第 1 講 基本的なプログラミングと電子回路の作成

1　本実習の目標

- Processing でお絵かきアプリを作成する。
- LED を光らせるために必要な抵抗値を理解する。
- 抵抗のカラーコードを理解する。
- プログラムの制御なしで、スイッチを押すと LED が光る回路を組む。

2　実習の準備

ノート PC の準備

本実習では、各個人に 1 台のノート PC を割り当てます。次回の授業からは開始までに、あらかじめロッカー内から自分が使用するノート PC を取り出して準備を行なってください。授業終了時には、ノート PC を収められていた箱に収納し、ロッカー内の指定位置まで返却してください。

このノート PC は、**他の実習授業でも使用します**ので、まず最初に各個人のアカウントを作成します。**授業内の作業は、各個人アカウントで行うようにしてください。**指示にしたがって、共有アカウントでログインし、個人用アカウントの作成、パスワードの設定を行なってください。

実習で用いる部品

図 1: Arduino

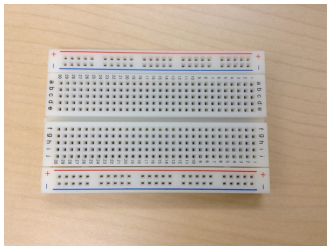


図 2: ブレッドボード

図 3: ジャンプワイヤ

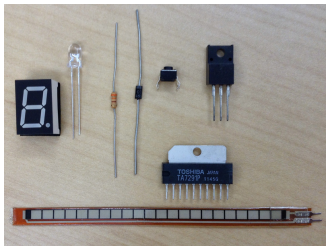


図 4: 抵抗、スイッチ、センサ etc.

注意!

Arduino などの電子部品は壊れやすいので取り扱いに注意してください。

- 静電気 (Arduino が壊れてしまいます)
- 電子部品のピンは曲がりやすいので無理に差し込まないように注意! (壊れます)
- PC に刺したまま配線をいじらない (最悪、PC ごと壊れます …)

3　電子回路の基礎

電気、電流、抵抗のおさらい

よく、電気の流れは水の流れに例えられます。厳密にいえばさまざまな違いはありますが、その性質を大まかにとらえるには、水の流れに例えて理解するのは有効な方法です。

電圧

電圧は 2 点間の高度 (電位) の違いを表す用語です。水は高度の高いところから低いところに向けて流れますが、電気も電位の高いところから低いところに向けて流れます。水の場合には、それぞれの地点の高さを比較するのに、海抜などを基準として用います。電気の場合には、グランド (GND) を基準として比較します。電子回路ではよく「グランドを接続する」ということが行われます。これは、基準であるグランドを共通にしないと、回路の部分ごとの電圧の基準が共通にならず、意図した通りに電気が流れてくれないためです。電圧の単位はボルト (V) です。数字が大きければ大きいほど、電圧が高いことを示します。

電流

電流は、水の流れが水流であるのと同じように電気の流れです。電流は、電圧の高いところから低いところに向けて流れます。水流も多い場合少ない場合がありますが、電流も多い場合や少ない場合があります。電流の単位はアンペア (A) です。数字が大きければ大きいほど、多くの電気が流れることを意味します。

抵抗

水の場合にも、何も障害物がない場合と、くねくね曲がって流れにくくしている場合では、水の流れにくさが異なります。同様に、電気の場合にも電流が流れやすい場合と流れにくい場合があります。この電流の流れにくさを表すのが抵抗です。抵抗の単位はオーム (Ω) です。数字が大きければ大きいほど、電流が流れにくいことを示します。

よく出てくる補助単位

電圧、電流、抵抗の単位はそれぞれボルト、アンペア、オームですが、実際にはこれに接頭辞がついた補助単位が用いられる場合が多くあります。以下は、登場する補助単位の例です。

- 1,000 倍を表す単位がキロ　　(例: 10,000 Ω = 10 k Ω)
- 1,000,000 倍を表す単位がメガ (例: 1,000,000 Ω = 1 M Ω)
- 1/1,000 を表す単位がミリ　　(例: 0.01A = 10 mA)

