2　章

ロゴのプログラミング

1. ロゴの基礎

■この節の内容はオンラインヘルプからも見ることができます。
EX　　→メニューバーの［ヘルプ］をクリック→［リファレンス」
EX（ジュニア）　→メニューバーの［ヘルプ］をクリック→［リファレンス」

・例示された命令は、コマンドセンターに入力して試します。Enterキーを押すことで命令が実行されます。(* コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter キーを押します。)
・例示された手順は、手順タブエリアに入力します。コマンドセンターに手順名入力し、Enter キーを押すことで手順が実行されます

マイクロワールド EX は、教育用コンピュータ言語のロゴ（Logo）をベースとしています。ロゴはわかりやすい言葉でプログラムを書くことができます。しかし、やさしい言葉であっても、コンピュータ言語である以上、いくつかのルールを守る必要があります。ここではそのルールについて、実践も交えて説明していきます。

(1) 基本用語

ロゴでは、あらゆる単語が、「あることを行うように」という指示と解釈されます。ロゴには、あらかじめ組み込まれ、コンピュータが理解できるよう定義された単語があり、これらを**基本用語(プリミティブ)**といいます。
基本用語を、その働きによって分類すると**コマンド**と**レポータ**、そして、ごく少数の特殊な単語の 3 種類になります。
ロゴのプログラムは**手順 (proce dure)** と呼ばれますが、手順はこれらの基本用語、すなわち**コマンド**と**レポータ**をつなげて作成します。

```
前へ　100
下へ書く　向き
```

各行が1つの命令ですが、**前へ**、**下へ書く**がコマンドであり、**向き**はレポータになります。

コマンドとレポータ

上に書いたように基本用語には「コマンド」と「レポーター」の 2 種類があります。
コマンドとは、あることを行わせるものです。

前へ　100の「**前へ**」は、指定した量（ドット数)、カメを動かします。

下へ書く　向きの「**下へ書く」**は、コマンドセンターにカメの向きを表示します。

このように実際に何かを動かしたり、状態を変化させたり、という仕事をコンピュータにさせる基本用語がコマンドです。

レポータは、計算の答えやオブジェクトの状態などの情報を報告します。簡略な言い方をするなら、私たちがロゴから何か情報を得たいときに使うのがレポータです。
レポータは情報だけを報告するので、コマンドを使って、その情報を受けとってやる必要があります。

下へ書く　向きの「**向き」**は、カメの現在の向き（情報）を報告します。

下へ書くというコマンドが「カメの現在の向き」という情報を受け取って、私たちが見える形でコマンドセンターに書きます。

次の命令の**最初**は、あとに続く**hellow**（インプット）という単語の1つ目の要素、"h" を報告します。それを受けて**書く**というコマンドは指定された位置に、"h" を書きます。

```
書く　最初　”hello
```

基本用語のどれがコマンドでどれがレポータかを、初心者のうちから分類しようとして考え込む必要はありません。例示されるサンプルプログラムや、それを自分でアレンジしたプログラムを作り、試していくうちに、コマンドとレポータの違いは少しずつわかってくるものです。もちろん、あなたが最初から複雑なゲームプログラムを組んだり、事務処理用の実用的なプログラムを組もうと考えるなら別ですが。
とはいえ、変数を使ったプログラムや2節で紹介するような幾何学模様、さらにリカージョンを使ったプログラムを作る場合には、コマンドとレポータの概念はある程度、身につけておくことが必要です。また、自分以外の誰かが作ったプログラムを分析していく場合にも、この概念は必要になります。

(2) タートルグラフィックスと基本操作

ロゴのカメ（Turtle）は、マイクロワールドEXの画面上に本物のカメに似たアイコンで表示されます。

このカメは、その位置と向き（カメが向いている方角）が分かるようになっています。この位置と向きは、カメの状態の基本要素です。
カメの状態を変更するにはコマンドを使います。
例えば、**前へ**あるいは**後ろへ**はカメを動かし、**左へ**あるいは**右へ**はカメの向きを変えます。
カメはまた、体の中央にペンを持っています。カメを作成した初期状態では、ペンは上がった状態になっていますが、**ペンを下ろす**でペンを下げた状態にすると、カメは線を引きながら動きます。

用語や構文について説明する前に、最も初歩的なルールを体験してみましょう。

ページの上にカメを1つ作り、コマンドセンターから以下に例示する命令を入力してください。1 行入力するたびに行末で Enter キーを押します。各行で Enter キーを押すたびに、ページ上でカメが命令を実行するのを見ることができるでしょう。
＊コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter キーを押します。

入力の基本

○基本用語(コマンドやレポータ)のあとには必ずスペースキーを押して1字分のスペースを入れます。
スペースは全角、半角いずれも使用できます。例では見やすいように全角スペースを入れています。(英語の基本用語を使う場合は、スペースも半角のみ使用可能となります)。

○インプットとしての数字を入力する場合も､基本用語のあとにスペースをいれることが必要です。
インプットとして数字を入力する場合も全角/半角いずれも使用することができます。(英語の基本用語を使う場合は、数字も半角のみ使用可能となります)。

○日本語の基本用語は" ひらがな" と" 漢字" の両方があります。
＊基本用語の一覧は巻末を参照してください。また、[ヘルプ] メニューの基本用語の検索から一覧を見ることもできます。
漢字の基本用語を使用する場合、基本用語として組み込まれている表記以外の「送り仮名」で入力しても、コンピュータはそれを基本用語として受け取ることができません（エラーメッセージが表示されます)。初心者の方は、漢字の表記があいまいな場合、「ひらがな」で入力することをおすすめします。(その一方で、漢字の基本用語を使うメリットには、あとでプログラムを分析したり修正する場合にわかりやすい、プログラム全体の長さが短くなる、などがあります)。

○基本用語を英語で記述する場合には､文字､数字ともに半角を使用してください。

○1つの作品や手順｡さらに命令の中に､英語､日本語(ひらがな､漢字)が混在しても､それぞれの基本用語の表記が正しければプログラムは正しく動きます。

```
前へ　100
右へ　45
前へ　100
帰る
```

今度は、次を試してみてください。

```
ペンを下ろす
前へ　100
右へ　45
前へ　100
```

前へは、**インプット**を1つ必要とします。コマンドの後ろに入力される「**インプット**」は**コマンドに必要な情報を与えて、目的の仕事をさせるもの**です。

次の命令は、**前へ　100**および**右へ　90**というコマンドを4回繰り返すことによって、カメに正方形を描かせます。

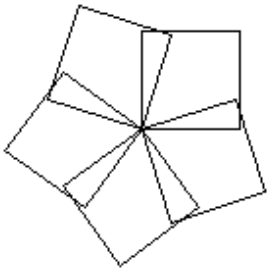
くりかえすは、インプットとして**命令リスト**を取るコマンドです。**命令リスト**は、ロゴが実行する命令をカッコ（「」）で、まとめたものです。

くりかえす 4 「前へ　100　右へ　90」

カメの回転は外角で考える

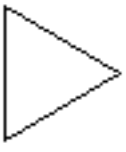
右へと**左へ**の働きについて少し考えてみましょう。
前へ 100がカメを100歩前進させるのを確認するのは簡単です。では、**右へ　90**はどうでしょうか？　90歩分の長さの横線を描くのでしょうか？
右へ　90は、位置を変えずに、カメを90度右回りに回転させます。カメの移動と向きの変更が別々に行えるということは、非常に便利であることはプログラミングを進めていくうちに理解されるでしょう。
例えば、この機能によって、同じ形であっても、向きの異なる形を簡単に描くことができます。

くりかえす　5　「くりかえす　4　「前へ 100　右へ 90」　右へ　72」



また、非常に簡単な命令を使って、カメに三角形を描かせることもできます。

くりかえす 3 「前へ　100　右へ　120」



この三角形の命令の「120」は不思議な感じがするかもしれません。
多くの人は、三角形を描くときに、**右へ　120**ではなく、**右へ　60**を使いたくなります。

しかし､ロゴの世界では､カメが回転する角度は外角です。

カメの動きの感覚は、人間が実際に動くのと同じようになっています。仮に、あなたが校庭で白線をひきながら、一辺が 100 歩の正方形を描くとしましょう。最初の一辺を描き終えて右へ曲がるとき、あなたは、今、自分が向いている向きから右へどのくらい身体を回したらいいかを考えるでしょう。

三角形と正方形のどちらも、カメが回転する角度の合計は一定であることに注意してください。自分でカメの軌跡を辿って正方形を作ると、カメが完全に 1 回転、つまり、360 度回転していることに気付くでしょう。実際、簡単な多角形を描くとき、カメはつねに合計で 360 度回転します。

・4 x 90 = 360
・3 x 120 = 360

ロゴでは、この現象をカメの完全移動の法則（Total Turtle Trip Theorem）と呼びます。

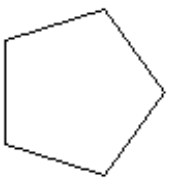
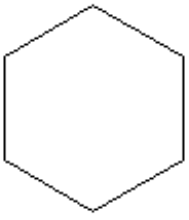
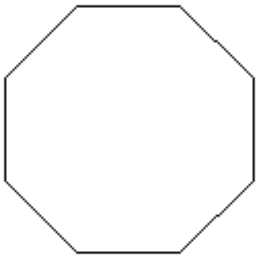
(仮定)カメの完全移動の法則
単純な閉じた図形の周囲を回って、出発したときと同じ位置と向きに戻る場合、カメは完全に 1 回転（360 度の回転）する。

このことについて、もう少し考えてみましょう。

この法則を使って命令のインプット（数字）を変えると、他の多角形を描くことができます。

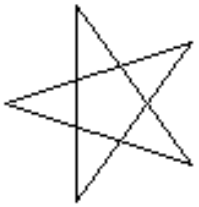
くりかえす＿＿「前へ　100　右へ ＿＿」

次のような図形を描くにはどのような数値をインプットすればよいでしょう？　アンダーラインの部分の数値を変えて試してみましょう。

実験していくと、この法則に従わないケースもあることを発見することができます。つぎのような場合です。

くりかえす 5 「前へ 100　右へ 144」



この命令はカメに星を描かせますが、カメは完全に2回転（合計で720度）しています。いろいろな図形を描いてみてください。結果として次のような結論が導かれるでしょう。

カメの完全移動の法則

> 閉じた図形の周囲を回って、出発したときと同じ位置と向きに戻る場合、カメは少なくとも1回、1回転する。

（3）手順

手順は、手順名の付いたロゴの命令の集まりです。
次は手順の例です。手順タブエリアを開き、入力してください。

```
手順は　四角
くりかえす　4 「前へ　100　　右へ　90」
終わり
```

ロゴで手順を作るということは、あらかじめ定義された基本用語を使って、単語を増やしていくことです。基本用語も、新たに定義した単語も、**コマンドセンター**から実行すれば、まったく同じように使うことができます。ただし、**ある作品の中で定義した単語は、通常はその作品の中でのみ使用できます。基本用語**と**手順で定義した単語**の違いはその点だけといえます。
手順を作ったら、ロゴにその手順を実行するように指示をだしましょう。コマンドセンターに次のように入力したあと、Enterキーを押して実行します。

```
四角
```

手順の中に書かれた命令をすべて実行して、四角を書きます。
＊カメのペンが下りていない場合は線を描きません。その場合はコマンドセンターから次のように命令します。そのあとで「四角」を実行してみてください。

```
ペンを下ろす
四角
```

ａ．手順の構造と手順作成

「手順」は、3つの部分に分けられます。

```
手順は　四角                          …（手順名）
くりかえす　4 「前へ　100　　右へ　90」　…（手順の内容）
終わり                                …（終了）
```

「手順は」と「終わり」

「手順」は、必ず**手順は__〇〇**というように手順名から始めます。

・この行を**タイトル行**といいます。この行には、ほかの単語を入力しません。Enterキーを押して改行し、次の行から命令（手順の内容）を書きは　じめます。
・手順名は必ずスペースを含まない単語1つの名前を付けてください。（数字だけを手順名に使ったり、基本用語と同じ単語を手順名に使うことはできません）。
・手順の最終行には、必ず**終わり（おわり）**という単語1つだけ入力し、Enterキーを押します。
終わりが入力されていないと、マイクロワールドEXは手順を認識することができません。（この場合も日本語のひらがな表記なら「おわり」、漢字表記なら「終わり」、英語の場合には「end」です。仮に「終り」と入力してもマイクロワールドEXはそれを認識することはできません）。
このように「手順」の最初の行と最後の行は、特別な形式の行です。
手順はで始まるタイトル行と**終わり**の行におはさまれた部分が手順の本体になります。

手順の内容と改行

ロゴではスペース（空白）が区切り文字になっています。そのためにそれぞれの基本用語の間、また基本用語とインプットの間にはスペースが必要であることを前項目（タートルグラフィックスと基本操作）でも説明しました。では、改行はどのようになるのでしょう？

```
手順は　四角
くりかえす　4 「前へ　100　右へ　90」　　…
終わり
```

```
手順は　四角
くりかえす　4 「前へ　100　　…
右へ　90」
終わり
```

```
手順は　四角
くりかえす　4 「　　…
前へ　100
右へ　90」
終わり
```

```
手順は　四角
くりかえす　4 「前へ　100　　…
右へ　90」
終わり
```

```
手順は　四角
くりかえす　4　「
前へ　100　右へ　90　　…
」
終わり
```

手順タブエリアや**カメの手順エリア**に手順を書く場合、 から　まですべて正しく同じように動作します。
手順タブエリアやカメの手順エリアでEnterキーを押して改行した場合､改行はスペースと同じに認識されます。
手順は、エラーを調べたり、修正したりするときに見やすいように改行して書くようにしましょう。
＊　と　は他のコンピュータ言語のプログラミングでも使われる「字下げ」という方法です。
＊スペースは最低1つ入れることが必要ですが、このように字下げのために**スペースをつけ加えることは問題ありません**。

b . 手順の実行

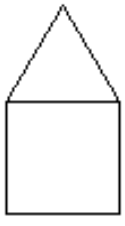
ロゴで手順をテストしたり実行する最も一般的な方法は､コマンドセンターに手順名を入力して､Enterキーを押して実行する方法です。
ロゴはその手順内のすべての行を実行します。
四角という手順を作ってある場合､コマンドセンターに**四角**と入力すると､**くりかえす　4　「前へ　100　右へ　90」**と入力したのと同じ結果が得られます。

入力された単語の定義（手順）が見つからない場合は、コマンドセンターにエラーメッセージが表示されます｡例えば､定義されていない**大きな四角**という単語を実行しようとして､コマンドセンターに**大きな四角**と入力すると､エラーメッセージが表示されます。

大きな四角のしかたは知りません。

手順を実行するもう1つの方法として､他の手順の中から手順を実行する方法もあります｡手順の定義の中で、すでに定義した手順を呼び出すことができます。例えば、三角という手順が定義されているとしましょう。

```
手順は　三角
くりかえす　3　「前へ　100　右へ　120」
終わり
```



続いて、**家**という手順を作ります。

```
手順は　家
```

四角 
前へ 100 
右へ 30 
三角 
終わり 

コマンドセンターから家を実行すると、間接的に**四角**と**三角**の2つの手順を実行することになります。つまり、**家**によって自動的にこれら2つの手順が実行されます。**家**でロゴが行うことを説明すると、次のようになります。
家手順の命令が1つずつ実行されます。**四角**の入った命令があると、その手順の定義が探し出され、その本体が実行されます。*四角*の本体は、**くりかえす 4 「前へ 100 右へ 90」**です。
続いて、**前へ**と**右へ**に進みます。
三角にくると、その手順の定義が探し出され、その本体が実行されます。
家の最後の行、**終わり**の行に達して、**家**の実行が完了します。
四角と**三角**は**家**の***子手順***、**家**は**四角**と**三角**の***親手順***です。
上記の例の子手順の**三角**を定義しないで、**家**を実行しようとすると、次のようなエラーメッセージが表示されます。

三角のしかたは知りません。

ヒント ：手順を作ると、その手順名はロゴのボキャブラリーとなり、その作品内で自由に使用することができますが、他の作品内や、新しい作品では使用できません。しかし**手順は、別の作品の手順タブエリアにコピーすることができます。**

(4) インプットを取る手順

四角という手順は、実行されるたびにまったく同じ正方形を描きます。**家**は、まったく同じ家の形を描きます。
これらの手順を**前へ**の命令と比較してみましょう。コマンドセンターから次のように入力し、実行してみます。

前へ 100 

この命令は、次の命令の2倍の長さの直線を描きます。

前へ 50 

前へというコマンドのインプットを変えれば異なる長さの直線を描くことができます。そこで、インプットを変更すれば辺の長さの違う四角を描くことができる汎用性のある手順を考えてみましょう。

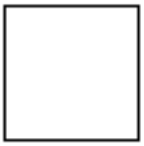
インプットを取る手順の作成

手順も、インプットを取る手順を作ることができます。四角という手順を、一辺の長さが100の正方形ではなく、いろいろな辺の長さの正方形を描く手順に変更します。

```
手順は　四角　：辺
くりかえす　4 「前へ　：辺　右へ　90」
終わり
```

このようにすると**四角**という手順は、インプットを取るコマンドとまったく同じようにして実行することができます。ただし、**四角**を実行する場合に、描く正方形の一辺の長さ（**：辺**）を指定する必要があります。
例えば、一辺が50の長さの正方形を描くには、次の命令を入力します。
＊先に**絵を消す**を使って画面に描かれたものを全部消してから実行しましょう。

```
絵を消す
ペンをおろす
四角　50
```



小さな正方形を描くには、次の命令を入力します。

```
四角　10
```

この**四角**の定義は、インプットを取る手順を作るときの一般的なルールを表しています。

インプットを取る手順の場合は､タイトル行の手順名の後にインプット名を指定します。インプットには自由に名前を付けることができますが､必ずコロン(:)で始める必要があります。

四角のタイトル行は、**四角**が**辺**という一つのインプットを取ることを表しています。手順の本体には、**辺**をインプットとするコマンド、**前へ**があります。この命令は、**：辺**の長さの直線を描きます。

インプットを取る手順を作った時点では、インプットには値はありません。インプットに値が与えられるのは、手順を実行するときだけです。

複数のインプットを取る手順

次の長方形は、2つのインプットを必要とする手順です。

```
手順は　長方形　：高さ　：幅
くりかえす　2 「前へ　：高さ　右へ　90　前へ　：幅　右へ　90」
終わり
```

長方形を実行するには、名前に続けて2つのインプット値を入力します。

```
長方形　50　100
```

インプットを取る親手順、子手順

ここで、親手順と子手順を使った例を紹介しましょう。**手順の作成**で使った**家**を編集して、家の形を異なる大きさで描けるようにします。

```
手順は　家　：大きさ
四角　：大きさ
前へ　：大きさ
右へ　30
三角　：大きさ
終わり
```

子手順にあたる**四角**と**三角**は、次のように定義しなおします。

```
手順は　四角　：大きさ
くりかえす　4　「前へ　：大きさ　右へ　90」
終わり
```

```
手順は　三角　：大きさ
くりかえす　3　「前へ　：辺　右へ　120」
終わり
```

1辺の大きさが150の家を描く手順を実行してみましょう。

```
絵を消す
家 150
```

大きさというインプットを取るように定義した**四角**は、親手順である**家**の中でも、**大きさ**というインプットを取ることに注意してください。インプットの名前は値を持つ入れものをあらわすにすぎません。

家の手順の中で**四角**、**前へ**、**三角**がそのインプットとしてすべて150を受け取るということは、どのようにして認識されるのでしょうか。ロゴが行うことを説明すると、以下のようになります。

四角の命令にくると、子手順である**四角**の定義を探し出し、150をインプットとして実行します。続いて、同じく150をインプットとして**前へ**を実行し、さらに**右へ　30**を実行しま

す。子手順である**三角**に達すると、三角の定義を探し出し、再び150をインプットとして実行します。
家の最後の行、**終わり**の行に達して、**家**の実行が完了します。

インプットを受けた親の手順は、そのインプットを子の手順に渡すことができます。どのようにして手順は、自身のインプット値を保持しながら、別の手順にインプットを渡すことができるのでしょうか。これは、インプットを名前と見なし、それぞれの手順に専用の名前ライブラリが存在すると考えると理解できます（エーベルソン、1982年）。つまり、インプット名は手順ごとに存在しますから、複数の手順で、値について混乱することなく、1つのインプットに同じ名前を使うことができます。
手順を作成すると、その手順専用のライブラリが作成され、手順内に定義されたインプット名と、その手順が呼び出された時点で与えられた実際の値の対応情報を保存します。そして、手順実行中にインプット名の入った命令があると、その手順専用のライブラリからそのインプットの値を探し出します。このため、2つのライブラリに同じ名前が存在していても、それぞれに異なる情報を持たせることができます。
インプットは、その手順の"ローカル変数"となります。専用のインプット名が必要なのは、手順の定義の詳細を正確に知らなくても、手順を実行できるということです。手順が行うことに関心を集中することができます。
親手順の家を作る時点では、四角は、単に正方形を描く"ブラックボックス"と考えることができます。四角がインプットに使用している名前に関心を払う必要はありません。

Abelson, H. (1982). Apple Logo. Peterborough, NH: Byte Publications Inc.

