All Rights Re:

リスト：仲間に分けたり，一緒にまとめたり

リストという重要な概念を学びましょう．

グループ（リスト）を作ろう

■ **「食事」に関するグループを作ってみる**

グループを作るには，中括弧（「{」と「}」）を使います．例えば，お子さんの七五三をする年齢を表す数のグループを作るには，次のように中括弧の中にカンマ（「,」）で区切って書いていきます．カンマを使うのは，どこまでが最初の数で，どこからが次の数か判るようにするためです．

```
{7, 5, 3}
```

小学校から大学までの標準的な在学年数を表す数から構成されるグループであれば，次のように同じ数が含まれることもあります．

```
{6, 3, 3, 4}
```

さて，本題に戻って食事に関するものとして朝食のグループを作ってみましょう．これまでの数のグループを真似て作ると次のようになると思いがちですが，これは少しだけ書き直す必要があります．

```
{ごはん, 味噌汁, 焼き魚}
```

朝食の内容に関するグループを正しく作るには，次のように食事の内容を二重引用符（ダブルクオーテーション：「"」）で囲む必要があります．

```
{"ごはん", "味噌汁", "焼き魚"}
```

これはちょっと不思議な感じがしますが，次のように，小括弧，中括弧，大括弧を含むような括弧のグループを作ると必要なことがわかります．まずは，引用符を付けないで作ってみましょう．

```
{), (,}, {,], [}
```

このように引用符を付けないと，どこまでがグループなのか判らなくなってしまいます．見方によっては，二つのグループを併記しているようにも見えますし，一つの大きなグループのようにも見えます．今度はちゃんと引用符を付けて作ってみます．非常に判りやすくなりました．

```
{")", "(", "}", "{", "]", "["}
```

【数と数字】朝食の内容や括弧は二重引用符で囲む必要があるのに，数の場合には囲んでいませんでした．なぜでしょうか．実は，数と言っても数値と数字では意味が違

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

います．数値は算数とか数学で言う数ですが，数字は，その数を表す「文字」です．ですから，数値として扱いたいときは二重引用符はいりません．逆に，朝食の内容のように文字や文章として数を扱いたいときは，数字と区別するために，二重引用符が必要になります．

せっかく朝食のグループを作るのですから，グループに名前を付けましょう．名前を付けるには，グループの前に，グループ名と等号（「=」）を書き加えてから，SHIFTを押しながらENTERを押してください．そうすると，次のように入力されていたものが，

朝食 = {"ごはん", "味噌汁", "焼き魚"}

次のように変化します．

In[1]:= 朝食 = {"ごはん", "味噌汁", "焼き魚"}

Out[1]= {ごはん，味噌汁，焼き魚}

何やら入力した覚えのないものが出てきました．最初の行の左側に表示されている「In[1]」は，*Mathematica*が命令（グループに名前を付けろ！）を受け取りました，その整理番号は何番です，という意味です．次の行の左側に表示されている「Out[1]」は，その整理番号の命令を実行した結果，という意味です．

【評価と計算】*Mathematica*は計算のできる利口なワープロソフトだと考えると良いでしょう．普通の文章を入力することもできますし，電卓のような計算をさせることもできます．ワープロソフトと違うのは，入力した数式を計算して欲しいと思ったら，その場所でSHIFT+ENTERを押すだけ計算結果を求めてくれます．この操作のことを「評価する」と言います．最近の*Mathematica*では命令をしなくても自動的に計算を行ってくれる便利な方法もありますが，この章では扱いません．

さて，朝食のグループには名前を付けたので，名前を呼べば確認できます．今度は，名前だけを入力して，SHIFT+ENTERを押してみましょう．次のように，朝食のグループを確認することができます．

In[2]:= 朝食

Out[2]= {ごはん，味噌汁，焼き魚}

同じように，昼食のグループにも名前を付けましょう．次のようにグループ名と等号（「=」）を入力をしてから，SHIFT+ENTERを押してみましょう．

In[3]:= 昼食 = {"サンドイッチ", "サラダ", "コーヒー"}

Out[3]= {サンドイッチ，サラダ，コーヒー}

All Rights Re:

これで朝食と昼食の準備ができました．どんな食事の内容かを確認したい場合は，グループ名を入力してから，SHIFT+ENTERを押してください．このように*Mathematica*と会話をすることが，*Mathematica*を使うということです．

```
In[4]:= 朝食
Out[4]= {ごはん，味噌汁，焼き魚}

In[5]:= 昼食
Out[5]= {サンドイッチ，サラダ，コーヒー}
```

