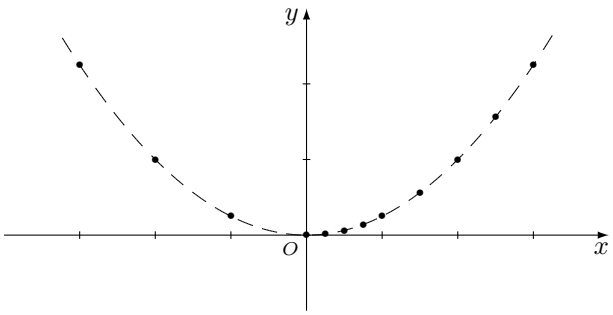
3.4 関数 $y = ax^2$

$y = ax^2$ のグラフが放物線になることは理解できただろうか。a、b の値だけ見て線を引けばよい直線の方程式 $y = ax + b$ とは異なり、a の値だけで正確なグラフを描くのは難しい。どうしてもいくつかの点を打った上で、点を**なめらかに**つなげないといけないからだ。

x	$\cdots$	-3	-2	-1	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$	3	$\cdots$
$y = x^2/4$	$\cdots$	$\frac{9}{4}$	1	$\frac{1}{4}$	0	$\frac{1}{64}$	$\frac{1}{16}$	$\frac{9}{64}$	$\frac{1}{4}$	$\frac{9}{16}$	1	$\frac{25}{16}$	$\frac{9}{4}$	$\cdots$



点をつなげる方法は面倒だが、あらゆる関数のグラフを描くときには有効な手法となる。たとえば $y = ax^2$ のグラフを描くプログラムは以下のようにするとよい。

[Parabola.playground]

```
 1 import UIKit
 2 import PlaygroundSupport // *
 3
 4 class ViewController: UIViewController {
 5
 6     let W: CGFloat = 400
 7     let H: CGFloat = 400
 8     let E: CGFloat = 20
 9     let dx: Float = 0.25
10     let graph = CAShapeLayer()
11
12     override func viewDidLoad() {
13         super.viewDidLoad()
14
15         drawAxis()
16         drawGraph(a: -1)
17     }
18
19     func drawAxis() {
20         let gline = UIBezierPath()
21         gline.move(to:    CGPoint(x: 0, y: H/2))
22         gline.addLine(to: CGPoint(x: W, y: H/2))
```

```
23          gline.move(to:    CGPoint(x: W/2, y: H))
24          gline.addLine(to: CGPoint(x: W/2, y: 0))
25
26          let axis = CAShapeLayer()
27          axis.strokeColor = UIColor.black.cgColor
28          axis.path = gline.cgPath
29          view.layer.addSublayer(axis)
30
31          let OLabel = UILabel(frame: //__
                    CGRect(x: W/2, y: H/2, width: 12, height: 14))
32          OLabel.text = "O"
33          view.addSubview(OLabel)
34      }
35
36      func drawGraph(a: Float) {
37          let line = UIBezierPath()
38          var x: Float = -Float(W) / Float(2 * E)
39          while x < Float(W) / Float(2 * E) {
40              let PL = calcFunction(with: a, and: x)
41              let PR = calcFunction(with: a, and: x + dx)
42
43              line.move(to: PL); line.addLine(to: PR)
44              x += dx
45          }
46          graph.strokeColor = UIColor.blue.cgColor
47          graph.lineWidth = 1.0
48          graph.path = line.cgPath
49          view.layer.addSublayer(graph)
50      }
51
52      func calcFunction(with k: Float, and x: Float) -> CGPoint {
53          // def of function //
54          let y = k * x * x
55
56          return CGPoint(x: W/2 + CGFloat(x) * E, //__
                           y: H/2 - CGFloat(y) * E)
57      }
58  }
59
60  let viewController = ViewController()                    // *
61  viewController.view.backgroundColor = UIColor.white      // *
62  PlaygroundPage.current.liveView = viewController         // *
63  PlaygroundPage.current.needsIndefiniteExecution = true // *
```

このプログラムは座標軸やグラフを描くためのものだが、実は汎用的なひな型でもある。いちばん初めに画面の横・縦である `W・H` を与えてあるが、ここを iPhone などの画面サイズを取得するようにしておけば、ひとつのプログラムで多機種に対応できる。次のようにすれば、さまざまな種類の画面だけでなく、機器の縦置き・横置きにも対応させられるので都合がよい。