(5) インプットとデータ

ロゴの命令は、基本用語や手順とそのインプットという基本単位で構成されます。
(中にはインプットを必要としない基本用語や手順もありますが、複雑で高度なプログラミングに進むにしたがって、この基本単位を知っておくことが作成や内容の理解を助けます)。

これを式になぞらえるなら、**命令　＝　基本用語(または手順)　＋　インプット**　ということができます。

インプットとして使用できるデータには**数字(数値)、ワード、リスト**の3種類があります。

a. 数字（数値）

数字（数値）は最も一般的にインプットとして使用されます。計算機能を利用した高度なプログラム（かなり正確な計算結果を必要とするもの）のために、ロゴでは浮動小数点形式の数字を使うことができます。
＊数字の表記方法は、IEEE 754　64ビット倍精度の規格に準拠しています。

数字の例

113
15.2

-16.03
323.
1E4

浮動小数点数は、次の各部から構成されます。

・負符号（省略可）
・整数部: 値がゼロの場合は必要ありません。ゼロ以外の場合は、数字の組み合わせになります。
・小数点: 数字に小数部がある場合にだけ必要です（数字の区切り記号は、Windows の地域プロパティの設定によって異なることがあります）。
・小数部: 値がゼロの場合は必要ありません。ゼロ以外の場合は数字の組み合わせになります。
・指数（省略可）: 英字の E と符号、300 までの整数。例えば、1E4 は 1 x 10 の 4 乗（= 10000）を意味します。

注意　：空白（スペース）は数字として、あるいは数字の途中で使うことはできないことに注意してください。

正符号（＋）を使って正数であることを示すことができますが、必須ではありません。正数あるいは負数であることを示すには、それぞれの符号（＋、－）を数字のすぐ前に付ける必要があります。

下へ書く　－2　＋　5

3

"－"と数字の間に空白がある場合、引き算の記号と見なされ、エラーメッセージが表示されます。

下へ書く　－　2　＋　7　（"－"と"2"の間に空白があります。）

－にインプットが足りません。

b. ワード

日常生活で使用している言葉も、ロゴの命令の中でインプットとして扱われる場合には、**ワード**と呼ばれます。**ワード**は文字で構成されています。数字を通常の文字として使用することもできます。

ワードの例

hellow
x
3.14
314

環境システム
Ｂig.Apple
日本-アジア
Bri3an
誰か?
やあ！

これらはすべてワードです。文字の１つ１つが、ワードの要素です。Bri3an は、６つの要素で構成されています。

Ｂｒｉ３ａｎ

ダブルクォーテーションマーク（ "）

ワードをインプットとして扱う場合の例として、よくあるのは次のような場合です。

下へ書く　"hello

hello

ワードの前には**ダブルクォーテーションマーク(")**が必要です。
・ワードの前のダブルクォーテーションマークは､そのあとに続く単語がコマンドやレポータではないことをコンピュータに伝えます。コンピュータに対して、ワードをその文字通りに扱うようにという指示になります。
・ダブルクォーテーションマークとワードの間はスペースを開けません。
・ダブルクォーテーションマークはワードの前に１つだけ付けます。ワードの後ろに引用符があると、ワードの一部と見なされてしまいます。

下へ書く　"hello"

hello"

スペース（空白）とカッコを含むワード

ロゴではスペース（空白）は区切り文字として使われ、また、下のようなカッコもプログラムの中で使用されます。

「」()[]

スペースや、これらのカッコを含んだワードを使う場合は、その文字を含むワードを**縦線**(¦) で囲みます。**縦線**は、その中の全内容を１つのワードと見なすという指示になります。

下へ書く　"¦Big Apple¦

Big Apple

c．リスト

リストとは、ワードやリストを間にスペースをいれて並べ、全体をカッコ（「 」または［］）で囲んだものです。

```
下へ書く 「これは　リスト　です」
```

これは　リスト　です

リストの例

```
「はじめまして。　山田です。　よろしく」
「1 X 2 Y 3 Z」
「こんにちは」
「「中庭」「噴水　池」「犬小屋]　花壇」
「「インターネット　「java　html」　「ワールド　ワイド　ウェッブ」「gif jpeg」」
「「1　「1　2」「17　「17　2」」」
「」
```

最初のリスト、「はじめまして。　山田です。　よろしく」には、3つの要素が含まれています。

```
はじめまして。
山田です。
よろしく
```

「「中庭」「噴水　池」「犬小屋]　花壇」のリストには、4つの要素が含まれ、その要素のそれぞれに別のリストが含まれています。

リスト内のワードには、ダブルクォーテーションマークは必要ありません。また、余分なスペース（空白）は無視されます。
また、コンピュータから出力される際にはリストの外側のカッコが表示されないことに注意してください。

```
下へ書く 「x　 y　z」
```

x y z

```
下へ書く 「こんにちは」
```

こんにちは

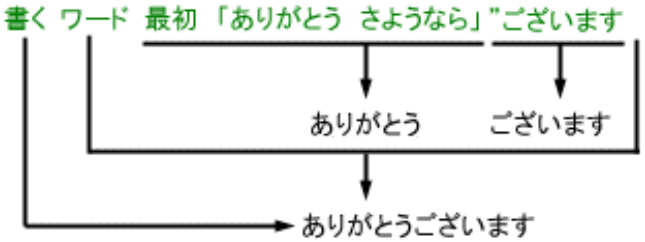
(6) ロゴの文法

ロゴの文法を理解することは、ロゴの命令を正しく作成したり、理解したりする上で役に立ちます。

- 各命令の先頭は常にコマンドです。
- 命令の先頭以外の単語はすべてコマンドに対するインプットとなります。
- 個々のインプットは、コマンドあるいはロゴの式が必要とするデータの種類によって決まります。
- コマンドあるいはレポータに対するインプット数はそれぞれに決まっており、複数のインプットの場合でも、括弧は必要ありません。ただし、命令内の計算は例外です。

以下に、複雑なロゴの命令の例を示します。

書く　ワード　最初　「ありがとう　さようなら」　”ございます



上記の文法規則に従って、この命令を左から順に読み取ると、次のように解釈することができます。

書くは、インプットを1つ取るコマンドです。インプットは**ワード** が報告した結果です。
ワード は、インプットを2つ取るレポータです。
1つ目のインプットは、**最初**というレポータが報告する**「ありがとう　さようなら」**の**ありがとう**です。
2つ目のインプットは、**ございます** という単語です。
最初　「ありがとう　さようなら」は、その結果として**ありがとう**という単語を**ワード**に伝えます。
ワード は、そのインプットの**ありがとう** と**ございます** で1つのワードを作り、その結果を**書く** に伝えます。
この結果、**書く** は**ありがとうございます** と表示します。

ロゴの命令とレポータ

ロゴの命令では、レポータから報告される値またはデータをどのように処理するかを指示する必要があります。
レポータは、それ自体がインプットを取ることもありますが、主にコマンドや手順に対するインプットとして使います。命令は左から順に読み取られるので、ロゴの命令は必ずコマンドから始まる必要があります。レポータが命令の先頭にある場合は、エラーメッセージが表示されます。コマンドセンターから、次のように入力してみましょう。

向き

0 をどうするのかわかりません。

では、**下へ書く**というコマンドを先頭に付けると、どうでしょうか。

下へ書く 向き

0

ここでは、**向き**というレポータの報告を**下へ書く**が受け取っているのでエラーメッセージは出ません。

次の例では、レポータそのものがインプットを取る例です。

下へ書く　最初　"Logo

L

この例では、**最初　"Logo**が**下へ書く**に対するインプット、**"Logo**が**最初**に対するインプットです。**最初　"Logo**を実行すると、**最初**の結果が**下へ書く**に伝えられます。

レポータには、複数のインプットを必要とするものがあります。例えば**和**がそうです。

下へ書く 和　2　3

5

この例では、**和**は2つのインプットを必要としており、結果を**下へ書く**に報告します。

インプットを必要としないレポータもあります。例えば、状態変数の**向き**はそれだけで値を返します。

ヒント1：それぞれのレポータの使い方は、［ヘルプ］メニューの **［基本用語の検索（めいれいをしらべる）］**から、その機能や用例を調べることができます。
ヒント2：計算式はすべてレポータです。計算式はつねに値を報告します。

（7）変数

変数は、外側に名前が付き、内側に値（数字、ワード、リスト）の入った箱（コンテナ）と考えることができます。

変数を作るということは、入れものを作ると同時に、中に値を入れることです。従来のロゴには、「ローカル変数」と「グローバル変数」の2種類の変数がありましたが、『マイクロワールドEX』のロゴでは、さらに 「状態変数」があります。
＊第3章「2. 変数のテクニック」(p.295)もご覧ください。

a. ローカル変数

手順のタイトル行に付けるインプットは**ローカル変数**になります。
ローカル変数は、その手　順が実行されている間だけ値を保持する変数です。
ローカル変数の簡単な例を示します。ここでは**サイズ**がローカル変数です。

```
手順は　四角　：サイズ
繰り返す　4　「前へ　：サイズ　右へ　90]
終わり
```

コマンドセンターに次のように入力して、この手順を実行してみてください。

```
四角　70
```

ローカル変数には好きな名前を使うことができますが、手順の途中で名前を変えることはできません。また、ローカル変数名の前には**必ずコロン（：）を付ける**必要があります。手順の中のコロンは、「欲しいのは、サイズという単語そのものではなく、サイズという入れものの中身（値）」であることを意味します。

ローカル変数は、仮の名前はやローカルはを使って作ることもできます。

b. グローバル変数

グローバル変数は、**名前は**あるいは**中身は**を使って作成することができます。

グローバル変数は、別の作品を開いた場合でもその値を保持します。
グローバル変数の値が失われるのは、**名前を皆なくす**を使って意図的に値を消去するか、『マイクロワールドEX』を終了した場合です。

グローバル変数を作る

作品と一緒にグローバル変数が保存されることはありません。変数の名前には、頭にダブルクォーテーションマークを付けます。

```
名前は　”友達　[洋子　Tom　ミカ　健一]
```

コマンドセンターに上の命令を入力して実行します。これで、**友達**という名前のついた変数の箱に、洋子、Tom、ミカ、健一という値が保管されたことになります。

グローバル変数を使う

変数の**値**が必要な場合は、変数名の前にコロン（：）を付けます。

```
下へ書く　：友達
洋子　Tom　ミカ　健一　…変数の値を表示します。
```

コロンを付けないと、『マイクロワールドEX』は、友達というコマンドを探して実行しようとします。

```
下へ書く　友達
友達のしかたは知りません。　…エラーメッセージが表示されます。
```

”友達と入力した場合はどうでしょう？

```
下へ書く　”友達
友達　　……友達という単語そのものを表示します。
```

ダブルクォーテーションマーク（”）とコロン（：）の違いに注意してください。

c．状態変数

状態変数はオブジェクトのそれぞれの状態情報を保持します。
1つの状態変数は、オブジェクトの特定の1つの状態情報を保持します。通常、状態変数には、状態を変更するコマンドと、その状態を報告するレポータがあります。

状態変数には次のようなものがあります。
・カメ
・スライダーとテキストボックス
・作品

カメの状態変数

カメには、向きや色などのいくつかの**状態変数**があらかじめ用意されています。
カメの状態変数は、**カメの中の名前**はを使って新たに定義することができます。

カメの中の名前はは、現在の作品のカメの1つ1つに状態変数を作ります。この状態変数には、そのカメ専用の値を設定することができます。
例えば、**カメの中の名前は　"スピード**という命令を実行すると、**スピードは**というコマンドが作成され、

```
かめ1、 スピードは　12
```

というように使うことができます。

特定のカメの変数の値を知る方法には次のような方法があります。

・作品タブエリアで、カメのアイコンをクリックし、変数の状態を参照する。
・コマンドセンターから、カメの名前と変数名を使い、値を報告させる。

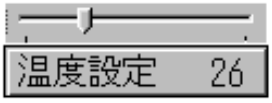
```
かめ1、 下へ書く　スピード
```

・コマンドセンターから、カメの名前に続けて"の"を付け、命令する。

```
下へ書く　かめ1の　”スピード
```

スライダーの状態変数

スライダーの状態変数とは、そのスライダーによって報告される値です。
温度設定という名前のスライダーを作ります。



マウスでスライダーのボタンを動かして26の位置にします。これで**温度設定**というスライダーの状態変数は**26**という値になりました。

温度設定というスライダー名を使い、設定されている値を報告させることができます。コマンドセンターから、確認してみましょう。

```
下へ書く　温度設定

2 6
```

温度設定はというように、スライダー名のあとに、**は**を続けることによって設定値を変えることができます。

```
温度設定は　45

下へ書く　温度設定

4 5
```

テキストボックスの状態変数

テキストボックスの状態変数とは、その**テキストボックスの内容**です。

いくつかの名前の入った、**友達**というテキストボックスを作ります。

友達というテキストボックス名を使い、このテキストボックスの内容（状態変数）を報告させることができます。

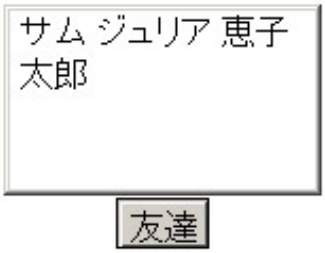
```
下へ書く　友達
```

洋子
Tom
ミカ
健一

友達はというように、テキストボックス名のあとに、**は** を続けることによって、その内容を置き換えることができます。

```
友達は 「サム　ジュリア　恵子　太郎」
```

テキストボックス名(友達)　+　は は､現在のテキストを完全に置き換えます。



ヒント　：これらの状態変数の値は、スライダー上あるいはテキストボックス内に常に表示され、普通の状態で見ることができます。
これは、**名前は**や**中身は**で作ったグローバル変数と違うところです。グローバル変数の内容はコンピュータのメモリにだけ存在しますから、抽象的です。

作品の状態変数

マイクロワールドEXでは、作品自体もオブジェクトのひとつです。
グローバル変数は、マイクロワールドＥＸを終了した時点でメモリからクリアされてしまいますが、作品の状態変数は作品と一緒に保存されます。
次回、同じ作品を開いた場合、作品の状態変数には保存した値が保持されています。

作品の状態変数は、作品内の名前はというコマンドを使って作成します。

作品内の名前は　"ポイント

この命令によって、マイクロワールドEXには、**ポイント**と**ポイントは**という2つの単語が追加されます。
以降、同じ作品で、次のような命令を使うことができます。

ポイントは　10　　…ポイント変数に10を設定します。

下へ書く　ポイント

10　　…ポイント変数の現在の値を表示します。

ポイントは　ポイント　+　5　　…現在の値に5を加えた値をポイント変数に設定します。

下へ書く　ポイント

15　　…ポイント変数の現在の値を表示します。

作品内の名前は、作品に含まれる作品の変数名をすべて報告するレポータです。

下へ書く 作品内の名前

ポイント　　…現在の作品の作品の状態変数名を表示します。

(8)計算のルールと特殊な記号

a．計算のルール

命令内の計算（＋ －＊ / < > =）に関するルールを説明します。このルールを知らないと正しい結果が得られないことがあります。

ルール1　：演算記号の両側には必ず空白を入れます。

空白がない場合、演算記号は単なる文字と見なされます。

（誤）下へ書く　100/2

100/2 のしかたは知りません。

（正）下へ書く　100　/　2

5 0

ルール2　:四則演算は､一般的な順序で計算され､掛け算と割り算が足し算と引き算に優先します。

一般的な順序とは異なる順序で計算を行いたい場合は、カッコ　"()"を使ってください。カッコ内が最初に計算されます。

```
下へ書く　3　＋　2　＊　4
```

1 1

```
下へ書く　（3　＋　2）＊　4
```

2 0

ルール3　:ロゴのコマンドは､計算を終えた後で実行されます。

例えば次は、サイコロの数字をシミュレートする命令の例です。**乱数** は、0からそのインプットより1小さい数字の範囲の乱数を報告します。

```
下へ書く　乱数　6　＋　1
```

この例では、6＋1がまず計算され、その後で**乱数**が実行されます。このため、**乱数　7** が実行されて、0から6の範囲の数字が報告されることになります。しかし、
サイコロの数字をシミュレートするには、数字の0が発生するのを防がなければなりませんから、この命令は修正する必要があります。

次のように､**乱数の結果に1を加える**ことによって0が発生するのを防ぎます｡つまり､正しくサイコロの数字をシミュレートするには、次のようにカッコを使って、計算の順序を変更します。

```
下へ書く　（乱数　6）＋　1
```

これは、次のように書くこともできます。

```
下へ書く　1　＋　乱数　6
```

＋の計算をするために、コンピュータは2つ目のインプットを要求します。そこで乱数6の結果が報告されます。

カッコの利用

カッコは、等号（=）の入った命令の実行にも影響します。例えば、2つの数字の先頭の数字が等しいかどうかを調べたいとします。次の命令では、正しい結果は報告されません。

```
（誤）下へ書く　最初　3.14　＝　3
```

これは、**最初**を実行する前に、**3.14**と**3**が等しいかどうかが調べられ、その報告である真

偽値「**うそ**」の最初の文字、**う** が**下へ書く** に報告されるためです。カッコを使うと、正しい結果が得られます。

```
（正）下へ書く （最初　3.14 ）　=　3
```

ほんとう

もう一つ、別の例を紹介しましょう。数字の2乗値を求める次のような手順があります。

```
手順は　二乗　：x
表示　：x　＊　：x
終わり
```

この手順を使って、2つの2乗値の合計を求めたいとします。次の命令では、正しい結果は得られません。

```
（誤）下へ書く　二乗　5　+　二乗　3
```

1 9 6

この結果が正しくないのは、最初に5に3の2乗（9）が加えられ、その後で、その合計（14）の2乗が求められるためです。正しくは、それぞれの数字の2乗を最初に求める必要がありますから、次のようにカッコを使って、計算の順序を変更します。

```
（正）下へ書く （二乗　5） + （二乗　3）
```

3 4

b. 特殊な記号

ロゴで使う記号には、次のような特別な意味を持つ記号があります。

記号(呼び方)	意　　味
" （ダブルクォーテーションマーク）	この後のワード（単語）をその文字通りに扱うときに使います。
「」または[] （かぎカッコ）	囲まれた内容がリストであることを示します。
： （コロン）	この後の単語が変数名であることを示します。入れものの内容を参照します。
() （カッコ）	通常とは異なる方法でグループ分けするか、基本用語に対するインプット数を変えるときに使います。
| | （縦線）	囲まれた内容（スペースを含む）を1つのワードとして扱うよう指示します。

(9) 繰り返し実行と条件付き実行

ロゴには、リストを一続きの命令として処理するコマンドがあります。これらは次のように分類することができます。

・繰り返し実行…あることを繰り返し行うコマンド
・条件付き実行 …ある条件が満たされた場合にあることを行うコマンド

繰り返し実行

1つの命令を繰り返し実行するコマンドとしては、例えば「**くりかえす**」があります。

```
くりかえす　3 「書く　”おはよう」
```

くりかえすは、1つ目のインプットとして数字、2つ目のインプットとして命令リストを取ります。

```
くりかえす　10「前へ　3　待つ　1」
```

繰り返し実行するコマンドとしては、他に**それぞれをやる**および**それぞれの数字をやる**があります。これらのコマンドは、インプットの内容を変更しながら命令を繰り返すことができます。

条件付き実行

条件付き実行では、ある条件が**真**であるかどうかに基づいて命令を実行することができます。条件を検査するためのコマンドとしては、**もし**と**もしどちらかを**があります。

```
もし　縦の位置　>　100　「色は　”あか」
```

ページにカメを1つ作ってから、コマンドセンターにこの命令を入力して、実行してください。その後で、カメをページの最上部にドラッグし、再び命令を実行します。

もしには、2つのインプットが必要です。1つ目のインプットは真または偽を返す条件、2つ目のインプットは命令リストです。**もし**の命令リストは条件付きであり、条件が満たされた場合にだけ実行されます。

もしどちらかをは、3つのインプットを取ります。1つ目のインプットは真か偽を返す条件、2つ目のインプットは条件が真の場合に実行する命令、3つ目のインプットは条件が偽の場合に実行する命令です。

次を試してみてください。

```
無限に　「前へ　1」
無限に　「もしどちらかを　縦の位置　>　0　[色は　”あか」「色は　”あお」」
```

ストップルール

もしおよび**もしどちらかを**は、ストップルールとして利用することができます。ストップルールは、次のように表されます。

「特定の条件が満たされた場合は、この手順をストップしなさい。」

次の手順は、ストップルールとして**もし**命令を使っている例です。

```
手順は　らせん　：幅　：角度　：増加
もし　：幅　＞　200 「止まる」
前へ　：幅
右へ　：角度
らせん　：幅　＋　：増加　：角度　：増加
終わり
```

コマンドセンターから次のように入力して実行してみましょう。

```
ペンを下ろす
らせん　2　50　1
```

止まる は、手順を止めるコマンドです。この他にも、**これを止める**、**皆止まる**、といったコマンドは手順を止めます。

2. 幾何図形：カメの幾何学

・例示された命令は、コマンドセンターに入力して試します。Enter キーを押すことで命令が実行されます。（＊コマンドセンターで命令を実行しないで次の行に移りたい場合は、 Ctrl を押しながら Enter キーを押します。）
・例示された手順は、手順タブエリアに入力します。コマンドセンターに手順名入力し、Enter キーを押すことで手順が実行されます

ロゴ（Logo）の開発者であるシーモア・パパート教授（MIT）は、タートルグラフィックスで表現されるコンピュータ上の世界を、小宇宙すなわち「Micro Worlds（マイクロワールド）」と呼び、自由に試行錯誤できる無限の可能性をもった環境と位置づけました。特にプログラミングの導入や数学的思考には適した環境であるといえましょう。
ロゴの特徴であるカメは、数学的、幾何学的なチャレンジに対して美しい図形で応えてくれます。ここでは、円を描く例を通して、インプットの引き渡しと幾何図形の重要な概念を学びます。

