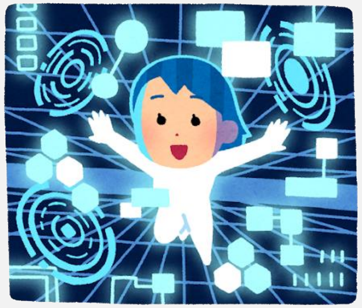
高校生向け夏休み研究体験

コンピュータシミュレーションをしてみよう



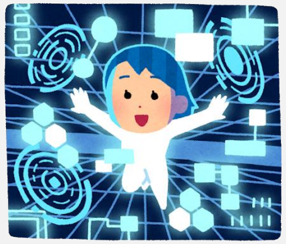
2019/08/08

慶應義塾大学理工学部物理情報工学科
渡辺研究室

今日やること

プログラムを組んで

コンピュータ上で



シミュレーションをします

シミュレーションとは

あるルールに従う系の振舞を

再現・予想すること

それをコンピュータ上でやるのがコンピュータシミュレーション

いろいろなシミュレーション

訓練のためのシミュレーション

フライトシミュレータ
ドライブシミュレータ

娯楽のためのシミュレーション

シミュレーションゲーム (戦略、経営等)
ロールプレイングゲーム

予測のためのシミュレーション

天気予報・自動車の設計・創薬

物理と支配方程式

物理とは

我々が住むこの世界を**理解・記述**する学問

この世界の**ルール**

この世界は**微分方程式**で記述されている
これを**支配方程式**(Governing Equation)と呼びます

支配方程式の例

拡散方程式

$$\frac{\partial T}{\partial t} = -\kappa \frac{\partial^2 T}{\partial x^2}$$

物質やエネルギーなどが広がって(拡散して)いく様子を表現する方程式

∂ は偏微分記号。とりあえず微分の一種と思ってください

支配方程式の例

拡散方程式 $$\boxed{\frac{\partial T}{\partial t}} = -\kappa \frac{\partial^2 T}{\partial x^2}$$

温度の時間変化を表す

ある点での温度が上がるか下がるか？

あがるならどれくらい上がるか？

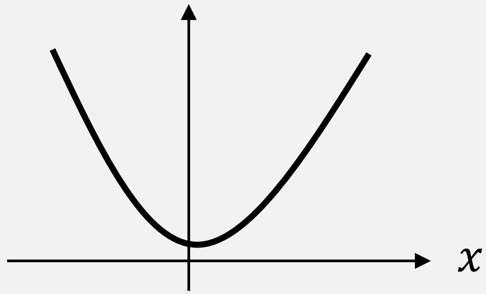
支配方程式の例

拡散方程式

$$\frac{\partial T}{\partial t} = \boxed{\kappa \frac{\partial^2 T}{\partial x^2}}$$

ある地点での**二階微分**の値

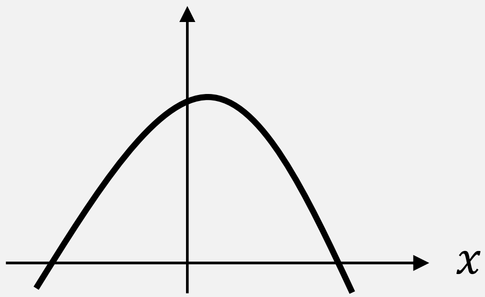
$y = f(x)$



$$\frac{d^2 f}{dx^2} > 0$$

下に凸なら正

$y = f(x)$



$$\frac{d^2 f}{dx^2} < 0$$

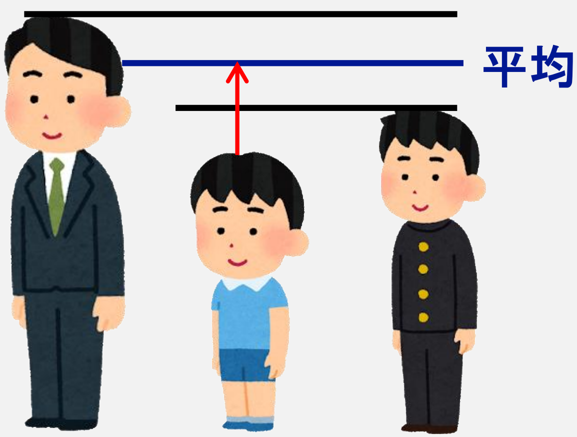
上に凸なら負

支配方程式の例

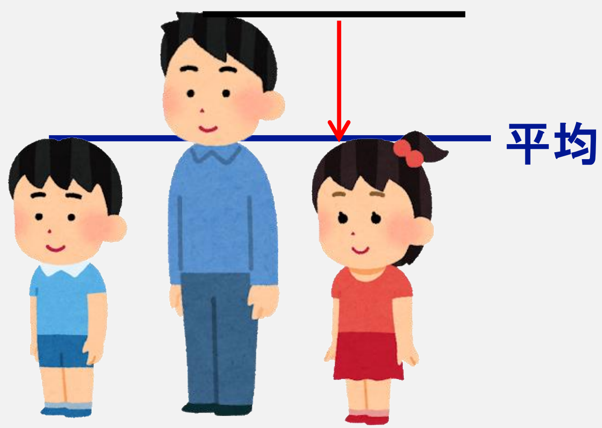
拡散方程式

$$\frac{\partial T}{\partial t} = \boxed{\kappa \frac{\partial^2 T}{\partial x^2}}$$

