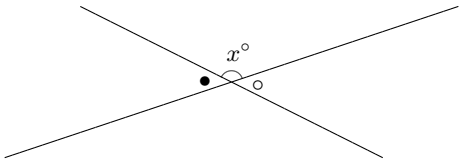
2.4 図形と証明 I

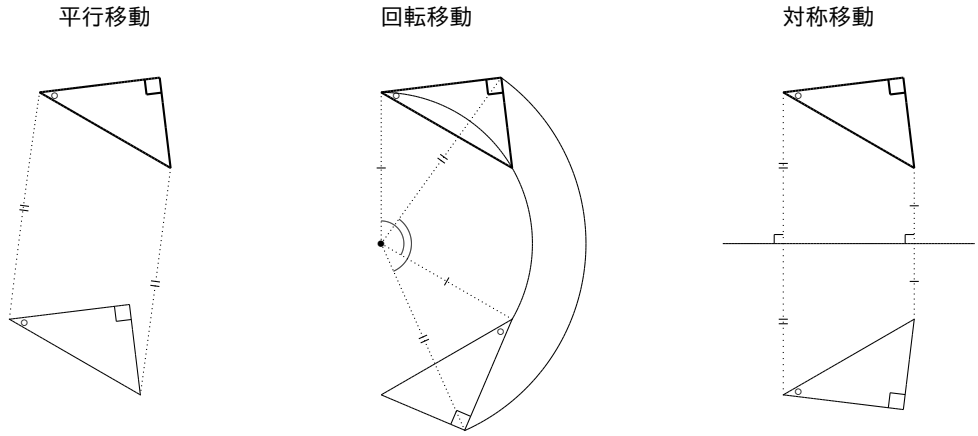
中学校の数学が算数と違うところは、いちいち**証明**の手続きをとることだろう。小学校では、たとえば三角形の内角の和が 180° である理由は説明されても、そのことを筋道立てた証明の形で示すことはない。およその理屈がわかればよいのだ。

ところが中学校ではそうはいかない。内角の和が 180° であることを示すための準備として**対頂角**が等しいことから証明したはずだ。



見ればわかるじゃん、ってのはダメなのだ。ちゃんと、$A(\bullet) = 180^\circ - x^\circ$、$B(\circ) = 180^\circ - x^\circ$ だから $A(\bullet) = B(\circ)$ と言うのだ。それから平行線における**錯角**が等しいことも証明して、ようやく三角形の話になって、**合同**を用いた証明へ進むわけだ。ああ面倒、なんてこれっぽっちも思ってはいけない。

三角形の合同条件は証明の基本みたいなところがあるが、そもそも合同条件自体は証明していないだろう。なぜなら "合同の条件" だからだ。条件とは、どんなときにどうなるかという話だから、ふたつの図形を移動によって重ねることができるときふたつの図形は合同である、という内容が合同条件である。



重ね合わせるとは、数学的には辺の長さや角の大きさが一致することだが、Swift 的には本当に移動させることだ。iPhone の画面で何かを移動させるときは指でなぞるだろう。だが場合によっては Swift 備え付けの部品を用いてもできる。たとえばスライダーを用いて回転移動をしてみよう。