(1) 円と半径

カメに円を描かせてみましょう。

```
手順は　円
くりかえす　360 「前へ　1　右へ　1」
終わり
```

カメはごく短い距離だけ前進し、すこし向きを変えるという動作を繰り返すことによって、曲線（弧）を描きます。

上のように、円は、カメを 360 度回転させることによって描くことができます。とはいえ、幾何学上の厳密な意味では、これは円でなく 360 の辺の多角形です。これは、円を短時間に描くためのちょっとした妥協です。

```
手順は　円
くりかえす　36 「前へ　10　右へ　10」
終わり
```

この円はもっと不完全で、36 個の辺の多角形を描きます。
前へに対するインプットと右へに対するインプットの両方を変えていることに注意してください。**前へ**のインプット値を変えられるように手順を書き直すことによって、異なるサイズの円を描くことができます。

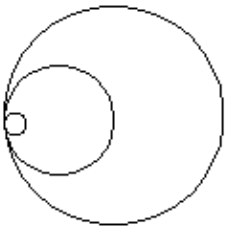
```
手順は　円　：歩幅
くりかえす　36 「前へ　：歩幅　右へ　10」
終わり
```

手順の入力が終わったら、コマンドセンターから次のように入力して実行してみましょう。

```
円　1

円　5

円　10
```



円のサイズは、インプット値に応じて変化します。インプット値が大きいほど、円は大きくなります。それぞれの円で繰り返される**前へ**の回数は同じ36回ですから、これは驚くことではありません。円周の長さは、**前へ**のインプットである**歩幅**によって決まります。

歩幅によって円の半径が決まるようにすると、**円**という手順はずっと便利なものになります。

円周は、2πrで求められます。πは円周率であり、rは半径です。
手順**円**では、円周は36個の**前へ**で描かれます。このため、次の式が成立します。

```
36　＊　：歩幅　＝　2　＊　円周率　＊　：半径
```

または

```
：歩幅　＝　2　＊　円周率 ＊ ：半径　/　36
```

この式は、**：歩幅**の値と円の半径の関係を表しています。

「**円周率**」はマイクロワールドEXの基本用語にその値が登録されています。コマンドセンターから次のように試してみましょう。

```
下へ書く　円周率
```

手順タブエリアに円の手順が作成されていれば、円を子手順にして、円に対するインプットでこの計算式を使用することができます。

```
手順は　円描き　：半径
円　2　＊　円周率　＊　：半径　/　36
終わり
```

絵を消すで画面をいったんきれいにしてから、次のようにコマンドセンターに入力し、実行してみましょう。

```
円描き　30
```

円描き　60

手順に対するインプットで計算式を使用できることに注目してください。

2　＊　pi　＊　:半径　/　36

ロゴは、**円**に対するこのインプットを次のように計算します。

最初に 2 x pi の計算が行われます。
pi は 3.14159265359 を報告するボキャブラリーであることを認識します。
計算結果と：半径の掛け算が行われ、その結果が 36 で割られます。

(2) 円の一部としての弧

作品でよく使われる幾何図形に弧があります。
弧は円の一部にですから、回転量を変えることによって、異なる長さの弧を描くことができます。**円**の手順を書きなおして、インプットを追加し、弧を描く手順を作成しましょう。

手順は　弧　：歩幅　：回数　　…手順名を変更して、インプットを追加しています。
くりかえす　：回数　「前へ　：歩幅　右へ　10」
終わり

次のようにインプット値を変えて、異なる弧を描くことができます。

弧　10　18

弧　10　9

弧には、描こうとする円周の角度をインプットとして与える方が、分かりやすくなります。
与えられた角度に基づいて歩数と回転回数を計算できるようにしましょう。
弧の手順を書きなおすと、次のようになります。

手順は　弧　：歩幅　：角度
くりかえす　：角度　/　10　「前へ　：歩幅　右へ　10」
終わり

命令リストが繰り返し実行されるたびにカメが10度右に向きを変えるため、**角度**を10で割っていることに注意してください。コマンドセンターから実行してみましょう。

弧　10　90　　…円の4分の1の弧を描きます。
弧　10　180　　…半円を描きます。

ヒント　：カメを新しく作成した場合は、コマンドセンターから手順を実行する前に、**ペンを下ろす**と入力し図形を描けるようにしましょう。また、カメが軌跡を描かないようにするには**ペンを上げる**と入力します。

(3) スライダーを使った多角形

ページ上に作成したスライダーを使って、手順の実行にインタラクティブな制御を加えることができます。

スライダー名は、現在の値を報告します。スライダーをインプットとして利用することのメリットは2つあります。1つはその値を画面で確認できること、もう1つはマウスで簡単に値を変更できることです。

例えば、スライダーを作って、**ステップ**という名前を付けてください。最小値を0、最大値を100に設定します。これで、**ステップ**という名前を使って、スライダーの現在の値を報告させることができます。

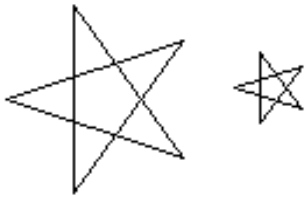
```
くりかえす　5　「前へ　ステップ　右へ　144」
```

スライダーの値を70にしてから、この命令を実行してみてください。星の1辺は70ドットに描かれます。

スライダー名「**ステップ**」は変数ではないので、コロン（：）は付けません。

ステップというスライダーを作るということは、結果として、そのスライダーの値を報告する、**ステップ**という名前の**状態変数**を作ることを意味することに注意してください。
しかしながら、スライダーを使ってステップの値を設定し、命令を入力するのは、手順のインプットを利用するより手間がかかるように思われます。もっと、実用的な例を紹介しましょう。

・カメの設定ボックスに**くりかえす　5　「前へ　ステップ　右へ　144」**という命令を設定します。
・スライダーで星のサイズを調整します。
・カメをクリックして、星を描きます。



星を作るカメからさらに進んで、多角形を作る別のカメを作ってみましょう。

多角形の辺の数を表す、**辺の数**という名前のスライダーを作ってください。新しいカメの設定ボックスに、次の命令を設定します。

```
くりかえす　辺の数　「前へ　ステップ　右へ　360　/　辺の数」
```

これで2つのスライダーを利用して、様々な多角形を作ることができます。試してみてください。

(4) 繰り返しとリカージョン（再帰）

リカージョン（再帰）

前項までの円、弧、多角形に関する手順では「くりかえす」を使って一定の回数だけ「前へ」や「右へ」を使いました。しかし、命令を繰り返し実行するもう1つの方法として、**リカージョン(再帰)**があります。これは**手順の最後で自身を子手順として呼び出す方法**です。例えば、次の**連続**という手順は、応用の仕方によっては驚くほど美しい図形を描くことができます。

```
手順は　連続　：幅　：角度
前へ　：幅
右へ　：角度
連続　：幅　：角度
終わり
```

連続を試してみましょう。コマンドセンターから次のように入力し、実行します。

```
連続　50　120
```

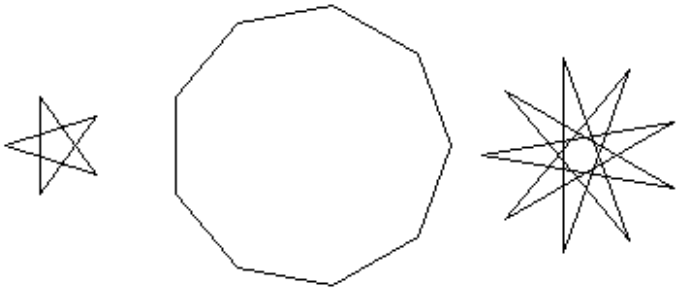
カメは、[**すべてを止める**]  をクリックしない限り、三角形を描き続けます。

この種の繰り返しのデメリットは、無限に続く点ですが、**連続**だけで、多数の異なる形を簡単に描くことができます。

```
連続　50　144
```

連続　50　40

連続　100　160



リカージョン（再帰）は、手順がその中で自身を呼び出す機能を意味するコンピュータ用語です。

上の**連続**は、その中で自身を呼び出していますから、***再帰的***です。

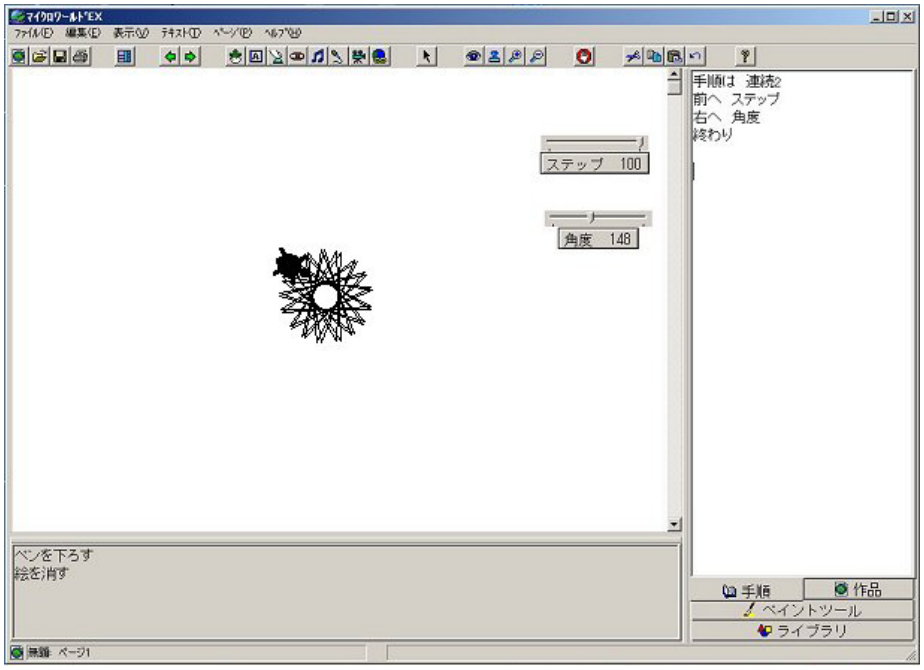
a．「無限に」とリカージョン

連続の手順は、最も簡単な形式のリカージョン（再帰）の例です。マイクロワールドEXでは、再帰手順の代わりに、**無限に**コマンドを使うか、カメまたはボタンのダイアログボックスで[無限に］を選択することによって命令を無限に繰り返し実行することができます。例えば、**連続**は次のように書くこともできます。
この**連続2**のステップと角度はそれぞれ**ステップ**と**角度**というスライダー名です。

手順は　連続2
前へ　ステップ
右へ　角度
終わり

手順を試す前にページ上に2つのスライダーを作成してください。
ステップは「(3) スライダーを使った多角形」で作ったものと同じ、最小値を0、最大値を100に設定します。**角度**は最小値を0、最大値を360に設定します。**連続2**を手順タブエリアに入力したら、カメの設定ボックスで**連続2**を命令として設定し、［無限に］欄にチェックマークを付けます。

この後、マウスでスライダーのボタンを適当な値に設定し、カメをクリックします。



再帰手順の**連続**と同じ結果が得られましたね。

この方法の短所は、[**すべてを止める**]  をクリックすることによってしか、繰り返しを止められないことです。

b．リカージョンでらせんを描く

リカージョン（再帰）は、高度なプログラミングで使うことができる非常に強力なツールです。

連続という手順を、もう少し発展させてみます。すなわち、自身を呼び出す行を実行するたびに**ステップ**というインプットに一定の増分が加えられるように手順を変更します。

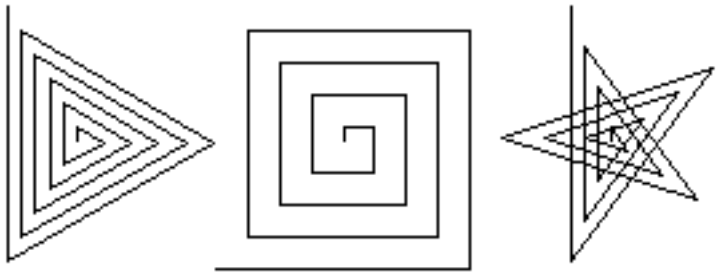
```
手順は　連続模様　：ステップ　：角度
前へ　：ステップ
右へ　：角度
連続模様　：ステップ　+　6　：角度
終わり
```

次のようなインプット値で試してみましょう。

```
連続模様　5　120

連続模様　5　90

連続模様　5　144
```



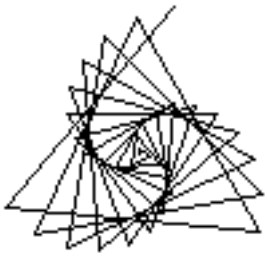
連続という手順では、カメは閉じた形を描きました。すなわち、前進し、向きを変えて、スタートした位置に戻ります。ただし、カメが向きを変える角度を0度や360度（または360の倍数）にすると、1周するたびに例外が発生します。この場合は、直線を描きます。

一方、**らせん**は開いた形です。カメがスタートした位置に戻ることはありません。らせんを描く場合、カメは1周するたびに前進する歩数（ステップ）を多くして、スタートした位置から遠ざかります。
3つ目のインプットを追加することによって、**連続模様**が自身を呼び出すたびに増やす**ステップ**の数を調節することができます。

```
手順は　連続模様　：ステップ　：角度　：増加
前へ　：ステップ
右へ　：角度
連続模様　：ステップ　+　：増加　：角度　：増加
終わり
```

これで、辺を大きくしていく割合を変更することができます。

連続模様　2　125　3



連続模様には、多くのバリエーションがあります。例えば、歩数（ステップ）を増やす代わりに、減らしてみてください。あるいは、**連続模様**の頂点だけ異なる色になるようにしてみてください。

c．ストップルール

ストップルールは、再帰手順を止めるために設定する条件のことです。
再帰手順にストップルールを設定することは重要です。再帰手順にストップルールを組み込むことによって、さらに複雑な形や美しい形の再帰を実現することができます。

連続模様に組み込むべきストップルールは次のようなものです。
：ステップというインプットが非常に大きくなったとき、例えば、200を超えたときに、止まるようにするのです。
ここでは、**止まる**というコマンドを使って実行中の手順を止めます。ストップルールは、次のような書式になります。

ロゴでは、**if文（もし～**で始まる仮定文）を***条件式***と呼びます。

マイクロワールドEXは、**連続模様**が自身を呼び出すたびに**：ステップ**値が200を超えているかどうか監視します。
この条件が**真**になると、**連続模様**は停止します。
マイクロワールドEXのボキャブラリーには、演算記号があります。
大なり記号（>）もそのような***テストレポータ***です。テストレポータは真（しん）または偽（うそ）を返します。
=（等号）、**>**（大なり記号）、**<**（小なり記号）はすべてテストレポータです。

上記のストップルールを**連続模様**に追加すると、次のようになります。

```
手順は　連続模様　：ステップ　：角度　：増加
もし　：ステップ　＞　200　「止まる」
前へ　：ステップ
右へ　：角度
連続模様　：ステップ　＋　：増加　：角度　：増加
終わり
```

連続模様の場合は、どのような基準でストップルールを決めるか、判断が難しかったかもしれません。

しかし、概ねリカージョンを使って描かれるような模様はスタート位置に戻ったときに1つの図形を完成します。
複雑な多角形の場合、カメは360度の回転を何回か行いますが、簡単な多角形の場合、カメが完全に1回転（360度の回転）すると一つの図形を描きます。従って1つの方法として、カメが元の向きに戻ったときに描くのを止めるストップルールを作ります。**連続模様**を一部書きなおして、別のストップルールで止めてみましょう。

そのためにはスタートするときのカメの向きを知らなければなりません。
向きを変えるたびに、そのときのカメの向きと比較します。**連続模様**の3つめインプットを、変えることによって、スタートする前のカメの向きを調べることができます。現在の向きがスタートするときの向きと同じになると、手順の実行は止まります。このストップルールは、次のような形になります。

```
もし　向き　＝　：スタート 「止まる」
```

このストップルールは、**右へ**の後に挿入する必要があります。これ以外の場所に挿入すると、描くのを始める前であってもただちに停止してしまいます。
別の手順名「**模様**」を作って試してみましょう。
(コピー＆ペーストを使って前の手順のコピーを作成し、手順名など必要な部分を修正すると簡単です)。

```
手順は　模様　：ステップ　：角度　：スタート
前へ　：ステップ
右へ　：角度
もし　向き　＝　：スタート 「止まる」
模様　：ステップ　：角度　：スタート
終わり
```

向きは、カメの現在の向きを返すレポータです。

コマンドセンターから実行してみましょう。

```
模様　50　120　　向き
```

これで手順としては、一応完成です。しかし、コマンドセンターから3つもインプットを入力するのはエレガントではないようです。
次の**模様実行**という手順は、**模様**を子手順にして**ステップ**と**角度**だけ入力すれば**模様**を実行できるように作成してあります。

```
手順は　模様実行　：ステップ　：角度
模様　:ステップ　:角度　向き
終わり
```

コマンドセンターから次のように実行してみましょう。

```
模様実行　50　120
```

d．条件文について

条件文は次ように表されます。

もし条件が真ならば、［あることを行いなさい］

条件を調べ、その条件が真の場合に命令リストを実行するコマンドとして、**もし(if)**と**もしどちらかを(ifelse)**の2つがあります。**もしどちらかを**は、条件が偽の場合に代替命令リストを実行します。

if(もし～)命令については､次の点に注意してください。

もしは条件を調べ、その条件が**真**の場合にのみ命令リストを実行します。条件が**偽**の場合は、命令リストを無視して手順の次の命令に進みます。
手順には、複数の条件文を使うことができます。この場合、複数の if 文は前から順に調査されます。
ストップルールの条件式には、**止まる**、**これを止める**、**皆止まる**、のいずれかを必ず含めます。条件式にこれらのどのコマンドも含まれていない場合、ストップルールは正しく動作しません。

(5) 複雑な再帰パターン

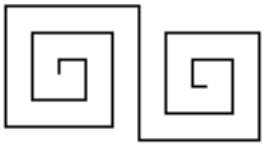
これまでに紹介した再帰手順はすべて、その手順の最後で自身の手順を呼び出していました。これは、命令が無限に実行される一般的なリカージョンと見なすことができます。

複雑な再帰手順になると、再帰行の後にも命令が存在します。簡単な例として、**連続模様**を変更したものを見てみましょう。

```
手順は　四角模様　：ステップ
もし　：ステップ　＞　50 「止まる」
前へ　：ステップ
右へ　90
四角模様　：ステップ　＋　5
前へ　：ステップ
左へ　90
終わり
```

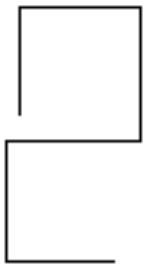
再帰行の後に、**前へ**と**左へ**の2つの命令があることを注目してください。

四角模様　5



どのようになるか確認するには、四角模様に大きな値のインプットを与えてみます。

```
四角模様　40
```



:ステップが50より大きくなって、子手順としての**四角模様**が終了しても、親の手順がただちに停止することはできません。命令の残りの部分の実行を完了する必要があるのです。
四角模様はそれぞれ独自の**:ステップ**値を保持します。つまり、各**四角模様**は独自の**:ステップ**値で**前へ**を実行し、**左へ　90**を実行します。
以下では、四角模様の命令がどのように実行されるかを見てみます。

四角模様 40 (ステップは 40) もし 40 > 50 [止まる] (40 > 50 か? 嘘) 前へ 40 右へ 90 四角模様 40 + 5			
→	四角模様 45 (ステップは 45) もし 45 > 50 [止まる] (45 > 50 か? 嘘) 前へ 45 右へ 90 四角模様 45 + 5		
	→	四角模様 50 (ステップは 50) もし 50 > 50 [止まる] (50 > 50 か? 嘘) 前へ 50 右へ 90 四角模様 50 + 5	
		→	四角模様 55 (ステップは 55) もし 55 > 50 [止まる] (55 > 50 か? 本当) 四角模様 停止
		前へ 50 左へ 90 四角模様 50 停止	←
	前へ 45 左へ 90 四角模様 45 停止	←	
前へ 40 左へ 90 四角模様 40 停止	←		

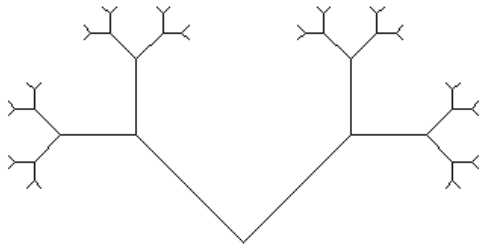
これで、ルールが巻き戻されます。

子手順が呼び出されると、親手順はこの手順が止まるのを待ち、その後、呼び出しの次の命令に進みます。

このルールは簡単に思えるかもしれませんが、以下の場合には、複雑になる可能性があります。

- 同じ手順に複数の再帰呼び出しがある場合
- すべての再帰手順がそれぞれ独自のインプット値を保持する場合
- 動作の順序が予想とは逆の順序の場合。これは最後に呼び出された手順が最初に止められるためです。

こうした構造で有名なのは、バイナリツリーです（1982年、Abelsonから抜粋）。バイナリツリーの再帰の仕方は、すべての先端に小さなツリーが存在するV形で表されます。小さなツリーの先端にはさらに小さなツリーが存在します。例えば、次のような具合です。



左図の最も大きいV形を描く命令は、次のようになります。

```
左へ　45
前へ　：長さ
後ろへ　：長さ
右へ　90
前へ　:長さ
後ろへ　：長さ
左へ　45
```