まわりの平均との差を表す



自分は平均より低い



自分は平均より高い

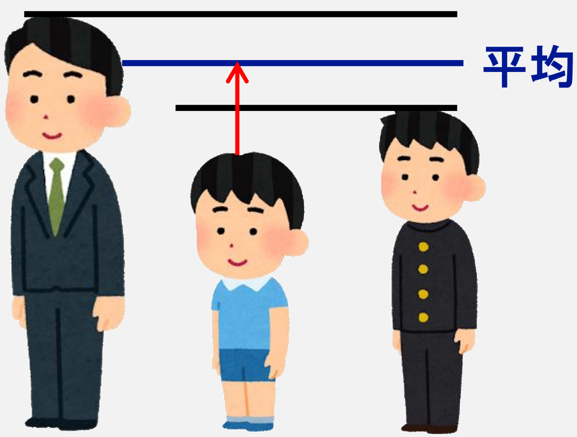
支配方程式の例

拡散方程式

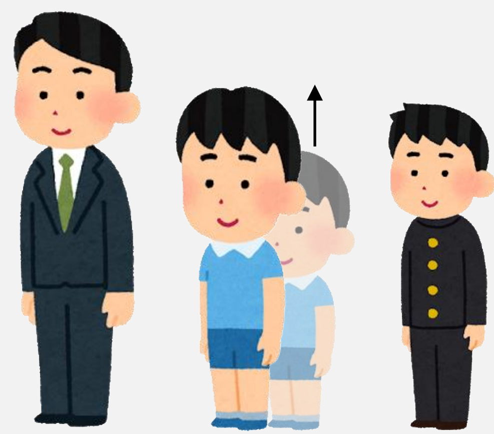
$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$$

まわりを見て

自分が平均以下なら増えようとする
自分が平均以上なら減ろうとする



自分は平均より低い



自分

様々な支配方程式

ニュートンの運動方程式

$$m\frac{dv}{dt} = F$$

古典的な物体の運動を記述する

例：天体の運動、弾道計算

ナビエ・ストークス方程式

$$\frac{Du}{Dt} = -\frac{1}{\rho}\nabla p + F$$

流体の流れを記述する

例：天気予報

マクスウェル方程式

$$\nabla \cdot D = \rho, \nabla \cdot B = 0$$

$$\nabla \times E = -\frac{\partial B}{\partial t}, \nabla \times H - \frac{\partial D}{\partial t} = J$$

