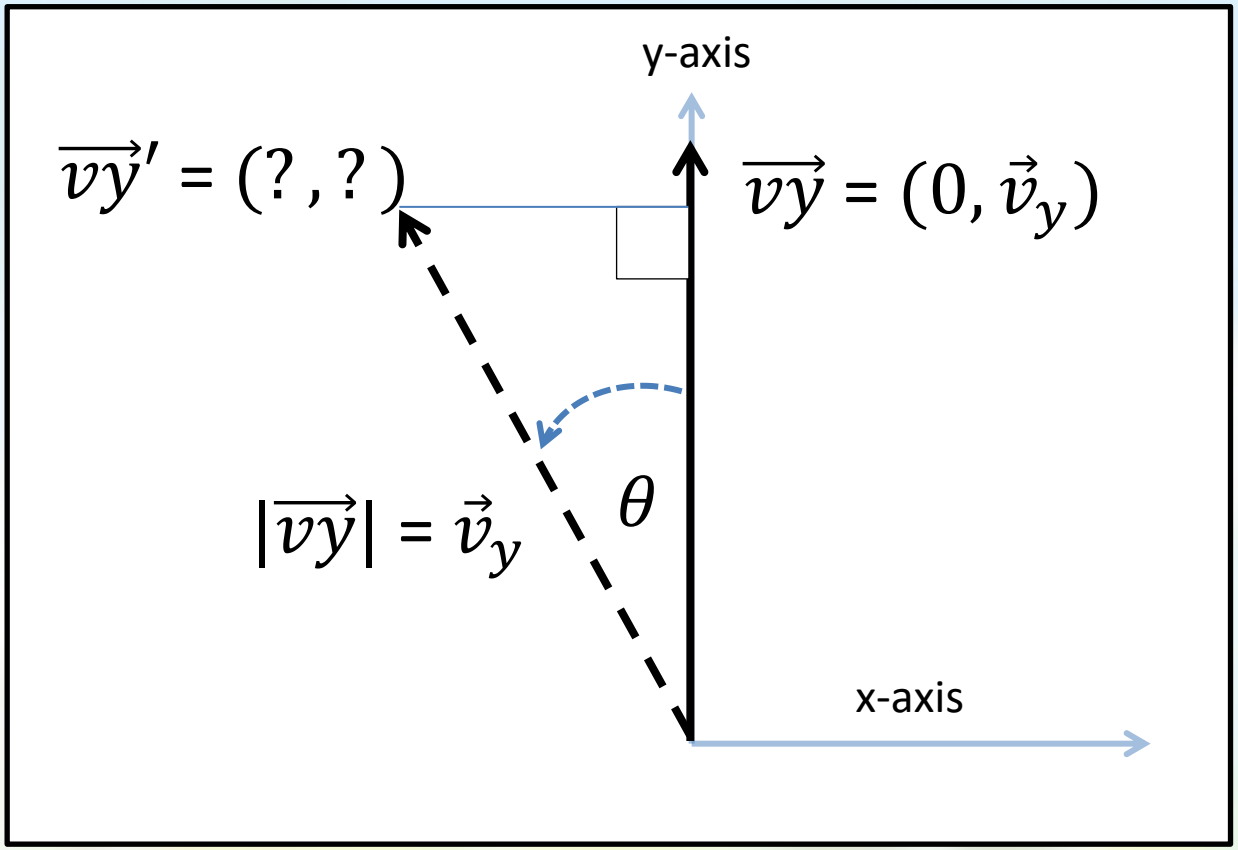


Calculating $\overrightarrow{vx'}$



$$\overrightarrow{vx'} = (\vec{v}_x * \cos\theta, \vec{v}_x * \sin\theta)$$

Calculating $\overrightarrow{vy}'$



$$\overrightarrow{vy}' = (-\vec{v}_y * \sin\theta\,, \vec{v}_y * \cos\theta)$$

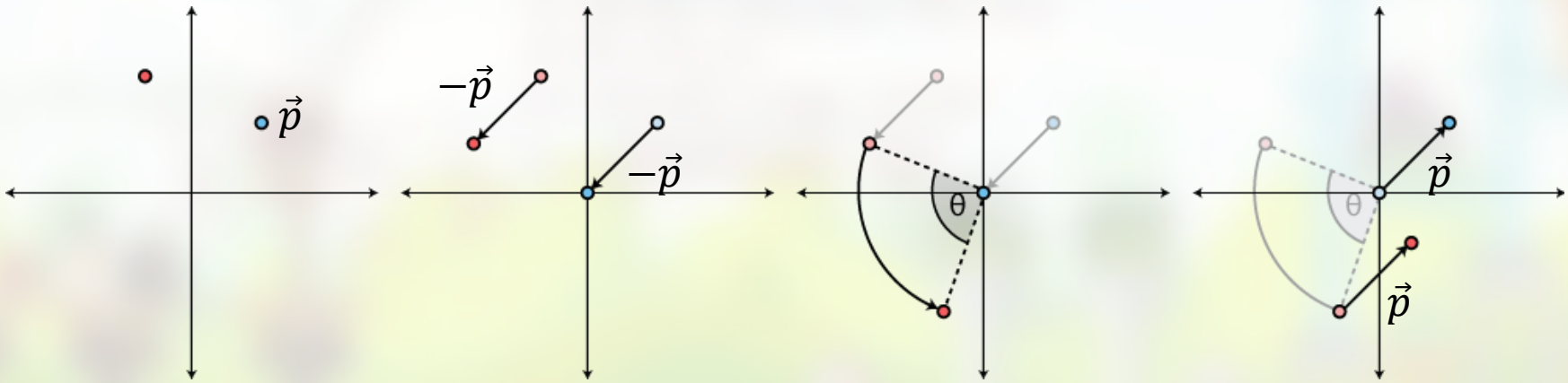
2D rotation formula for $\vec{v}$ and angle θ

- $\vec{v}' = \overrightarrow{vx}' + \overrightarrow{vy}'$ where,
  - $\overrightarrow{vx}' = (\vec{v}_x * \cos\theta\ , \vec{v}_x * \sin\theta)$
  - $\overrightarrow{vy}' = (-\vec{v}_y * \sin\theta\ , \vec{v}_y * \cos\theta)$
- gives:
  - $\vec{v}' = (\vec{v}_x * \cos\theta - \vec{v}_y * \sin\theta\ , \vec{v}_x * \sin\theta + \vec{v}_y * \cos\theta)$

- We have to calculate $\cos\theta$ / $\sin\theta$ only once!

Rotation around an arbitrary point?

- *Rotation around point $\vec{p}$:*
  - Translate everything over $-\vec{p}$
  - $\overrightarrow{p'}$ is now at (0,0)
  - Apply standard 2d rotation formula
  - Translate back over $\vec{p}$



Rotation around an arbitrary point

- Usage:
  - Strafing/Orbiting NPC's?
  - Rotating a vector relatively
  - Possibly: Assignment 2 (though when using SetOrigin correctly, the GXPEngine takes care of this: much easier!)

Euler Integration and Acceleration

Creating Realistic Movement

- Last week: realistic movement: *position* only changes through *velocity*
  - *See code sample 002*
- Even better: *velocity* only changes through *acceleration*
  - *See code sample 003*