この再帰ツリーのすべての先端にサイズが半分のツリーを持たせるとしましょう。
このため、すべての**前へ**の後に、長さを半分にして自身を呼び出す手順、**Tree**を作ります。

```
手順は　Tree　：長さ
左へ　45
前へ　：長さ
Tree　：長さ　/　2
後ろへ　：長さ
右へ　90
前へ　：長さ
Tree　：長さ　/　2
後ろへ　：長さ
左へ　45
終わり
```

しかしながら、この手順にはストップルールがないため、呼び出されるすべての**Tree**が無限に繰り返され、どれも命令の実行が終了しません。このため、簡単なストップルールを挿入して、**：長さ**が非常に小さくなったときに**Tree**を止めるようにします。

```
手順は　Tree　：長さ
もし　：長さ　<　2　[止まる]
左へ 45
前へ　：長さ
Tree　：長さ　/　2
後ろへ　：長さ
右へ　90
```

```
前へ　：長さ
Tree　：長さ　/　2
後ろへ　：長さ
左へ　45
終わり
```

Treeを試してみてください。コマンドセンターから次のように入力します。

```
Tree　40
```

Tree　12を入力した場合の内部的な動きは以下のようになります。

Tree 12 もし 12＜2 [止まる] (12 ＜ 2 か? 嘘) 左へ 45 前へ 12 Tree 12 / 2			
→	Tree 6 もし 6＜2 [止まる] (6 ＜ 2 か? 嘘) 左へ 45 前へ 6 Tree 6 / 2		
	→	Tree 3 もし 3＜2 [止まる] (3 ＜ 2 か? 嘘) 左へ 45 前へ 3 Tree 3 / 2	
		→	Tree 1.5 もし 1.5＜2 [止まる] (1.5 ＜ 2 か? 本当) Tree 1.5 止まる
		後ろへ 3 右へ 90 前へ 3 Tree 3 / 2	←
		→	Tree 1.5 もし 1.5＜2 [止まる] (1.5 ＜ 2 か? 本当) Tree 1.5 止まる
		後ろへ 3 左へ 45 Tree 3 停止	←
	後ろへ 6 右へ 90 前へ 6 Tree 6 / 2	←	
	→	Tree 3 もし 3＜2 [止まる] (3 ＜ 2 か? 嘘) 左へ 45 前へ 3 Tree 3 / 2	
		→	Tree 1.5 もし 1.5＜2 [止まる] (Tree 1.5 止まる)
		後ろへ 3 右へ 90 前へ 3 Tree 3 / 2	←
		→	Tree 1.5 もし 1.5＜2 [止まる] Tree 1.5 止まる
		後ろ 3 左へ 45 Tree 3 停止	←
	後ろへ 6 左へ 45 Tree 6 停止	←	

一般に、リカージョンを使うと手順が簡潔でエレガントなものになります。
リカージョンのプロセスを完全に理解するのは難しいことですが、この概念に慣れておくことは、プログラミングにおいて、とても有利なことです。
Treeは一見すると簡単に見えますが、すべての枝を辿るのはかなり困難です。**Tree**では、状態の透明性（すなわち、カメの開始と終了状態が同じ）が特に重要です。最後の**後ろへ**と**左へ**は、すべてのツリーが正しい位置に描かれるよう、カメを最初の位置と向きに戻します。

3. プログラムからのテキスト操作

■この節の内容はオンラインヘルプからも見ることができます。
EX　　→メニューバーの［ヘルプ］をクリック→［リファレンス」
EX（ジュニア）　→メニューバーの［ヘルプ］をクリック→［リファレンス」

・例示された命令は、コマンドセンターに入力して試します。Enterキーを押すことで命令が実行されます。（＊コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter キーを押します。）
・例示された手順は、手順タブエリアに入力します。コマンドセンターに手順名を入力し、Enter キーを押すことで手順が実行されます

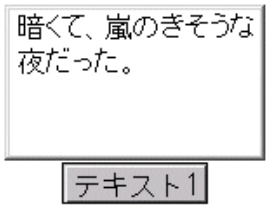
マイクロワールド EX では、マウスやメニューを使ってテキストを操作できるばかりでなく、プログラムの中でテキストボックスを操作することもできます。ここでは、テキストボックスにおけるカーソル移動や、テキストの挿入・削除、フォントの変更などについて説明します。
（＊これから取り上げる基本用語は、テキストボックス内のテキストに対してのみ使用することができます。）

(1) テキストボックスの内容

テキストボックス名は、1つの文字列（ロングワード）の形式でその内容を報告します。
リストにしたものを使えば、文字列（例えば、上記のテキストボックスの内容）をリストに変換することもできます。テキストボックスの内容を、文字列ではなく、一塊りの行と見なす特殊なプリミティブもあります。
コンピュータの世界では、単語は**文字列**（ロングワード）と呼ばれます。実際、1つ1つの単語は文字列ですが、ここで「文字列」という言葉を使っているのは、個々の単語の集まりに見えるものが、実際には、空白を含めた個々の文字の集まりであることに注目するためです。

a. 文字列とテキストボックス

ここでは、**テキスト1**という名前のテキストボックスを例に説明をします。テキストボックス名である**テキスト1**は、それ自体がレポータとして機能します。
テキスト1というレポータは1つの文字列（ロングワード）として、テキストボックス内の空白を含むすべての文字を報告します。
まず、**テキスト1**に　”暗くて、嵐のきそうな夜だった。”と入力します。



コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　テキスト 1
```

次の行に、下のように表示されます。
暗くて、嵐のきそうな夜だった。

文字として数えられるものをすべて数えると、文字は 15 個あります。(句読点も含めます)。
コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　数　テキスト 1
```
1 7

注　：この 17 という数字には、行末の Enter キーのコード（2 文字分）が含まれています。

テキストボックスの内容が 1 つの文字列として扱われていることを確かめてみます。
コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　最初　テキスト 1
```

暗

最初というレポータによって文字が抽出されました。マイクロワールド EX は、**テキスト 1** の文字列を、1 つの長いワードとして捉えていることがわかります。

注:　：空白を含むロングワードは、縦線（|）を使って作ることもできます。

```
下へ書く　"|暗くて、嵐のきそうな夜だった。|
```

b. 文字列のリスト化

テキストボックスの内容に何らかの処理を行う場合は、1 つの文字列であるよりも、複数の単語のリストとして扱った方が便利です。
リストにしたものというレポータは、空白を含む文字列を単語のリストに変換して、報告します。
まず、**テキスト 1** の文を次のように変更します。　"暗い　嵐の　夜だった。"
コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　数　テキスト 1
```

次の行に、下のように表示されます。
1 5

同様にして次の命令を試します。

```
下へ書く　リストにしたもの　テキスト1
```
暗い　嵐の　夜だった。

何も変化がないように見えますが、要素の個数を数えると、違いが出てきます。

```
下へ書く　数　リストにしたもの　テキスト1
```

3

テキスト1は改行コードも含めて、15文字が報告されましたが、**リストにしたもの　テキスト1** は、3を報告しました。

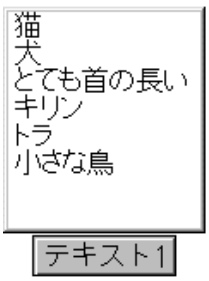
```
下へ書く　最初　リストにしたもの　テキスト1
```

暗い

ここでは、マイクロワールドEXが ”暗い　嵐の　夜だった。” という文字列を、”暗い ”、”嵐の ”、”夜だった。”
という3つの要素をもったリストとして、扱っていることがわかります。

c. テキストボックスの行操作

テキストボックスの内容を文字列としてではなく、行の集まりと見なす特殊なプリミティブがいくつかあります。
ここでは、**テキスト1** の内容が次のようになっているとします。



行の数は、インプットとしてテキストボックス名を取り、テキストボックスの行数を報告します。
コマンドセンターから次のように入力し、Enterキーを押して実行します。

```
下へ書く　行の数　”テキスト1
```

次の行に、下のように表示されます。

5
ここには、6行書かれているように見えます。しかし、3行目と4行目の ”とても首の長いキリン ”は、実際には1つの行で、テキストボックスの表示幅に収まらなかったために、2行に分けて表示されているのです。テキストボックスを広げると、5行であることが分かります。
マイクロワールドEXの基本用語では、**論理行**だけ認識します。論理行とは、Enterキーを押したときに生成されるコード（Enterキーのコード）によって終了している行です。

テキスト1の前の引用符（ ”）は、インプットとして名前を与えるようにという指示です。テキスト1を引用符なしで使うと、そのテキストボックスの内容が返されます。
行の項目は、行番号を指定することによって、特定の行を報告します。

下へ書く　行の項目　3　”テキスト1　　…3は行番号です。

とても首の長いキリン

行のどれかはテキストボックスからランダムに選択した行を報告します。

下へ書く　行のどれか　”テキスト1

犬

下へ書く　行のどれか　”テキスト1

小さな鳥

行の項目と**行のどれか**は、リストではなく、単語またはロングワードを報告します。

d. 行操作の基本用語

例は**テキストボックスの行操作**の**テキスト1**の内容に基づいています。

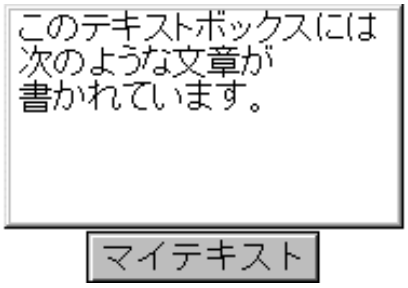
基本用語	働き	例	出力
行の数	テキストボックスの行数（論理行）を報告します。	行の数　”テキスト1	*5*
行の項目	指定された行番号の行を報告します。	行の項目　1　”テキスト1	*犬*
行のどれか	ランダムに選択した行を報告します。	行のどれか　”テキスト1	*小さな鳥*

(2)カーソルの移動

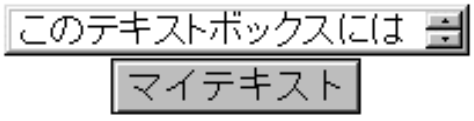
ここでは、カーソルの移動に関係する基本用語をまとめています。これらの基本用語は、**文末か**以外はすべてコマンドです。**文末か**は真偽値を報告するレポータです。

基本用語	種類	働き
一字右へ	コマンド	カーソルを次の文字に移動します。
一字左へ	コマンド	カーソルを前の文字に移動します。
一字上へ	コマンド	カーソルを1行上に移動します。
一字下へ	コマンド	カーソルを1行下に移動します。
文頭へ	コマンド	カーソルをテキストの先頭に移動します。
文末へ	コマンド	カーソルをテキストの末尾に移動します。
行頭へ	コマンド	カーソルを行の先頭に移動します。
行末へ	コマンド	カーソルを行末に移動します。
文末か	レポータ	カーソルがテキストの終わりにあるかどうかを調べます。

一字下へコマンドを使い、テキストボックスの内容を1行ずつ表示する手順を作ってみます。例として取り上げるテキストボックスの名前は**マイテキスト**です。



このテキストボックスは3行のテキストが含まれています。テキストボックスのサイズを変更することによって、1行目だけ表示されるようにしましょう。



カーソルをテキストの先頭に移動します。

文頭へ

コマンドセンターに次の命令を入力すると、1行ずつ表示されます。

くりかえす　3　「一字下へ　待つ　5」

待つを使って、テキストを読む時間を確保しましょう。インプットの数値は自由に調整することができます。テキストボックス内にカーソルは表示されないことに注意してください。

テキストボックスに多数の行が含まれている場合は、**くりかえす**のインプットとして、**行の数**を使うと便利です。
行の数などの行操作の基本用語は、論理行、すなわち、Enter キーのコードで終了している行に対して働くのに対し、**一字下へ**などのカーソル移動に関する基本用語は物理行、すなわち、画面の表示行に対して働きます。行操作の基本用語を使うときは、テキストボックスの各行が Enter キーのコードで終了しているかどうか確認してください。

```
文頭へ
くりかえす　行の数　"マイテキスト　「一字下へ　待つ　5」
```

この命令行は、"マイテキスト "という名前の部分を、**テキストボックス名**というレポータに置き換えることによって、ずっと汎用性の高いものにすることができます。
テキストボックス名は、アクティブなテキストボックスの名前を報告します。
テキストボックスをアクティブにするには、画面上でテキストボックスをクリックするか、**カメは**コマンドを使って切り替えることができます。次の手順は、テキストボックスの各行を順に表示するツールとして利用することができます。

```
手順は　スクロール
文頭へ
待つ　5
くりかえす　行の数　テキストボックス名　「一字下へ　待つ　5」
文頭へ
終わり
```

a. テキストの挿入と削除

書くと書き入れる

書くは文字を書いたあとに改行しますが、**書き入れる**は、カーソル位置に文字や数字を書き、改行しません。
一字消すは、カーソルの右側の文字を削除します。
これらの2つのコマンドを使い、テキストを横方向に1文字ずつスクロールすることができます。テキストボックスのテキストが1文字ずつ削除されるか、テキストボックスに文字が1文字ずつ挿入されます。

横方向のスクロール手順

例えば、**ことわざ**というテキストボックスに次のテキストを書いておき、1文字づつ消してみましょう。

後で簡単に元に戻せるように、テキストボックスの内容を、**”テキスト保存**という変数に保存しておきましょう。

```
名前は　”テキスト保存　ことわざ
```

一字づつの削除

```
文頭へ
くりかえす　数　ことわざ 「一字消す　待つ　2」
```

この命令は、前項の「カーソルの移動」にあった「スクロール」という手順に似ていますが、ここでは**行の数**ではなく、**数**を使って、**ことわざ**の文字数を数えます。
一字下へではなく、**字を消す**を使ってカーソルの右側の文字を削除します。

変数からの復元

```
書き入れる　：テキスト保存
```

テキストを削除しても、**書き入れる**を使って復元することができます。
これらを手順にまとめると次のようになります。

```
手順は　ロール
名前は　”テキスト保存　ことわざ
文頭へ
くりかえす　数　：テキスト保存 「一字消す　待つ　2」
書き入れる　：テキスト保存
終わり
```

文字の挿入

1文字ずつのテキスト挿入は、**最初**と**最初以外**を使った**再帰手順**を作ることによって実現することができます。挿入するものがなくなるまで、テキストを１文字ずつ挿入します。
ページに空のテキストボックスをひとつ作ります（名前は「テキスト1」とします)。
手順タブエリアに次の手順を作ります。

```
手順は　一個づつ　：文字
もし　空か　：文字 「止まる」
テキスト1、 書き入れる　最初　：文字　待つ　1
一個づつ　最初以外　：文字
終わり
```

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
一個づつ　ことわざ
```

「テキスト1」の中に、ことわざの内容が書かれ、自動的に止まります。
一個づつでは、「ことわざ」の中の文字がなくなるまで１文字ずつ削除することによって

文字数が減っていきます。ストップルールは、「単語が空になったら、停止しなさい」という意味です。
ここでのインプットには、引用符を付けないテキストボックス名を使います。これはテキストボックス名の内容をインプットにするからです。
テキストボックスに書かれていない文字を一個づつで表示させる場合は、直接インプットにします。

```
一個づつ　”あいうえおかきくけこ
```

親手順を作ることによって、変数にテキストを保存したうえで、テキストを削除し、子手順から一個づつを呼び出すことができます。

```
手順は　入力
名前は　”テキスト保管　ことわざ
字を消す
一個づつ　：テキスト保管
終わり
```

b．フォントとスタイルの変更

基本用語を使って、テキストボックス内のテキストのフォントやスタイルを設定することもできます。
テキストのフォントを変更するには、まず、テキストを選択する必要があります。
選ぶを実行するとは、マイクロワールドEXは選択を開始します。この後、カーソルを動かして、開始位置からカーソル位置までのすべてのテキストを選択します。


```
文頭へ
選択
文末へ
```

この命令は、テキストボックス内のテキスト全体を選択します。
テキストを選択すると、フォントとサイズ、スタイル、色を変更することができます。

```
フォントは　"Arial
フォントのサイズは　20
フォントのスタイルは　”ふとじ
字の色は　”あか
```

注:　：複数の単語からなるフォント名を使う場合は、次の例に示すように、フォント名を縦線で囲む必要があります。

```
フォントは　”¦Times New Roman¦
```

ロゴには、フォントのサイズとテキストの色を調べるレポータがあります。また、**選んだもの**というレポータを使い、選択されているテキストを報告させることもできます。

c．テキスト選択の基本用語

基本用語	種類	働き	例
選ぶ	コマンド	選択を開始します。	文頭へ　選ぶ　文末へ （テキストボックス内の文の全内容を選択します。）
選ばない	コマンド	**選ぶ**または**探す**による選択状態を解除します。	文頭へ　選ぶ　文末へ選ばない (選択されている状態を元に戻します。)
選んだもの	レポータ	現在選択されているテキストを報告します。	リストにしたもの　選んだもの (選択されているテキストをリストに変換します。)

d．テキスト設定の基本用語

基本用語	種類	働き	例
フォントは	コマンド	フォントを設定します。	フォントは　"Arial
フォントのサイズは	コマンド	フォントのサイズを設定します。	フォントのサイズは 20
フォントサイズ	レポータ	選択されているテキストまたはカーソル位置のテキストのフォントサイズを報告します。	フォントのサイズはフォントのサイズ　+　4
フォントのスタイルは	コマンド	テキストのスタイルを 標準、斜体、太字、太字 斜体、に設定します。	フォントのスタイルは　”ふとじ
字の色は	コマンド	テキストの色を指定された番号または名前に設定します。	字の色は　”あか
字の色	レポータ	選択されているテキストまたはカーソル位置のテキストの色番号を報告します。	字の色は　字の色　+　10

注　:**字の色は**と**フォントのスタイルは**で指定する色の名前やスタイル名はひらがなを使用してください。

e．サンプルプログラム

テキストボックスのテキストを1文字ずつ連続的に大きくして、色を変える再帰手順を紹介します。

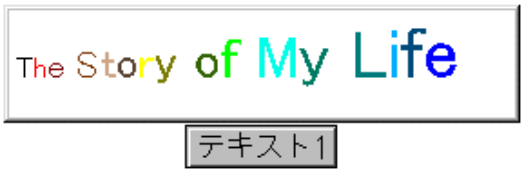
```
手順は　グロー　：サイズ　：色
もし　文末か　「止まる」
選ぶ　一字右へ
フォントのサイズは　：サイズ
字の色は　：色
選ばない
グロー　：サイズ　＋　1　：色　＋　5
終わり
```

ストップルールで**文末か**を使い、テキストの終わりを調べていることに注意してください。**選ぶ、一字右へ**は、テキストを1文字ずつ選択します。**選ばない**は、文字のサイズと色を変えた後でいったん選択を解除します。
再帰呼び出しのたびに、フォントサイズは1ずつ大きくなり、テキストの色番号は5ずつ増えます。
親手順を作り、グローを子手順として実行するようにしてみましょう。

```
手順は　文字グロー
グロー　フォントのサイズ　字の色
終わり
```

テキストボックスに”The Story of My Life ”と入力してから、実行してみましょう。

```
文頭へ
文字グロー
```

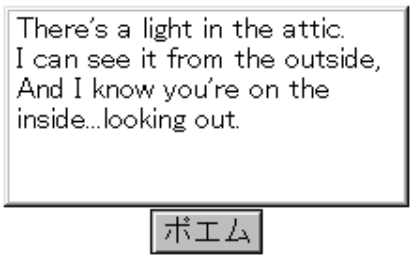


(3)テキストボックス内のテキスト編集

a. テキスト編集の基本用語

基本用語	種類	働き	例
探す	コマンド	指示された単語または文字を、検索して、選択状態にします。	探す　”a
見つけたか	レポータ	直前に行われた「探す」が成功したかどうかの結果を報告します。	下へ書く　見つけたか
切り取る	コマンド	選択されているテキストを切り取って、クリップボードにコピーします。	選ぶ　行末へ　切り取る
コピー	コマンド	選択されているテキストをクリップボードにコピーします。	文頭へ　選ぶ　文末へ　コピー
貼りつける	コマンド	カーソル位置にクリップボードの内容を貼り付けます。	テキスト1、貼りつける
クリップボード	レポータ	現在のクリップボードの内容を報告します。	設定する　”テキスト１"じ　クリップボード

テキスト編集の基本用語を使って、テキストの検索やコピー、カット＆ペーストを行うことができます。
ここでは、**ポエム**というテキストボックスを例に説明します。



このテキストボックスの編集作業を行う前に、このテキストをグローバル変数に保存しておきます。

名前は　”オリジナル　ポエム

ポエムの元の内容を復元するには、次の命令を使います。

設定する　”ポエム　”じ　：オリジナル

検索

探すは、単語または文字を、カーソルのある位置から文末方向に検索します。見つかると選択状態にします。

```
文頭へ
探す　"a
```

2つの単語を検索するには、縦線（|）を使うか、**ワード**を使って2つの単語を1つにする必要があります。
ワードは、そのインプットを1つの単語にまとめ、その単語を報告します。

```
文頭へ
探す　"|the attic|
```

または、

```
文頭へ　探す （ワード 「the attic」）
```

探すに対する括弧は、**ワード**のインプットが1つのときでも必要です。

見つけたかは、直前に「**探す**」を使って行われた検索が成功したかどうかを報告します。

```
探す　"a
下へ書く　見つけたか
本当
```

コピー&ペースト

テキストを切り取ったり、コピーするには、そのテキストを選択する必要があります。**選ぶ**で選択してから**コピー**を使います。