電場や磁場の振る舞いを記述する

例：スマートフォンの開発等

コンピュータシミュレーション

支配方程式 ＝ 知りたい現象を記述する微分方程式

これを解けば未来がわかる

⇩

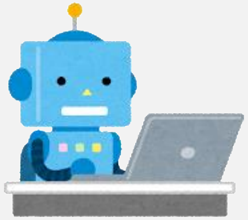
ほとんどの微分方程式は厳密に解くことができない

⇩

数値的に**近似解**を求める

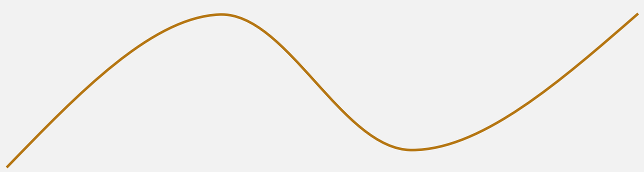
⇩

コンピュータシミュレーション



離散化

この世界は**連続的**



コンピュータは
離散的値しか扱えない

計算機が扱えるように連続的な値をとびとびの値にすることを**離散化**と呼ぶ

離散化

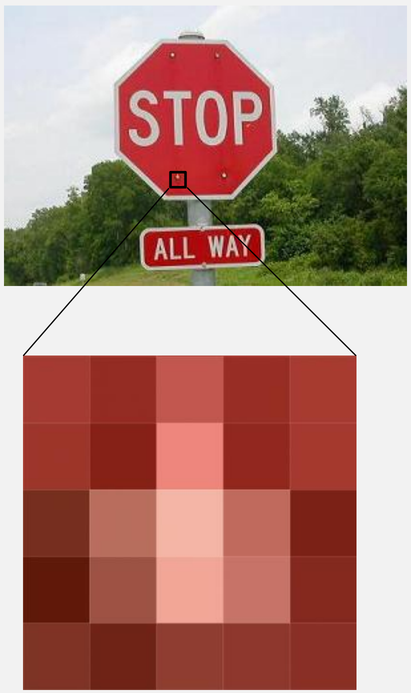
時間　　　　　　　　　　空間

$$\boxed{\frac{\partial T}{\partial t}} = -\kappa \boxed{\frac{d^2 T}{dx^2}}$$

離散化には空間の離散化と時間の離散化がある

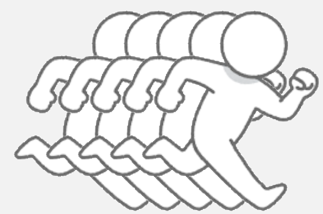
離散化

空間の離散化



拡大するとピクセルに

時間の離散化

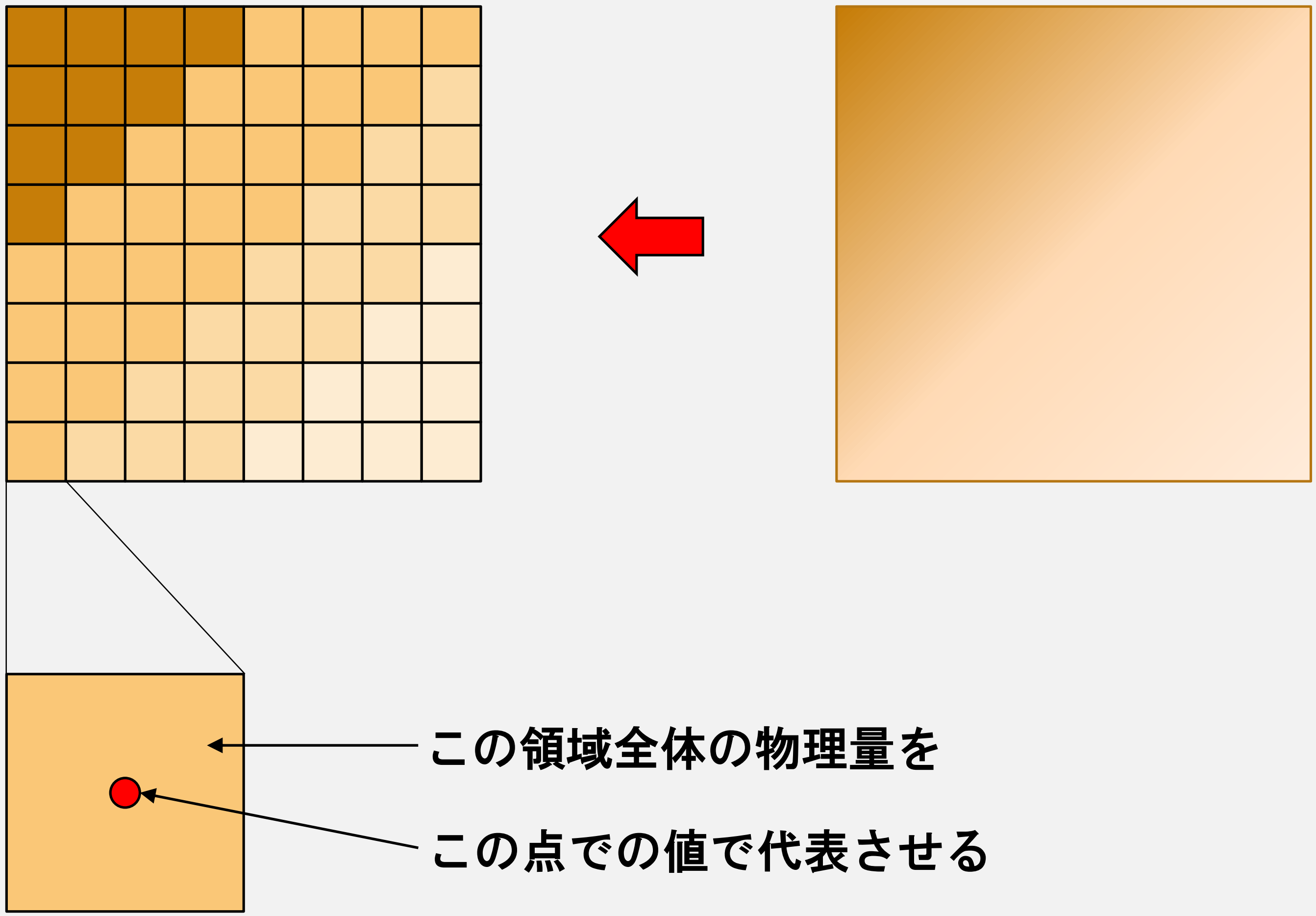


静止画像を高速コマ送り

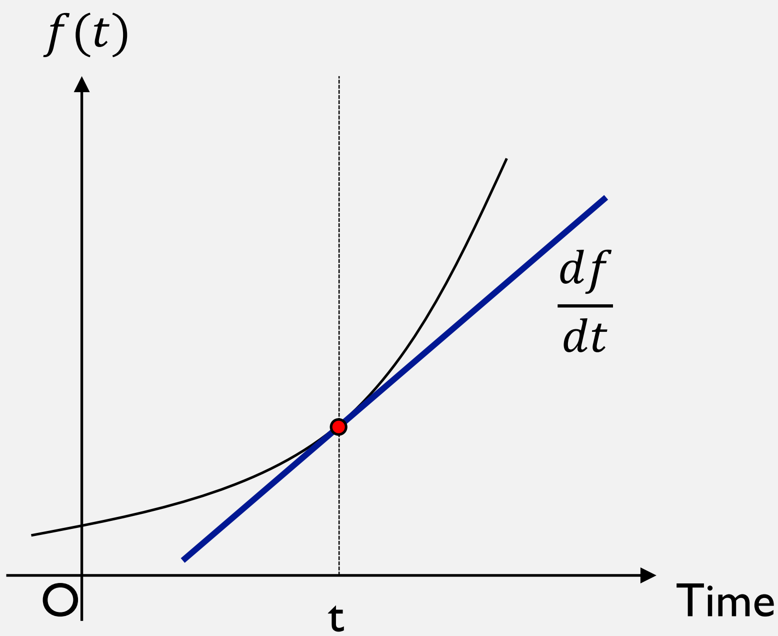
我々が計算機を通して目にするものは離散化されている

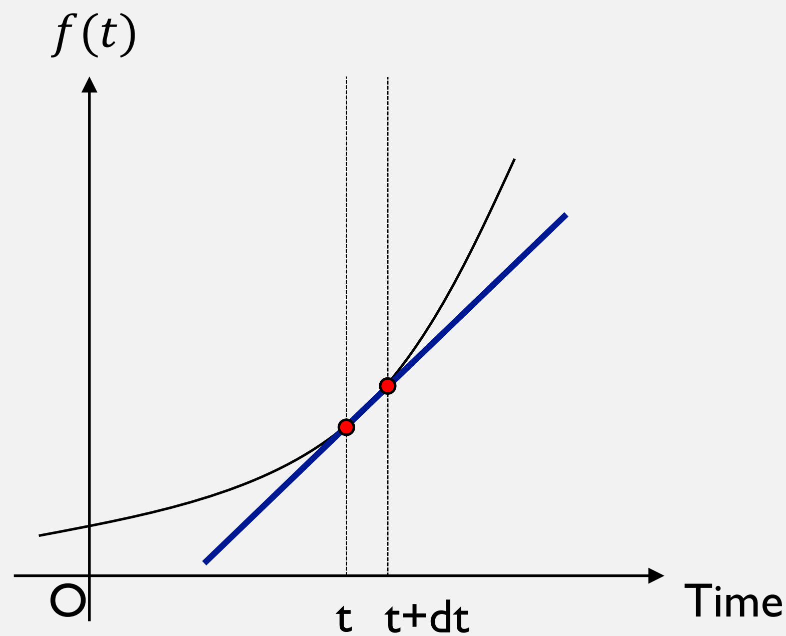
空間の離散化

離散的な世界　　　　　　連続的な世界



時間の離散化





$$\frac{df}{dt}$$ 時刻tにおける傾き

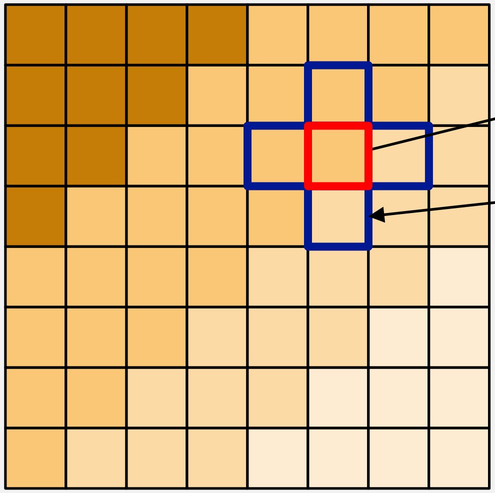
$$\frac{df}{dt} \approx \frac{f(t+dt)-f(t)}{dt}$$