[Triangle.playground]

```
import UIKit
import PlaygroundSupport // *

class ViewController: UIViewController {

    let W: CGFloat = 400
    let H: CGFloat = 400
    var P = CGPoint(x: 250, y: 200)
    var Q = CGPoint(x: 290, y: 200)
    var R = CGPoint(x: 290, y: 170)
    var aRadius: Float = 0; var aTheta: Float = 0.0
    var bRadius: Float = 0; var bTheta: Float = 0.0
    var cRadius: Float = 0; var cTheta: Float = 0.0
    let triang = CAShapeLayer()

    override func viewDidLoad() {
        super.viewDidLoad()

        putTriangle(A: P, B: Q, C: R)

        let slider = UISlider(frame: //__
                            CGRect(x: 0, y: H, width: W, height: 30))
        slider.addTarget(self, action: //__
                        #selector(sliderChanged), //__
                                  for: UIControlEvent.valueChanged)
        slider.minimumValue = -Float.pi
        slider.maximumValue =  Float.pi
        view.addSubview(slider)
    }

    func putTriangle(A: CGPoint, B: CGPoint, C: CGPoint) {
        let gline = UIBezierPath()
        gline.move(to: A)
        gline.addLine(to: B)
        gline.addLine(to: C)
        gline.addLine(to: A)

        triang.strokeColor = UIColor.blue.cgColor
        triang.fillColor = UIColor.white.cgColor
        triang.lineWidth = 2.0
        triang.path = gline.cgPath
        view.layer.addSublayer(triang)
    }

    func getTriElems(A: CGPoint, B: CGPoint, C: CGPoint) {
        aRadius = sqrt(Float((A.x - W/2)*(A.x - W/2) + //__
                                        (A.y - H/2)*(A.y - H/2)))
        bRadius = sqrt(Float((B.x - W/2)*(B.x - W/2) + //__
                                        (B.y - H/2)*(B.y - H/2)))
```

```
        cRadius = sqrt(Float((C.x - W/2)*(C.x - W/2) + //__
                                        (C.y - H/2)*(C.y - H/2)))
        aTheta = Float(acos((A.x - W/2)/CGFloat(aRadius)))
            if A.y > H/2 { aTheta = -aTheta }
        bTheta = Float(acos((B.x - W/2)/CGFloat(bRadius)))
            if B.y > H/2 { bTheta = -bTheta }
        cTheta = Float(acos((C.x - W/2)/CGFloat(cRadius)))
            if C.y > H/2 { cTheta = -cTheta }
    }

    @objc func sliderChanged(sender : UISlider) {
        getTriElems(A: P, B: Q, C: R)

        aTheta += sender.value
        bTheta += sender.value
        cTheta += sender.value
        let ax = W/2 + CGFloat(aRadius * cos(aTheta))
        let ay = H/2 - CGFloat(aRadius * sin(aTheta))
        let bx = W/2 + CGFloat(bRadius * cos(bTheta))
        let by = H/2 - CGFloat(bRadius * sin(bTheta))
        let cx = W/2 + CGFloat(cRadius * cos(cTheta))
        let cy = H/2 - CGFloat(cRadius * sin(cTheta))

        putTriangle(A: CGPoint(x: ax, y: ay), //__
                    B: CGPoint(x: bx, y: by), C: CGPoint(x: cx, y: cy))
    }
}

let viewController = ViewController()                  // *
viewController.view.backgroundColor = UIColor.white    // *
PlaygroundPage.current.liveView = viewController       // *
PlaygroundPage.current.needsIndefiniteExecution = true // *
```

三角形を回転移動させるプログラムなので、三角形の頂点を表す変数を始めとして、たくさんの変数を用いることになった。W, H はビューサイズ用、P, Q, R は三角形の頂点の座標で適当な値で初期化してある。名前に Radius を含む変数 3 個は回転の中心からの距離のための、名前に Theta を含む変数 3 個は回転角のための変数である。また triang は三角形を描画するための台紙にあたる。

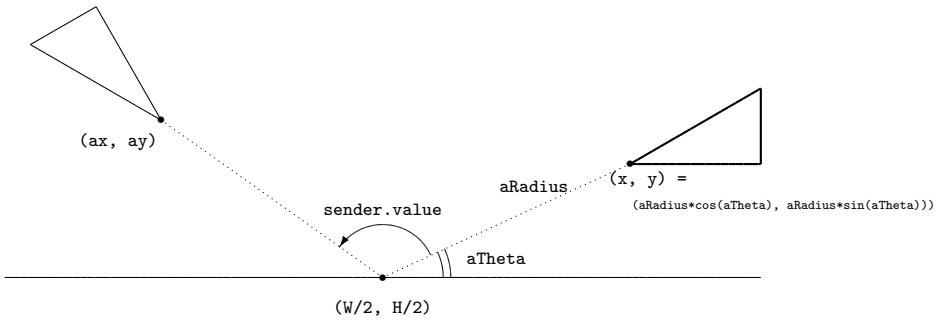
プログラムは回転の計算に不可欠な**三角比**を用いているが、これは高 1 数学で学ぶことである。平行移動と対称移動には必要ないことなので、回転移動はそれらと比較してもだいぶ複雑なことをしていることになる。Swift は回転移動のためのコンポーネントを持っているのかな？ よくわからないが、数学の勉強もするつもりで自分で関数を書くのも悪くないと思うけどね。