```
文頭へ
選ぶ　文末へ
コピー
```

クリップボードは、クリップボードの内容を1つの文字列（ロングワード）として報告します。

```
下へ書く　クリップボード
```

貼りつけるは、カーソル位置にテキストを貼り付けます。

b . サンプルプログラム

単語の置き換え

ここでは、**探す**と**見つけたか**を使って、置換の手順を作ります。
1つ目のインプットの単語をすべて2つ目のインプットの単語に置き換えます。

```
手順は　置換　：単語　：新規
探す　：単語
もし　反対　見つかったか　「止まる」
書き入れる　：新規
置換　：単語　：新規
終わり
```

ポエムテキストボックスを使って試してみます。コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
文頭へ
置換　”I　”you
```

この手順には問題があるようです。単語内の文字も含めて、"I" に一致するものをすべて置換してしまうことです。同様に、"you" がスペースなしで挿入されてしまいます。
主語としての "I" をスペースの入った "you" に置き換えるには、インプットを、縦線（|）を使って文字とスペースとの組み合わせにします。テキストボックス内の文字をいったん全部消し、復元してから試してみましょう。
コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
字を消す
書き入れる　：オリジナル
文頭へ
置換　”I |　”you |
```

特定の文字削除

探すと**見つけたか**を使って、テキストから特定の文字を削除する手順を紹介します。

一字消すはカーソルの右側の文字を削除しますから、**探す**を実行した後、検索された文字を削除するには、**一字左へ**を使ってカーソルの位置を戻す必要があります。

```
手順は　消去　：文字
探す　：文字
もし　反対　見つけたか　「止まる」
一字左へ　一字消す
消去　：文字
終わり
```

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
文頭へ　消去　”a
```

(4) 暗号化と解読

テキスト処理と従来の基本用語を組み合わせることによって、メッセージを簡単に暗号化することができます。
簡単な暗号の例として、シーザーの暗号があげられます。ジュリアス・シーザーはアルファベットを3つ右にシフトすることによって交信しました。このシーザーの暗号では、AがD、BがE、CがFというようになります。

アスキー

コンピュータでは、すべての文字がアスキー（ASCII）コードという数値コードに対応づけられています。ここでは簡単な例として、英大文字だけ利用します。
コマンドセンターから次のように入力し、Enterキーを押して実行します。

```
下へ書く アスキー "A

6 5
```

マイクロワールドEXは、文字の**アスキー**コードを報告します。コンピュータプラットフォームには、標準で英字が組み込まれています。

字

字というレポータは、数値のインプットを取り、その数値（アスキーコード）に対応した英字を報告します。

```
下へ書く　字　65

A
```

英大文字のアスキーコードは65から90の範囲です。

```
下へ書く アスキー "Z

9 0
```

下の**code**という手順は、インプットとしてアスキーコードの数値を受け取り、その数値を3つずつ大きくしていきます。**code**は実行中にコードの上限も調べます。数値コードが90より大きくなった場合、英大文字のアスキーコード範囲を超えてしまうので、26を引くことによって英大文字のアスキーコード範囲に戻します。

```
手順は　code　：文字
名前は　”文字　：文字　+　3
もし　：文字　>　90 「名前は　”文字　：文字　-　26」
表示　：文字
```

```
終わり

下へ書く　code 65
```

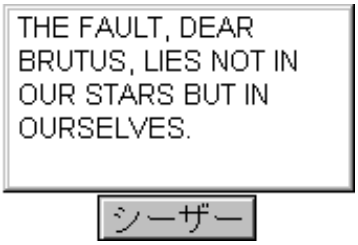
6 8

codeには、**表示**が使われています。このことは、**code**という手順がレポータとして定義されていることを意味します(P.○○**アウトプットを持つ手順**を参照)。

```
下へ書く　code 90
```

6 7

code手順を使い、テキストの１つ１つのアルファベットを変換する手順を作成してみましょう。まず、次のシーザーというテキストボックスを使って考えます。



選ぶと**一字右へ**は、英字１文字を選択します。

```
文頭へ
選ぶ　一字右へ
```

"T"が選択されます。

```
下へ書く　アスキー　選んだもの
```

8 4

選んだものは、選択されているものを１つの単語として報告します。

```
下へ書く　code　アスキー　選んだもの
```

8 7

次の命令は、シフトすることによって、選択されている英字（T）を3つ先の英字（W）に置き換えます。

```
書き入れる　字　code　アスキー　選んだもの
```

この命令は、次のように処理されます。

書き入れるコマンドの対象は**字**です。
字は、そのインプットを調べます。インプットは**Code**です。
Codeは、その手順のインプットを調べます。インプットは**アスキー**と**選んだもの**です。
選んだものは、**アスキー**に**T**を報告します。
アスキー　選んだものは、**code**に84を報告します。
code　アスキー　選んだものは、**字**に87を報告します。
字　87は、**書き入れる**にWを報告します。
書き入れるは、現在のカーソル位置にWを挿入します。

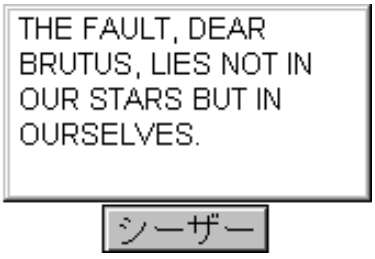
次の暗号は、テキストに含まれるすべての英字を3文字シフトします。

```
手順は　暗号
もし　文末か　「止まる」
選ぶ　一字右へ
書き入れる　字　code　アスキー　選んだもの
暗号
終わり
```

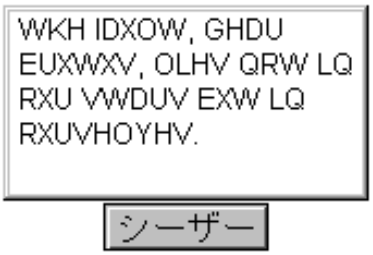
文末かのストップルールで、カーソルがテキストの終わりにきたかどうかを調べていることに注意してください。
この**暗号**は、完全ではありません。空白やピリオド（.）も暗号化されてしまいます。これらのアルファベット以外のものには何の処理もしないようにしましょう。
codeを改良することによって、空白、.、,、;、?、!、:　などのアスキーコードを見つけることができます。
新しい**code**では、**あるか**を使って、選ばれた文字のコードが、空白、.、,、;、?、!、:のコードかどうかを調べます。

```
手順は　code　：文字
もし　あるか　：文字　「32 33 44 46 58 63」「表示　：文字」
名前は　”文字　：文字　+　3
もし　：文字　>　90　「名前は　”文字　：文字　-　26」
表示　：文字
終わり
```

これで、暗号が正しく動作するはずです。




```
文頭へ
暗号
```



このテキストを解読する手順(2つ)は、**暗号**の実行結果を取り消します。**解読準備**はインプットとして英字のアスキーコードを取り、そのコードから3を引きます。**解読**は暗号化されたアルファベットを元のアルファベットに置き換えます。

```
手順は　解読準備　：文字
もし　あるか　：文字 「32 33 44 46 58 63」「表示　：文字」
名前は　”文字　：文字 - 3
もし　：文字 < 65 「名前は　”文字　：文字 + 26」
表示　：文字
終わり
```

```
手順は　解読
もし　文末か 「止まる」
選ぶ　一字右へ
書き入れる　字　解読準備　アスキー　選んだもの
解読
終わり
```

メッセージを解読するには、次の命令を使います。

```
文頭へ
解読
```

4. プログラムからのオブジェクト操作

■この節の内容はオンラインヘルプからも見ることができます。
EX　　→メニューバーの［ヘルプ］をクリック→［リファレンス］
EX（ジュニア）　→メニューバーの［ヘルプ］をクリック→［リファレンス］

・例示された命令は、コマンドセンターに入力して試します。Enterを押すことで命令が実行されます。(* コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter を押します。)
・例示された手順は、手順タブエリアに入力します。手順タブエリアの入力が終わったら、コマンドセンターに手順名を入力し、Enter を押すことで手順が実行されます

カメ、テキストボックス、スライダーなどのオブジェクトの作成や編集は、一般的にはマウスを使って行います。
しかし、同時にオブジェクトは、基本用語を使って作成、編集、操作することもできます。
設定するは、オブジェクトを制御するためのコマンドです。この設定するに対応するレポータの**設定**は、オブジェクトの状態を報告します。オブジェクトを作成あるいは削除したり、オブジェクトの状態を変更したりするコマンドやレポータを使えば、対話形式でオブジェクトを変更する作品を作ることができます。

(1) プログラムからのカメの操作

新しいカメ

新しいカメ は、カメを新規に作るコマンドです。**新しいカメ** はインプットとしてカメの名前を取ります。
どのような名前でも使うことができます。ただし、マイクロワールドEXが基本用語として定義している単語と同じ名前は付けることができません。
特に命名しない場合は、自動的に**かめ1**、**かめ2** というような標準名が付けられます。
コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
新しいカメ　"たろう
```

カメを作っただけでは、カメは画面に表示されません。マウスで作成したカメは、ページ上ですぐに見ることができますが、**新しいカメ**で作成した場合には、**出てくる**と命令します。
カメを画面に表示する前に状態変数を設定することもできます。カメには、色、位置、向き、サイズ、ペンの色、ペンの太さなどに関する状態変数があらかじめ用意されています。例えば、画面に表示する前に、カメの色や位置、向きを設定することができます。

コマンドセンターから試しましょう。

色は　15　位置は 「10　100」 向きは　45　出てくる

例えば、カメの群れを作る場合は、次のような命令を使うことができます。

名前は　”群れ 「緑1　緑2　緑3　緑4」　　…**群れ**という変数を作ります。

それぞれをやる 「i :群れ」「新しいカメ　:i　色は　55」…異なる名前を持つカメを作ります。

全部 「出てくる」　　…新しいカメがすべて表示されるようにします。4つとも「0　0」の位置に表示されるため、1つに見えます。

頼む 「緑1　緑2　緑3　緑4」「向きは （最後　今のカメ）＊30」」
…カメがそれぞれ違う方向を向くようにします。**最後　今のカメ**は、アクティブなカメの名前の最後の文字（整数）を報告します。

全部 「前へ　100」　　…すべてのカメを前進させます。

命令はは、カメに命令を出します（カメのダイアログボックスで設定するのと同じです）。この**命令は**では、カメのダイアログボックスの［一回］、［無限に］に対応する**始める** または**無限に** を使うことができます。**命令は**によって、動いているカメの命令を変更することができます。

命令は 「始める 「前へ　50」」

例えば、上記のすべてのカメを点滅させる命令を設定することができます。

全部 「命令は 「無限に 「かくれる　まつ　1　でてくる　まつ　1　」」」

全部 「クリックオン」

クリックオンは、カメの命令を起動します。この命令は、画面上でカメをクリックするのと同じことです。**クリックオフ**は、カメの命令を止めます。これらのコマンドを使って、クリックに対応したカメを、コマンドセンターや手順から制御することができます。

```
全部 「クリックオフ」
```

カメの中の名前は

カメの中の名前はを使って、新しいカメの状態変数を定義することもできます。
カメの中の名前はは、作品のカメの1つ1つに新しい状態変数を作ります。

```
カメの中の名前は　”スピード　　…すべてのカメにスピードという変数を割り当てます。
```

カメの中の名前はのインプットは、すべてのカメに割り当てられる変数になります。作った状態では、この変数には、値はありません。
カメの変数には、変数名と語尾の"は"を使って、値を割り当てることができます。

```
かめ1、スピードは　2　　…かめ1というカメの速度を2に設定します。
かめ2、スピードは　10　　…かめ2というカメの速度を10に設定します。
```

この後、これらのカメに**前へ　スピード**という命令を実行させた場合、かめ2の動きはかめ1より速くなります。

```
頼む 「かめ1　かめ2」「命令は 「無限に 「前へ　スピード」」」
```

```
頼む 「かめ1　かめ2」「クリックオン」
```

状態変数名を単独で使った場合は、アクティブなカメのその変数値が報告されます。

```
かめ1、下へ書く　スピード
```

```
2
```

カメの中の名前で設定したカメの変数の値は、語尾に"の"を付けた短縮形を使って確認することもできます。
この場合、スピードにダブルクォーテーションマーク(”)を付けることに注意してください。

```
下へ書く　かめ2の　”スピード
```

```
10
```

設定というレポータは、アクティブなカメの変数と値のリストを報告します。

```
下へ書く　設定　”かめ1　"なまえ　　…かめ1の カメの中の名前 変数と値を表示します。
```

スピード　2

無くす　”スピード　　…すべてのカメからスピードという変数を削除します。

設定する を使って、カメの命令を制御することができます。また、カメの表示や固定の状態を変更することもできます。(一般的には**隠れる、出てくる、固定、解除**に対して利用されます。) 以下は、設定する使用例、同じ働きをする命令をまとめています。

命令	**同等の命令**
設定する　”かめ１　”命令 「はじめる　「前へ　1」」	かめ1、 命令は 「始める 「 前へ　１」」
設定する　”かめ１　”はじまったか　”ほんとう	クリックオン
設定する　”かめ１　”はじまったか　”うそ	クリックオフ
設定する　”かめ１　”みえるか　”ほんとう	出てくる
設定する　”かめ１　”みえるか　”うそ	隠れる
設定する　”かめ１　”こていか　”ほんとう	固定　”かめ１
設定する　”かめ１　”こていか　”うそ	解除　”かめ１

設定 を使い、**カメの中の名前** で作られた状態変数も含めて、カメのいろいろな状態を調べることができます

式	**働き**
設定　”かめ１　”みえるか	カメが表示されているかどうかどうかについて、**真**または**偽**を報告します。
設定　”かめ１　”めいれい	カメのダイアログボックスに設定されている命令リストを報告します。
設定　”かめ１　”はじまったか	カメがクリックされているかどうか、すなわち、プロセスを起動しているかどうかに従って**真**または**偽**を報告します。
設定　”かめ１　"なまえ	**カメの中の名前**で設定された状態変数（カメの変数）と、その値を報告します。
設定　”かめ１　"こていか	カメが固定されているかどうかに従って**真**または**偽**を報告します。

作品タブエリアには、上のようなカメの状態のすべてが表示されます。

注　:**設定する**と設定でインプットとして使用するレポータはすべて**ひらがな**で入力してください。

(2) プログラムからのテキストボックスの操作

テキストボックス

テキストボックスは､新しいテキストボックスを作ります。
テキストボックスというコマンドは3つのインプットをとります。1つ目のインプットはテキストボックス名です。(カメの場合と同様､テキストボックス名に基本用語を使うことはできません)。2つ目のインプットはテキストボックスの位置（画面中央を0、0とするx-y座標)、3つ目のインプットはテキストボックスのサイズです。

```
テキストボックス　”手紙　「10　90」「200　50」
```

字の表示および**字を隠す**は、テキストボックス自体の表示/非表示を制御します。
不透明および**透明**を使うと､テキストボックスの内容である文字の表示/非表示だけ制御します。
テキストボックスを作ると、そのテキストボックス名の語尾に"は"を付け加えて、その内容を設定することができるようになります。例えば、上記の例では、**手紙は** として次のように使うことができます。

```
手紙は　「次回の例会について」
```

設定するを使ってテキストボックスのあらゆる状態を制御することができます｡テキストボックスの位置とサイズは、ページの中に収まる値を指定する必要があります。
以下に、**設定する** の使用例とその働き、または同じ働きをする命令をまとめています。

命令	**同等の命令または働き**
設定する　”テキスト1　「やあ　こんにちは」	テキスト1は　「やあ　こんにちは」
設定する　”テキスト1　"みえるか　”ほんとう	字の表示
設定する　”テキスト1　”みえるか　”うそ	字を隠す
設定する　”テキスト1　”とうめいか　”ほんとう	透明　”テキスト1
設定する　”テキスト1　”とうめいか　”うそ	不透明　”テキスト1
設定する　”テキスト1　”いち　「0　0」	テキストボックスの位置を設定します。
設定する　”テキスト1　”おおきさ　「100 100」	テキストボックスのサイズを設定します。
設定する　”テキスト1　”なまえをひょうじするか　”ほんとう	名前タグを表示します。
設定する　”テキスト1　”なまえをひょうじするか　”うそ	名前タグを隠します。
設定する　”テキスト1　”こていか　”ほんとう	固定　”テキスト1
設定する　”テキスト1　”こていか　”うそ	解除　”テキスト1

対応するレポータ、**設定**は、**行をやる**が使われているかどうかを含めて、テキストボックスの状態を報告します。

式	同等の式／働き
設定　”テキスト1　”みえるか	テキストボックスが表示されているかどうかを報告します。
設定　”テキスト１　”いち	テキストボックスの位置を報告します。
設定　”テキスト１　”おおきさ	テキストボックスのサイズを報告します。
設定　”テキスト１　”とうめいか	テキストボックスが透明であるかどうかを報告します。
設定　”テキスト1　"なまえをひょうじするか	テキストボックスの名前タグが表示されているかどうかを報告します。
設定　”テキスト１　" じ	テキスト１の内容を報告します。
設定　”テキスト１　”こていか	テキストボックスが固定されているかどうかを報告します。
設定　”テキスト1　”ぎょうをやる	アクティブな**行をやる**命令を報告します。

次に**設定する**（コマンド）と**設定**（レポータ）を使ってテキストボックスを配置する例を紹介しましょう。
同じ２つのテキストボックスを横に並べます。

```
テキストボックス　”テキスト１　「－200　125」「100　130]」

テキストボックス　”テキスト２　「－80　125]」 設定　”テキスト１　”大きさ
```

２行目の命令の[－80　125]は、**テキスト１**よりも 120 ドット右寄りの位置を指定します。3つ目のインプットは、**設定**を使って**テキスト１**の大きさを報告させ、それを利用します。
（注意：サイズは、作品タブエリアで右クリックを使って直接、設定することができます）。

設定するおよび**設定**によって、テキストボックス名を変数として使うことができるため、複数の用途を持つツールを作ることができます。
次の例では、異なるテキストボックスを使って計算中のデータを表示します。

```
設定する　”AA　”じ （設定　”AA　”じ） ＋　1
```

この命令では、「AA」というテキストボックスの内容を１ずつ大きくします。この命令は実際には次と同じです。

```
AA は　 AA ＋ 1
```

次に、**設定する**の命令で変数を使って、テキストボックス内の数字を一定の数ずつ大きくする例を紹介します。

```
手順は　増加　：対象
設定する　：対象　”じ　（設定　：対象　”じ）　＋　１）
終わり
```

数字を内容としたテキストボックスを作っておき、**増加　テキストボックス名**を入力すると、テキストボックス内の数字が１ずつ大きくなります。

(3)プログラムからのボタンの操作

ボタン

ボタンは、新しいボタンを作ります。他のオブジェクト同様、ボタン名に基本用語を使うことはできません。**ボタン**も３つのインプットを取ります。１つ目はボタン名、2つ目は位置（画面中央を「0　0」とするx-y座標）、3つ目はボタンの命令です。
ボタンというコマンドで作ったボタンの実行モードは、自動的に［一回］に設定されます。

```
ボタン　”マイボタン　「90　90」　「前へ　50」
```

他のオブジェクトと違って、ボタンの場合、通常その名前は重要ではありません。マウスで［ボタン作成］ツールを使ってボタンを作ると、自動的に"ボタン１"、"ボタン２"などの標準名が付けられます。（名前を変更することはできません）。

ボタンを隠すことはできません。ボタンは、命令とともにつねに画面に表示されます。ボタンの位置とサイズ、状態、命令は個々に設定することができます。
コマンドでボタンを作る場合、ボタンのダイアログボックスの［一回］と［無限に］に対応する機能を実現するには、それぞれ**始める**、**無限に**を使います。
ボタンの状態には、ONとOFFがあります。この状態を調べるレポータに**始まったか**があります。
ボタンがクリックされ、命令が実行されている場合、"始まったか"は**真**になります。実行を終えると、"始まったか"は**偽**に戻ります。作品の別のページのボタンから命令を実行することもできます。この場合には、**設定する**を使ってそのボタンの状態を制御します。
以下は、**設定する**の使用例とその働き、または同じ働きをする命令をまとめています。

命令	**同等の命令／働き**
設定する　”ボタン１　”いち　「0　0」	ボタンの位置を設定します。
設定する　”ボタン１　”おおきさ　「100　100」	ボタンのサイズを設定します。
設定する　”ボタン１　”めいれい　「始める　「前へ　１」」	ボタンの命令実行モードを［一回］に設定します。
設定する　”ボタン１　”めいれい　「無限に　「前へ　１」」	ボタンの命令実行モードを［無限に］に設定します。
設定する　”ボタン１　”はじまったか　”ほんとう	ボタンの命令を実行します。
設定する　”ボタン１　”こていか　”ほんとう	固定　”ボタン１
設定する　”ボタン１　”こていか　”うそ	解除　”ボタン１

対応するレポータ、**設定**はボタンの状態を報告します。

式	同等の式/働き
設定　”ボタン１　”いち	ボタンの位置を報告します。
設定　”ボタン１　”おおきさ	ボタンのサイズを報告します。
設定　”ボタン１　”めいれい	ボタンの命令を報告します。
設定　”ボタン１　”はじまったか	ボタンの命令が実行されているかどうかを報告します。
設定　”ボタン１　”こていか	ボタンが固定されているかどうかを報告します。

注　:**設定する**と**設定**でインプットとして使用するレポータはすべて**ひらがな**で入力してください。

次の例に示すように、**設定する**（コマンド）と**設定**（レポータ）を使って複数のボタンを同じサイズにすることができます。

この例では、それぞれに命令が設定された、ボタン１とボタン２という２つのボタンがあるとします。ここでは、ボタン２をボタン１と同じサイズにします。

