Python練習問題

問題1 (関数定義)

4つの数 a, b, c, d を受け取り, ベクトル (a, b) と (c, d) の内積を計算する関数, `inner(a, b, c, d)` を書き, 適当な例を使って確かめよ.

```
def inner(a, b, c, d):
    ...

print inner(...)
```

問題2 (関数定義, import)

微分係数の定義に従えば，h を 0 に近い数 (例えば 0.001) とすれば

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}$$

が成り立つ．すなわち微分係数の近似値を，右辺を利用して求められる．

これを利用して，関数 e^x の，$x = a$ における微分係数の近似値を求める関数 `g(a)` を書け．同じ方法で，$\log x$ の，$x = a$ における微分係数の近似値を求める関数 `h(a)` を書け.

なお，e^x や $\log x$ の値を計算する関数は，`math` モジュールに，`exp`, `log` という名前で提供されている．

問題3 (関数の関数)

上記の微分係数の計算方法は f の形に依存しない一般的な方法で, $f(x) = e^x$ だろうが $f(x) = \sin x$ だろうが, 同じように計算できる．しかしプログラム上は，関数が変わるたびに似たようなプログラムを書かなくてはならない．それよりも「(任意の) 関数 f(と a) を受け取り，その $f'(a)$ を求める」関数を書くことができれば便利である．

Python においては def で定義された関数を，他の関数に渡したりできるのでこのようなことが素直に可能である．例えば以下は,「(任意の) 関数 f と値 a」を受け取り，$f(f(a))$ を計算する関数である．

```
def twice(f, a):
    return f(f(a))
```

たとえば以下で, e^{e^x} を計算する関数が出来上がる．

```
def double_exp(x):
    return twice(math.exp, x)
```

以上を元に,「f と a を受け取り，$f'(a)$ を求める」関数 deriv を書け.

```
def deriv(f, a):
    ...
```

これに，math.exp, math.log などを渡して，微分係数を計算してみよ.

```
deriv(math.exp, 1)
deriv(math.log, 1)
```

自分で定義した関数も渡すことができる.

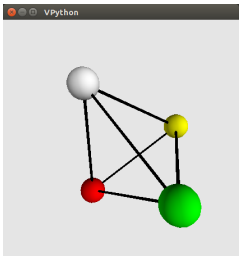
```
def p(x):
    return x ** 3 + x ** 2

deriv(p, 2)
```

問題 4 (変数を使って順に計算する)

数学関数を適宜利用して正四面体の 4 頂点 (辺の長さや配置は, 計算のしやすいよう任意に決めてよい) の座標を求め, 正四面体の各頂点に一個ずつ球面がついている, 以下のようなオブジェを visual python で作れ (頂点は sphere, 辺は curve を使って書く).

図は, 表示された後見やすいようにマウスで視点やズームを変えたもので, 最初から位置をこのとおりにせよということではない (計算のしやすい位置に置けば良い).



問題 5 (繰り返し for, range)

次で定まる数列の第 n 項を計算する関数 a(c, n) を書け.

$$a_0 = c, \tag{1}$$

$$a_n = \frac{1}{3}\left(2a_{n-1} + \frac{c}{a_{n-1}^2}\right) \quad (n > 0). \tag{2}$$

$$\tag{3}$$

これは $\sqrt[3]{c}$ に収束する数列で, ある程度大きな n に対して, a(c, n) を計算すればそれが, $\sqrt[3]{c}$ の近似値となる.

a(c, n) を 3 乗して, 結果がほぼ c と同じになるか確かめよ.

参考: 一般に, $f(x) = 0$ の解を求めるのに以下のニュートン法が用いられる.

$$a_0 = c, \tag{4}$$
$$a_n = a_{n-1} - f(a_{n-1})/f'(a_{n-1}) \ \ (n > 0). \tag{5}$$
$$\tag{6}$$

これが**収束するならば**, $x = x - f(x)/f'(x)$ を満たす x, すなわち $f(x) = 0$ を満たす x に収束することは明らか. 本問はこれを, $f(x) = x^3 - c$ に適用したものである.

ただし, 収束するとは限らないので無条件に使えるわけではない.

計算機で何かを求めるとき, このようなに「欲しい答えに収束するような数列」を使って計算することは非常に多い.

問題 6 (数値積分)

積分の定義式:

$$\int_a^b f(x)\,dx = \lim_{\forall i|x_{i+1}-x_i|\to 0} \sum_0^{n-1} f(x_i)(x_{i+1} - x_i) \tag{7}$$

に従えば，十分大きな n に対して，右辺を計算すればそれが積分の近似値となる:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f(x_i)(x_{i+1} - x_i). \tag{8}$$

ここで, $a = x_0 < x_1 < \cdots < x_n = b$ である.

関数 f と，積分区間 a, b, 点の数 n を受け取り，積分:

$$\int_a^b f(x)\,dx$$

の近似値を, $[a, b]$ を n 等分して式 (8) の右辺を計算することで求める関数 `integrate(f, a, b, n)` を書け.

書けたらそれを使って好きな積分, 例えば

$$\int_0^1 \sqrt{1 - x^2}\,dx$$

の近似値を求めよ.

```
def circ(x):
    return math.sqrt(1 - x * x)

integrate(circ, 0.0, 1.0, 1000000)
```

参考: この方法は, 計算機で積分を計算する常套手段で, 手計算の時のように, 原始関数を求めずに実行することができる. 必要なのは任意の点に対して, $f(x)$ が計算できることだけである.