オームの法則

電子部品に電圧をかけすぎたり、電流を流しすぎると壊れてしまいます。そのために、電源側に抵抗器を配置し、電流量を調節する必要があります。

オームの法則は、

$$V(\text{電圧}) = I(\text{電流}) \times R(\text{抵抗}) \tag{1}$$

R (抵抗) を求めるために式を変形すると、

$$R = \frac{(\text{電源電圧} - LED\ \text{にかかる電圧})}{LED\ \text{に流したい電流}} \tag{2}$$

となります。

表 1: LED のデータシートの例

順方向電圧 VF		最大定格 IF(mA)	最大定格 VR(V)
typ(V)	IF(mA)		
1.9	20	30	5

LED 抵抗値計算

例えば、電源が 5V で順方向電圧 1.9V の LED (表 1) を 20mA で光らせたい場合

$$R(\Omega) = (5V - 1.9V) \div 0.02A = 155\ \Omega \tag{3}$$

となり、155 Ωに近い 180 Ωの抵抗器を用いれば良い (OK) ということになります。

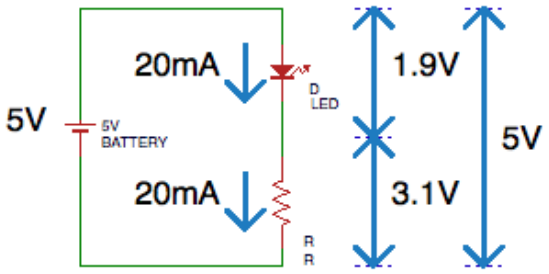


図 5: 抵抗値の求め方

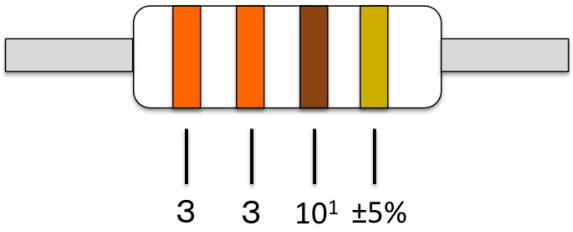


図 6: 330 Ωの抵抗

抵抗器の見方

抵抗のカラーは図 6 のように 4 本 (5 本) のカラーの帯が印刷してあります。この帯で抵抗値と誤差がわかるようになっています。抵抗値は 2 桁の数値と乗数とのかけ算で求めることができます。抵抗のカラーの読み方は帯が片寄っている方が始まりで、左から 10 の位、1 の位、乗数、誤差を表しています。色と数値、乗数、誤差との対応は表 2 となります。図 6 の抵抗の場合、左から 1 本目が橙色なので 3、2 本目が橙色なので 3、3 本目が茶色なので × 10^1 となり、抵抗値は 33 × 10^1=330 となります。また、4 本目が金色なので、誤差± 5%の 330 Ωの抵抗ということがわかります。

表 2: カラーコード表

色	数値	乗数	公称誤差	覚え方
黒	0	× 10^0	–	黒い礼 (0) 服
茶	1	× 10^1	± 1%	お茶を一杯
赤	2	× 10^2	± 2%	赤いに (2) んじん
橙	3	× 10^3	–	み (3) かんは橙
黄	4	× 10^4	–	四季 (黄) の色
緑	5	× 10^5	± 0.5%	五月ミドリ
青	6	× 10^6	–	青二才のろく (6) でなし
紫	7	× 10^7	–	紫式 (7) 部
灰	8	× 10^8	–	ハイヤー (8)
白	9		–	ホワイトク (9) リスマス
金		× 10^{-1}	± 5%	
銀		× 10^{-2}	± 10%	
色なし			± 20%	

スイッチを押したら LED を点灯させる

Arduino、スイッチ、抵抗、LED を用いて、スイッチを押したら LED を点灯させる回路を作ります。

- **電源**

  今回は Arduino から電源を取ります。Arduino は 5V と 3.3V を出力することができます。今回は 5V のピンを電源として用います。(乾電池 1 本は 1.5V)

- **スイッチ**

  今回用いるのは、押した時だけ通電するスイッチ (タクトスイッチ) です。内部でつながっている部分の方向に注意! (図 7 参照)