```
設定する　”ボタン２　”おおきさ　設定　”ボタン１　”おおきさ
```

（注意：ボタンのサイズは、作品タブエリアで右クリックを使って直接、設定することができます。）

続いて、ボタンの命令を使いながら変更する例を紹介しましょう。

タイル式のパズルの例です。タイルの１つ１つがカメになっています。ボタンの命令の**シャッフル**と**リセット**を"交互に"切り替えることができます。作品を開くと、パズルが作られます、当初のボタンの命令は**シャッフル**です。

```
手順は　シャッフル
全部 「動く」
設定する　”ボタン１　”命令　「始める　「リセット」」
終わり

手順は　動く
向きは　乱数　360　前へ　乱数　2000
終わり

手順は　リセット
全部 「位置は　ホームポジション」
設定する　”ボタン１　”命令 「始める　「シャッフル」」
終わり
```

ボタンを押すと、ボタンの**シャッフル**が次の命令を持つ**リセット**に変わります。

```
設定する　”ボタン１　”命令　「始める　「リセット」」
```

同様に、**リセット**が実行されると、**リセット**が**シャッフル**に変わります。

4）プログラムからのスライダーの操作

スライダーは、あらかじめ設定した範囲内の値を報告します。スライダーの出力はコマンドや手順のインプットとして利用することができます。

スライダー

スライダーは、新しいスライダーを作ります。**スライダー**は３つのインプットを取ります。１つ目はスライダー名です（他のオブジェクトと同様、スライダー名に基本用語を使うことはできません）。２つ目はスライダーの位置（画面中央を「0　0」としたx-y座標）、３つ目のインプットは３つの数字からなるリストです。このリストには、最小値、最大値、現在の値 の順に値を指定します。

```
スライダー　”スピード　「200　−150」「0　50　25」
```

ページ上の「200　−150」の位置に、"スピード"という名前の、"最小値0、最大値50、現在の値25" のスライダーが作成されます。

設定する を使って、スライダーのいろいろな状態を制御することができます。以下は、**設定する** の使用例をまとめています（"スピード"はスライダー名です）。

命令	**同等の命令／働き**
設定する　”スピード　”いち　「0　0」	スライダーの位置を設定します。
設定する　”スピード　”なまえをひょうじするか　”ほんとう	名前タグを表示します。
設定する　”スピード　”はんい　「0　100」	スライダー値の範囲を設定します。
設定する　”スピード　”あたい　50	スピードは　50
設定する　”スピード　”こていか　”ほんとう	固定　”スピード
設定する　”スピード　”こていか　”うそ	解除　”スピード

設定（レポータ）は、スライダーの状態を報告します。

式	同等の式/働き
設定　”スピード　”いち	スライダーの現在の位置を報告します。
設定　”スピード　”なまえをひょうじするか	スライダー名が表示されているかどうかを報告します。
設定　”スピード　”はんい	スライダー値の範囲を報告します。
設定　”スピード　”あたい	スピード
設定　”スピード　”こていか	スライダーが固定されているかどうかを報告します。

注　:**設定する**と**設定**でインプットとして使用するレポータはすべて**ひらがな**で入力してください。

上の命令を使って、2つのスライダーを上下に重ねて配置する例を紹介します。2つのスライダーがあり、一方が"**スピード**"、もう一方が"**角度**"という名前であるとします。

"**スピード**"スライダーをページの右下角に配置します。

設定する　”スピード　”いち　「200　–150」

"**角度**"スライダーをそのすぐ上に配置します。

設定する　”角度　”いち　「200　–120」

注意：位置は、作品タブエリアで右クリックメニューから直接、設定することができます。

(5) プログラムからの色の操作

マイクロワールドEXでは、色をオブジェクトとしてプログラミングすることができます。色に対して命令を設定することができるのです。つまり、ある色の上をカメが通過したり接触した場合に実行する命令を設定したり、あるいはある色の上でマウスをクリックしたときに実行する命令を設定することができます。

設定するを使って、プログラムからも、こうした命令を設定することができます。**マウスクリック**は、色の上でマウスがクリックされた場合に実行する命令です。

カメのめいれいは、色の上をカメが通過したり接触した場合に実行する命令です。**カメのモード**の［一回］と［毎回］の設定は、**カメのめいれい**に対してだけ有効です。

注意：色への命令を設定をすると、その色のすべての階調が影響を受けます。例えば、赤色（色番号15）に命令を設定すると、色番号10から19（カラーパレットの赤の列）のすべての赤がその設定の影響を受けます。

命令	働き
設定する　"あか　"カメのめいれい　「後ろへ　50」	カメを検出したときの命令を設定します。
設定する　"あか　"カメのモード　"いっかい	実行モードを「一回」に設定します。
設定する　"あか　"カメのモード　"まいかい	実行モードを「毎回」に設定します。
設定する　"あか　"マウスクリック　「前へ　50」	マウスがクリックされたときの命令を設定します。

設定（レポータ）は、上記の命令の設定内容を報告します。

式	働き
設定　"あか　"カメのめいれい	カメを検出したときの命令を報告します。
設定　"あか　"カメのモード	実行モードを報告します。
設定　"あか　"マウスクリック	マウスがクリックされたときの命令を報告します。

(6) プログラムからのメディアオブジェクトの操作

音声とメロディー、音楽、ビデオのどれにも、プログラムから変更できるパラメータがあります。ただし、これらのメディアオブジェクトは、**マウスを使ってのみ作成**することができます。

以下は、パラメータとその意味、**設定する**の使用例をまとめています。

パラメータ	意味	命令
いち	位置を設定します。	設定する　"メロディー1　"いち　「50　50」
みえるか	アイコンの表示／非表示を制御します（真または偽）。	設定する　"メロディー1　"みえるか　"うそ
なまえをひょうじするか	名前タグの表示／非表示を制御します（真または偽）。	設定する　"メロディー1　"なまえをひょうじするか　"ほんとう
はじまったか	アイコンのオン／オフを制御します（真または偽）。	設定する　"メロディー1　"はじまったか　"ほんとう
こていか	アイコンの固定／解除を制御します（真または偽）。	設定する　"メロディー1　"こていか　"ほんとう

オブジェクトのオン／オフ状態を制御する**設定する**は、実際にアイコンを制御します。
例えば、音声を再生する次の命令が実行されると、アイコンが"押された状態"で表示されます。（"**サウンド1**"は音声アイコンの名前）。

```
設定する　"サウンド1　"はじまったか　"ほんとう
```

音声の録音や読み込みで、ページ上にアイコンが作られると同時に、同等の働きをするコマンドが、同じ名前で作成されます。

設定は、上記のパラメータを報告します。

パラメータ	**意味**	**式**
いち	オブジェクトの現在の位置を報告します。	設定　”メロディー１　”いち
みえるか	アイコンの表示／非表示の状態を報告します。	設定　”メロディー１　”みえるか
なまえをひょうじするか	名前タグの表示／非表示の状態を報告します。	設定　”メロディー１　”なまえをひょうじするか
はじまったか	オブジェクトのオン／オフの状態を報告します。	設定　”メロディー１　”はじまったか
こていか	オブジェクトが固定／解除の状態を報告します。	設定　”メロディー１　”こていか

注　:**サウンド１**をコマンドから実行しているときに次の命令を実行すると、**偽**が報告されます。

```
下へ書く　設定　”サウンド１　”はじまったか
```

これは、サウンド１のアイコンが"オン"になっていないためです。

メロディーの場合は、**設定**を使ってその他のパラメータを報告させることもできます。

式	**意味**
設定　”メロディー１　”がっき	現在の楽器を報告します。
設定　”メロディー１　”ボリューム	現在の音量を報告します。
設定　”メロディー１　”テンポ	現在のテンポを報告します。

次はプレゼンテーション用の作品の中でビデオを表示し、再生してから、隠す、というアイデアの例です。

```
設定する　”ビデオ１　”みえるか　”ほんとう
ビデオ１
設定する　”ビデオ１　”みえるか　”うそ
```

(7) プログラムからのハイパーリンクの操作

ハイパーリンクは、ツールバーの［ハイパーリンク］を使って作ることができます。
ハイパーリンクでは、**設定する**を使って、表示や位置などの標準的なパラメータを制御できるばかりでなく、URLや電子メール、FTPのアドレスを設定することもできます。ハイパーリンクを作ると、"ハイパーリンク1"などの標準名が付けられますが、他のオブジェクト同様、この名前は変更することができます。

以下は、**設定する**の使用例をまとめています。

命令	**同等の命令／働き**
設定する　"ハイパーリンク1　"なまえをひょうじするか　"ほんとう	名前タグを表示します。
設定する　"ハイパーリンク1　"いち　「0　0」	アイコンの位置を設定します。
設定する　"ハイパーリンク1　"みえるか　"うそ	アイコンを隠します。
設定する　"ハイパーリンク1　"みえるか　"ほんとう	アイコンを表示します。
設定する　"ハイパーリンク1　"アドレス　"http://www.lcsi.ca/	ハイパーリンクにURLアドレスを設定します。URLアドレスは、http:// から始めます。
設定する　"ハイパーリンク1　"アドレス　"mailto:info@lcsi.ca	ハイパーリンクに電子メールアドレスを設定します。電子メールアドレスはmailto:から始めます。
設定する　"ハイパーリンク1　"こていか　"ほんとう	固定　"ハイパーリンク1
設定する　"ハイパーリンク1　"こていか　"うそ	解除　"ハイパーリンク1

設定（レポータ）は、上記のパラメータの現在の設定を報告します。

式	**働き**
設定　"ハイパーリンク1　"なまえをひょうじするか	名前タグの表示/非表示の状態を報告します。
設定　"ハイパーリンク1　"いち	アイコンの現在の位置を報告します。
設定　"ハイパーリンク1　"みえるか	アイコンの表示/非表示の状態を報告します。
設定　"ハイパーリンク1　"アドレス	ハイパーリンクのアドレスを報告します。
設定　"ハイパーリンク1　"こていか	アイコンの固定/解除の状態を報告します。

(8) プログラムからのページの操作

ページに対しては、**設定する** を使って、**はいけいのこていか** と **きりかえこうか** の2つのパラメータを設定することができます。
ページの切り替え効果には番号が振られており、0は標準、つまり、特殊効果なしになります。

命令	**同等の命令／働き**
設定する　”ページ１　"はいけいのこていか　”ほんとう	背景の固定
設定する　”ページ１　"はいけいのこていか　”うそ	背景の解除
設定する　”ページ１　”きりかえこうか　1	ページの切り替え方を１に設定します。
設定する　”ページ１　”きりかえこうか　0	ページの切り替え方を標準に設定します。

設定（レポータ）は、指定されたページ上の各種オブジェクトの名前ばかりでなく、背景が固定されているかどうか、あるいはページの切り替え効果が設定されているか、といった情報も報告することができます。

式	**働き**
設定　”ページ１　”カメ	このページに含まれるカメの名前を報告します。
設定　”ページ１　”テキスト	このページに含まれるテキストボックスの名前を報告します。
設定　”ページ１　”ボタン	このページに含まれるボタンの名前を報告します。
設定　”ページ１　”スライダー	このページに含まれるスライダーの名前を報告します。
設定　”ページ１　”メロディー	このページに含まれるメロディーの名前を報告します。
設定　”ページ１　”サウンド	このページに含まれるサウンド（音声/録音）の名前を報告します。
設定　”ページ１　”おんがく	このページに含まれる音楽の名前を報告します。
設定　”ページ１　”ビデオ	このページに含まれるビデオの名前を報告します。
設定　”ページ１　”いろのめいれい	プログラミングされた色の名前を報告します。
設定　”ページ１　”はいけいのこていか	背景画像が固定されているかどうかを報告します。
設定　”ページ１　”きりかえこうか	ページの切り替え方の番号を報告します。

注　:**設定する**で使用するパラメータはすべて**ひらがな**で入力してください。

(9) プログラムからの作品の操作

作品に対しては、**設定する** を使って、作品の表示の仕方を制御することができます。
プレゼンテーションモードでは、画面に作品のページ部分だけが表示されます。また、ツールバーやタブエリア、ステータスバー、コマンドセンターを表示または隠すことができます。作品の場合、**設定する**、**設定** のインプットは特殊な単語の**さくひん**になります。

命令	**同等の命令／働き**
設定する　”さくひん　”プレゼンテーションモードか　”ほんとう	プレゼンテーションモード
設定する　”さくひん　”プレゼンテーションモードか　”うそ	プレゼンテーションモード（解除するには再度、命令する）
設定する　”さくひん　　”ツールバーか　”うそ	ツールバーを隠します。
設定する　”さくひん　”コマンドセンターか　”うそ	コマンドセンターを隠します。
設定する　”さくひん　”タブか　”うそ	タブエリアを隠します。
設定する　”さくひん　”ステータスバーか　”うそ	ステータスバーを隠します。

設定（レポータ）は、上記のパラメータの設定を報告します。

式	**働き**
設定　”さくひん　”プレゼンテーションモードか	プレゼンテーションモードであるかどうかを報告します。
設定　”さくひん　”ツールバーか	ツールバーが表示されているかどうかを報告します。
設定　”さくひん　”コマンドセンターか	コマンドセンターが表示されているかどうかを報告します。
設定　”さくひん　”タブか	タブエリアが表示されているかどうかを報告します。
設定　”さくひん　”ステータスバーか	ステータスバーが表示されているかどうかを報告します。

5. プログラミングテクニック

■この節の内容はオンラインヘルプからも見ることができます。
EX　　→メニューバーの［ヘルプ］をクリック→［リファレンス］
EX（ジュニア）　→メニューバーの［ヘルプ］をクリック→［リファレンス］

・例示された命令は、コマンドセンターに入力して試します。Enterキーを押すことで命令が実行されます。(*コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter キーを押します。)
・例示された手順は、手順タブエリアに入力します。コマンドセンターに手順名を入力し、Enter キーを押すことで手順が実行されます

ここでは、実際にプログラミングにするときのテクニックや注意事項について説明します。

(1) オブジェクトの名前

ロゴのプログラミングでは、ネーミングは重要です。手順や変数には必ず名前を付けますが、意味のある名前を付けることによって、プログラムや分かりやすく修正しやすくなります。

マイクロワールドEXは、ネーミングによる様々な制御機能が拡張されたため、ネーミングを必要とするオブジェクトが数多くあります。
ほとんどのオブジェクトは、作成時に自動的に**標準名**が付けられるか、独自の名前を付けない場合に標準名がつけられます。しかし、カメやテキストボックスなど、プログラムの中で対象を特定して、プログラムで使うオブジェクトには独自の名前を付けるようにしましょう。

それぞれのオブジェクトは基本的に他のオブジェクトと重複しない名前を付ける必要があります。同じページに含まれるオブジェクトはそれぞれに異なる名前である必要がありますが、ページが異なれば、同じ作品内でも、同じ名前のオブジェクトが存在することができます。
言いかえれば、**同じページ内で、重複するオブジェクト名を使うことはできません。**

オブジェクト名の重複

オブジェクト	標 準 名	ページ内	作品内
カメ	かめ1、かめ2、...	X	
テキストボックス	テキスト1、テキスト2、...	X	
スライダー	スライダー1、スライダー2、...	X	
ボタン	ボタン1、ボタン2、...	X	
ページ	ページ1、ページ2、...	X	
音声（録音）	サウンド1、サウンド2、..		X
メロディー	メロディー1、メロディー2...	X	
ビデオ	ビデオ1、ビデオ2、...	X	
音楽	音楽1、音楽2、...	X	
ハイパーリンク	ハイパーリンク1、...	X	

注意：メディアオブジェクト（音楽、音声、ビデオ）の場合、作品への1回目の読み込み時には自動的なネーミングは行われません。ファイルの保存時に、ファイルとして付けた名前が付けられます。

同じメディアオブジェクトを（他のページなどで）2回目に読み込むと、標準名が付けられます。

他のページのオブジェクト操作

自動的に付けられる標準名をそのまま使った場合は、**かめ1**、**かめ2**といったカメの名前や**テキスト1**、**テキスト2**といったテキストボックス名が作品内の複数のページに含まれる可能性があります。従って、表示中のページから、他のページのテキストボックスに対して、テキストの表示や挿入を行うには、次のような注意が必要です。

・カンマ命令（**テキスト2**、など）あるいは**カメは**を使って、操作するテキストボックスを特定する。
・現在のページに同じ名前のテキストボックスがないことを確認する。

例えば、現在開いているページに、"テキスト3"というテキストボックスが無い場合は、上記の方法で"テキスト3"を特定することで、他のページにある"テキスト3"にテキストを表示、挿入したり、内容の報告をさせたり、さらには"テキスト3は"というコマンドを使って内容を置きかえることができます。

自動的にオブジェクトに付けられた標準名をそのまま使っていた場合、次のような問題が発生する可能性があります。

ページ1とページ2の両方のページに、同じ**テキスト1**というテキストボックスが含まれている場合を例とします。

テキスト1というテキストボックスがないページ3を開いているときに、**テキスト1、下へ書く　テキスト1**などの命令を使うと、次のエラーメッセージが表示されます。

テキスト1は2つ以上あります。

これは、2つある"テキスト1"というテキストボックスのどちらを処理してよいのかを判断できないためです。
それぞれのテキストボックスに異なる名前を付けらていれば、この問題が起きることはありません。

他のオブジェクトについても、このような場合、同様のエラーメッセージが表示されます。

それぞれのカメに違った名前を付けることによって、間違ったカメが命令を実行するといった問題は避けられます。

(2) 質問と答え：対話の実現

質問 および **答え**は、マイクロワールドEXが自動的に質問ダイアログボックスを開いて、入力された単語（または文）を答えとして利用するユニークな基本用語です。質問が出され、答えが入力された後で**答え**というレポータを使うと、その答えが報告されます。**答え**は、その情報を保持していますから、再び質問 が使われるまで繰り返し報告することができます。
手順タブエリアに、次の手順を作ってください。

```
手順は　あいさつ
質問 「あなたのお名前は？」
メッセージを出す　文　”はじめまして　答え
終わり
```

コマンドセンターに手順名を入力し、Enter キーを押して実行します。

```
あいさつ
```

ページにダイアログボックスが表示されるので、答えを入力し、［OK］をクリックします。この後、答えは繰り返し利用することができます。
上の手順では、**文**は1つ目のインプットである**はじめまして**と、2つ目のインプットである**答え**を1つのリストにまとめます。そのリストを**メッセージを出す**が、ダイアログボックスに表示します。

コマンドセンターに次の命令を入力し、実行してください。

```
下へ書く　答え
```

質問と答えの活用

質問と答えを使って、算数のクイズを作ることもできます。

```
手順は　クイズ
質問 「5　+　9　はいくつですか」
もしどちらかを　答え　＝　14 「メッセージを出す　「正解です！」」
「メッセージを出す 「この次はがんばろう！」」
終わり
```

答え が数字の場合、カメの命令のインプットとして**答え** を使うことができます。

```
前へ　答え                    …カメが入力された数字の歩数前進します。
```

答え が報告するデータは単語1つです。この単語には空白を含むことができ単語1つと見なされます。

質問に対して、"はい"または"いいえ"の入力を求めることが、よくあります。

```
質問 「作業を継続しますか」
```

これに対する答えとして、"y" が入力されることもあれば、 "yes" と入力されることもあるかもしれません。また、誤って、"yes" の前や後に空白があるかもしれません。
こうした場合は、等号 (=) ではなく、**あるか** を使って、**答え** の内容を調べた方が賢明です。例えば、次の命令は、答えが yes の場合にだけ機能します。yes の前後に空白が付いていてはいけません。

```
もし　答え　=　"yes 「これを実行」
```

これに対し次の命令は、答えに y（大文字でもよい）が含まれている限り機能します。

```
もし　あるか　"y　答え 「これを実行」
```

特定の種類の答えが必要なことがあります。

次の例では、数字以外の答えが入力されると、**もし　答え　>　10** から始まる命令があるために、エラーメッセージが返されます。これは、**">（大なり記号）**" が数字に対してだけ有効なためです。また、答えの欄に、11 ではなく、"eleven" が入力された場合もエラーになってしまいます。

```
もし　答え　>　10　「下へ書く　「古いね！」」
```

> への eleven のインプットがおかしいです

この種の問題が発生すると、対話型のプログラムは途中で終了してしまいます。中途終了を防ぐには、答えが正しい種類の情報であるかどうかを調べて、正しくない場合には入力をやり直すように求めるようにします。

次は、答えとして数値の入力を求める手順の例です。

```
手順は　例題　：リスト
質問　：リスト
もし　数字か　答え 「止まる」
例題　：リスト
終わり
```

答えが数字以外の場合は、再び同じ質問が表示されます。

質問ダイアログボックスの質問に答えずに、単に［OK］がクリックされると、プログラムが止まってしまうという問題が起きたとします。この種の問題は、手順を繰り返し "ループ" させることによって避けることができます。

```
手順は　例題　：リスト
質問　：リスト
もし　空か　答え 「例題　：リスト」
終わり
```

次の手順は、答えとして単語1つの入力を求めます。つまり、**もし** は、答えにスペース（空白：ASCII コード 32）が含まれているかどうかを調べ、含まれている場合は、再び同じ質問を出します。

```
手順は　例題　：リスト
質問　：リスト
もし　反対　あるか　じ　32　答え　「止まる」
例題　：リスト
終わり
```

ダイアログボックスの表示位置は、設**定する** で設定することができます。

(3) テキストボックスのコマンドセンター化

行をやる を使って、テキストボックスをコマンドセンターと同じように使うことができます。

ウェブプレイヤーにはコマンドセンターがないので、この機能をインターネットに公開する作品で利用すると便利です。

行をやる は2つのインプットを取ります。1つ目のインプットは、対象となる（コマンドセンターとして使う）テキストボックス名です。2 つ目のインプットは命令です。

テキスト1 というテキストボックスを作って、実際に試してみましょう。このテキストボックスをコマンドセンターのようにするには、コマンドセンターに次の命令を入力します。