問題 7 (微分方程式)

y が x の (未知の) 関数で, 以下の式 (微分方程式)

$$y(a) = c, \tag{9}$$
$$\frac{dy}{dx} = \frac{1}{y} \tag{10}$$
$$\tag{11}$$

を満たすとする. この解が $y = \sqrt{2(x-a)+c^2}$ であることは手計算でもすぐにわかる (変数分離法).

以下はそれを計算機にやらせる方法. 一般に,

$$\frac{dy}{dx} = f(x,y) \tag{12}$$

という方程式は, その定義に戻れば,

$$\lim_{h\to 0} \frac{y(x+h)-y(x)}{h} = y \tag{13}$$

ということで, つまり, $h \ll 1$ のとき,

$$y(x+h) \approx y(x) + f(x,y)h \tag{14}$$

ということである.

これを, $y(a) = c$ から出発して, $y(a+h), y(a+2h), y(a+3h), \ldots$ を次々に与える漸化式だと解釈すれば, x における $y(x)$ (の近似値) を計算することができる.

与えられた関数 f, 実数 a, b, c に対し,

$$y(a) = c, \tag{15}$$
$$\frac{dy}{dx} = f(x,y) \tag{16}$$
$$\tag{17}$$

を満たす y の $x = b$ における値 $y(b)$ の近似値を計算する関数 `ord_solve(f, a, b, c)` を書け.

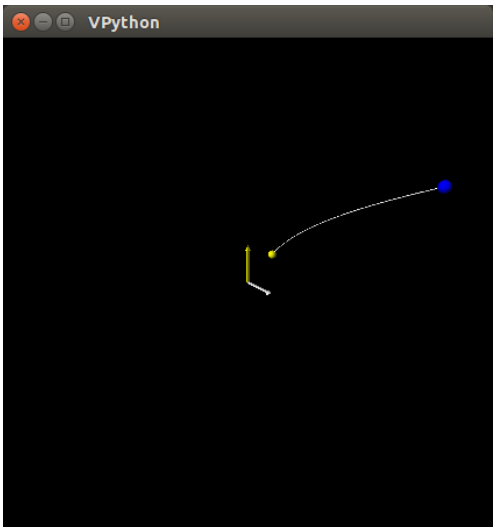
それを用いて, 最初にあげた微分方程式を解き, $y(2.5)$, $y(5.0)$ などを計算してみよ. それを厳密解と比較してみよ.

```
def f(x, y):
    return 1.0 / y

ord_solve(f, 1.0, 1.0, 2.5)
ord_solve(f, 1.0, 1.0, 5.0)
```

可視化: 式 (14) を繰り返し適用して得られるひとつひとつの x と y に対して, 点 $(x, y, 0)$ を中心とした小さな球を描いていけば, (x, y) の軌跡が得られる.

初期値と, (手計算を利用して求める) 最終値に目立つオブジェを置いて, このようなオブジェを作ってみよう.



普通に球を書くだけだと最終結果だけが表示されるが，繰り返しの中で，

```
for ... :
   visual.rate(50)
   ...
```

という一文を入れてやることで，アニメーションになる (visual.rate(50) は，そこで画面を更新すると共に，それが 1 秒間に 50 回呼ばれるよう時間調整 (1/50 秒休む) をする).

問題 8 (乱数)

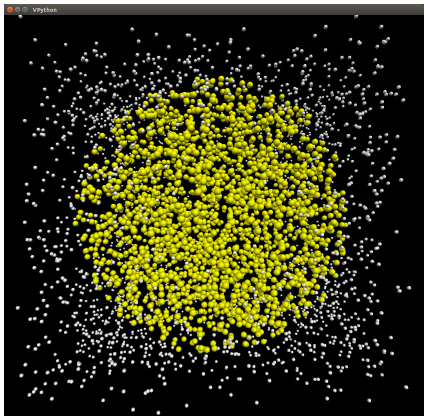
random というモジュールを import すると, random() という関数が使える. これは, 区間 [0,1] から数をひとつランダムに生成する.

```
$ python
Python 2.7.3 (default, Sep 26 2013, 20:03:06)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from random import *
>>> random()
0.8408639918721788
>>> random()
0.7644349032373015
>>> random()
0.6271941721108247
>>>
```

乱数を用いて, 箱 $[-1,1] \times [-1,1] \times [-1,1]$ に多数の点を発生させ，visual python で可視化してみよう. 例えば，

- 原点からの距離 ≤ 1.0 ならば黄色で大きめに表示
- そうでなければ白で小さめに表示

とかやって，こんなアート (??)[1] を作ってみよう (今までの知識でできることをやってみているだけで，アートはなんでもよい).



問題 9　乱数は確率の問題を解く基本かつ最強の武器

モンティ・ホール問題という, TV 番組で物議をかもした問題が有る.

ゲームのルール (Wikipedia より):

1. 3 つのドア (A, B, C) に（景品、ヤギ、ヤギ）がランダムに入っている。
2. プレイヤーはドアを 1 つ選ぶ。(プレイヤーはドアの中身を知らない)
3. モンティは残りのドアのうち 1 つを必ず開ける。
4. モンティの開けるドアは、必ずヤギの入っているドアである。(モンティはドアの中身を知っている)
5. モンティはプレーヤーにドアを選びなおしてよいと必ず言う。

このときプレイヤーはドアを選び直したほうが良いのか?

という問題.

乱数を使ったシミュレーションを行い「選びなおす」「選び直さない」それぞれのケースで，景品をひく確率を求めよ.

こうして正解がわかるという話と，納得しない人をどう納得させるかというのは別の話ですが w

[1] 本物のアートをやっている人，ごめんなさい