$$velocity = \Delta p / \Delta t$$

$$acceleration = \Delta v / \Delta t$$

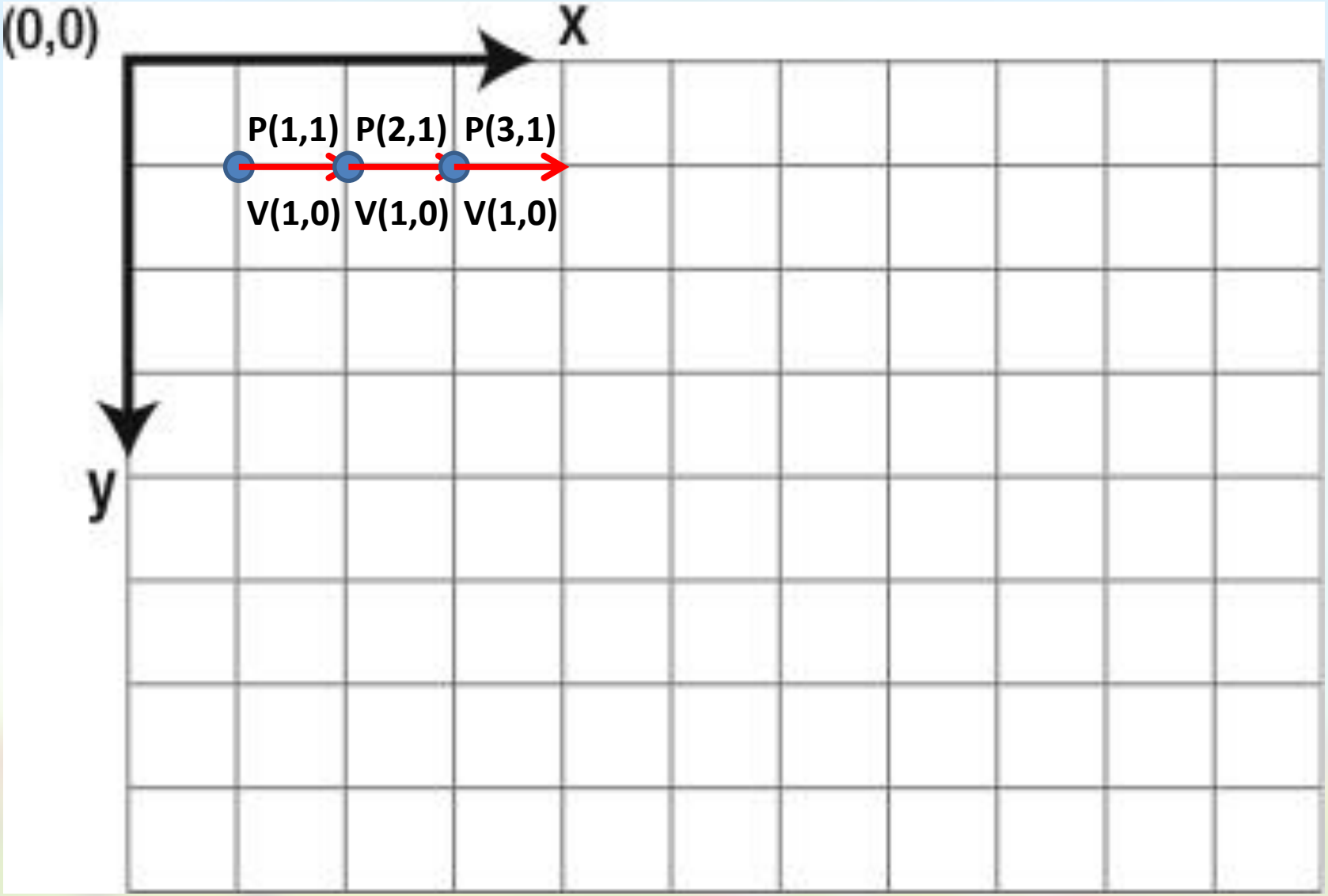
Updating our code loop:

