ネットワーク分野　研究室ローテーション

processingを用いたプログラミング 基礎編

Raspberry PiとProcessingによるIoTデバイス作成演習用

予備資料

編集履歴
前田　2017年5月22日
福嶋　2017年5月24日

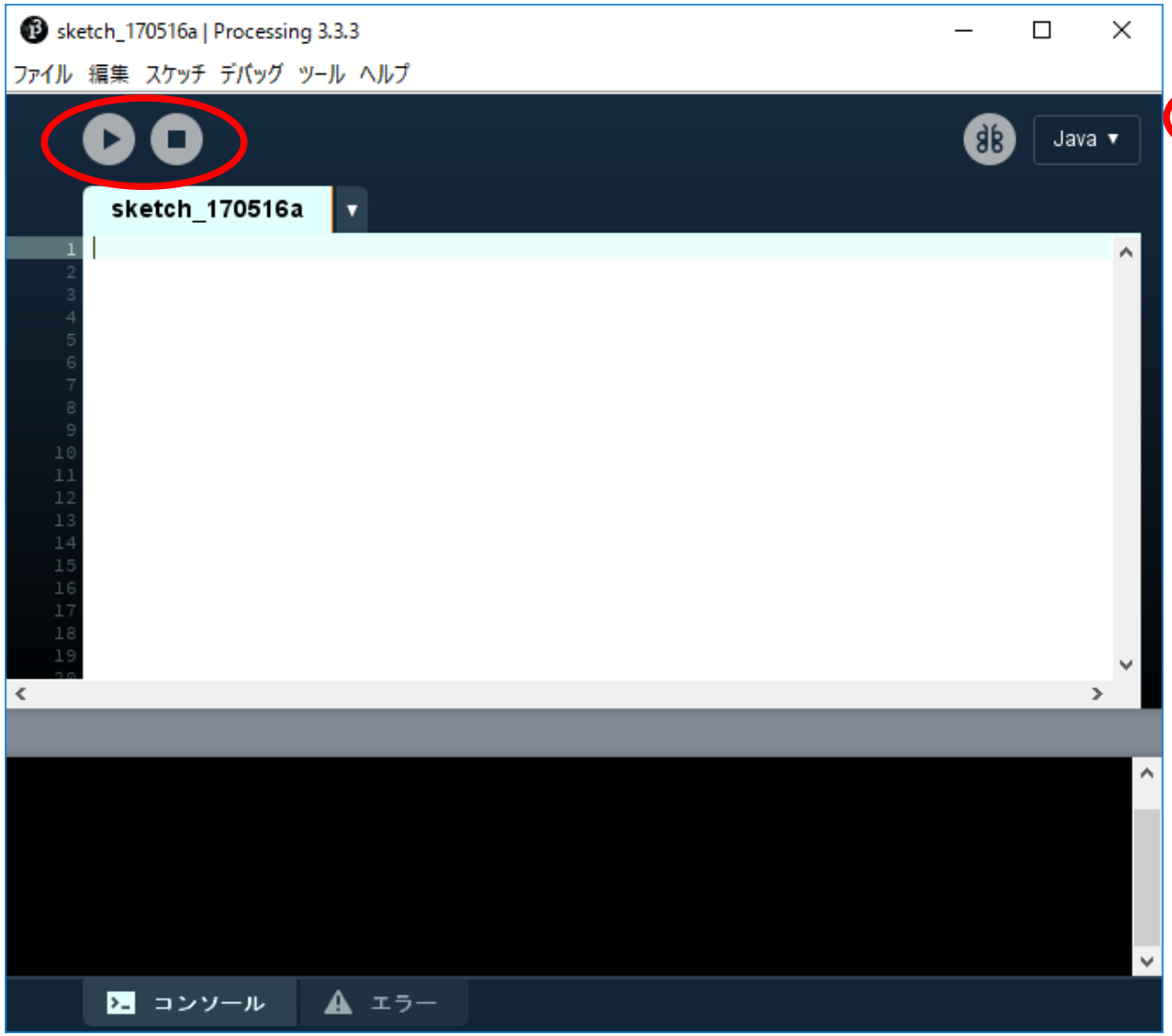
プログラムって何？

- コンピュータにやってほしいことを記述したもの
  - これ計算してほしい！
  - 画面にこういうものを表示してほしい！
  - このファイルを探してきてほしい！

