Media Programing 1 - Processing

- proc.01 - 2009.04.21 hrr

目的

Processingを使って、プログラミングの概念と要素を学ぶ。
プログラミングの基本がわかれば、他の言語（Max, C, C++, perl,python)でもきちんとしたプログラムが作れるようになる。
Processingらしいプログラムではなく、オブジェクト指向プログラミングやネットワークの基礎的な学習を中心とします。本当の基礎はやりません。

この授業では、まずは手を動かしてください。
少しでもコードを書いて、書く事に慣れてください。
授業内でも、どんどん書いてもらいます。

内容

Processing 全６回

- ・プログラムの基礎の復習
- ・基本的なアルゴリズムの実現（サーチ、ソートなど）
- ・配列、データ構造、
- ・オブジェクト指向、
- ・アニメーション、衝突判定など制御
- ・ネットワーク対応

最終回のテスト
最終課題：対戦型ゲーム作品？　ネットワーク型作品？

\\ad.iamas.ac.jp\common\MediaProgramming1_20100331\Processing

開発環境

Processing1.03以降。
processing はJavaを元に一枚皮をか被せたようなものなので、eclipseなどを使ってJavaプログラムとして開発することもできる。
1.0からは見えなくなったのでちょっと面倒。

processing:
　http://processing.org/
eclipse:
　http://www.eclipse.org
　http://www.eclipsewiki.net/eclipse/

参考文献：

前川 峻志 , 田中 孝太郎 ,Built with Processing[改訂版] 3570円（日本語）
Ben Fry, Visualizing Data, 3702円
ビジュアライジング・データ —Processingによる情報視覚化手法、オライリージャパン　3780円

今日のテーマ

いよいよ本題。
プログラミングの基礎を復習。

Processing を使ってみる

まずは、四角いエリアを作成し、真ん中に点を書く。
セミコロンは、一つの命令の区切り（一つの命令文の終わり）として重要。
忘れないように！

まずは定番、HelloWorld!

```
println("HelloWorld!");
```

下のコンソール部分に文字が表示される。
ついでに、小さいウインドウも表示される。

ウインドウの大きさを決めるには、

```
size(200,200);
point(100,100);
```

線を描く。

```
line(0, 0, 200, 200);
```

四角を書く。

```
rect(10, 10, 100, 100);
```

丸を描く。

```
ellipse(150, 150, 50, 50);
```

色を付ける。
　背景色、図形の色、塗りつぶし

```
background(128, 128, 255);   //背景色
```

```
stroke(255, 0, 0);            //線の色
noFill();                     //塗りつぶししない
fill(128);                    //塗りつぶし色、この場合グレー
colorMode(HSB, 100);          //カラーモード、RGB,HSB と色指定の範囲
fill(0, 0, 255, 50);          //塗りつぶし、アルファ付き
colorMode(HSB, 100);          //HSBカラーモードにする
```

いろんな場所にいろんな色でいろんな形を描いてみる。
実行環境上部の三角ボタンを押すと実行し、四角ボタンで停止する。
エラーがあると、下部にメッセージが出る。
こまめにcmd-S または、saveボタンを押して保存すること。

なお、コマンド等の詳細については、以下を参照。
processingのreference: ローカル or
http://processing.org/reference/index.html
参考図書

プログラムの基本要素

変数

値を入れるための”もの”
箱があってそれに名前がついてる

変数の使い方

変数は使う前に宣言をする必要があります.
宣言には、変数名と変数の型を指定します.

```
int i;        // 整数型のiという変数
```

変数の型には,

int	整数型
float	浮動小数点型
char	文字型
byte	1byteの整数
boolean	論理型(true/false)
color	色型

などがあります.

整数型には整数を浮動小数点型には実数を入れることができます.

--

```
int i;     //  変数の宣言　一回すればよい

i = 10;    //  変数へ値を代入する
```

```
   i = 11.5; // エラーが起こる
--

   float f;

   f = 11.5;

--

   i = i * 5;
   println(i);
   i++;
   println(i);

--

   char c;

   c = 55; //文字コード55の文字
   println(c);

   colorMode(RGB, 255);
   color cl;
   cl = color(128, 255, 255, 50);
--
```
一度宣言した変数をまた宣言しようとするとエラーになる。

命令文(statement)

命令語を組み合わせて、命令文を作る。

```
--
   int i;
   i = i + 10;

   line(0, 0, 100, 100);

--
```