$$velocity = \ \ldots some\ value\ \ldots$$
$$position = position + velocity$$

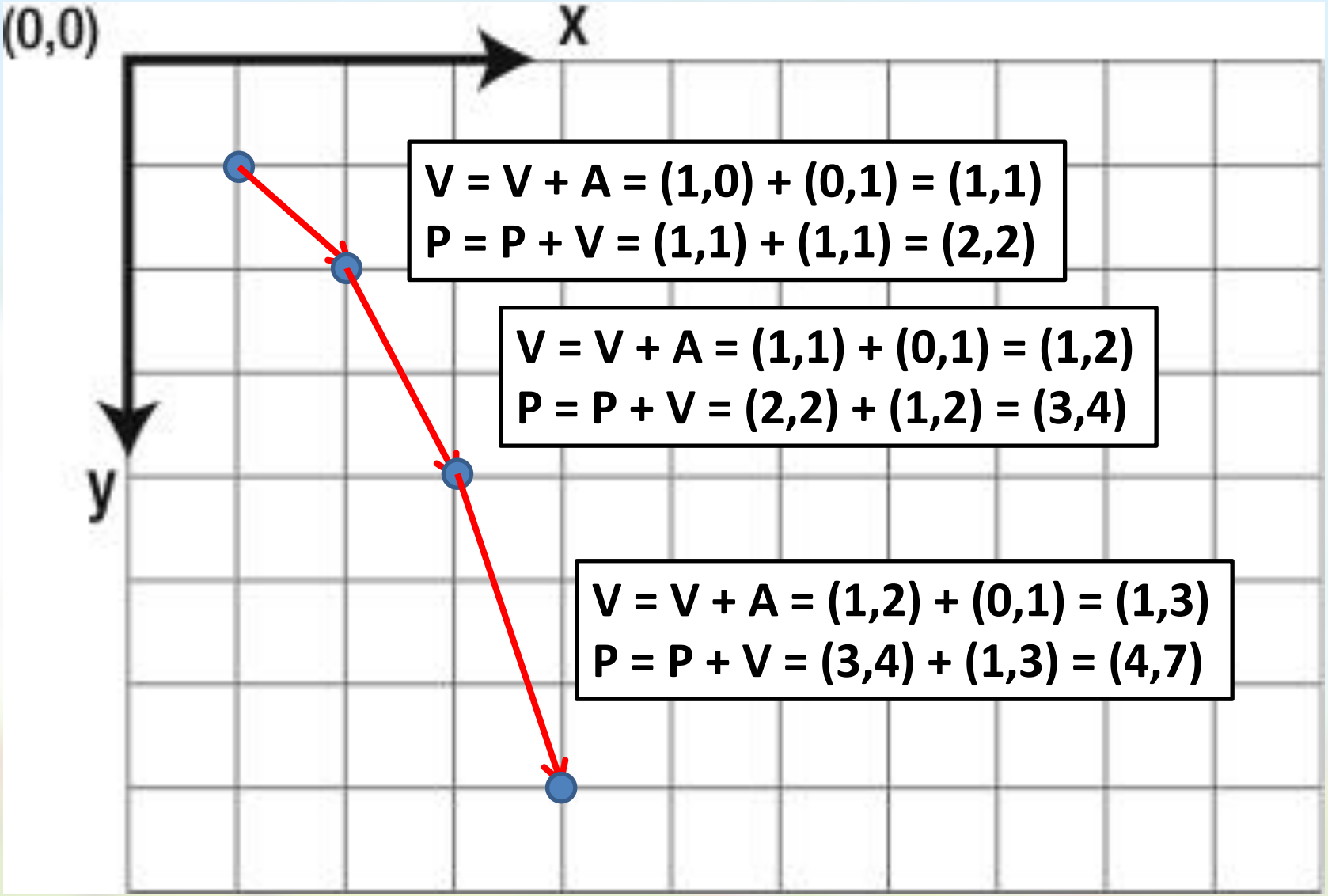
becomes

$$acceleration = \ \ldots some\ value \ldots$$
$$velocity = velocity + acceleration$$
$$position = position + velocity$$