などなど・・・

まずはプログラムを作るために使うエディタと
簡単なプログラムの例を見てみよう

プログラムの作成に使うエディタ



プログラムの実行と停止

ここにプログラムを書く

ここに文字（実行結果）が出力される

図：使用するエディタ

画面に文字を表示するプログラムの例


```
void setup()
{
        println(“こんにちは。”);
        println(“さようなら。”);
}
```

- このプログラムを実行すると，以下のように画面に文字が表示される．
  こんにちは。
  さようなら。
- 次は，プログラムの書き方の基本ルールを確認する．

（※printlnは文字を表示する命令．後から演習をする）

画面に文字を表示するプログラムの例


```
void setup()
{
        println(“こんにちは。”);
        println(“さようなら。”);
}
```

- void setup(){}　の中括弧の中に処理（文）を書く.
- 文は上から順番に実行される.
- ;（セミコロン）は文の終わりを表す.
- 基本的に使っていいのは半角英数字だけ.
  - ただし，“”で囲んだ部分は文字列として扱われ、全角文字を使ってもOK.

文のコメントアウト


```
void setup()
{
        //println(“こんにちは。”);
        println(“さようなら。”);
}
```

- 「//」の後ろに書かれている文は無視される（コメントアウトという）
  つまり，上のプログラムを実行すると，1つ目の文は無視されることになり，
  出力は以下のようになる．

        さようなら。

演習（その１）

1. 先ほどのサンプルコードをエディタに入力し、実行してみよう.
    1. setupの部分から確実に入れよう
2. “”を外したらエラーが出ることを確認しよう.
3. ;を外したらエラーが出ることを確認しよう.
4. サンプルコードの一部をコメントアウトして、コメントアウトした処理が行われなくなることを確認しよう.

文字の表示（println関数）

- 画面に文字を表示させるときにはprintln関数を使う.
  （関数についての説明は後述）
- println()の括弧の中に、表示させたい文字列、数値を記述する.

```
void setup()
{
        println(“Hello, World!”);    // 「“”」を忘れないように注意
        println(3.141592);           // 数値は「“”」をつけなくてもOK
}
```

演習（その２）

1. println関数を使って「Hello, World!」以外の文字を出力してみよう

データの種類（型）

- データには様々な種類（型）がある.
  - 整数　　0, 1, -1など
  - 実数　　1.1, 3.14, -0.5など
  - 文字列　　“あいうえお”, ”abcde”など
  - 真偽値　　true, false

- それぞれのデータの型には名前がついている.
  - 整数  int
  - 実数  double
  - 文字列  String
  - 真偽値  boolean
- 次は、データの入れ物（変数）について理解しよう.

変数と宣言

- 変数とは，データの入れ物のこと．
- データと同じく，変数にも型という概念がある．
- 変数の型に応じて，その変数に入れられるデータも決まる．
  - ※int型の変数にはint型のデータしか入らない
- 変数には名前をつけなければならない．
- 変数を使うには，使う前にどこかで変数の使用宣言をしなければならない．
  - 「変数の型 変数名」で宣言が出来る．

```
void setup()
{
        int x;  // int型の変数「x」の宣言
}
```

変数へのデータの代入１

- 変数にデータを入れることを代入という．
- 代入には「=」を使う．
  - 左辺の変数に右辺のデータが入る
  - 数式の，「等価である」の意味ではないので注意！等価であるは「＝＝」で書く．後に登場．
- 宣言したばかりの変数の中身は不定となるため，宣言後には極力適切なデータを入れるよう心がけること．（変数の初期化）

```
void setup()
{
        int x;          // int型の変数「x」を宣言（何が入っているかわからない）
        x = 10;         // int型のデータ「10」を変数「x」に代入
        println(x);     // xの中身を表示（10が出力される）
}
```

変数へのデータの代入２

- 変数の宣言と同時に代入をすることもできる.
- 「=」の右辺に変数を使ってもよい.

```
void setup()
{
        int x = 10;    // int型の変数「x」を宣言するとともに、10で初期化
        int y = 20;
        println(x); println(y);
        x = y;         // xにyを代入（xの中身が20になる）
        println(x); println(y);
        x = 30;        // xの中身が30になる（yの中身は20のまま）
        println(x); println(y);
}
```

演習（その３）

1. int型の変数に整数を代入し、printlnしてみよう.
2. double型の変数に小数を代入し、printlnしてみよう.
3. String型の変数に文字列を代入し、printlnしてみよう.
4. boolean型の変数に真偽値を代入し、printlnしてみよう.
5. int型の変数に小数や文字列を代入できるか確認してみよう.