文の区切りはセミコロン。

複数の文を{ } で囲むと一つの文となる。

＊後から実例が出てきます！

変数の有効範囲（スコープ）

変数には、有効範囲がある。
一番外側で宣言すると、プログラムの全体で有効。
{ } で括られたブロック（文をまとめた文）内で宣言すると、
そのブロックの範囲内だけで有効。

このような変数の範囲のことを変数のスコープと呼びます.

```
--
   {
      int j;
      j = 20;
   }
   j = 10; //エラーになる
--
```

変数をスコープを意識しながら，自分で宣言し値を代入し出力してみよう.
同じ変数名で変数の宣言をした場合、スコープによって同じ名前だけど違うものとなっていることに注意する。

関数
ある機能を実現する手続きを名前を付けて定義したもの。

クラス
オブジェクト指向のときにでてきます。

参考：コメントの書き方
`//`：`//`以降の一行をコメントにする
`/*  */`：`/*` と `*/` の間をコメントにする。複数行も可。

制御構造

条件分岐

if　（もし、〜なら〜〜する）

```
if (条件式) 命令文;

--
   if (i > 10) i = 0;

   if (i > 10) {
      i = 0;
      j++;  // j = j + 1;
   }                          //{ } で囲むと一つの文とみなされる
--
```

if else　（もし、〜なら〜〜する、そうじゃなければ〜〜〜する）

```
   if （条件式）文1 else 文2

--
```

```
   if ( x > 200) {
      x = 0;             // xが200より大きいとき
      y = y + 10;
   } else {
      x = x + 10;        // そうじゃない(xが200以下）のとき
   }
--
```

switch case　（式の値に応じて実行する文を決める）

```
   switch(式) {
      case 値: 文１;
               break;
      case 値: 文２;
               break;
      default: 文３;
   }

--
   int num;

   switch (num) {
      case 1: println("1");
               break;
      case 2: println("2");
              break;
      default: println("default");
   }
--
```

条件式

```
   i < 0                   // i より小さいとき
   i <= 0                  // i 以下のとき
   i > 0                   // i より大きいとき
   i >= 0                  // i 以上のとき
   i == j                  //  i と j が等しいとき
   i != j                  // i と j が等しくないとき
   (i < 0) && (j > 0)      // 且つ  AND
   (i < 0) || (i > 10)     // または OR
   !i                      // 否定  not
```

授業内課題１：

processing では、マウスの座標値は、mouseX, mouseY という変数に
自動的にセットされている。
マウスの座標値に応じて、色の違う点や四角を描いてみる。
まずは if を使って、右半分に描くときは赤、左半分に描くときは緑で描いてみる。
点の色はstroke(255,0,0)などで変えられます(r, g, b)の順番。

```
--
   void setup(){      // 最初の一回だけ実行される
      size(200,200);
   }

   void draw(){       // この部分はずっと繰り返し実行されている
      point(mouseX, mouseY);     // ここをif文を使って書き換える
   }
--
```

注意：
プログラムを書くときは、{ があったら次からはタブでインデントするようにして
プログラムの制御構造、スコープが見ただけでわかるように書きましょう。

繰り返し（ループ）
for　初期値と、終了条件、繰り返しごとの処理を指定して繰り返す
for (初期化; 式がtrueの間繰り返す条件式; 繰り返し毎の処理){
　繰り返しをする文;
}

```
--
   int i;
   int j = 0;

   for ( i=0; i < 10; i++){     // 最初はiを0にする、iが10より小さい間繰り返す
                                // 一回繰り返すごとにi++を実行
      println(i);                  // この文を10回繰り返す
   }
--
```

while　条件式を満たす間繰り返す
while (式がtrueの間繰り返す){
　繰り返しをする文;
}

```
--
   while ( i < 100){
      point(i++, 10);
   }
--
```

練習：
（１） １から１０００までを足した値をprintln()を使って出力するプログラムを
　　for文を使って書く。
　　int i;
　　int sum = 0;

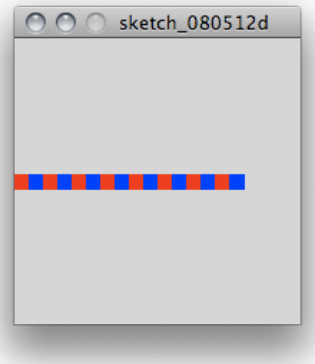
```
for(i =1;,,,){
    sum = sum + i;
}
```