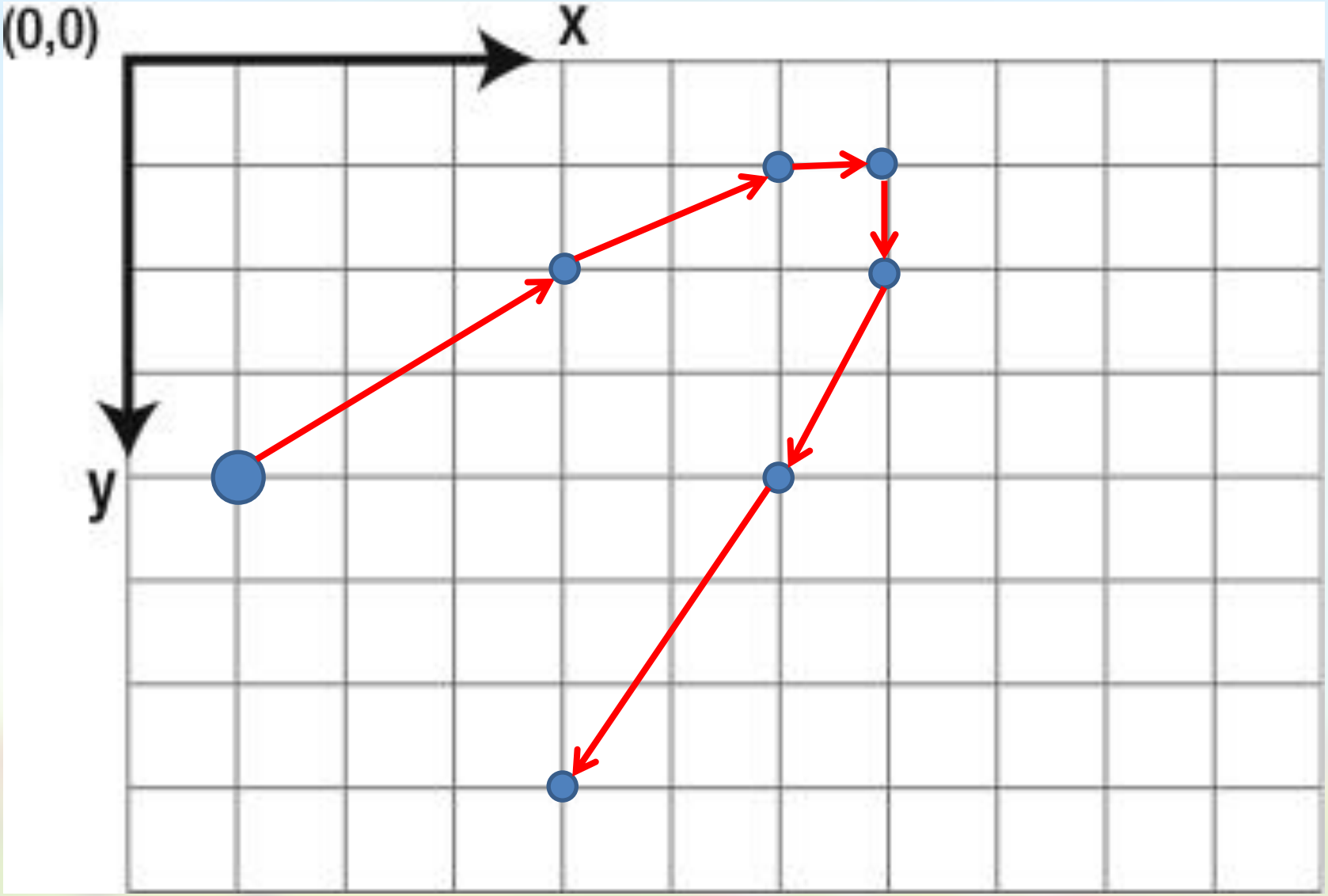
P = (1,1), V = (1,0), A = (0,0)



P = (1,1), V = (1,0) , A = (0,1)


(0,0)
X
y
V = V + A = (1,0) + (0,1) = (1,1)
P = P + V = (1,1) + (1,1) = (2,2)
V = V + A = (1,1) + (0,1) = (1,2)
P = P + V = (2,2) + (1,2) = (3,4)
V = V + A = (1,2) + (0,1) = (1,3)
P = P + V = (3,4) + (1,3) = (4,7)

P = (1,4), V = (4,-3) , A = (-1,1)



Integration

$$acceleration = \ \ldots some\ value \ldots$$
$$velocity = velocity + acceleration$$
$$position = position + velocity$$

- This core step is also called "semi-implicit Euler integration"
- It is just an *approximation* of the true movement! (Piecewise linear instead of parabola)
- There are many alternatives, see e.g. https://gafferongames.com/post/integration_basics/
- …but this is the best choice for now.