演習（その３）　解答例

1. int i = 10;
   println(i);
2. double d = 3.14;
   println(d);
3. String s = “はろーわーると”;
   println(s);
4. boolean b = false;
   println(b);
5. 変数とデータの型が一致しないとエラーが出る

変数（注意書き）

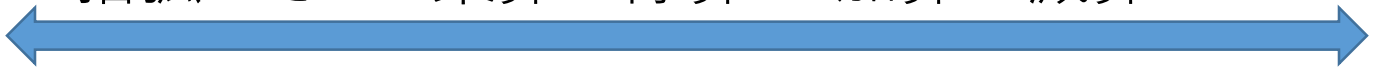
- int：整数
  - int a = -1;
  - int a = １; //全角はだめ
- float：実数
  - float b = -1.2;
- String：文字列
  - String c = “aaaa”;
  - String c= “1.2”; //これも文字列
- Boolean：真偽
  - Boolean d = true;

四則演算

- 四則演算を行う際には、それぞれに対応する演算子を使用する：
  - 加算：+
  - 減算：-
  - 乗算：*
  - 除算：/
- 括弧をつけることで演算の優先順序を制御できる.

優先度高　　優先度低

括弧つき　＞乗算・除算＞加算・減算



```
void setup()
{
        int x = 5 * 6 + 10 / 2 - (3 + 6);    // 30 + 5 – 9 となり、xには26が入る
}
```

四則演算の備考

- データの型によって計算結果に違いが出ることがあるので注意しよう.
  - 例えば、3/2の出力は1になるが、3.0/2ならば出力は1.5になる.
  - 計算結果が小数になりそうな場合はdouble型を使おう.

- 一部の計算は簡略化した書き方ができる. （以下はint型の変数「i」の例）
  - i = i + 5 ⟷ i+=5
  - i = i – 10 ⟷ i-=10
  - i = i * 3 ⟷ i*=3
  - i = i / 2 ⟷ i/=2

  特に、変数に1加算する、もしくは1減算する時は以下のように書ける.
  - i = i + 1 ⟷ i+=1 ⟷ i++
  - i = i – 1 ⟷ i-=1 ⟷ i--

+を用いた文字列の結合

- 「+」を使うと、文字列同士の結合が出来る.
  - 「-」、「*」、「/」は文字列操作に使えない.

```
void setup()
{
        String str1 = “ピカ”;
        String str2 = “ライ”;
        String str3 = “チュウ”;
        String str4 = str1 + str3;  // “ピカ” + “チュウ”
        String str5 = str2 + str3;  // “ライ” + “チュウ”
}
```

演習（その４）

1. 以下の計算結果を変数に代入し、printlnで確認してみよう.
    1. 5 + 10
    2. 5 + 10.2
    3. 7 – 3.5
    4. 4 × 5.5
    5. 3 ÷ 2
    6. 3.0 ÷ 2
    7. (1 + 2) × 3 + 10
    8. “5月” + “22日” + “月曜日”

2. 以下のコードの3行目を簡略化した書き方で書いてみよう

```
int x = 10;
int y = 20;
x = x + y;
println(x);
```

演習（その４）　解答例

1. 省略

2.
```
int x = 10;
int y = 20;
x += y;
println(x);
```

関数

- 関数とは，文をひとまとめにし，それに名前をつけたもの．
  - 今までに出てきたsetupやprintlnも関数である

- 関数は，他の関数から呼び出すことができる．
  - 今まではsetup関数からprintln関数を呼び出していた
  - println関数は偉い人が予め作っておいてくれた

- 関数では，以下のようなことをやっている．
  1. 呼び出し元の関数からデータを受け取る
  2. 受け取ったデータを使って何らかの処理を行う
  3. 処理の結果にしたがって、呼び出し元の関数にデータを返す

