マニュアルに戻る。

はじめてのプログラミング

ドリトルのプログラムを作ってみよう。プログラムは**編集画面**に入力し、「実行！」と書かれた**実行ボタン**を押すことで実行できる[1)]。編集画面と実行画面は、Ctrl-Tで切り替えられる（MacではCtrlの代りにCommandキーを使用する） &show(edit.png,nolink,画像の説明,40%); &show(run.png,nolink,画像の説明,40%);

オブジェクトを作る

ドリトルでは、**オブジェクト**を作り、それに**命令**を送る形でプログラムを動作させる。ここでは**タートル**オブジェクトを使い、**タートルグラフィックス**を扱う。タートルオブジェクトは標準ではカメの姿をしており、歩いた軌跡が線として残る。そこで、画面上を動かすことで絵を描くことができる。 タートルオブジェクトを作るには、**タートル**という名前の**オブジェクト**に**作る**という命令を送る。文の最後には「。」を書く。

```
タートル！作る。
```

&show(kameta.png,nolink,画像の説明);

実行すると、画面にタートルオブジェクトが表示される。ドリトルではオブジェクトを複製する形で新しいオブジェクトを生成する。そのために、**プロトタイプオブジェクト**と呼ばれるひな形となるためのオブジェクトが用意されている。プロトタイプオブジェクトは画面上には表示されないが、生成したすべてのオブジェクトの親の役割をする重要なオブジェクトである。 この例では、プロトタイプオブジェクトである「タートル」に「作る」を送ることで新しいタートルオブジェクトを生成した。

オブジェクトに名前を付ける

ドリトルでは、オブジェクトを指定して命令を送る。先ほど作ったプログラムでは、生成したオブジェクトに**名前**を付けていないため、画面に表示されたオブジェクトに命令を送ることはできない。オブジェクトに名前を付けるには、

> 名前 = **オブジェクト**。

のように、**=** の左辺に名前を書き、右辺にオブジェクトを指定する。

```
かめた＝タートル！作る。
```

&show(kameta.png,nolink,画像の説明);

実行すると、先ほどの例と同じく画面にタートルオブジェクトが表示される。見かけは変わらないが、このタートルオブジェクトには「かめた」という名前が付いている。

名前は**変数**と呼ばれることがある。名前には英字と数字のほかに、仮名や漢字などの日本語文字を使うことができる。ただし、名前の先頭に数字を使うことはできず、名前の中に空白や記号を含めることはできない。また、**タートル**といった最初から用意されている名前を使わないように注意する必要がある。

名前が示すオブジェクトは常に1個である。複数のオブジェクトに同じ名前を付けた場合には、最後のオブジェクトだけに名前が残り、他のオブジェクトは名前がなくなってしまう。

オブジェクトに命令する

ドリトルでは、オブジェクトに命令を送ることでプログラムを実行する。命令を送るオブジェクトは!で指定し、その右側に送る命令を書く。たとえば、

> かめた！100　歩く。

は、「かめた」に対して「100　歩く」という命令を送っている。ここで「歩く」は命令である。「100」は命令とともに送られる値であり、**パラメータ**と呼ばれる。数値に「100歩」のように単位を付けると、単位は無視されて数字の部分だけが使われる。パラメータと命令の間は**空白**で区切る必要がある。

オブジェクトが理解できる命令はオブジェクトの種類ごとに決まっている。オブジェクトの種類と命令はレファレンスにまとめられている。次の表はタートルが理解できる命令の一部である。命令は**メソッド**とも呼ばれる。自分で命令を追加することもできる。この方法は、後の章で扱う。

命令	用途	使用例
作る	オブジェクトを作る	かめた＝タートル！作る。
歩く	前進する	かめた！100　歩く。
右回り	右に回る	かめた！90　右回り。
左回り	左に回る	かめた！90　左回り。