（２） １から１００までの整数のうち、３で割り切れる値と７で割り切れる値を出力するプログラムをprint()、for文、if文を使って書く。
ヒント： 余りを求める演算子% 、print(i + " "); と書けば数値の後にスペースを出力してくれる。3で割り切れ且つ７でも割り切れる値を二重に出力しないこと。

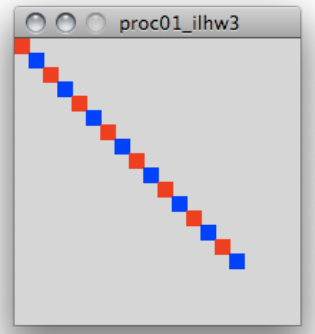
3 6 7 9 12 14 15 18 21 24 27 28 30 33 35 36 39 42 45 48 49 51 54 56 57 60 63 66 69 70 72 75 77 78 81 84 87 90 91 93 96 98 99

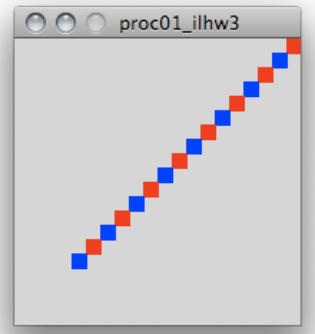
参考：
% は割った余りの値を求める演算子ですので、
10%2 は10割る２なので、余り0 となり、0を返します。
11%2 なら余りは1なので、1となります。

授業内課題２：
200x200のウインドウに10x10の赤と青の四角を並べて計１６個描く。
for文を使って１６回のループを作り、その中で偶数/奇数をif文で判定し、
赤い四角か青い四角を描く。

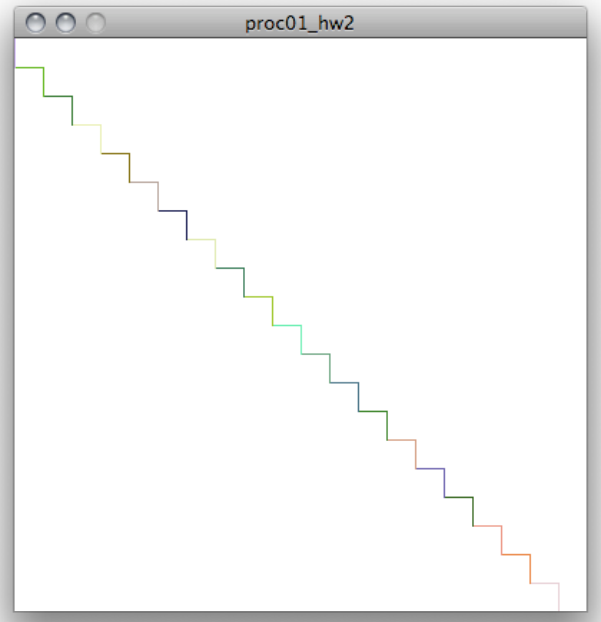


授業内課題３：
さらに四角を横に並べるのではなく、斜めに並べてみる。
各方向（左上→右下、右上→左下）で試してみる。





授業内課題４：
条件判断(if)とループを使って、階段状に点を描いていく。
階段の各々の段に別の色を付けてみる。



どうやって描いていけば良いか考えてみる。
描いていく順番をイメージし、書き出してみる。
描くための手順を繰り返しや条件を意識しながら書き出してみる。
プログラムとして使えるもの(ifやfor)によって書き直す。
プログラムを書いて試してみる。エラーが出たら直す。
試行錯誤する。

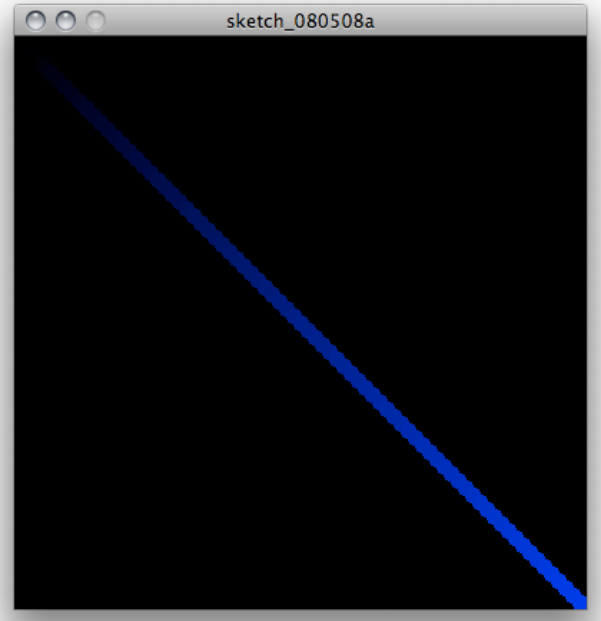
このようにプログラムとして実現可能な手順を見つけることが、
アルゴリズムを考えること。