```
行をやる　”テキスト１　”やる
```

テキスト1 の名前タグが青色に変わり、**行をやる** がアクティブであることが示されます。
やるは、そのインプットを命令として実行します。テキスト 1 には、どのような命令でも入力することができます。例えば**前へ　50** という命令を入力してみてください。カメが前進します。

この機能を無効にするには、行をやるの 2 つ目のインプットを空にします。

```
行をやる　”テキスト１　”
```

行をやるで手順名を使うとどうなるのでしょうか。
まず、手順タブエリアで**逆転** と**逆に書く** という手順を作ってください。

```
手順は　逆転　：単語
もし　空か　：単語 「表示　”」
表示　ワード　最後　：単語　逆転　最後以外　：単語
終わり

手順は　逆に書く　：単語
書く　逆転　：単語
終わり
```

ページに**テキスト1**というテキストボックスがあることを確認し、コマンドセンターに次の命令を入力します。

```
行をやる　”テキスト１　”逆に書く
```

続いて、テキスト１に単語１つを入力して、Enter キーを押します。

注　:**行をやる** で使っているテキストボックス名は、作品タブエリアで確認することができます。

(4)「注意してやる」

命令が正しく機能するかどうか確信が持てない場合は、**注意してやる** を使ってください。命令が正しく実行されない場合でも、手順が止まることはありません。

ただし、**注意してやる** を挿入した場合、エラーメッセージは表示されませんから、実際に**注意してやる** を使う前に、プログラムをデバッグしてください。

注意してやる は2つのインプットを取ります。ともに単語またはリストです。**注意してやる** は、それぞれのインプットを命令として実行します。最初の命令にエラーが含まれている場合は、2つ目の命令を実行し、発生したエラーの情報を**エラーメッセージ** に設定します。最初のインプットにエラーがなかった場合、2つ目のインプットは無視されます。

次の例では、**注意してやる** を使って、テキストボックス内のテキストを実行します。このテキストはロゴの命令です。実際にはこの例は、ウェブプレイヤーで実行する対話形式の作品ですから、コマンドセンターはありません。従って命令にエラーがあっても、そのエラーを表示する場所がありません。しかし、2つ目の命令が、コマンドセンターではなく、**メッセージを出す** のダイアログボックスにエラーを表示するよう指示します。(**はじめ** は、ボタンから実行する手順です)。

```
手順は　はじめ
注意してやる 「やる　テキスト1」「メッセージを出す　エラーメッセージ」
終わり
```

注意してやる は、次のようなケースでも利用することができます。冒険ゲームで、プレーヤーに目的地を尋ねる質問のダイアログボックスを表示するとします。プレーヤの答えは、**ページを出す** に対するインプットとして使われます。

```
手順は　次へ
質問　「お城と洞窟 ... どちらへ行きますか？]
ページを出す　答え
終わり
```

プレーヤが綴りを間違えるか、選択肢にはない単語を入力した場合、手順が止まり、エラーメッセージが表示されます。例えば"お城"を"おしろ"と入力すると、

ページを出すへの おしろのインプットがおかしいです。

次へという手順で問題がありそうな命令は、**ページを出す　答え**です。このため、**注意してやる**に対する最初のインプットとして[**ページを出す　答え**]を指定して、2つ目のインプットで、メッセージを出すようにします。

```
手順は　次へ
質問　「お城と洞窟 ... どちらへ行きますか？]
注意してやる　「ページを出す　答え　止まる」
「メッセージを出す　「そのような場所はありません」」
終わり
```

(5) スタートアップ手順

スタートアップ手順は、その作品が開かれると**最初に実行される特別な手順**です。

スタートアップ手順を作ることによって、作品が開かれると同時に特定のページを表示することができます。
次のスタートアップ手順は、作品が開かれると同時に**コンテンツ**というページを表示して、アニメーションを開始します。

```
手順は　スタートアップ
コンテンツ
メッセージを出す　「ようこそ　私の作品館へ！」
かめ1、クリックオン　待つ　80　クリックオフ
終わり
```

(この手順の**かめ1**には、アニメーションの命令が設定されているものとします。)

最初に表示されるページは次のようなルールで決まります。
(1) 作品内に**ページ1**というページがある場合は、そのページが最初に表示される。
(2) 作品に**ページ1**というページがない場合、前回、作品を保存したときに表示されていたページを最初に表示する。

"**ページ1**"という名前ではないページを、必ず最初に表示するには、スタートアップ手順を使うことが必要です。

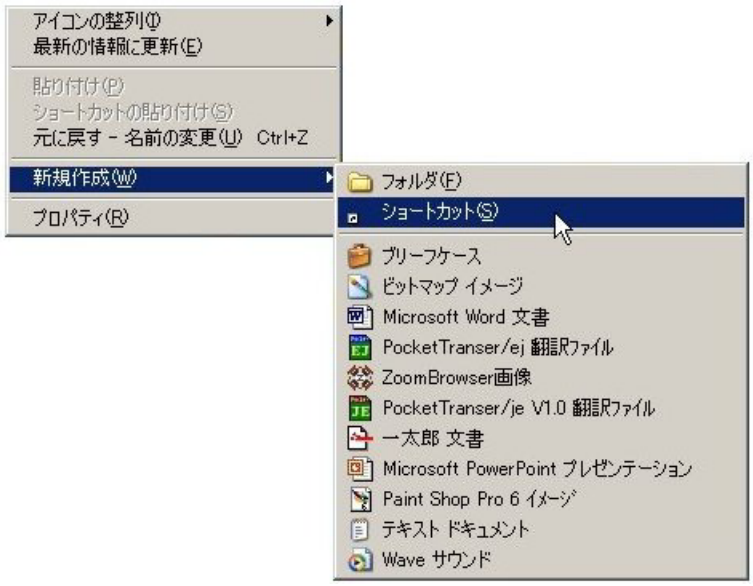
スタートアップ手順はまた、作品にグローバル変数を作ったり、状態変数を設定したりする手段でもあります。
次のスタートアップ手順の例は、グローバル変数を作って、**スピード**というスライダーを0に設定し、**コメント**というテキストボックスの内容を消去します。(ページ上には"スピード"という名前のスライダーと、"コメント"という名前のテキストボックスが作成されているものとします)。

```
手順は　スタートアップ
名前は　”ポイント　0
スピードは　0
コメント、 字を消す
終わり
```

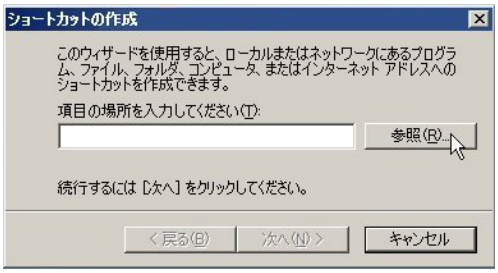
(6) スタートアップ作品

Windows の“ショートカット”機能を利用して、マイクロワールド EX の起動時に特定の作品を開くようにすることができます。

1.Windows のデスクトップ上の何もないところで右クリックします。
2. メニューの［新規作成］から［ショートカット］を選択します。



3. ショートカット作成のダイアログボックスが開き、ファイルの場所と名前を求められます。
4. ［参照］をクリックして、スタートアップ作品に設定したい作品の場所を選択します。



5. デスクトップ上に作品名のショートカットアイコンが作成されます。

(7) オブジェクトの固定

オブジェクトを固定すると、マウスを使ってオブジェクトを移動したり、また、サイズ変更、削除ができなくなります。完成した作品を勝手に変更されたくない場合や、また、半完成のテンプレートなどを作成する場合に利用できます。

テキストボックスを固定すると…

マウスを使ったテキストボックスも移動、拡大縮小、切り取りができなくなります。
字を消す、書く、書き入れる、また、設定するを使ってテキストを入力したり、削除したり、編集することはできます。

カメを固定すると…

マウスを使ってカメの形を変更したり、移動することができなくなります。
カメのダイアログボックスを開いたり、**スタンプ、前へ、形は**などは使うことができます。

ボタンを固定すると…

クリックで命令の実行を止めることができなくなります。例えばボタンから音楽の再生を開始した場合、ボタンをクリックして、再生を止めることはできなくなります。
これらのオブジェクトの固定情報は作品と一緒に保存されます。このため、次回作品を開いたときも、固定したオブジェクトは固定されたままになります。

マウス操作による固定 / 解除の設定と変更

・ページ上でオブジェクトを右クリックして、［解除］を選択する。（オブジェクトが固定されている場合は、メニューの［切り取り］が［解除］に変わります。）
・作品タブエリアでオブジェクトのアイコンを右クリックして、［固定］または［解除］を選択する。

プログラムからの固定 / 解除の設定と変更

固定はインプットとして単語またはリストを受け取り、オブジェクトを1つずつ、あるいはリストのオブジェクトをすべて固定します。また、ページの全オブジェクトを固定することもできます。オブジェクト名は、作品タブエリアで調べることができます。

```
固定　”テキスト１
```

```
固定 「テキスト１　かめ１　スライダー１　僕の曲」
```

```
固定　”ページ１
```

解除は、オブジェクトの固定を解除して、再び動かしたり、編集したりできるようにします。解除するオブジェクトの名前が分からない場合は、作品タブエリアを参照します。

```
解除　”テキスト１
```

```
解除 「テキスト１　スライダー１」
```

```
解除　”ページ１
```

背景画像の固定

背景画像を固定すると、消しゴムや**絵を消す**で消去できなくなります。
背景の固定は、ページごとにコマンドセンターから実行し、開いているページの背景を固定します。固定した背景を消去することはできませんが、画像を追加することはできます。テンプレートの作成などに役立ちます。

注　：**背景は**を使って背景の色を設定した場合、その色を消去することはできません。
固定した画像上に図を描いた場合、その新しい図を削除することはできますが、**背景の固定**を使ったときに存在していた画像が削除されることはありません。**背景の解除**は、固定を解除します。

(8)Microsoft Excel とのデータ交信

Microsoft Excel を利用できる場合は、表を開く、表を閉じる、セルの値は、セルの値の 4 つの基本用語を使って Excel とデータをやりとりすることができます。

表を開く は、新しい Excel のワークシート、あるいは既存の Excel のワークシートを開きます。表を開く は 2 つのインプットを取ります。1 つはファイル名、もう 1 つはシート名です。

```
表を開く　"4 月データ　'sheet1　「4 月データ」というワークシートの Sheet1 を開きます。
表を開く　"年間平均　'sheet2　「年間平均」というワークシートの sheet2 を開きます。
```

ワークシートを開くと、データを追加したり、報告させたりすることができます。
セルの値はというコマンド は、行、列、値の 3 つのインプットを取ります。セルの値というレポータは、行と列の 2 つのインプットを取ります。

```
セルの値は　1　1　2　　1 行目の 1 列目のセルに 2 を書き込みます。
セルの値は　1　2　4　　1 行目の 2 列目のセルに 4 を書き込みます。
セルの値　1　1　　1 行目の 1 列目のセルの値を報告します。
2
セルの値　1　2 「」　　1 行目の 2 列目のセルの値を削除します。
```

注意：上記の基本用語を使って Excel とデータのやり取りをしているときに Excel のワークシート上を直接クリックして、データを追加しようとするとエラーになります。

表を開く と表を閉じる を使って、複数のワークシートの値を変更したり、値を追加したりすることができます。また別のワークシートに情報を転送することができます。

```
表を開く　"年間平均　'sheet2　「年間平均」というワークシートの Sheet2 を開きます。
セルの値は　1　2　4　　1 行目の 2 列目のセルに 4 を設定します。
名前は　：当月値　セルの値　1　2　1 行目の 2 列目のセルの値に名前を付けます。
表を閉じる　　ワークシート（年間平均）を保存して、閉じます。
表を開く　"降雨量　'sheet1　「降雨量」というワークシートを開きます。
セルの値は　2　3　：当月値　　2 行目の 3 列目のセルに当月値を挿入します。
表を閉じる　　「降雨量」というワークシートを閉じます。
```

注意：パス名を指定してワークシートが開かれたのでない場合、表を閉じるはマイクロワールド EX の作品と同じ場所にワークシートを保存して、閉じます。パス名が指定されている場合は、元と同じ場所にワークシートを保存します。

マイクロワールド EX で収集したデータは、Excel を使ってスプレッドシートやグラフの形式で表示することができます。つまり、マイクロワールド EX でデータを収集しながら、そのデータを Excel でグラフ表示することができます。

6. 数字、ワード、リストの操作

■この節の内容はオンラインヘルプからも見ることができます。
EX　　→メニューバーの［ヘルプ］をクリック→［リファレンス］
EX（ジュニア）　→メニューバーの［ヘルプ］をクリック→［リファレンス］

・例示された命令は、コマンドセンターに入力して試します。Enterを押すことで命令が実行されます。（＊コマンドセンターで命令を実行しないで次の行に移りたい場合は、Ctrl を押しながら Enter を押します。）
・例示された手順は、手順タブエリアに入力します。手順タブエリアの入力が終わったら、コマンドセンターに手順名を入力し、Enter を押すことで手順が実行されます

ロゴのデータは、基本的に数字とワード、リストの3種類に分類されます。ロゴには、データをまとめたり、分けたり、調べたりする基本用語があります。こうした基本用語を使って、簡単なクイズやゲーム、データ操作のプログラムを作ることができます。ここでは、リカージョン（再帰）を使った多数の例を紹介しています。

(1) ワードの操作

次のレポータは、ワードから文字を抽出します。

基本用語	働き	例	結果
最初	ワードの先頭の文字を報告します。	最初　"おはよう	お
最後	ワードの最後の文字を報告します。	最後　"おはよう	う
最初以外	ワードの先頭の文字以外のすべての文字を報告します。	最初以外　"おはよう	はよう
最後以外	ワードの最後の文字以外のすべての文字を報告します。	最後以外　"おはよう	おはよ
幾つめ	ワードの指定された位置の文字を報告します。	幾つめ　2　"おはよう	は

書くまたは**下へ書く**を使い、コマンドセンターから上の例を試してみましょう。

最初と最初以外

コマンドセンターから次のように入力し、Enter キーを押して実行します。

下へ書く　最後　"おはよう

次の行に、下のように表示されます。

う

```
下へ書く　最後以外　"おはよう
```

おはよ

```
下へ書く　最初　最初以外　"おはよう
```

は

最初以外は、その仕事として、**おはよう**の先頭の要素以外のすべての要素（**はよう**）を、**最初**に報告します。
最初にとって、インプットは**はよう**ですから、最初の要素である**は**を、**下へ書く**に報告します。これで、**下へ書く**はコマンドセンターにはを表示します。

ワード

ワードというレポータを使って、複数のワードをまとめて1つのワードを作ることができます。

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　ワード　"国際　"理解
```

次の行に、下のように表示されます。

国際理解

次の命令を使って、リストをワードに分けることができます。

```
下へ書く　(ワード　[This is a list])
```

このカッコは、**ワード**のインプットが 1 つの場合にも必要です。

ロゴでは数字はワードと見なされますから、同じレポータを使って数字を操作することもできます。

```
下へ書く　最後　1998
```

8

```
下へ書く　ワード　"Nagano　98
```

Nagano98

数字の前には引用符（"）を付ける必要はありません。

空のワード

ワードで大切なことは、文字が存在しない**空白もワード**であるということです。
ロゴでは、そうしたワードを***空のワード***と呼びます。

```
show　"
```

この命令は空白を表示します。この場合、引用符を使ってインプットがワードであることをロゴに伝える必要があります。ワードから文字を抽出するレポータのインプットに空のワードを使うことはできません。
下のようなエラーメッセージが表示されます。

```
下へ書く　最初　”
```

最初へのインプットがおかしいです。

「空のワード」という概念は、再帰的なテキスト操作手順のストップルールを作るのに非常に役立ちます。

ワードから文字を抽出するおもしろい手順を紹介します。

```
手順は　字の三角　：ワード
もし　：ワード　＝　”「止る」
下へ書く　：ワード
字の三角　最初以外　：ワード
終わり
```

コマンドセンターから実行してみます。
(**：ワード**で指定するワードには「 ”」を付ける必要があります)。

```
字の三角　”microworlds
```

次の行に、下のように表示されます。

microworlds
icroworlds
croworlds
roworlds
oworlds
worlds
orlds
rlds
d s
d s
s

字の三角では、ワードの文字がなくなるまで、ワードの文字数が1つずつ減っていきます。ストップルールは、「：ワードが空になったら、止まりなさい」という意味です。

この手順で注目したいのは、次の 2 点です。

・レポータ（**最初以外**）が手順にインプットを渡す方法
・再帰手順で繰り返しながらワードのインプットを渡す方法

実際には、**字の三角**は呼び出されるたびに異なるインプットを受け取ります。すなわち、前のワードから 1 文字を減らしたインプットを受け取るのです。

次の手順は、ワードの文字を逆の順序で表示します。

```
手順は　逆書き　：要素
もし　：要素　＝　”「書く　” 止まる」
書き入れる　最後　：要素
逆書き　最後以外　：要素
終わり
```

ページ上にテキストボックスを作ってから、コマンドセンターで逆書きを実行してみてください。

```
逆書き　”マイクロワールド
```

ドルーワロクイマ

逆書きは、前の**字の三角**に比べて少し複雑です。**逆書き**は呼び出されるたびに、**「最後以外　：ワード」**から異なるインプットを受け取ります。**書き入れる**は、カーソルのある位置に**最後**で抽出された文字を書きます（改行はしません）。

(2) リスト処理

ロゴの強力な機能の 1 つに、**リスト処理**があります。リストは、ワード、数字、リストの並びです。

リストでは、他のワードとリストを含むデータ構造体を作ることができます。ロゴでは、かぎカッコ（「」）を使ってリストであることを示します（英語でプログラムを書く場合は半角の []）。

数というレポータを使って、ワードまたはリストの要素数を数えることができます。

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　数　「1　「1　2」「17　「17　2」」」
```

次の行に、下のように表示されます。

3

リストの**「1 「1 2」「17 「17 2」」」**には、6つではなく、3つの要素が含まれています。1つ目の要素は**1**、2つ目は**「1 2」**というリスト、3つ目の要素は、**「17 「17 2」」**というリストです。
このように、リスト自体が要素であることもあるのです。
ワードから要素を抽出するレポータは、リストに対しても同じ働きをします。

基本用語	働き	例	結果
最初	リストの最初の要素を報告します。	最初　「1 「1 2」「17 「17 2」」」	*1*
最後	リストの最後の要素を報告します。	最後　「1 「1 2」「17 「17 2」」」	*「17 「17 2」」*
最初以外	リストの最初の要素以外のすべての要素を報告します。	最初以外　「1 「1 2」「17 「17 2」」」	*「1 2」「17 「17 2」」*
最後以外	リストの最後の要素以外のすべての要素を報告します。	最後以外 「1 「1 2」「17 「17 2」」」	*1 「1 2」*
幾つめ	リスト内の指定された位置の要素を報告します。	幾つめ　3　「1 「1 2」「17 「17 2」」」	*「17 「17 2」」*

空のリスト

ワードには、「空のワード」があるように、リストにも***空のリスト***があります。空のリストは空白行を報告します。

下へ書く 「 」

結果が同じに見えても、「空のリスト」と「空のワード」は同じものとは見なされないことに注意してください。
1つのワードを含むリストとそのワードそのものを表示するのとでは、結果は同じになりますが、ロゴでは、異なる種類のデータと見なされます。

下へ書く　"happy

happy

下へ書く　「happy」

happy

下へ書く　"happy　＝　「happy」

うそ

前に紹介した**字の三角**は、インプットとしてワードまたはリストを受け取るように変更することができます。

```
手順は　字の三角　：データ
もし　空か　：データ　「止まる」
下へ書く　：データ
字の三角　最初以外　：データ
終わり
```

コマンドセンターから実行してみます。

```
字の三角　「ウサギは　ジャンプが　上手です」
```

ウサギは　ジャンプが　上手です
ジャンプが　上手です
上手です

空かというレポータが使われていることに注目してください。
空かは、そのインプットが「空のワード」、または「空のリスト」であるかどうかを調べます。

リストを作る

ロゴには、複数の要素からリストを作成する基本用語があります。最もよく使われるのは、**文**という基本用語です。**文**は、インプットされたすべての要素を使ってリストを作ります。

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　文　「私は」「幸運な　旅人だ」
```

次の行に、下のように表示されます。

私は幸運な旅人だ

ロゴには、この他にも、**リスト**というレポータがあります。**文**と**リスト**の違いは、**リスト**がインプットの要素の構造を保持する点です。

同様にコマンドセンターを使って試します。

```
下へ書く　リスト　「私は」「幸運な　旅人だ」
```

[私は] [幸運な　旅人だ]