- 何度も使うような処理を関数として作成（定義）しておくことで，文の量を少なくなり，また管理もしやすくなる．

関数の構成

- 関数は，以下の4要素から構成されている：
  1. 呼び出し元の関数に返すデータ（戻り値という）の型
  2. 関数名
  3. 呼び出し元の関数から受け取るデータ（引数という）の型と仮の名前
  4. 処理内容
- 2つのint型のデータを足し合わせ，その結果を返す関数addは以下のようになる．

```
int add(int a, int b)
{
        int sum;                // 関数内でint型の変数sumを宣言
        sum = a + b;            // sumに引数の和を代入
        return sum;             // sumの中身を呼び出し元の関数に返す
}
```

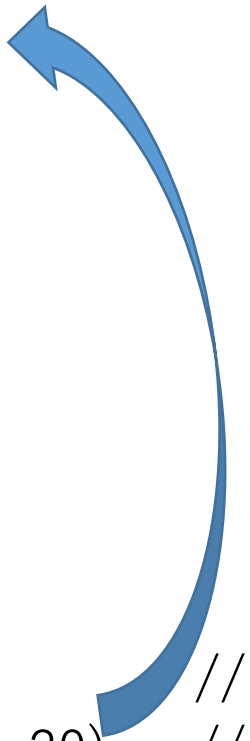
関数の定義方法

- 関数の定義は，関数の外側で行う．
- 値を返す時はreturn文を使う．（返す値を戻り値と呼ぶ）
  - 「return 戻り値」で返せる（先ほどの例ではsumを返した）
  - return文を実行すると呼び出し元の関数に処理が戻る
- 戻り値は返さなくてもよいが、多くとも１つしか返せない．
  - 戻り値を返さない時は、戻り値の型を「void」とする．
- 引数は複数個受け取ることが出来る．（受け取らない場合は空欄にする）
  - 複数個指定する時はカンマで区切る

```
int  add ( int a, int b )
{
        int sum;                    // 関数内でint型の変数sumを宣言
        sum = a + b;                // sumに引数の和を代入
        return sum;                 // sumの中身を呼び出し元の関数に返す
}
```

関数の呼び出し

- 関数を呼び出したい時は，呼び出す関数の名前と引数の値を記述する．
  - 関数名(引数1, 引数2, …)



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

void setup()
{
        int result;                 // 計算結果を入れるための変数を宣言
        result = add(10, 20);       // addを呼び出し10と20を引数として渡す　戻り値はresultへ代入
        println(result);            // resultには30が入っている
}
```

演習（その５）

画面に文字を出力する関数を自分で定義してみよう

1. int型のデータを戻り値とし、常に1を返すように作る
2. 引数として文字列を１つ受け取る
3. 自作の関数内でprintln関数を呼び出し、引数として受け取った文字列を表示する
4. 定義できたら、setup関数から自作の関数を呼び出して表示がされるか確認する
(引数は文字列であれば”Hello, World!”でもなんでもOK)
5. 正しく動作することが確認出来たらいろんな文字列を入れて表示を変えてみる

演習の解答例


```
int myPrint(String str)
{
        println(str);           // 引数strの中身を表示
        return 1;               // 呼び出し元に1を返す
}
void setup()
{
        myPrint(“Hello, World!”);  //  「Hello, World!」と表示される
        String s = “おはよう！”;
        myPrint(s);                     // 引数に変数を渡してもOK
}
```

数値の大小比較

- 数値の大小比較の際には等号「=」と不等号「<, >」を用いる.
  - 等しい　：　==
  - 大なり　：　>
  - 小なり　：　<
  - 大なりイコール　：　>=
  - 小なりイコール　：　<=

- 等しいことを示す時は「イコール２つ」なので注意しよう.

- 比較結果はboolean型で返される.

```
int x = 10;
boolean result;
result = x > 5;　// 比較「x>5」の結果をresultに代入（この場合はtrue）
```

条件分岐（if文）

- 条件に応じて処理の方法を変えたい時にはif文を使う

```
if(条件式)
{
        //  条件式がtrueの時に行う処理
}
else
{
        // 条件式がfalseの時に行う処理
}
```

- 条件式の部分にはBoolean型のデータが入る
- 条件式がfalseの時にやるべき処理が無ければelse以下は省略しても構わない
- else以下にさらにif文を入れることもできる → else if(条件式)

if文の使用例

- int型の変数xとyを引数とし、大小関係を判定する関数を示す