processing には、描画関係にはたくさんの命令があります。
特別に紹介することはないので、自分でいろいろ試してみてください。

参考：
乱数を使って、色や座標を決めるとバリエーションがでます。
float f;
f = random(100);　　// 0 から 100までの実数を返す。

次回までに、if 文やfor文などを使って、いろいろと描画をして、
使い方に慣れておいてください。

授業内課題５：
右上から左下にかけて徐々に大きくなりながら、
色も段々と濃くなりなが少しずつ重なっている円を描く。



きちんと色の濃さが０から255までになるように計算して色を決定する。
ウインドウの大きさを変えても円の数は増えても、グラディエーションは
変わらないようにする。
ウインドウの幅は width という変数に入っています。
ちなみに、heightにはウインドウの高さが入っています。

関数

一連の手続き（命令文の固まり）をまとめて、名前をつけたもの。
名前は自由に決める事ができます。ただし、すでに予約されている
setup(), draw() など使うべきではありません。
関数は、値を返すことができ、返す型を宣言して定義する。
関数は、引数（ひきすう、パラメータ）を取ることができる。

この先、関数を使うときには、いきなり　size(100,100)などとは
書けなくなります。
必ず、setup()関数内に記述してください。

関数の定義

関数の型　関数名(引数の型と変数名，引数の型と変数名)

```
int myfunction(int x, int y){
    //関数の中身
}
```

引数の数は自分で決める。上の例では２つ。
つまり、この関数では、整数型の引数を２つ与えて呼び出すと、
整数の値を返してくることになる。

関数の呼び出し

引数を与えて、呼び出すだけ。
size()やrect()も関数なので、既に使っていることになる。
void 型は何も値を返さないので、変数を呼び出すだけで良い。

それ以外の型では、値を使う必要があるので、別の変数に代入したり、
別の関数のパラメータとして使ったりする。

```
--
   void setup(){
      int i,j;

      size(200,200);
      noFill();

      myfunc();          // 関数myfuncを呼んで色を設定
      i = j = 5;
      rect(i, j, 10, 10);

      myfunc();
```

```
   i = myfunc2(i);    // 関数myfunc2を呼んで、計算
   rect(i, j, 10, 10);
}

void myfunc(){        // 値を返さない myfunc という名前の関数
   stroke(random(255), 0, 0);
}

int myfunc2(int x){ // 整数の値を返すmyfunc2という名前の関数
   x = x * 10;      // 引数として受け取った値(xに入っている)
                    // にかけ算を行う、xはこの関数内だけで有効
   return x;        // 結果を返す
}
```

--

練習：

--

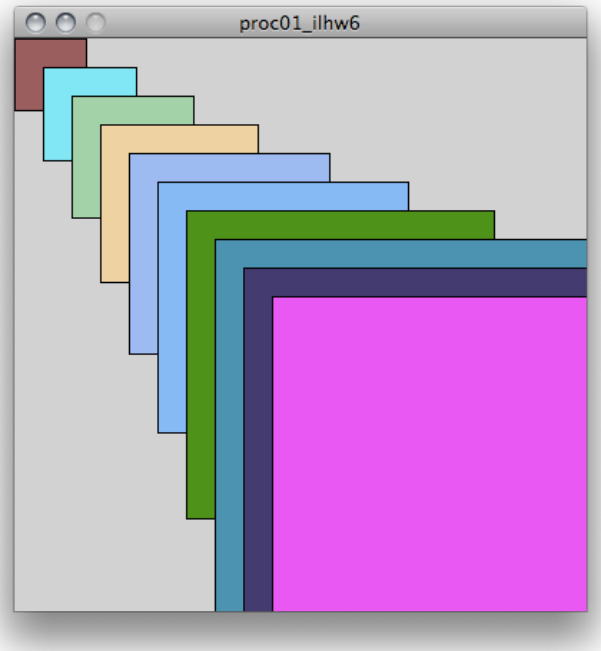
```
void setup(){

   int i = 0;
   int result;
   while (i <= 300){
      i = i + 100;
      result = sumOdd(i);
      println("odd_" + i + ": " + result);
      result = sumDiv7(i);
      println("div7_" + i + ":  " + result);
   }
}

int sumOdd(int num){
   int i;
}

int sumDiv7(int num){
}
```