プログラムが複雑になっているのは三角比だけが理由ではなく、三角形の作り方にも原因がある。もっとよい方法があるけれど、それはあとから学ぶことにして、いまは限られた知識でできる

範囲のプログラムで我慢してもらいたい。

21 行目で slider に代入した UISlider() はテキストフィールドやボタン同様、Swift が持っている部品だ。だから使い方は似たようなもので、置く位置を決め、何をしたときどうするかを.addTarget() で与えてやればよい。スライダーによる回転は $-180°$ から $+180°$ までに設定した。それは 23 行目と 24 行目の.minimumValue と.maximumValue のプロパティで設定するのだが、Swift は角度を “度数” で測るのではなく “ラジアン” で測る。ラジアンという単位は高校から使うもので、$180° = \pi$ ラジアンと決められいる。π はもちろん円周率なので、私たちが日常使っている “一周 360°” は、数学では “一周 2π ラジアン” という。

さて、このプログラムのキモは回転操作にある。viewDidLoad() では、三角形を置いてスライダーを配置しただけで、三角形を回転させるのは 54–68 行目の sliderChanged() の仕事だ。



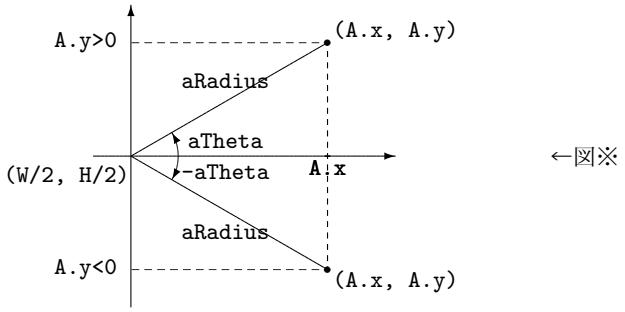
これは画面の中心 (W/2, H/2) を回転の中心として、スライダーの移動量 sender.value 分の回転をする仕様にしてある。そのために回転前の三角形の位置を取得する関数 getTriElems と回転後の位置に三角形を描く関数 putTriangle を作った。と言っても、putTriangle() は最初に三角形を置いたときにも使っている。こういうところが関数の便利なところである。

問題は、回転前の三角形の位置を取得する計算とスライダーの位置から回転角を求める計算だ。軽く説明するが、三角比を習ってないとよくわからないと思う。高校 1 年になったらしっかり勉強しよう。putTriangle() は 3 個の頂点の座標を受け取り、頂点を結んで三角形を描く関数である。変数 gline に与える属性が少し増えているが前節と同じことをしている。

getTriElems() は頂点の位置を逆算する関数である。頂点の位置なら P, Q, R を CGPoint 型で表しているんじゃない？と思うかもしれない。もちろんそうだが、座標系で点の位置を表す方法は $(x_0,\ y_0)$ の組だけとは限らない。平面の点の位置は、原点 O から点までの距離 r_0 と水平軸となす角度 θ_0 がわかれば特定できる。単に点の位置を示すだけなら x-y 座標系で十分だが、回転する点の位置を表すなら r-θ 座標系のほうが便利である。なぜなら、回転するときは回転半径 r_0 は一定

