

この教材は『手を動かしながらやさしく学べるはじめてのAIデータサイエンスリテラシー』（技術評論社）の実践編として本書のv～viiページで紹介している発展的な実践教材です。本書で学んだことを活かして、発展的で面白いプロジェクトに挑戦できます。PDFで提供しており、Wolframノートブックでの配布はありません。必要に応じて印刷し、Wolframノートブックに自分でコードを入力して実行してください。

なお、ここでは、本書独自のフォーマットを使い、プログラムの入力部分にオレンジの枠、出力部分にブルーの枠をつけてプログラムの部分をわかりやすく表示しています。新規でノートブックを開いた時にはこれらの色枠は付きません。

実践編1：コンピュータの基本 ～身のまわりの情報を0と1で表現してみよう～

コンピュータは、電気のONとOFFというしくみで動作しているため、すべての処理を2進数（0と1）で行います。この0と1を使って、テキストや画像など、身のまわりの情報を白と黒のデジタル形式で表現してみよう。

ドット絵を描いてみよう

みなさんは、ドット絵をご存知ですか？ドット絵とは、コンピューター上で四角い点をマス目状に並べて表現した絵のことです。1980年代から1990年代のコンピュータやゲーム機では、一度に扱うことができる情報量が少なかったため、画像やキャラクターは限られた数の点（ドット）で表現されることになり、出力画面はドット絵のように見えました。ドット絵は、コンピュータの基本そのものです。なぜならば、コンピュータはすべてのデータを「0」と「1」の組み合わせで処理しており、ドット絵もまた、画面上の小さな点（ピクセル）を「白=0, 黒=1」のように0と1で表現できるためです。



Wolfram言語の`ArrayPlot`関数を使い、64ビット（8バイト）の情報量をもつ白黒のドット絵を描いてみましょう。`ArrayPlot`関数は、リストの0と1の数字を「白=0, 黒=1」で出力します。64は8×8なので、0か1のどちらかの値を8個もつリスト（例 {0,0,0,0,0,0,0,0}）を8つ作ると、8x8のマス目を作ることができます。まず、値が全ての0の8つのリストを`ArrayPlot`関数に与えて、表示してみましょう。なお、第2引数に `Mesh→True` を与えることによって、マス目を四角で表示することができます。

In[1]:=

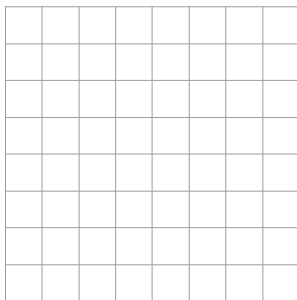
ArrayPlot[

配列プロット

```
{ {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0} }, Mesh → True]
```


Out[1]=



8×8の白のマス目が表示されました。では、この中に丸を描くには、どの0を1にすればよいでしょうか？上記の`ArrayPlot`関数のプログラムをコピーペーストし、黒くしたい位置の0を1に変更して、丸を描いて下さい。

In[2]:=

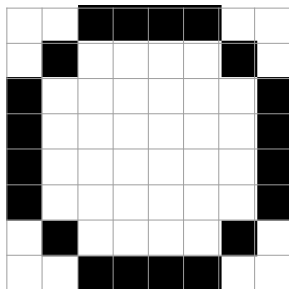
ArrayPlot[

配列プロット

```
{ {0, 0, 1, 1, 1, 1, 0, 0}, {0, 1, 0, 0, 0, 0, 1, 0}, {1, 0, 0, 0, 0, 0, 0, 1},
  {1, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 0, 0, 0, 0, 1},
  {0, 1, 0, 0, 0, 0, 1, 0}, {0, 0, 1, 1, 1, 1, 0, 0}}, Mesh → True]
```




Out[2]=



上記では、最初の行は、{0,0,1,1,1,1,0,0}、つまり、白白黒黒黒黒白白とし、次は、{0,1,0,0,0,0,1,0}、次は{1,0,0,0,0,0,0,1}として円を描いていますが、いろいろなやり方があると思います。皆さんも丸のドット絵が描けましたか？