--

（1）引数として与えられた値までの、奇数の値の合計を計算する関数sumOddを
　　完成させる。
（2）引数として与えられた値までの、７で割り切れる値の合計を計算する
　　関数sumDiv7 を完成させる。

授業内課題６：
先ほどの例のうち、「色をランダムに設定し、四角を描く」部分を関数としてまとめる。
引数として、四角の始点の座標と終点の座標を与える。
返す値は特にないので、void 型とする。

つまり、
void mykansu(int sx, int sy, int ex, int ey)
の関数を完成させる。
この関数を使うプログラムとして、段々と大きくなりながら
ランダムの色の四角を描く。



アニメーション

アニメーションを行うためには、Processingで既に用意されている
関数を元にしてプログラムを書いていく必要があります。
プログラム全体で使う変数（グローバル変数）の宣言など以外には、
関数の外で実行用のコードを書く事は基本的にありません。

用意されている関数はreference にもあるようにsetup(), draw()などや、
マウスやキーボードのイベントに対応した関数
であるmousePressed(), keyPressed()などがあります。

setup() 関数は、プログラムの開始時に一回だけ行われるため、
全体の初期化などの準備処理を行います。
draw() 関数は、frameRateに応じて、自動的に繰り返し呼び出されます。
この関数の中身を記述することで、アニメーションを描画できます。

```
void setup(){
   size(400,400);
```

```
    background(255);
    frameRate(10);
  }

  int x = 0;
  int y = 0;

  void draw(){
    stroke(0, 255, 0);
    point(x, y);       // rect(x, y, x+10, y+10); に変えてみると？
    x++;
    y++;
  }

  void fadeToWhite(){     // Processing本にある例　フェードアウト
    stroke(255);
    fill(255, 30);
    rect(0, 0, width, height);
 }
```

授業内課題７：
三角関数 sin() を使って、上下に動きながら移動する円を描いてみる。
三角関数を扱うには、値をfloat（浮動小数点）で使うため、
座標値もfloat で宣言してしまった方が良いかも。
あるいはキャストすることで強制的に型変換を行う。

キャストの例

```
--
    int x;
    float f;
    x = int(f* 12.5);          // int() 関数を使う
       または、こういうやり方もできる( java, C 風)
    x = (int)(f * 12.5);       // (int) を付ける事でint にキャスト
--
```

マウスイベント

マウスの座標値は、mouseX, mouseYに自動的に格納される。
マウスのクリックの検出は、用意されている関数によって行う。

```
  void mousePressed() {   // マウスをクリックすると
    noLoop();             // アニメーションが止まる
  }
```

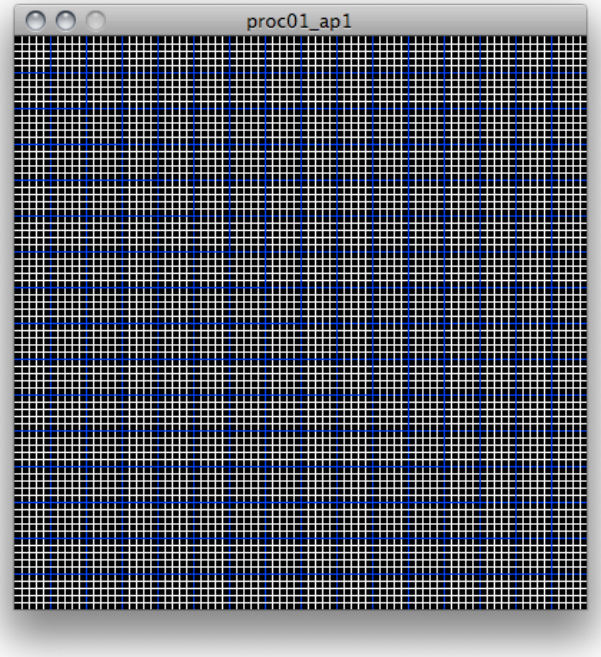
マウスがクリックされると、mousePressed関数が呼ばれるように
なっている。
プログラムとしては、マウスのイベントを常に監視していて、

イベントが発生すると、関数を呼ぶようになっている。
グラフィカルなインターフェイスを持つプログラムでは、
このようなイベント駆動（event driven）によるプログラムが
一般的に使われている。