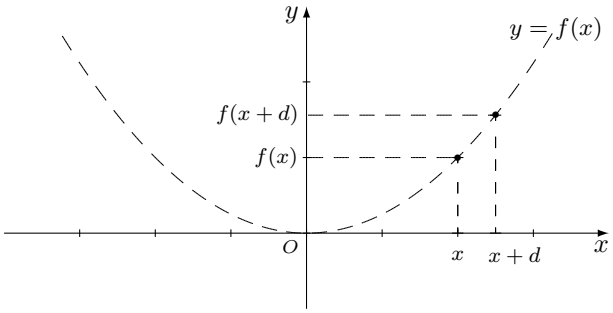
```
let W = UIScreen.main.bounds.size.width
let H = UIScreen.main.bounds.size.height
```

また、点をつなぐための座標計算も関数 calcFunction にしてあるので、ここの式を書き換えれば多様な関数が書けるし、引数を増やせばさらに関数の式を複雑にできる。いろいろと試してみよう。

ex. 1 viewDidLoad() 内の drawGraph(a: -1) と、calcFunction() 内の y = k * x * x が合わさって $y = -x^2$ のグラフを描く仕様になっている（a の値が k へ渡される）。a: -1 の値を変えて、グラフがどのように変わるか確かめてみよう。

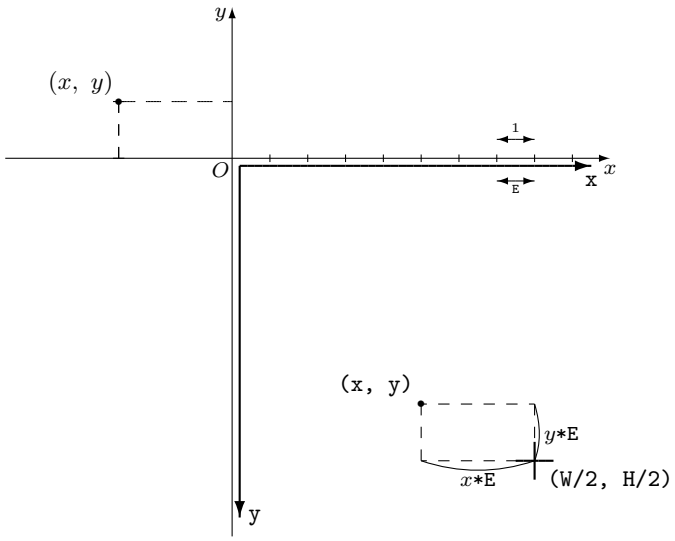
ex. 2 y = k * x * x を、たとえば y = k*x*x + 2 とか、y = x*x*x - k*x とかに変えると、どのようなグラフが描けるか試してみよう。ちなみに、$y = x^3$ などのグラフは高校で学ぶ。

プログラムは、関数のある x の値 $f(x)$ と、その僅（わず）か先の $x + d$ の値 $f(x + d)$ を計算して、$(x,\ f(x))$ と $(x + d,\ f(x + d))$ を直線でつないでグラフを描く。$(x + d,\ f(x + d))$ みたいな表現を見るとビビっちゃうかもしれないが、$f(x) = x^2$ において $x = 5$ のときが $(5,\ f(5))$、すなわち点 $(5,\ 5^2)$ を求めていることと対応させると、自然な表現であるとわかるはずだ。



数学における関数のグラフをコンピュータの画面に表示する場合、面倒なのが座標変換であろう。たとえばこの [Parabola.playground] は、機器の画面サイズを 400×400 と想定し 1 目盛を 20 に設定したので、横・縦共に 20 目盛の大きさになる。画面中央を原点 O とすれば x 軸、y 軸共に $-10 \leqq x,\ y \leqq 10$ の領域になる。func drawGraph() 内で開始点となる var x と終了点まで回す while x の値に Float(W) / Float(2 * E) が使われているのは、このためである。

また、描画したいグラフが $y = -2x^2$ なら点の計算はこの式で行いたいが、結果の座標 $(x,\ y)$ はそのままビューの値には使えない。ここでは値の変換を、関数の定義と共に calcFunction() 内に記述した。座標を固定すれば難しい変換ではないので、応用は楽にできるだろう。