```
void compare(int x, int y)          // 戻り値はなし
{
        if(x > y)
        {
                println(“xはyよりも大きいです。”);      // 「x > y」がtrueの場合
        }
        else
        {
                println(“xはy以下です。”);              // 「x > y」がfalseの場合
        }
}
```

演習（その６）

1. int型の変数2つを引数とし，それらが等しいか否かを判定する関数を作ってみよう．
2. double型の変数2つを引数とし，大きい方の値を出力する関数を作ってみよう．（等しい時は「2つは等しいです」と表示させてみよう）

演習（その６）　解答例

1.
```
void compare(int x, int y)
{
        if(x == y)
        {
                println(“等しいです。”);
        }
        else
        {
                println(“等しくありません。”);
        }
}
void setup()
{
        compare(5, 5);        // 「等しいです。」と表示される
}
```

演習（その６）　解答例

2.
```
void printBigger(double a, double b)
{
        if(a > b)
        {
                println(a);    // 「a > b」がtrueの時はaを表示
        }
        else if(a < b)
        {
                println(b);    // 「a < b」がtrueの時はbを表示
        }
        else                   // 「a > b」と「a < b」が共にfalseの時
        {
                println(“2つは等しいです。”);
        }
}
```

論理演算

- 2つ以上の真偽値を組み合わせたい時には論理演算子を使う

- aとbはそれぞれBoolean型のデータとした時：
  - a かつ b　：　a && b
  - aまたはb　：　a || b

- a && bは両方ともtrueである時にのみ出力がtrueとなる
- a || bは少なくともどちらか一方がtrueであれば出力がtrueとなる

論理演算を用いたプログラムの例

- int型のデータが1桁の正の整数か否かを判定する関数

```
void checkDigit(int x)
{
        if(x >= 1 && x <= 9)    // xが1以上　かつ　xが9以下　のとき
        {
                println(“1桁の正の整数です。”)
        }
        else                            // それ以外の時
        {
                println(“1桁の正の整数ではありません。”);
        }
}
```

演習（その７）

1. int型のデータを引数に取り、それが3桁の正の整数であるかどうかを判定する関数を作ってみよう（論理演算子&&を使うこと）

演習（その７）　解答例

1.
```
void checkDigit(int x)
{
        if(x >= 100 && x <= 999)
        {
                println(“3桁の正の整数です。”)
        }
        else
        {
                println(“3桁の正の整数ではありません。”);
        }
}
```

処理の繰り返し（for文）

- プログラムの中で，同じような処理・似たような処理を何度も繰り返したい時はfor文，もしくはwhile文を使う．

- for文は以下のように使う：

```
for(前処理; 制御式; 後始末)
{
    // ここに繰り返したい処理を書く
}
```

  - 前処理には繰り返し処理を始める前に行う処理を書く．
  - 制御式にはループを繰り返す条件を書く．
  - 後始末には１ループする度に行う処理を書く．

for文の使用例

- 以下に「Hello, World!」を3回出力するプログラムを示す.

```
void setup()
{
	for(int i = 0; i < 3; i++)
	{
		println(“Hello, World!”);
	}
}
```

- （前処理）最初のループに入る前にint型の変数iを宣言し、0で初期化
- （制御式）iが3よりも小さいうちはループし続ける
- （後始末）ループが一周する度にiに1を足す
- 3周し終わるとiが3になるため、ループから抜ける

処理の繰り返し（while文）

- while文はfor文と書き方は異なるが，繰り返し処理のために使うという点は同じ
- while文は以下のように使う：

```
while(true)
{
	// ここに繰り返したい処理を書く
	if(ループの終了条件)
	{
		break;
	}
}
```

- break文を使うとループから抜けることができる．
- 上の例では，if文の条件式が満たされた時にループから抜ける．

while文の使用例

- 1から10までの正の整数の総和を求めるプログラムを以下に示す.