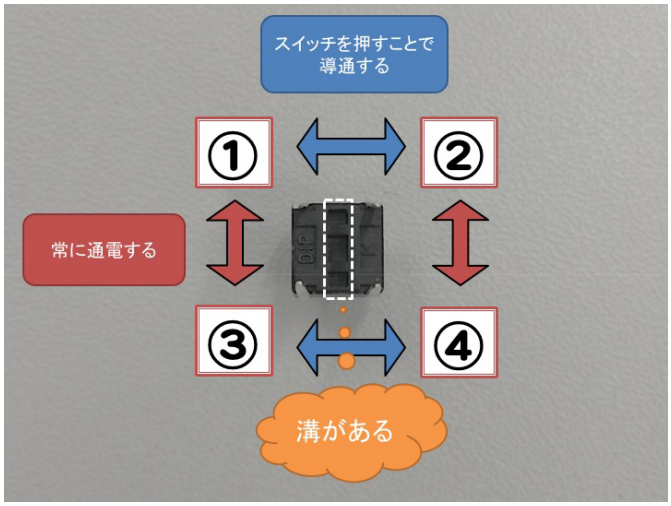


図 7: タクトスイッチ

- **抵抗器**

  電流の流れにくさをコントロールします。抵抗で LED に流れる電流を制御することができます。オームの法則を用いて抵抗値を算出します。実際には、計算結果にぴったりの抵抗器があるとは限りません。その場合、近い値の抵抗器を用います。

- **LED**

  正式名称:Light Emitting Diode(発光ダイオード)。電圧をかけると発光するダイオードで、消費電力が小さく寿命が長いです。赤色で 1.8V、青色で 3.6V 程度の電圧が必要です。極性があるので注意! (電流を流す方向を間違えると壊れます...) 足が長く、先端の小さいほうがアノード (プラス) 、足が短く、先端の大きいほうがカソード (マイナス) です (図 8 参照) 。

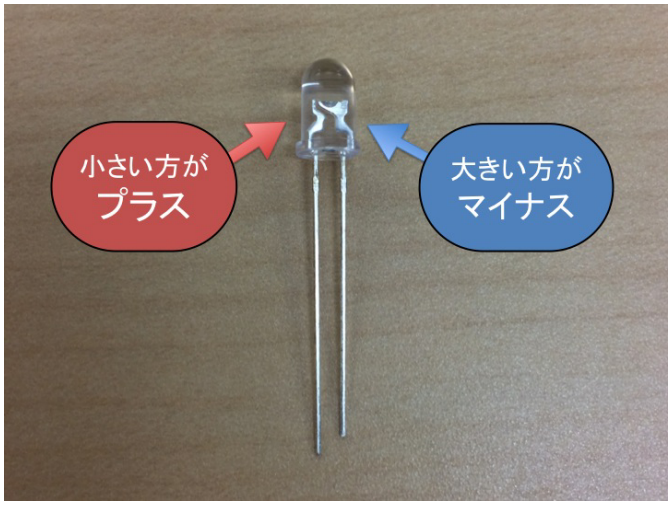


図 8: LED

- **ブレッドボード**

  通常の電子工作では、ハンダ付けなどを行いますが、この実習では部品の付け外しが簡単なブレッドボードを使います。このブレッドボードは、ソケット (穴) に部品を差し込むことで電子回路を作ることができます。穴はブレットボードの内部でつながっています。図 9 のように赤色の線と青色の線の下の一列の穴は横方向に電気的につながっています。黒い線の下の一列の穴は縦方向に電気的につながっています。一般的に、青色の線の下の一列の穴は GND ライン、赤色の線の下の一列の穴は電源ラインと呼ばれています。