Warning

- This type of Euler integration can be used to create good spaceship / hovercraft / … controls → sliding / floating
- Don't use it for car / tank controls! (This week's assignment) → *no sliding / floating allowed!*

Assignment 2

Assignment 2

- Unlocked by knowledge test on Blackboard
- Part I: Extending Vec2 even further
- Part II: Driving a tank and shooting
  - Assets have been provided
  - Specific requirements/limitations

Divide and Conquer

We have no movable tank

We have a movable tank

Divide and Conquer

We have no movable tank

First requirement:

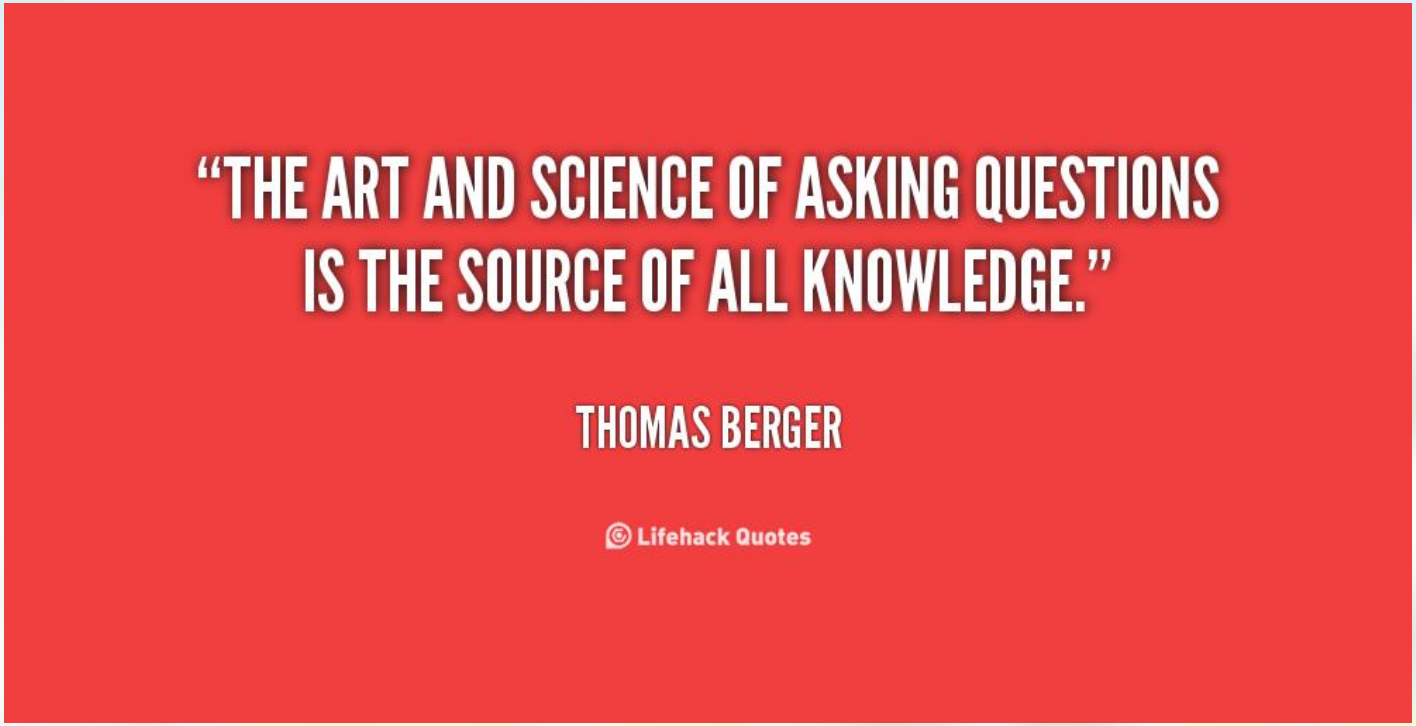
“forward/back accelerates/decelerates the tank in the direction the tank is facing, up to a maximum speed”

Implement a number which increases on Key.UP (print it!)
+ number reaches a max value
+ we can decrease it as well using Key.DOWN
+ number goes back to zero slowly on key release
+move tank in direction of current rotation using this speed

We have a movable tank

Divide and Conquer

- The number of steps to get somewhere is different for everyone
- What is your first step?
- What is your next step?
- Keep stepping


“THE ART AND SCIENCE OF ASKING QUESTIONS
IS THE SOURCE OF ALL KNOWLEDGE.”
THOMAS BERGER
Lifehack Quotes

More info

- If you want more (basic) info, check out e.g.:
  - https://www.khanacademy.org/math/geometry-home/right-triangles-topic/intro-to-the-trig-ratios-geo/v/basic-trigonometry
- Credits for images on radians/degrees and background material:
  - https://betterexplained.com/articles/intuitive-guide-to-angles-degrees-and-radians/