時間変化＝現在と少し未来の差

数値計算

$$\boxed{\frac{\partial T}{\partial t}} = -\kappa \boxed{\frac{\partial^2 T}{\partial x^2}}$$

ある場所の時間変化量は　　　まわりの平均との差をへらそうとする



この地点での次のステップでの値を

周りの値をみて決める

上記の操作をすべての地点について繰り返すと
次のステップ(少し未来)での「世界」がわかる

「ステップ」を繰り返せば、遠い未来の世界がわかる

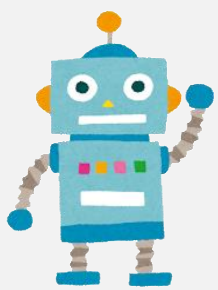
これをプログラミングしよう

プログラミングとは

コンピュータがわかる形で指示を与えること

コンピュータは**機械語**しか理解できない

```
554889e5
897dfc89
75f88b55
fc8b45f8
01d05dc3
```



機械語は数字の羅列

人間にわかりやすい言語から機械語に翻訳する

これを**プログラミング言語**と呼びます

プログラムの翻訳方式

プログラミング言語
(人間がわかる)

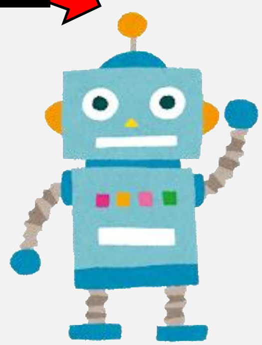
機械語
(コンピュータがわかる)

スクリプト言語

```
def add(a, b):
    return a + b
```

インタプリタ
(同時通訳)

53 17 7c 7c



コンパイラ言語

```
int add(int a, int b){
  return a+b;
}
```

コンパイラ
(翻訳)

```
554889e5
897dfc89
75f88b55
fc8b45f8
01d05dc3
```

プログラム言語の種類

実行方法 (翻訳の仕方)

スクリプト言語：Python, Ruby, Perl, JavaScript, ...
コンパイル言語：C, C++, Fortran, Java, Rust, ...