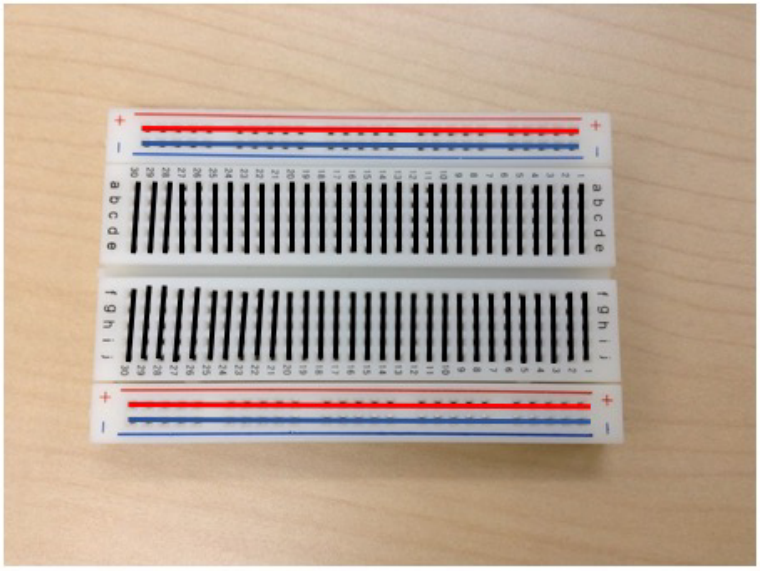


図 9: ブレットボードの仕組み

TRY1: スイッチが押されたら LED が点灯する回路を作成する

図 10 を参考にして回路を組んでみましょう。

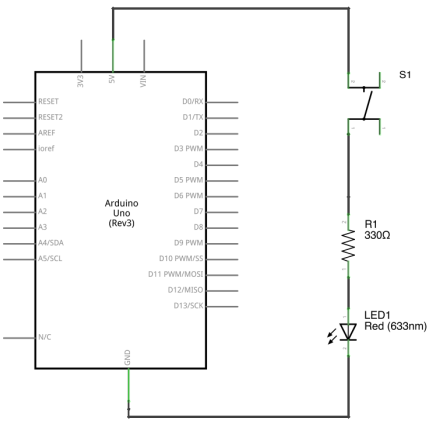


図 10: 回路図

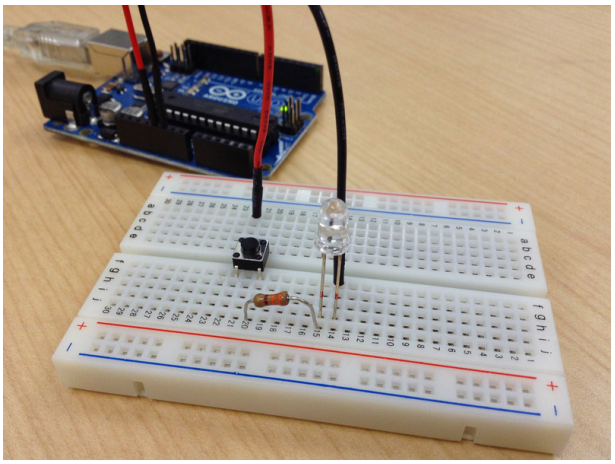
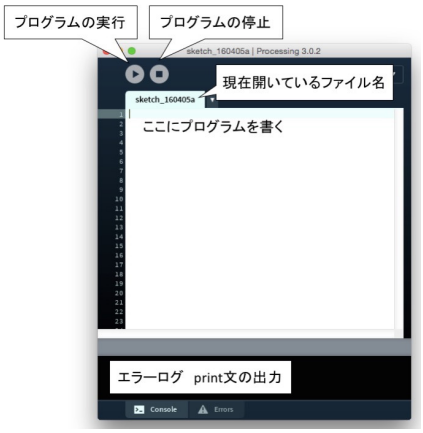


図 11: 配線例

4 Processing の基礎

本実習では Processing[1] というプログラミング言語を用います。Processing は、Java を単純化し、グラフィック機能に特化した言語です。

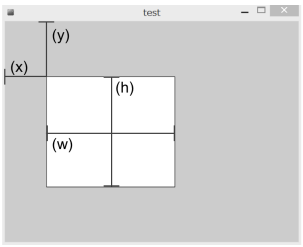
Processing による基本的なプログラミングを行います。なお、メニューバーの Help → Reference でインターネットブラウザが起動し、より詳しい説明が確認できますので、そちらも参考にしてください。ただし英語表記です。Processing にはあらかじめサンプルコード (お手本になるプログラム) が、たくさん用意されているので、それを動かしてみてどんなことができるかを見ておくのも良いかもしれません。サンプルコードは、メニューバーの File → Example から見ることができます。

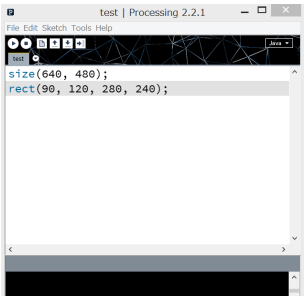


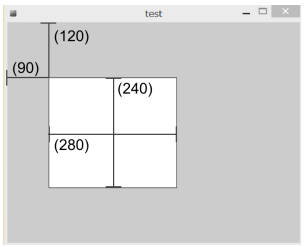
基本的な図形を描く

- 四角形を書く: rect(x, y, w, h);