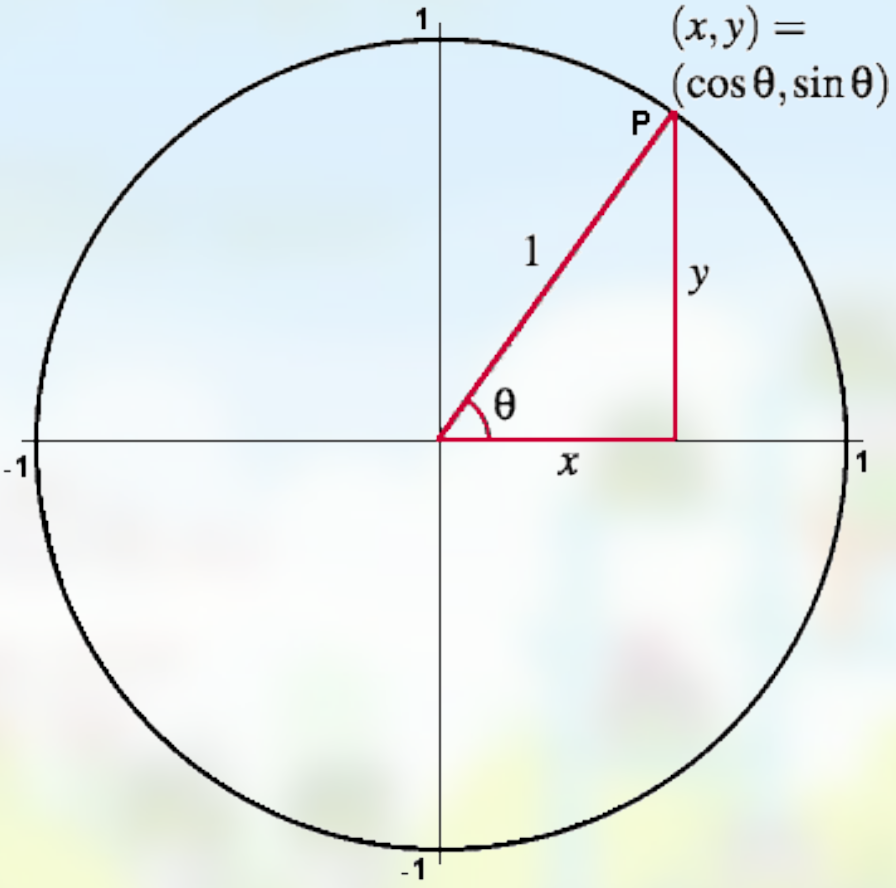
- Better Explained is a great math site with lots of in depth explanations on various math topics. Be sure to check it out and bookmark it!

Detailed info

You should not read on, if you are already overwhelmed…

Cos and Sin equalities on Unit Circle

- $a^2 + b^2 = c^2$
- $x^2 + y^2 = 1^2$
- $\cos(\theta)^2 + \sin(\theta)^2 = 1$
- $\sin(\theta)^2 = 1 - \cos(\theta)^2$
- $\cos(\theta)^2 = 1 - sin(\theta)^2$
- sin (-θ) == - sin (θ)
- cos (-θ) == cos (θ)



Proofs

- Can we prove that if we calculate $\vec{v}$ as:

  (speed * cos(α), speed * sin(α))

  that the length of $\vec{v}$ is actually speed?
- Length of (speed * cos(α), speed * sin(α)) ==
- $\sqrt{(\text{speed} * \cos(\alpha))^2 + (\text{speed} * \sin(\alpha))^2}$ ==
- $\sqrt{speed^2 * (\cos(\alpha)^2 + \sin(\alpha)^2)}$ ==
- $speed * \sqrt{(\cos(\alpha)^2 + \sin(\alpha)^2)}$ ==
- speed * 1 == speed

Algebraic 2d rotation proof

- https://www.youtube.com/watch?v=a0LvqflQMx4
- http://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/2drota.htm

Degrees / radians / Trig cheatsheet