※ 最近はこの二種類の区別は曖昧

パラダイム (設計思想)

手続き型、関数型、オブジェクト指向、...

※ 現代のプログラミング言語はほとんどがマルチパラダイム

ブロックの表現方法 (見た目)

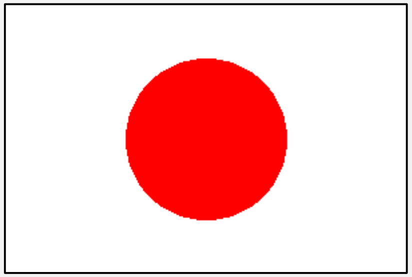
中括弧型：C, C++, Perl, Java, JavaScript, Rust, ...
キーワード型：BASIC, Ruby, シェルスクリプト, ...
インデント型： Python, Haskell, F#, ...

ブロックとは

プログラムは原則として上から順番に実行される

「日の丸」を描画するプログラム

```
from PIL import Image, ImageDraw
white = (255,255,255)
red = (255,0,0)
im = Image.new("RGB", (300, 200),white)
draw = ImageDraw.Draw(im)
draw.ellipse((90, 40, 210, 160), fill=red)
im.show()
```



ブロックとは

プログラムは原則として上から順番に実行される

```
from PIL import Image, ImageDraw
white = (255,255,255)
red = (255,0,0)
im = Image.new("RGB", (300, 200),white)
draw = ImageDraw.Draw(im)
draw.ellipse((90, 40, 210, 160), fill=red)
im.show()
```