授業内課題８：
boolean 型の変数 isAnim を使って、
マウスをクリックする毎に、アニメーションの停止/再開を繰り返すようにしてみる。
ヒント：isAnim はグローバル変数として宣言する。
isAnim がtrue ならは noLoop()を呼びループを停止し、
isAnim がfalseならば loop()を呼んでループを開始する。

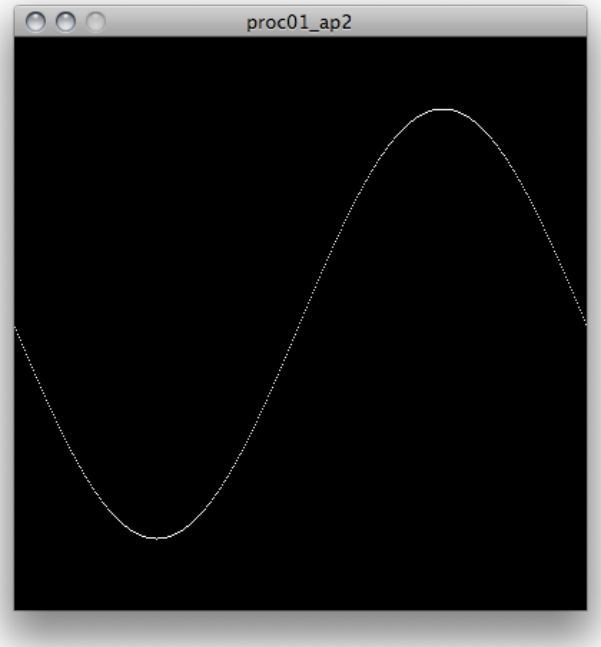
来週は休講、次回は５月１２日。
課題はたくさん出しておきます。
５月１１日までに、私あて(hrr@iamas.ac.jp)に提出すること。
授業内課題も同時に提出すること。

課題１：
400x400のキャンバスを作成し、その上に
5本毎に違う色を付けた線を引く事で、グリッドを描画する。

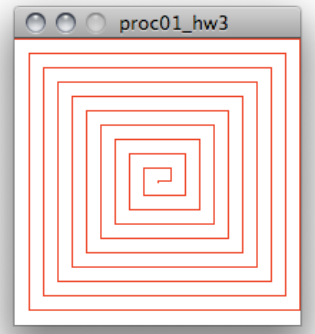


課題２：

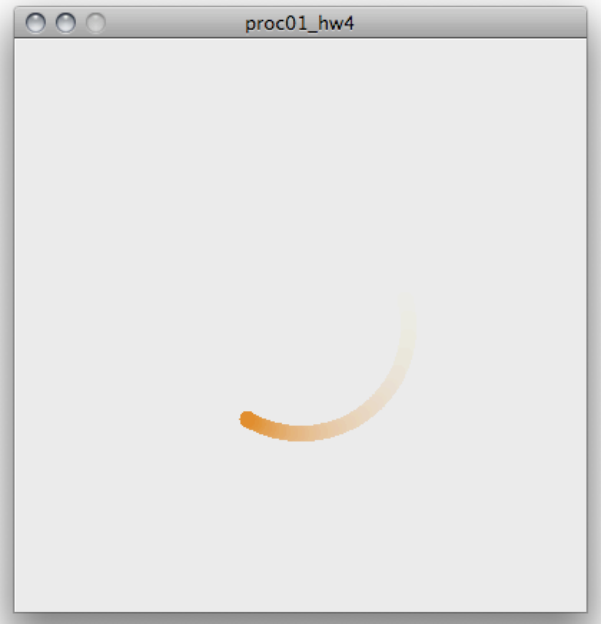
点でサイン波を描いてみる。
sin() は、float 型で使うこと。
パイは TWO_PI という定数を利用。



課題３：
渦巻き状の線を描く。
線毎に色を変えていく。
width , height という変数には、ウインドウの幅と高さが
自動的にセットされています。
これを使うと、ウインドウの大きさを変えても
大きさに応じて描画できるようになります。
ヒント：while文、絵を描いて何が変化しないといけないか確認する。
　　各線で視点と終点の座標を式で表してみる。



課題４：
円を描くようにグルグル回る円のアニメーションを作成する。
マウスをクリックすると、アニメーションが停止する。
再度マウスをクリックすると、違う色の円が回り始める。



課題５：
マウスポインタの周りを円を描くようにグルグル回る円のアニメーションを作成する。
マウスをクリックすると、アニメーションが停止する。
再度マウスをクリックすると、違う色の円が回り始める。