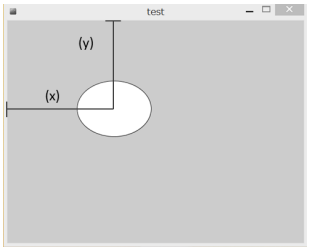
  x = x 座標、y = y 座標、w = 四角形の幅、h = 四角形の高さ

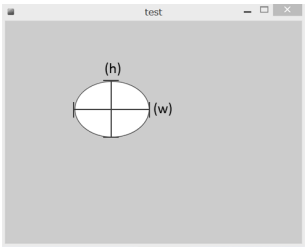




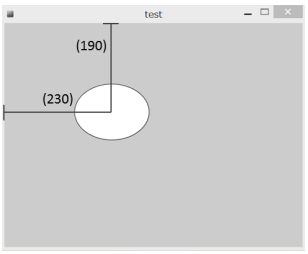


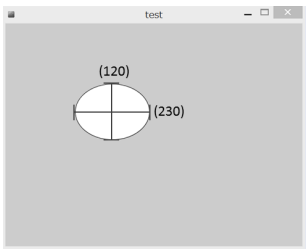
- 円を書く: ellipse(x, y, w, h);





[1] http://processing.org





- **線を書く**: line(x1, y1, x2, y2);

  x1 = 始点の x 座標、y1 = 始点の y 座標、x2 = 終点の x 座標、y2 = 終点の y 座標

図形の色を変える

Processing で図形などの色を指定したいときは、Red、Green、Blue をそれぞれ 256 段階で指定します。

```
// 四角形を赤で表示する
fill(255, 0, 0);          // 赤は r = 255, g = 0, b = 0
rect(10, 10, 100, 50);
```

また、グレースケールで指定する方法もあります。

```
fill(0);                  // 0（黒）から 255（白）で指定
```

白から黒の色で指定したいときはこちらの方が楽です。

- **図形の色を変える**: fill(r, g, b);
- **図形の色を消す**: noFill();
- **線に色をつける**: stroke(r, g, b);
- **線の色を消す**: noStroke();
- **ウィンドウの背景色を変える**: background(r, g, b);

初期化とループ処理

Processing では、プログラムを実行時に setup 関数が 1 度実行され (初期化) 、その後 draw 関数 が繰り返されます (ループ処理)。setup と draw を用いるとアニメーションなどの動的な表現ができます。

```
void setup(){
  // 初期化の処理をここに書く
  // プログラム開始時に1度だけ実行される
  // size(w, h); などはここへ
}

void draw(){
  // 繰り返したい処理をここに書く
  // setup()が実行された後にプログラムが終了するまで繰り返される
}
```

TRY2: Processing でベースとなるプログラムを作成する

1. ウィンドウサイズを幅 300 pixel、高さ 300 pixel に
2. ウィンドウの背景色を黒に
3. 幅 100 pixel、幅 100 pixel の円を青色で表示

```
void setup() {
  size(???, ???);
}

void draw() {
  background(???, ???, ???);

  fill(???, ???, ???);
  ellipse(???, ???, ??? ???);
}
```

TRY3: マウスボタンがクリックされたら図形を表示する

TRY2 で作ったプログラムを改造し、マウスが押されたときにウィンドウに図形が表示されるようにします。

Processing では、あらかじめ用意されている「mousePressed」という変数を用いることで、マウスのクリック動作を簡単に検出することができます。

```
if (mousePressed) {
  // マウスボタンを押したときの処理
} else {
  // マウスボタンを押していないときの処理
}
```

TRY4: マウスポインタで図形を操作する

TRY3 で作ったプログラムを改造し、マウスポインタの位置に合わせて図形が表示されるようにします。

Processing では、あらかじめ用意されている「mouseX」「mouseY」という変数を用いることで、マウスポインタの x 座標と y 座標を取得できます。

```
void setup() {
  size(300, 300);
}

void draw() {
  if (mousePressed) {
     ellipse(mouseX, mouseY, 32, 32);
  }
}
```