ライブラリの読み込み

色の定義

イメージの作成

円の描画

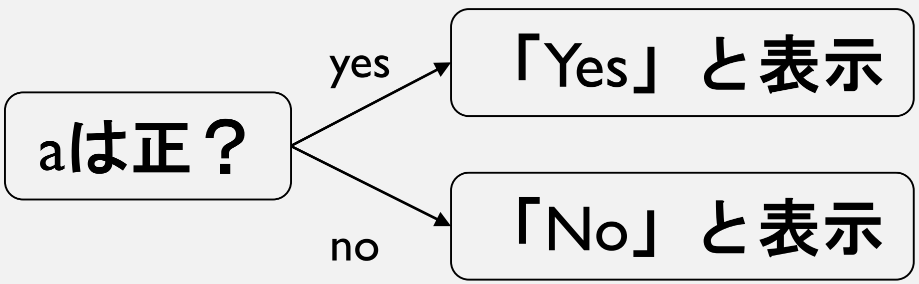
イメージの表示

実行順序

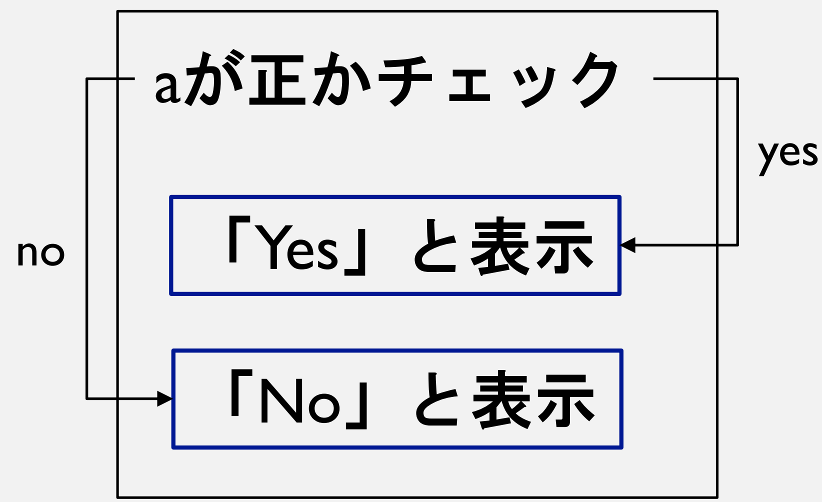
ブロックとは

しかし、順番に実行したくない場合もある

実装したいロジック



プログラムでの表現

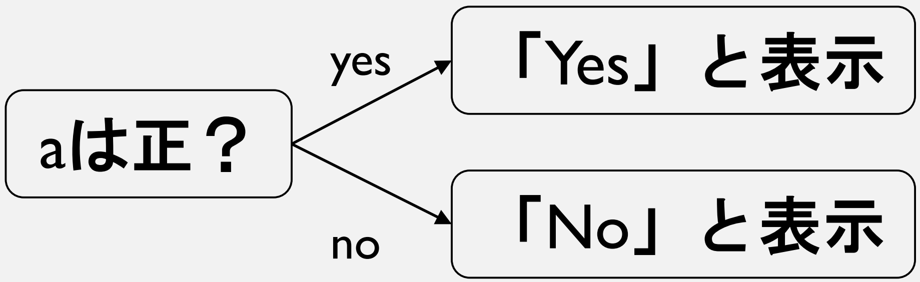


プログラムを「かたまり」にわけて
その「かたまり」に処理を飛ばす

ブロック＝プログラムのかたまり

ブロックの表現

ブロックの表現方法はプログラミング言語により異なる



C言語
カッコで表現

```
if (a>0){
puts("Yes");
}else{
puts("No");
}
```

Ruby
キーワードで表現

```
if a > 0 then
puts("Yes")
else
puts("No")
end
```

Python
インデントで表現

```
if a > 0:
  print("Yes")
else:
  print("No")
```

Pythonの文法

原則として「上から順番」に実行される

```
a = 1
b = 2
c = a + b
print(c)
```

3が表示される

Pythonの文法：変数

値につけるラベルを変数と呼ぶ

```
a = 1
b = 2
c = a + b
print(c)
```

a, b, cは、それぞれ1,2,3という値につけられたラベル

長い名前をつけることもできる

```
very_very_long_variable = 1
```

Pythonの文法：変数

Pythonは代入によって変数を宣言する

```
a = 1
```

初めて使われた変数はここで作られる

```
a = 1
a = 2
```

二度目以降は「上書き」される

Pythonの文法：制御構造

if文：条件が成立しているかどうかで処理を分岐させる

コロンを忘れないように

```
if 条件:
    条件が成立した時に実行
```

インデントする

インデントを
そろえること

```
if 条件:
    条件が成立した時に実行するコード
else:
    条件不成立時のコード
```

```
if 条件1:
    条件1が成立した時に実行
else if 条件2:
    条件1が不成立、かつ条件2が成立した時に実行
else:
    条件1と2、いずれも不成立時に実行
```

Pythonの文法：繰り返し

指定の内容を繰り返すことができる

```
for 変数 in リスト等:
        繰り返す内容1
        繰り返す内容2
```

```
for i in [0, 1, 2]:
        print(i)
```

0 1 2が表示される

```
for i in range(3):
        print(i)
```

こんな書き方もできる

```
for i in range(100, 200, 50):
        print(i)
```

100から200未満まで50刻み
→ 100, 150が表示される

Pythonの文法：関数

よくつかう処理を「関数」としてまとめることができる

```
def 関数名(引数):
    処理内容1
    処理内容2
```

```
def hello():
    print("Hello")

hello()
hello()
```

関数は定義時には実行されない

あとで何度でも実行できる

```
def add(a, b):
    return a + b

c = add(1,2)
```

関数の入力は「引数(ひきすう)」と呼ぶ

関数はreturnで値を返すことができる

Pythonの文法：ライブラリ

ライブラリ＝便利なツールをまとめたもの

Pythonはライブラリが豊富かつ強力

ライブラリを使いたい場合はimport文を使う

例：三角関数が使いたい場合

```
import math
a = math.sin(1.0)
b = math.cos(2.0)
```

「mathライブラリを使うよ」という宣言

mathライブラリのsin関数を使う

mathライブラリのcos関数を使う

「モジュール名.関数名(引数)」の形で
mathモジュールの中の関数を呼びだす

Pythonの文法：ライブラリ

いちいちモジュール名を書きたくない場合はfrom文を使う

```
from math import sin, cos
a = sin(1.0)
b = cos(2.0)
```

「*」で、そのモジュール内の関数すべて使うこともできる

(思わぬバグの元なので非推奨)

```
from math import *
a = sin(1.0)
b = cos(1.0)
c = tan(1.0)
```

Pythonの文法：ライブラリ

「as」インポートするライブラリに別名をつける

numpyをロードして「np」という別名をつける

```
import numpy as np
```

あとでnpを使ってライブラリ内の関数を呼び出せる

```
np.random.randint(1,7)
```

matplotlibのpyplotにpltという名前をつけるのも良く行われる

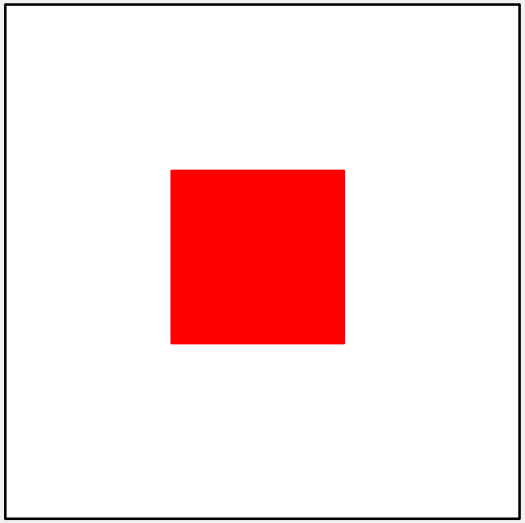
```
from matplotlib import pyplot as plt
```