```
void setup()
{
	int i = 1;			// iを1で初期化
	int sum = 0;			// sumを0で初期化

	while(true)
	{
		sum += i;		// sumにiを加算
		i++;			// iを1増加

		if(i > 10)		// iが10よりも大きくなったらループから抜ける
		{
			break;
		}
	}

	println(sum);			// 総和を出力（55と表示される）
}
```

演習（その８）

1. for文による繰り返し処理を書いてみよう.
    1. ”Hello, World!”を5回繰り返すプログラムを書いてみよう.
    2. 制御式を変えて、繰り返しの回数を変えてみよう.
    3. 変数の初期値を変えて、繰り返しの回数を変えてみよう.

2. While文による繰り返し処理を書いてみよう.
    1. 2の10乗を計算するプログラムを書いてみよう.
    2. break文を実行する条件を変えてみよう.

演習（その８）

1.
```
void setup()
{
        for(int i = 0; i < 5; i++)
        {
                println(“Hello, World!”);
        }
}
```

演習（その８）

2.
```
void setup()
{
        int product = 1;          // 積
        int count = 0;            // かけた回数を数えるための変数

        while(true)
        {
                product *= 2      // productに2をかける
                count++;          // countを1増加する

                if(count == 10)           // かけた回数が10になったらループから抜ける
                {
                        break;
                }
        }

        println(product);         // 結果を出力する（1024と表示される）;
}
```

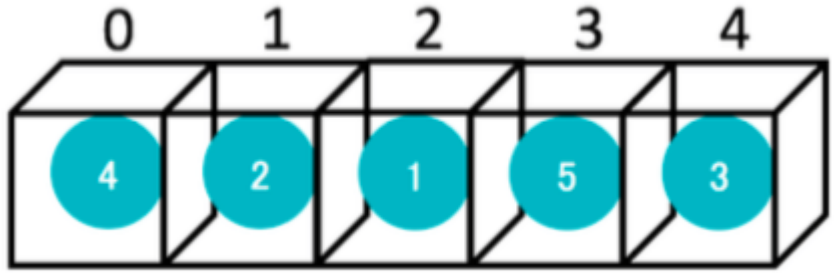
配列とは

- 配列とは，複数の同じ型の変数から連続して集まった変数のまとまり
- 配列の宣言は以下のように行う（型の部分には同じものを入れること）

型[] 配列名 = new 型[要素数];

- 例えば，int型の変数5個を要素に持つ配列arrayを宣言する時は以下のように書く．

int[] array = new int[5];



図：int array[5]で宣言した配列のイメージ
int型用の5つの入れ物があり，それぞれにデータを入れておくことができる

配列の添え字（インデックス）

- 配列の各要素には，添え字（インデックス）と呼ばれる番号が割り当てられており，先頭から順に0,1,2,3,4,…となっている．（全て整数）
- 配列名とインデックスを組み合わせることで、配列の各要素へアクセスすることができる．

```
int[] array = new array[5];        // 要素数5のint型配列arrayの宣言
array[0] = 10;                     // 先頭の要素に10を代入
array[1] = 20;                     // 2番目の要素に20を代入
array[2] = 25;                     // 3番目の要素に25を代入
array[3] = 5;                      // 4番目の要素に5を代入
array[4] = 90;                     // 5番目の要素に90を代入
```

配列とfor文の組み合わせ

- int型の変数100個に対して，偶数を昇順に代入していきたい場合を想定する.
- 以下のように，配列とfor文を組み合わせることで少ない文量で実現できる.

```
int[] array = new int[100]; // 要素数100のint型配列を宣言

for(int i = 0; i < 100; i++)        // ループを100回繰り返す
{
        array[i] = i * 2;           // iでインデックスの指定と偶数の算出
}
```

演習（その９）

1. 要素数10のdouble型の配列を宣言してみよう.
2. 宣言した配列にデータを代入してみよう.
3. データが代入されているか、printlnを使って調べてみよう.
4. インデックスに整数以外が使えないことを確認しよう.

演習（その９）

1.
```
double[] d = new double[10];
```

2.
```
for(int i = 0; i < 10; i++)
{
    d[i] = i*10;
}
```

3.
```
for(int i = 0; i < 10; i++)
{
    println(d[i]);
}
```

4. 省略

import文

- import文とは、他の人が作った関数を使えるようにする宣言である
  - 既に他の人が作っているような関数であれば、自分で作らずに楽できる
- import文は以下のように使う
  - 「import 読み込みたいファイルの場所」
  - 関数が集められたファイルのことをライブラリという
  - 読み込んだライブラリにある関数がプログラムの中で使えるようになる

- プログラム内でライブラリをインポートせずに、開発環境の設定をすることで、複数のファイルを関連付け（管理）する方法もある

よくある間違い

- 同じ関数の名前を２つ作ることができません.