次は三角形に挑戦です。完全な三角形を描くのは難しいので、三角っぽく見えたら良しとしましょう。

以下は、三角のドット絵のプログラム例です。第3引数に**ColorRules->{1→Red,0→White}**を与えて、ドットを赤色で表示しています。

In[3]:=

ArrayPlot[{{0, 0, 0, 1, 1, 0, 0, 0}, {0, 0, 1, 1, 1, 1, 0, 0},

配列プロット

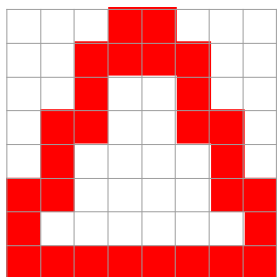
```
{0, 0, 1, 0, 0, 1, 0, 0}, {0, 1, 1, 0, 0, 1, 1, 0}, {0, 1, 0, 0, 0, 0, 1, 0},
  {1, 1, 0, 0, 0, 0, 1, 1}, {1, 0, 0, 0, 0, 0, 0, 1}, {1, 1, 1, 1, 1, 1, 1, 1}},
  Mesh → True, ColorRules → {1 → Red, 0 → White}]
```







Out[3]=



練習問題(1)：

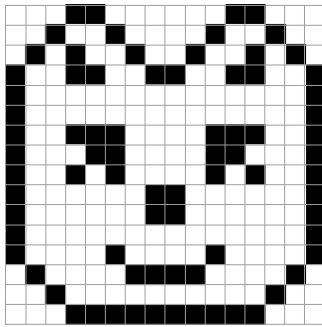
上記の例にならって、8×8のマス目でバツ印や四角形などの形を描いてみましょう。また、**ColorRules**の値を変更して、線の色や背景の色を変えてみましょう。

マス目の数を増やすと、さらに複雑な絵を描くこともできます。以下は、16×16のマス目を使い、猫を描いてみました。リストを変数`code`に代入し、`ArrayPlot`関数に`code`を与えることでドット絵を描画しています。猫に見えますか？

```
In[4]= code = {{0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
  {0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0}, {0, 1, 0, 1, 0, 0, 1, 0,
  0, 1, 0, 0, 1, 0, 1, 0}, {1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1},
  {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1},
  {1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1}, {1, 0, 0, 1, 0, 1, 0, 0,
  0, 0, 1, 0, 1, 0, 0, 1}, {1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1},
  {1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 0, 0, 0, 1, 0, 0,
  0, 0, 1, 0, 0, 0, 0, 1}, {0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 1, 0, 0}, {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0}};
```

```
In[5]= ArrayPlot[code, Mesh → True]
配列プロット 真
```

Out[5]=



第3章で説明したように、我々が見る高精細のデジタル画像も、実はこのドット絵と同じしくみでできています。違うのは、ドット（ピクセル）の数が格段に多いこと、かつ、ドットの色が白と黒の0,1ではなく、色の三原色（赤青黄）を使っているという点だけです。

1つ1つの小さな点に色の情報が記録され、それが集まって私たちが見ている写真や映像になっています。ドット絵の考え方を理解することは、デジタル画像の仕組みを知る一歩といえます。

文字コードをドットで表してみよう

では、次に、コンピュータに格納されている文字コードをドットで表すとどうなるか試してみましょう。第6章では、Wの文字はコンピュータ内部では2進数の7バイトで表現されていることを学びました。この数字を **ArrayPlot** 関数を使って、「白=0、黒=1」のオンオフで表してみましょう。

“W”の文字を2進数で表すには、まず **ToCharacterCode**["W"] を使って10進数で表し、さらにその値(87)を **IntegerDigits** 関数で2進数に変換しました(本書P84を参照)。この処理を1行のプログラムで書くと以下ようになります。

```
In[6]:= codeW = IntegerDigits[ToCharacterCode["W"], 2]
```