命令を順に実行する（1）

プログラムは上から順に実行されていく。そこで、複数の命令を順に実行したいときは、上から1行ずつ書いていけばよい。実際に書いてみよう。

まず、1行書いて実行する。画面にはカメが表示される。

```
かめた＝タートル！作る。
```

&show(kameta.png,nolink,画像の説明);

1行追加して実行する。画面のカメが前に歩く。

```
かめた＝タートル！作る。
かめた！100　歩く。
```

&show(fd100.png,nolink,画像の説明);

もう1行追加して実行する。カメは歩いた後に左を向く。

```
かめた＝タートル！作る。
かめた！100　歩く。
かめた！90　左回り。
```

&show(fd100lt90.png,nolink,画像の説明);

もう1行追加して実行する。カメは歩いてから左を向いてさらに歩く。

```
かめた＝タートル！作る。
かめた！100　歩く。
```

```
かめた！90　左回り。
かめた！100　歩く。
```

&show(fd100lt90fd100.png,nolink,画像の説明);

命令を順に実行する（2）

オブジェクトは命令を実行すると、結果としてオブジェクトを返す。たとえば、

> かめた＝タートル！作る。

というプログラムは、「タートル！作る」という命令を実行した結果、新しいタートルオブジェクトが返される。

この文では、生成された新しいオブジェクトに「かめた」という名前を付けていた。このように、新しいオブジェクトを生成したり数式を計算したりする場合は、返された結果の値に名前を付けたり画面に表示したりして活用することができる。

一方、タートルを動かすような命令では、元のオブジェクトがそのまま返されることが多い。そこで、

> かめた！100　歩く　90　右回り。

のように命令を並べて書くことで、ひとつのオブジェクトに続けて複数の命令を送り、実行することができる。このように、命令を実行した結果のオブジェクトに続けて命令を送ることを**カスケード**という。ここで、「90　右回り」は、「かめた！100　歩く」から返されたオブジェクトに送られている。

たとえば、

> タートル！作る　100　歩く。

では、「タートル！作る」から返されたオブジェクトに対して「100　歩く」が送られる。つまり、新しく作られたオブジェクトが100歩歩くことになる。

次のプログラムは、1行に1命令ずつ書いたプログラムである。内容の割にすぐに行数が長くなってしまい無駄が多い。

```
かめた＝タートル！作る。
かめた！100　歩く。
かめた！90　左回り。
かめた！100　歩く。
かめた！90　左回り。
かめた！100　歩く。
かめた！90　左回り。
かめた！100　歩く。
かめた！90　左回り。
```

&show(square.png,nolink,画像の説明);

次のプログラムは、できるだけ続けて書いたプログラムである。全体で2行と短くなったが、横に長くなり、分かりづらくなってしまった。

```
かめた＝タートル！作る。
かめた！100　歩く　90　左回り　100　歩く　90　左回り　100　歩く　90　左回り　100　歩く　90　左回り。
```

&show(square.png,nolink,画像の説明);

次のプログラムは、動作ごとにまとめて書いたプログラムである。このように、ひとまとまりの動作を1行にまとめて書くと分かりやすくなる。

```
かめた＝タートル！作る。
かめた！100　歩く　90　左回り。
かめた！100　歩く　90　左回り。
かめた！100　歩く　90　左回り。
かめた！100　歩く　90　左回り。
```

&show(square.png,nolink,画像の説明);

繰り返し

コンピュータは命令を高速に実行し、大量の命令を実行しても繰り返すことに疲れたり飽きたりすることはない。そこで、ある処理を何度も繰り返すプログラムはよく使われている。先ほどのプログラムにも、同じ記述が何度も出てきていた。