はじめに func viewDidLoad() を見ることにしよう。ここがプログラムの核である。それなのに肝心のコードはわずか 2 行で寂しいね。もちろんこの 2 行は中身が詰まった関数である。実際ふたつの関数を見ると、そこには何行ものコードが書かれている。おまけにみっつ目の関数まであるので、全体のコード量はこれまでの最大になってしまった。

drawAxis() は単に座標軸を描画する関数だが、目盛がなく見栄えもよくない。それなりの出来にするには手間を惜しんではいけない。ただ、今後いろいろなプログラムに座標軸を使うなら、以下のようなコードなどを加えて専用の関数に仕立てておけば、常に drawAxis()1 行で片がつくので便利だと思われる。

```
let dline = UIBezierPath()
var x = W/2+E
while x < W {
    dline.move(to:    CGPoint(x: x, y: H))
    dline.addLine(to: CGPoint(x: x, y: 0))
    x += E
}
x = W/2-E
while x > 0 {
    dline.move(to:    CGPoint(x: x, y: H))
    dline.addLine(to: CGPoint(x: x, y: 0))
    x -= E
}
var y = H/2-E
while y > 0 {
    dline.move(to:    CGPoint(x: 0, y: y))
    dline.addLine(to: CGPoint(x: W, y: y))
    y -= E
```

```
    }
    y = H/2+E
    while y < H {
        dline.move(to:    CGPoint(x: 0, y: y))
        dline.addLine(to: CGPoint(x: W, y: y))
        y += E
    }
    let grid = CAShapeLayer()
    grid.strokeColor = UIColor.black.cgColor
    grid.lineDashPattern = [1, 1]
    grid.path = dline.cgPath
    view.layer.addSublayer(grid)
```

このコードでは x 軸と y 軸に破線を描き重ねることを嫌って、たとえば x 軸なら`-W < x < W/2` と `W/2 < x < W` に分けて線を描いているが、あまり細かいことを気にしなければ、`var x = -W+E`、`while x < W` で済ませればコード量は半分になる。ちなみに破線はプロパティ `.lineDashPattern` で指定し、[実線長，間隔長] を指定する。

`drawGraph(a: Float)` は $y = ax^2$ のグラフを描く関数である。引数として a の値を与える。グラフは点 $(x,\ f(x))$ と $(x+d,\ f(x+d))$ を繰り返し結んで描くのだが、関数 `drawGraph` はビュー座標上に線を引くことを想定しているので、x 側の点の座標 `PL`、$x+d$ 側の点の座標 `PR` は共にピクセル単位である。したがって、$f(x)$ の計算は数学座標で行なっても、`PL` へ代入する際にはビュー座標に変換されている必要がある。そのため、$f(x)$ の値を計算する関数 `calcFunction` には、関数の計算式だけでなく座標変換の式も含めたのだが、座標変換は専用の関数にする手もある。関数の式や座標変換の式が複雑になるようなら分けるほうがよいだろう。また、初期設定として `dx = 0.25` ピクセルであるが、計算量や描画速度によって適度な値にしてもらいたい。`dx` や `W, H, E` が定数であるにも関わらず変数のように代入していたのは、のちの変更が容易になるからである。座標変換の式を

```
return CGPoint(x: 200 + CGFloat(x) * 20, y: 200 - CGFloat(y) * 20)
```

と書いてしまったときを想像すれば、重要な措置であることが理解できるだろう。

ところで、今回は関数をあえて `calcFunction(with k: Float, and x: Float)` と書いてみた。これは `calcFunction(k: Float, x: Float)` でもまったく問題がない。`PL` などへの代入が `calcFunction(k: a, x: x)` になるだけだからだ。しかし... と Swift の開発者は考えたのだろう。... これは読みにくい、と。日本人なら `k:` や `x:` の記述を変えれば済む話だ、と考えそうなものだ。でも実際、彼らはそう考えなかったようだ。読みやすいようにすればいんじゃね？

引数に補助的に単語を書き加えて、書き加えた単語が関数使用時に見えれば読みやすいはずだ。

`calcFunction(with: a, and: x)` ならば「この関数は `a` と `x` を用いて関数の計算をする」ってことが一目でわかる。サイコー。

`calcFunction(with: a, and: x)` どころか `calcF(a, x)` でも十分だと思っている私みたいな輩（やから）にしてみれば、そんなのは...。ま、仕方ない。それが Swift 仕様なんだから。

> **ex. 3** グラフは x の範囲を元に描いているので、場合によってはグラフが x-y 座標からはみ出ることがある。どのようなグラフを描いても x-y 座標の枠内に収まるようにコードを追加してみよう。