だから、変化する回転角だけ気にすればよい。

そこで getTriElems() は、x-y 座標の値から回転半径と回転角を逆算しているのである。その際、回転の中心は (W/2, H/2) なので調整が必要だが。回転角を求めるには三角比の知識が要る。見てのとおりの式で理解してもらいたい。回転半径 aRadius と水平方向の長さ A.x - W/2 から求める値は $-1 \leqq$ (A.x − W/2)/(aRadius) $\leqq 1$ の範囲であり、このとき arccos（acos()）で逆算される回転角は $0 \leqq$ aTheta $\leqq 180°$—ラジアンを単位とすれば $0 \leqq$ aTheta $\leqq \pi$ である。しかし回転半径と水平方向の長さだけでは、たとえば右回転の 60° も左回転の 60° も同じ値をとるため、どちらの回転かわからない。そのため A.y の位置によって回転の正・負を判定し、負の回転のときは -aTheta としているのである。



←図※

sliderChanged() は、スライダーの値から得た回転角を加算して、その角度を用いて移動先の三角形の頂点の x 座標と y 座標を計算する。あとは 3 個の頂点の座標を putTriangle() に渡せばよい。ざっと、こんなところだ。

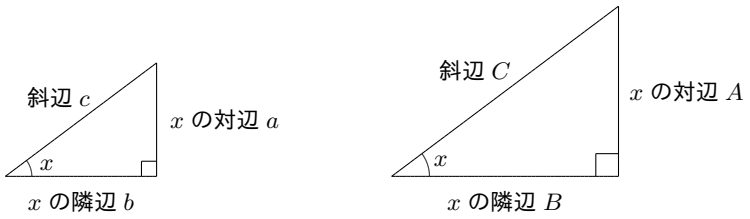
プログラミングは使うツールを選べば小学生にもできることで、シューティングゲームやロールプレイングゲームなんかも作れてしまう。だからプログラミングは、ある意味簡単なことなのだ。しかし、モノによっては高校や大学で学ぶ数学の知識が必要になる。1 日あればスケートが滑れるようになるとしても、スケートでジャンプしたり回転したりできるようになるには、単に滑るだけ

でない高度な技術が必要で、それには土台となる滑りを身につけないといけない。見た目は同じようにリンクを 1 周している 2 人であっても、経験が違えば見えない筋肉の使い方は異なるものだ。

プログラミングにおいても、高度な技術は数学の知識が土台にある。同じ処理をするプログラムを書いても、経験や数学の知識が違えばコードの書き方は異なる。砂遊びであっても他人をうならせるプログラムが書けるように精進しようじゃないか。

ex. **1** 三角比を学んでいない人のために、`[Triangle.playground]` において、座標 `(ax, ay)` の求め方と角度 `aTheta` の求め方を説明しよう。

まず、三角比の定義を簡単に述べよう。次の図を見てなんとか理解してもらえれば幸いだ。



$\frac{b}{c} = \frac{B}{C}$：この比の値を「$x$ の余弦（cos of x）」という

$\frac{a}{c} = \frac{A}{C}$：この比の値を「$x$ の正弦（sin of x）」という

相似な、つまり直角以外のひとつの角が等しい直角三角形においては（3 辺のうちの）2 辺の辺の比は一定している。2 辺 b, c を選んだときは $\cos(x) = b/c$ と表す。2 辺 a, c を選んだときは $\sin(x) = a/c$ と表す。

このことをコードでどう表していたか見てみよう。$\cos(x) = b/c$ の関係を図※に対応させると、`cos(aTheta) = (A.x-W/2)/aRadius` である。したがって、`aRadius*cos(aTheta)` の計算は `aRadius*(A.x-W/2)/aRadius` となって `A.x-W/2` を求めたことになる。

同様に、$\sin(x) = a/c$ の関係を図※に対応させて `A.y-H/2` が求められる。以上が座標の求め方である。

$\cos(x) = b/c$ は x から b/c の値を求めることができる。逆に b/c の値から x を求めることも可能で、その場合は $\arccos(b/c) = x$ と表す。arccos は Swift では `acos` と書く関数である。$\arccos(b/c) = x$ の関係を図※に対応させると、`acos((A.x-W/2)/aRadius) = aTheta` である。したがって、これで `aTheta` を求めたことになる。

高 1 数学で三角比を学べば済む話だが、プログラムで用いた計算の仕組みを述べてみた。よくわからないよね。高校でしっかり勉強しましょう。