ドリトルでは、プログラムの一部を「…」で囲むことで、**ブロックオブジェクト**として扱うことができる。たとえば、

> 「かめた！100　歩く　90　左回り」

はブロックオブジェクトである。ブロックの中には複数の文を書くことができる。ブロックの末尾では、文末の「。」を省略してもよい。

ブロックに対して自分自身を何度も実行させると**繰り返し**を実現できる。一定回数の繰り返しは

> 「...」！3回　繰り返す。

のように記述する。ここで、数字に続く数字以外の文字は無視されるため、「3回」は「3」と同じ意味である。

次のプログラムは、繰り返しを使った例である。

```
かめた＝タートル！作る。
「かめた！100　歩く　90　左回り」！4　繰り返す。
```

&show(square.png,nolink,画像の説明);

この例では、繰り返しを使うことで、四角を描く4行のプログラムを1行にまとめることができた。このように、同じ処理が何度も書かれているときは、繰り返しを使うことで、プログラムを簡潔にまとめることができる。

繰り返しの結果として、最後に実行された文の値が返される。よって、次のような記述も可能である。このプログラムを実行すると、三角形を描いた後で、かめたが200歩歩く。

```
かめた＝タートル！作る。
「かめた！100　歩く　120　左回り」！3回　繰り返す　200　歩く。
```

&show(repeat_result.png,nolink,画像の説明);

命令の定義

タートルなどのオブジェクトは、最初から「歩く」、「右回り」などの命令を使えるが、オブジェクトに新しい命令（メソッド）を追加して使うこともできる。

次のプログラムは正方形を描くプログラムである。よく読めば「100歩歩いて90度左に回る動作を4回繰り返すから正方形が描かれるのだな」と分かるが、慣れないと分かりづらい。

```
かめた＝タートル！作る。
「かめた！100　歩く　90　左回り」！4　繰り返す。
```

&show(square.png,nolink,画像の説明);

「正方形」という命令をかめたが理解できるように定義してみよう。オブジェクトに新しい命令を追加するには、追加したい命令を「…」で囲んだブロックで定義する[2)]。次のプログラムでは、かめたに正方形という名前の命令を定義した。命令定義のブロックでは、オブジェクト自身（ここではかめた）を自分と書くと便利である。定義した命令は、普通の命令と同様に使うことができる。

```
かめた＝タートル！作る。
かめた：正方形＝「「自分！100　歩く　90　左回り」！4　繰り返す」。
かめた！正方形。
```

&show(square.png,nolink,画像の説明);

作った命令は、プログラムの中で何度も使うことができる。次のプログラムでは、正方形を3回描いている。命令を定義して使うことで、プログラムを見たときに「ここで正方形を描いている」ということが一目瞭然であり、プログラムも短く書くことができた。

```
かめた＝タートル！作る。
かめた：正方形＝「「自分！100　歩く　90　左回り」！4　繰り返す」。

かめた！正方形。
かめた！120　左回り。
かめた！正方形。
かめた！120　左回り。
かめた！正方形。
```

&show(square3.png,nolink,画像の説明);

追加した命令には、「100 歩く」と同じようにパラメータを渡すことができる。パラメータを受け取るためには、変数をブロックの先頭で「|…|」という記号の間に記述する。複数のパラメータを受け取るときは空白で区切る。次のプログラムでは、パラメータとして辺の長さを受け取り、1辺がその長さの正方形を描いている。

```
かめた＝タートル！作る。
かめた：正方形□□|x|□自分□□x□歩く 90 左回り」！4 繰り返す」。

かめた！100　正方形。
かめた！120　左回り。
かめた！150　正方形。
かめた！120　左回り。
かめた！200　正方形。
```

&show(square32.png,nolink,画像の説明);

1)
プログラムはCtrl-Gでも実行できる。編集画面と実行画面は、Ctrl-Tで切り替えられる□Ctrl-Oでファイルを開く。文字の大きさはCtrl-↑□Ctrl-↓□Ctrl-0で変更できる□MacではCtrlの代りにCommandキーを使用する。
2)
詳しくは言語解説のブロックの説明を参照。

From:
https://dolittle.eplang.jp/ - **プログラミング言語「ドリトル」**

Permanent link:
https://dolittle.eplang.jp/ch_graphics?rev=1514987953

Last update: **2018/01/03 22:59**