数の各桁の数字 整数コードへ

```
Out[6]:= {{1, 0, 1, 0, 1, 1, 1}}
```

関数はこのように入れ子構造が可能です。上記では、この結果を **codeW** という変数に代入しました。

この0,1からなるリストを「白=0、黒=1」のオンオフで表してみましょう。
codeW を **ArrayPlot** 関数に与えて結果を出力します。

```
In[7]:= ArrayPlot[codeW, Mesh → True]
```

配列プロット 真

```
Out[7]=
```



では、“Wolfram”という文字列についても同様に結果を出力してみましょう。今度は出力結果を毎回確認するために、順を追って実行します。まず、“Wolfram”を **ToCharacterCode** 関数で10進数に変換し、それを変数 **code10** に代入します。

```
In[8]:= code10 = ToCharacterCode["Wolfram"]
```

整数コードへ

```
Out[8]:= {87, 111, 108, 102, 114, 97, 109}
```

さらに、**IntegerDigits** 関数を使って2進数に変換します。“Wolfram”の先頭文字“W”のところは、さきほどと同じ、1010111となっていますね。

```
In[9]:= code2 = IntegerDigits[code10, 2]
```

数の各桁の数字

```
Out[9]:= {{1, 0, 1, 0, 1, 1, 1}, {1, 1, 0, 1, 1, 1, 1},
{1, 1, 0, 1, 1, 0, 0}, {1, 1, 0, 0, 1, 1, 0},
{1, 1, 1, 0, 0, 1, 0}, {1, 1, 0, 0, 0, 0, 1}, {1, 1, 0, 1, 1, 0, 1}}
```

それでは、**ArrayPlot** 関数を使って **code2** を「白=0、黒=1」のオンオフ（白黒のデジタル形式）で表現してみましょう。

In[10]:=

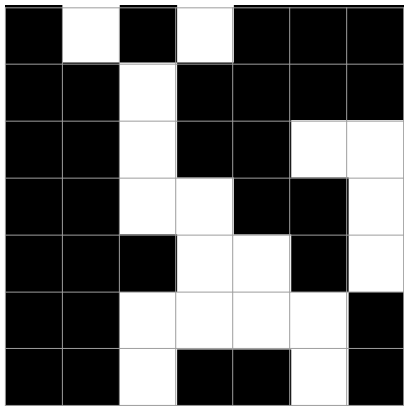
ArrayPlot[code2, Mesh → True]

配列プロット

[×…]

[真]

Out[10]=



上の結果は、上から1行目が1010111で”W”、2行目が1101111で”o”、というように2進数のオンオフを黒と白のマス目で表現しています。

ところで、このような白と黒のマス目の表現をどこかで見たことはありませんか？ーそう、QRコードです。QRコードは、Webページへの誘導やスマホのQRコード電子決済、コンサートの電子チケットなど、みなさんの身近な場面でもよく使われています。QRコードの原理は、先ほどの”Wolfram”を白と黒のマス目で表した仕組みと同じです。文字や数字の情報を2進数に変換し、「白=0, 黒=1」のパターンで表現しています。あらかじめ決められたルールに従ってマス目に配置されているため、コンピュータが正しく情報を読み取ることができます。

ここで紹介した**ArrayPlot**関数での出力結果やQRコードは、情報を「0と1」の形に置き換えるしくみを使ってコンピュータで視覚的に表現したものです。スマホの文字や写真、ゲームのアニメーションなど私たちが日ごろ目にするさまざまな情報は『コンピュータの内部ではすべて「0と1」で表現されている』ということが実感できたのではないのでしょうか。

練習問題(2)：

”W”を白黒のデジタル形式で表すプログラムを1行で書いて、実行してみましょう。また、”Wolfram”にについても白黒のデジタル形式であらわすプログラムを1行で書いてみましょう。

練習問題(3)：

自分の名前のアルファベット表記を上記のような黒白のタイルを使った形式で表現してみましょう。