たいていの**対話型プログラム**では、**文**を使って単語を表示します。これは、**リスト**では、カッコが表示されてしまうためです。

同じく、**最初に**および**最後に**も、要素からリストを作る基本用語です。

最初には、2つ目のインプットの先頭に1つ目のインプットを追加したリストを作成して報告します。2つ目のインプットはリストであることが必要です。

下へ書く　最初に　"インターネット　「「ウェブ」「gif　jpeg」」
インターネット「ウェブ」「gif　jpeg」

最後には、2つ目のインプットの最後に1つ目のインプットを追加したリストを作成して報告します。2つ目のインプットはリストであることが必要です。

下へ書く　最後に　"インターネット　「「java」「ウェブ」「gif　jpeg」」
「java」「ウェブ」「gif　jpeg」インターネット

最初にと**最後に**、**リスト**の3つの基本用語は、操作可能なデータ構造体としてリストを利用するときに非常に重要です。

次の表は、ワードとリストを結合する上記の4つの基本用語をまとめたものです。
［結果］欄は、コマンドセンターで**下へ書く**を使って試した結果です。
例：　下へ書く　リスト　"犬　"馬

レポータ	インプット1	インプット2	結果
最初に	"犬	"馬	*エラーメッセージ*
リスト	"犬	"馬	*犬馬*
最後に	"犬	"馬	*エラーメッセージ*
文	"犬	"馬	*犬馬*
最初に	"ロゴ	「すばらしい」	*ロゴすばらしい*
リスト	"ロゴ	「すばらしい」	*ロゴ［すばらしい］*
最後に	"ロゴ	「すばらしい」	*すばらしい　ロゴ*
文	"ロゴ	「すばらしい」	*ロゴすばらしい*
最初に	「私は」	「幸運な　旅人です」	*[私は] 幸運な　旅人です*
リスト	「私は」	「幸運な　旅人です」	*[私は] [幸運な　旅人です]*
最後に	「私は」	「幸運な　旅人です」	*幸運な　旅人です [私は]*
文	「私は」	「幸運な　旅人です」	*私は 幸運な　旅人です*
最初に	"キャンプ	「　」	キャンプ
リスト	"キャンプ	「　」	キャンプ *[ ]*
最後に	"キャンプ	「　」	キャンプ
文	"キャンプ	「　」	キャンプ

(3) ワードとリストで遊ぶ

これまでに学んだことを踏まえて、ワードとリストを操作する手順を作ってみましょう。

a. 文章を作るプログラム

これは、言葉を組み合わせて文章を作る簡単な手順です。

```
手順は　会話　：動物　：音
もし　どれか本当　空か　：動物　空か　：音
「書く 「これで全部！」 止まる」
書く　文　最初　：動物　最初　：音
会話　最初以外　：動物　最初以外　：音
終わり
```

ページにテキストボックスを作成してから、「犬　小鳥　馬」「ほえる　さえずる　いななく」というインプットを付けて、コマンドセンターから実行してみましょう。

```
会話 「犬　小鳥　馬」「ほえる　さえずる　いななく」
```

テキストボックスに次のように書かれます。

```
犬 ほえる
小鳥 さえずる
馬 いななく
これで全部！
```

会話は再帰手順です。再帰するたびに、リストは最初の要素を削除されて、順々に短くされていきます。

次の行に注目してください。

```
もし　どれか本当　空か　：動物　空か　：音
「書く 「これで全部！」 止まる」
```

どれか本当はレポータです。このレポータは2つのインプットを取ります。2つのインプットの両方について調べ、どちらかが**真**であれば、1つ目の命令リストを実行します。もし、真でなければ（偽）、2つ目の命令リストを実行します。
この例では、**：動物**と**：音**について空であるかどうかを調べます。どちらかが空の場合、**書く「これで全部！」 止まる**が実行されます。

このようにインプットに真偽値をとるレポータを***論理演算子***と呼びます。

どちらか本当は、そのインプットのいずれか一方が**真**の場合に**真**を報告します。
この他にも、**皆本当**と**反対**の2つの論理演算があります。

皆本当は、すべてのインプットが真の場合に**真**を報告します。
反対は、そのインプットの論理の逆を報告します。すなわち、インプットが真の場合は**偽**、偽の場合は**真**を報告します。

会話を変更して、でたらめな文章を作る「ジョーク」の手順を作って見ましょう。

2つのリストを要素ごとに対応させるのではく、両方のリストからランダムに単語を拾うようにロゴに指示します。
どれかというレポータは、ワードまたはリストからランダムに要素を報告します。**もし**の行をなくして、手順が際限なく実行されるようにしましょう。従って**最初以外**を削除して、インプットを変更しないようにします。

```
手順は　ジョーク　　：名詞　：動詞
書く　文　どれか　　：名詞　どれか　：動詞
ジョーク　：名詞　：動詞
終わり
```

2つのインプットを付けて、コマンドセンターから実行してみましょう。(ページにテキストボックスがあることを確認してから行ってください)。

```
ジョーク 「犬　小鳥　人　馬」「ほえる　さえずる　さけぶ　いななく」
```

テキストボックスには際限なく、次のようなデタラメな行が書かれていきます。

```
犬　さけぶ
人　いななく
小鳥　ほえる
犬　さけぶ
馬　さえずる
```

手順を止めるには、[すべてを止める]  を押します。

インプットと再帰行を削除して、**ジョーク**を次のような手順にすると、ボタンからこの手順を実行することもできます。

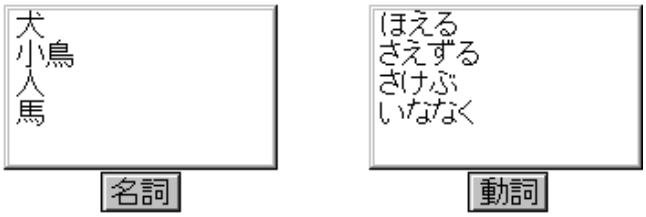
```
手順は　ジョーク
書く　文　どれか 「犬　小鳥　人　馬」　どれか 「ほえる　さえずる　さけぶ　いななく」
終わり
```

手順タブエリアに上の手順を作成したら、**ジョーク**という命令のボタンを作成して、[無限に]にチェックマークを付けます。

上のインプットを2つのテキストボックスに残す方法もあります。

ページ上に、**名詞**と**動詞**という名前の2つのテキストボックスを作り、それぞれにインプットのワードを入力します。

各テキストボックスが、ワードの入れものの働きをします。



手順は次のように変更します。

```
手順は　ジョーク２
書く　文　行のどれか　”名詞　行のどれか　”動詞
ジョーク２
終わり
```

名詞テキストボックスと**動詞**テキストボックスに、テキストボックスの内容としてワードが格納されていますから、**ジョーク２**にはインプットは必要ありません。

b . ワードの要素をつなげる

次の手順は、**ワード**というレポータを使って新しいワードを作り、**文**を使って、その単語を使った文章を作ります。

```
手順は　紹介　：県名
書く　文　「出身は」　：県名
書く　文　「私は」　ワード　：県名　”人
終わり
```

コマンドセンターから実行してみましょう。

```
紹介　”福岡県
```

テキストボックスに次のように表示されます。

出身は福岡県
私は福岡県人

c．文字の順を変える

プリントダウンという手順は、単に空になるまで、インプットの1つ目の要素を表示します。

```
手順は　プリントダウン　：要素
もし　からか　：要素　「止まる」
書く　最初　：要素
プリントダウン　最初以外　：要素
終わり
```

コマンドセンターから実行してみます。

```
プリントダウン　"wonderful
```

w
o
n
d
e
r
f
u
l

この手順に少し変更を加えて、**繰り返しとリカージョン（再帰）**で説明したリカージョンを試してみましょう。
再帰行の後に書く命令をもう1つ追加します。

```
手順は　プリントダウン　：要素
もし　からか　：要素 「止まる」
書く　最初　：要素
プリントダウン　最初以外　：要素
書く　最初　：要素
終わり
```

コマンドセンターから実行してみます。

```
プリントダウン　"rain
```

r
a
i
n
n
i
a
r

手順に対するすべての再帰呼び出しは、そのすべての行が処理された後で停止します。これによって、目的の効果が生み出されます。

d．ロゴのレポータ

プログラム内の計算を除いて、ロゴのレポータの最後の文字は**「〜か」**です。
レポータは、数字とワード、リストに使うことができます。

レポータ	**働き**
空か	インプットが空の場合に**真**を報告します。
同じか	インプットが等しい場合に**真**を報告します。英大文字と小文字の区別はされません。
より大きいか	最初のインプットが2つ目のインプットより大きい場合に**真**を報告します。
全く同じか	インプットが等しい場合に**真**を報告します。英大文字と小文字が区別されます。
より小さいか	最初のインプットが2つ目のインプットより小さい場合に**真**を報告します。
リストか	インプットがリストである場合に**真**を報告します。
ある	か最初のインプットが、2つ目のインプットの中にある場合に**真**を報告します。
数字か	インプットが数字の場合に**真**を報告します。
ワードか	インプットがワードの場合に**真**を報告します。
=	インプットが等しい場合に**真**を報告します。
<	最初のインプットが2つ目のインプットより小さい場合に**真**を報告します。
>	最初のインプットが2つ目のインプットより大きい場合に**真**を報告します。

(4) アウトプットを持つ手順

レポータの働きをする手順

ロゴでは、レポータの働きをする手順を作ることもできます。このためには、基本用語の**表示（**コマンド）を使います。
レポータの定義として使えるかどうかは、数字を扱うとすぐに分かります。例えば、**下へ書く**を使ってインプットを２乗する手順を作ってみましょう。

```
手順は　二乗　：x
下へ書く　：x　＊　：x
終わり
```

この手順を試すと、結果が表示されます。

```
二乗　5
```

2 5

しかしながら、この手順でできることは、これだけです。もっと複雑な手順の一部としてこの手順を利用することはできません。例えば、2つの数字の2乗の合計値を求めたいとします。このために、結果を別の手順に伝えられる必要があります。上記の二乗の**下へ書く**を**表示**に置き換えてみましょう。

```
手順は　二乗　：x
表示　：x　＊　：x
終わり
```

二乗はレポータになりました。コマンドセンターから試してみます。

```
二乗　5
```

25 をどうするのかわかりません。

レポータから返された答え（アウトプット）をどのようにすればよいのか分からないというエラーメッセージが返されます。これは行の先頭にコマンドがないためです。

次を試してください。

```
下へ書く　二乗　5
```

2 5

二乗は**下へ書く**というコマンドに値を渡すレポータとして機能しています。

二乗を和のインプットに使い、2つの数字の2乗の合計値を求めることができます。

```
下へ書く　和　二乗　5　二乗　3
```

3 4

次の手順は、2つの数字の平均値を求めるレポータです。

```
手順は　平均値　：x　：y
表示　（：x　+　：y）　/　2
終わり

下へ書く　平均値　20　50
```

3 5

2つのレポータを組み合わせることによって、平均値の2乗を求めることができます。

```
下へ書く　二乗　平均値　20　50
```

1225

ワードとリストを扱うレポータ手順はもっと役立ちます。多くの場合、レポータ手順はツールの役目を果たすことができます。ツールとは、様々な目的に使える汎用性ある手順です。次の手順は、同じ単語をつなげることによって単語のサイズを2倍にする簡単なツールです。

```
手順は　ダブル　：単語
表示　ワード　：単語　：単語
終わり

下へ書く　ダブル　”ヤッホー

ヤッホーヤッホー
```

このレポータを別の手順のインプットに使うことができます。下記は、**ワードの操作**で作った**字の三角**という手順に使用した例です。

```
字の三角　ダブル　”ヤッホー

ヤッホーヤッホー
ッホーヤッホー
ホーヤッホー
ーヤッホー
ヤッホー
ッホー
ホー
ー
```

a．アウトプットを持つ再帰手順

逆書きは、レポータではなくコマンドです。

```
手順は　逆書き　：要素
もし　：要素　＝　”「書く　”　止まる」
書き入れる　最後　：要素
逆書き　最後以外　：要素
終わり
```

逆転という新しい手順を作ってみましょう。**逆転**は、結果を表示するのではなく、結果を報告することを除けば、**逆書き**に似ています。第1段階としては、単に**書き入れる**（と**書く**）を**表示**に変更します。

```
手順は　逆転　：要素
もし　：要素　＝　”「表示　”」
```

```
表示　最後　：要素
逆転　最後以外　：要素
終わり
```

試してみましょう。

```
下へ書く　逆転　”cat

t
```

実行結果がtacになるようにするには、どこを修正すればよいでしょうか。ここで問題なのは、3行目の表示によって、結果が出力されてしまうことです。逆転の実行結果が最後にまとめて1つの単語として出力されるように手順を修正します。

```
手順は　逆転　：要素
もし　：要素　＝　”「表示　”」
表示　ワード　最後　：要素　逆転　最後以外　：要素
終わり
```

(5) 作品でワードやリストを操作する

条件文やレポータ、アウトプットを持つ手順に関する知識を活用することによって、便利なプログラムを作ることができます。ロゴで独自のレポータを作るのは簡単です。自分で定義したレポータを、作品を作るツールの手順として利用することもできます。

例えば、**逆転**(**アウトプットを持つ再帰手順**を参照)を作った後、前から読んでも後ろから読んでも同じになるワードを調べるレポータを作りたいとしましょう。次の**逆さ言葉**はそうした手順の1例です(最初に**逆転**を作っておく必要があります)。この手順に**もしどちらかを**が使われていることに注目してください。

```
手順は　逆さ言葉　：要素
もしどちらかを　：要素　＝　逆転　：要素 「表示　”本当」「表示　”嘘」
終わり
```

この手順の意味を説明すると次のようになります。

「インプットのワードがインプットのワードの文字を逆順に並べたものと同じ場合は、真を報告し、そうでない場合は偽を報告しなさい。」

＝は、それ自体がレポータなので、**逆さ言葉**はもっと簡潔に書くことができます。

```
手順は　逆さ言葉　：要素
表示　：要素　＝　逆転　：要素
終わり
```

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　逆さ言葉　"racecar
```

次の行に、下のように表示されます。
ほんとう

次の手順は、インプットが母音で始まるワードかどうかを調べます。

```
手順は　母音　：単語
表示　あるか　最初　：単語　[a e i o u]
終わり
```

コマンドセンターから試します。

```
下へ書く　母音　"eaon
```

ほんとう

次にワードの最初の英字に従ってワードの前に"a"または"an"を付ける手順を紹介しましょう。

```
手順は　冠詞　：単語
もし　母音　：単語 「表示　文　”an　：単語」
表示　文　”a　：単語
終わり
```

もしの代わりに**もしどちらかを**を使うこともできます。この場合、手順の2行目は、2つ目の命令リストの内容になります。

```
手順は　冠詞　：単語
もしどちらかを　母音　：単語 「表示　文　”an　：単語」
「表示　文　”a　：単語」
終わり
```

冠詞手順はレポータですから、別のコマンドやレポータのインプットに使うことができます。

```
下へ書く　冠詞　”airport
```

an airport

母音かどうかを調べるという考え方は、ピッグラテン語（英語圏で子供がふざけて使う言葉）を定義するプログラムに活かすことができます。

```
手順は　ラテン　：例文
もし　空か　：例文 「表示 「」」
表示　文　pig　最初　：例文　ラテン　最初以外　：例文
終わり
```

```
手順は　pig　：単語
もし　あるか　最初　：単語 「ａｅｉｏｕｙ」「表示　ワード　：単語　”ay」
表示　pig　ワード　最初以外　：単語　最初　：単語
終わり

下へ書く　ラテン　「I love Logo」

Iay ovelay ogoLay
```

ラテンは親手順です。この手順は、子手順である**pig**を呼び出して、そのインプットの最初の単語をピックラテン語に変換し、文のすべて単語の変化が終わるまでインプットを1単語ずつ減らしていきます。

pigは単語を調べて、その一部を変更します。**あるか**を使って単語が母音（yを含む）で始まっているかどうかを調べ、母音で始まっている場合は、単語の最後に"ay"を追加します。母音で始まっていない場合は、単語の先頭が母音になるまで、単語の最初の英字を切り取って、最後に追加する処理を繰り返します。

（このプログラムは単語に母音が含まれている場合しか機能しないことに注意してください。プログラムが誤りを犯さないようにするには、例外かどうかをチェックする手順を追加する必要があります。）

別のバージョンを紹介しましょう。**語尾**という手順は、**ight**で終わる単語の最後を**ite**に変更します。

```
手順は　語尾　：単語
もし　あるか　”ght　：単語 「表示　ワード　最後以外　最後以外　最後以外　：単語　”te」
表示　：単語
終わり
```

次の**文変更**という手順は、**語尾**を利用しています。

```
手順は　文変更　：原文
もし　からか　：原文 「表示 「」」
表示　最初に　語尾　最初　：原文　文変更　最初以外　：原文
終わり
```

この**文変更**は**ラテン**とほぼ同じですが、**文**ではなく**最初に**が使われています。

```
下へ書く　文変更 「Star light star bright」

Star lite star brite
```

語尾は、単語がghtで終わっているかどうかを調べます。ghtで終わっている場合は、単語からその部分を切り取って、te追加します。ghtで終わっていない場合は、インプットされた単語を単に表示します。

語尾の次の行に注目してください。

```
もし　あるか　"ght　：単語 「表示　ワード　最後以外　最後以外　最後以外　：単語　"te」
```

：単語がlightの場合、**最後以外　最後以外　最後以外　：単語**　はどのようになるのでしょうか。

ロゴの命令は後ろ側から順に読み取られるというルールを適用してみましょう

最初の**最後以外**のインプットは、最後以外　最後以外　"lightです。
2つ目の**最後以外**のインプットは、最後以外　"lightです。
3つ目の**最後以外**のインプットは"lightで、ligh が報告されます。
2つ目の **最後以外**インプットは"ligh で、lig が報告されます。
最初の**最後以外**のインプットは"ligで、liが報告されます。

文変更は親手順です。リストを受け取り、その中の単語の一部を別の英字に置き換えます。

(6) サンプルツール

ここでは、プログラムを作るときにツールとして利用できるレポータ手順を紹介します。
範囲は、ある値が特定の範囲内にあるかどうかを調べます。

```
手順は　範囲　　：x　：low　：high
表示 反対　どれか本当　（ ：x　<　 ：low ) ( ：x　>　 ：high ）
終わり
```

コマンドセンターから次のように入力し、Enter キーを押して実行します。

```
下へ書く　範囲　 5　3　9
```

次の行に、下のように表示されます。

本当

置き換えは、文中の単語（単語1）を別の単語（単語2）に置き換えます。

```
手順は　置き換え　：単語1　：単語2　：例文
もし　からか　：例文 「表示 「」」
もし　：単語1　＝　最初　：例文
「表示　文　：単語2　最初以外　：例文」
表示　最初に　最初　：例文　置き換え　：単語1　：単語2　最初以外　：例文
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　置き換え　”ジャンプ　”シュート　「ゴールに　向かって　ジャンプ」
```

ゴールに　向かって　シュート

ソートは単語のリストをアルファベット順に並べ替えて報告します。

```
手順は　ソート　：単語リスト
表示　ソート準備　：単語リスト 「」
終わり

手順は　ソート準備　：単語リスト1　：単語リスト2
もし　空か　：単語リスト1 「表示　：単語リスト2」
表示　ソート準備　最初以外　：単語リスト1　書きこみ1　最初　：単語リスト1　：単語リスト2
終わり

手順は　書きこみ1　：単語　：単語リスト
もし　空か　：単語リスト 「表示　最初に　：単語　：単語リスト」
もし　先頭　：単語　最初　：単語リスト 「表示　最初に　：単語　：単語リスト」
表示　最初に　最初　：単語リスト　書きこみ1　：単語　最初以外　：単語リスト
終わり

手順は　先頭to before? ：a　：b
もし　どれか本当　空か　：a　空か　：b 「表示　”うそ」
もしどちらかを （アスキー　最初　：a） ＝ （アスキー　最初　:b）
「表示　先頭　最初以外　：a　最初以外　：b」
「表示 （アスキー　最初　：a） ＜ （アスキー　最初　：b)」
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　ソート　「Ann Diane Zak Tom Odette」
```

Ann Diane Odette Tom Zak

何番目は、要素のリスト内の位置を報告します。要素はリストのメンバーである必要があります。

```
手順は　何番目　：要素　：文字リスト
もし　空か　：文字リスト 「表示　0」
もし　：要素　＝　最初　：文字リスト 「表示　1」
表示　1　+　何番目　：要素　最初以外　：文字リスト
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　何番目　”あ 「や　ま　あ　ら　し」
```

3

次の3つの手順は、2つのインプットのデータを操作するツールとして利用できます。

ダブリは、2つのインプットで共通するものを報告します。

```
手順は　　ダブリ　：リスト1　：リスト2
もし　空か　：リスト1「表示　「」」
もし　あるか　（最初　：リスト1）　：リスト2
「表示　文　（最初　：リスト1）
ダブリ　( 最初以外　：リスト1)　：リスト2」
表示　ダブり　（最初以外　：リスト1）　：リスト2
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　ダブリ　[1 2 3 4 5] [3 4 5 6]
```

3 4 5

Subsetは、1つ目のインプットのリストが2つ目のインプットのリストであるかどうかを調べます。

```
手順は subset　：リスト1 ：リスト2
もし　空か　：リスト1「表示　”本当」
もし　反対　あるか　最初　：リスト1　：リスト2「表示　”嘘」
表示　subset　最初以外　：リスト1　：リスト2
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　 subset　[2 3] [2 3 4 5]
```

本当

結合は、2つのインプットを結合させます。

```
手順は　結合　：リスト1　：リスト2
もし　空か　：リスト1「表示　：リスト2」
もしどちらかを　あるか　最初　：リスト1　：リスト2
「表示　結合　最初以外　：リスト1　：リスト2」
「表示　最初に　最初　：リスト1　結合　最初以外　：リスト1　：リスト2」
終わり
```

コマンドセンターから試しましょう。

```
下へ書く　結合　[1 2 3 4 5] [3 4 5 6 7]
```

1 2 3 4 5 6 7