他にもたくさん文法や機能があるけれど、とりあえず組んでみよう！

テーマ１:拡散方程式

$$\frac{\partial T}{\partial t} = \kappa \Delta T$$

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

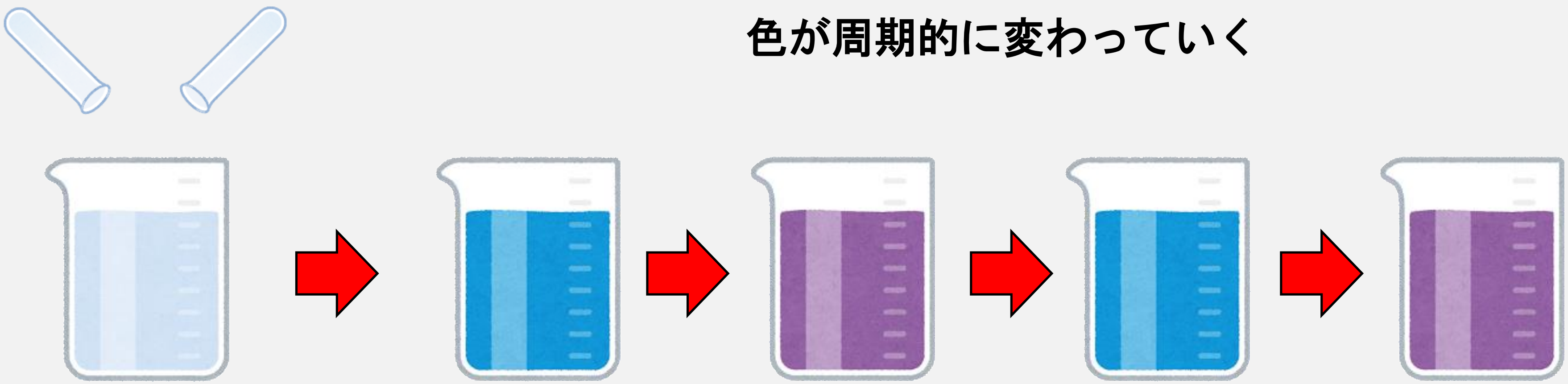


正方形領域の中央部分を正方形状に熱し、
熱がどのように伝わっていくか観察する

テーマ2：反応拡散方程式

Belousov-Zhabotinsky (BZ) 反応

いくつかの溶液を混ぜる



周期的な化学反応が拡散と組み合わさると？

テーマ2：反応拡散方程式

二種類の化学物質uとvがお互いに反応しながら拡散する式

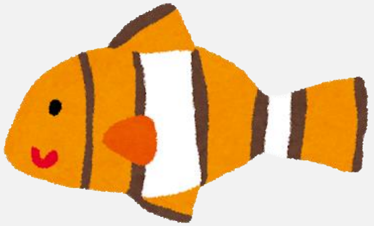
$$\frac{\partial u}{\partial t} = \boxed{D_u \Delta u} - \boxed{uv^2 + F(1-u)}$$

$$\frac{\partial v}{\partial t} = \boxed{D_v \Delta v} + \boxed{uv^2 - (F+k)v}$$

拡散　　　反応

チューリング・パターン

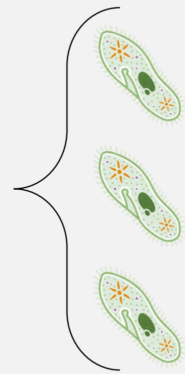
反応と拡散がおりなす不思議な模様



テーマ3：ロジスティックマップ

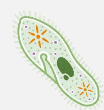
第i+1世代

子供を作って死ぬ



十分栄養を蓄えたら
たくさん子供を作る

第i世代



栄養がないと
子供も少ない

栄養がほとんどないと
子供を作ることができない

この振る舞いを数式で表現する

テーマ3：ロジスティックマップ

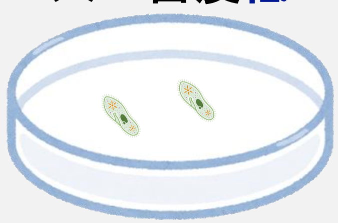
第i+1世代の個体数

第i世代の個体数

$$\boxed{N_{i+1}} = \boxed{a\left(1 - \frac{N_i}{N_{\max}}\right)} \boxed{N_i}$$

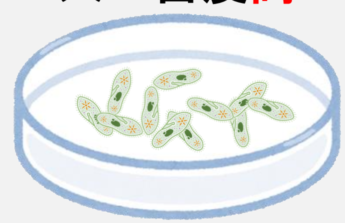
一匹が生む次世代の個体数

人口密度低



子供をたくさん生む

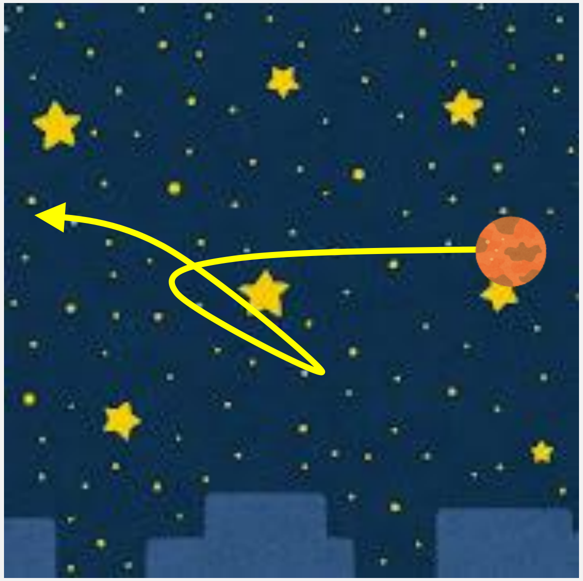
人口密度高



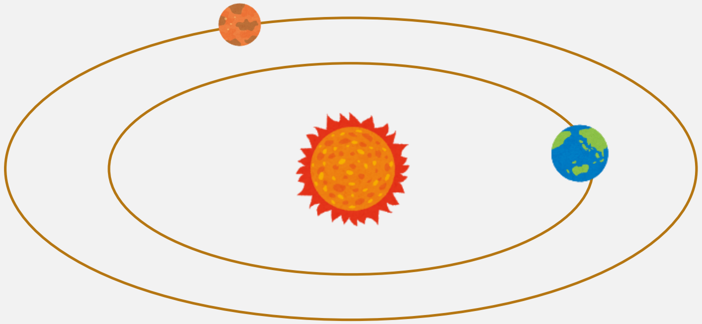
子供をほとんど産まない

そのうち個体数が落ち着く？

テーマ4：分子動力学法



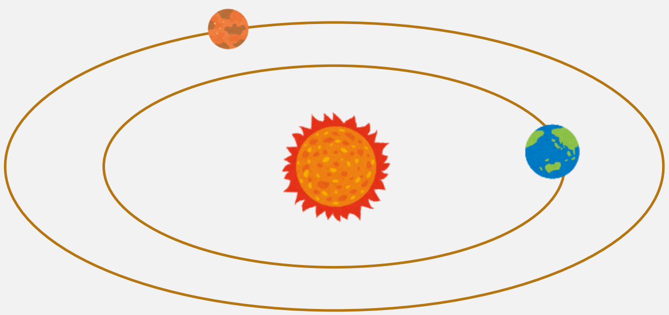
地球から見ると惑星は複雑に動く



太陽系の外から見るとその動きは単純

テーマ4：分子動力学法

太陽系の動きを**支配**しているのはニュートンの運動方程式



$$m\frac{d\vec{v}}{dt} = \vec{f}$$

速度の変化は力に比例する

$$\frac{d\vec{r}}{dt} = \vec{v}$$

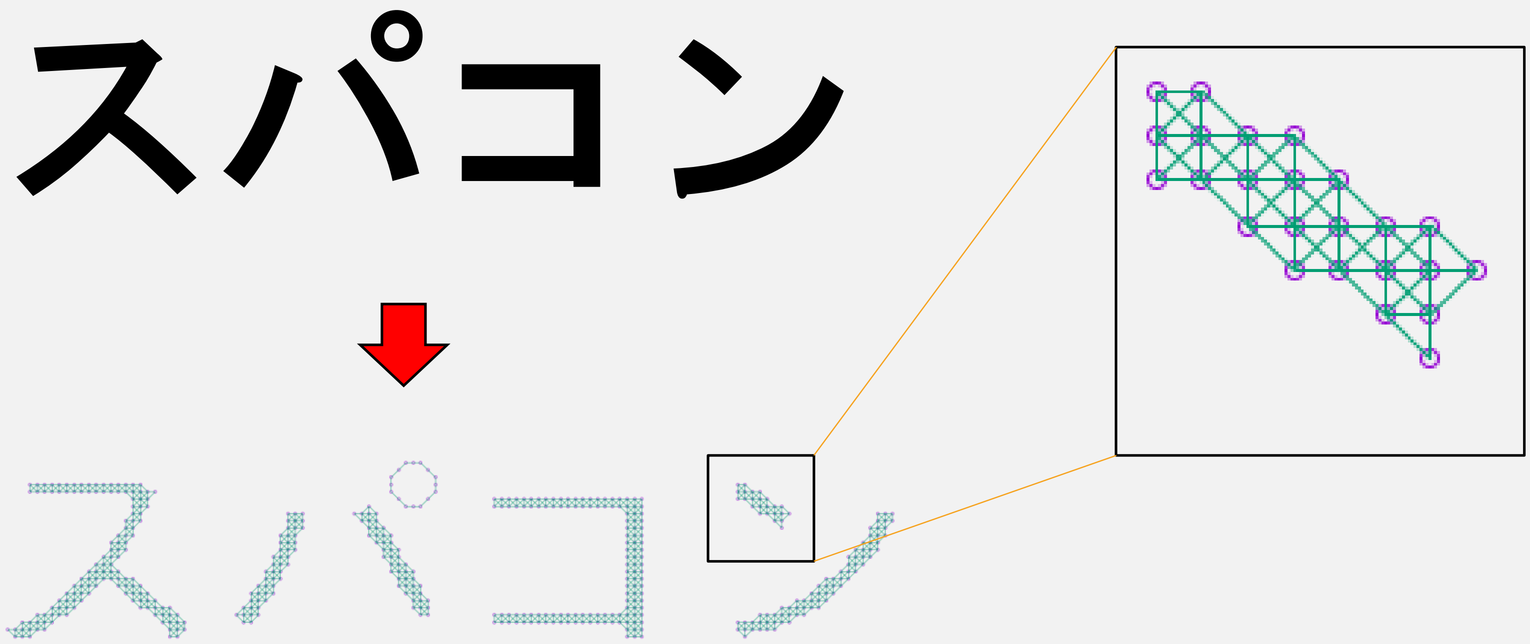
速度×時間が位置の変化

Molecular Dynamics method

これを数値的に解くのが分子動力学法

テーマ4：分子動力学法

「文字の形」の構造物を作り、重力をかけてその変化を調べる



ビーズをバネでつないだモデルに変換