これ以外にもおやつであったり，夕食であったり，様々なグループを作ることが出来ます．このように，中括弧（「{」と「}」）で作ったグループのことを「リスト（List）」と呼びます．リストは，*Mathematica*の非常に重要な概念になりますので，ぜひ名前を覚えるようにしてください．

```
In[6]:= おやつ = {"ショートケーキ", "ミルクティー"}
Out[6]= {ショートケーキ，ミルクティー}

In[7]:= 夕食 = {"肉じゃが", "生姜焼", "味噌汁", "ごはん", "漬物"}
Out[7]= {肉じゃが，生姜焼，味噌汁，ごはん，漬物}
```

【シンタックスカラーリング】*Mathematica* 6では，既に使用したグループ名を入力すると「黒色」で表示されますが，まだ使用されていないグループ名の場合は「青色」で表示されます．これは即ち，その時点で*Mathematica*に覚えさせているグループ名でなければ「青色」で表示されていることを意味します．プログラミング言語の用語でいえば，未定義の変数は青色，定義済みの変数は黒色で表示されます．

- **いろいろなグループを作ってみる**

先ほどの食事の内容は長坂家のものだったとします．長坂家だけでなく，いろいろな家庭の食事の内容に関するグループを作ってみます．岩見家の食事の内容のグループを作ってみましょう．あくまでも仮定の話ですが，朝食は次のようなグループになると思います．

```
{"食パン", "ハム", "コーヒー", "サラダ"}
```

このグループにも名前を付けたいので，グループ名と等号（「=」）を入力をしてから，SHIFT+ENTERを押してみましょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[8]:= 朝食 = {"食パン", "ハム", "コーヒー", "サラダ"}

Out[8]= {食パン, ハム, コーヒー, サラダ}
```

岩見家の朝食のグループにも名前を付けたので，名前を呼べば確認できます．今度も，名前だけを入力して，SHIFT+ENTERを押してみましょう．次のように，朝食のグループを確認することができます．

```
In[9]:= 朝食

Out[9]= {食パン, ハム, コーヒー, サラダ}
```

ここで疑問が一つ生まれます．長坂家の朝食のグループを知りたくなったらどうすれば良いでしょうか．長坂家の朝食のグループを教えてもらいたくて，次のように名前を呼んでも岩見家のグループしか*Mathematica*は教えてくれません．

```
In[10]:= 朝食

Out[10]= {食パン, ハム, コーヒー, サラダ}
```

実は，同じ名前のグループを複数定義しても，*Mathematica*は最後に覚えたものしか教えてくれません．既に，長坂家の朝食のグループは忘れ去られているので，いくら聞いても岩見家のグループしか出てきません．次のように河合家の朝食を覚えてもらうと，今度は岩見家の朝食グループを忘れてしまいます．

```
In[11]:= 朝食 = {"ごはん", "納豆", "味噌汁", "漬物"}

Out[11]= {ごはん, 納豆, 味噌汁, 漬物}

In[12]:= 朝食

Out[12]= {ごはん, 納豆, 味噌汁, 漬物}
```

【教えること＝定義＝覚えさせること】朝食グループのように，「等号（=）」を使って名前を付けて*Mathematica*に教えることを「定義」と言います．人が言葉を覚えるときに，その単語の意味を教えてもらうのと同じで，その名前が意味することを*Mathematica*に覚えさせることになります．数学で，変数の値を「$a = 1$」のように表記するのと同じですね．

人間であれば，話の文脈を理解して適切なものを教えてくれるのですが，コンピュータは人間とは違って融通が利かない石頭なので仕方がないのでしょうか．安心してください．実は，*Mathematica*にも「コンテキスト（文脈）」という仕組みが備わっています．そしてコンテキストごとに定義を覚えさせることが出来ます．

長坂家というコンテキスト（文脈）を「長坂家」という名前で表現するならば，長坂家の朝食や昼食のグループを次のように定義することが出来ます．ここで，コンテキ

All Rights Re:

スト名である「長坂家」とグループ名の間に始まりを表す単一引用符（「`」）が入っていることに注意してください.

```
In[13]:= 長坂家`朝食 = {"ごはん", "味噌汁", "焼き魚"}
Out[13]= {ごはん, 味噌汁, 焼き魚}

In[14]:= 長坂家`昼食 = {"サンドイッチ", "サラダ", "コーヒー"}
Out[14]= {サンドイッチ, サラダ, コーヒー}
```

続いて，岩見家の朝食と昼食のグループも定義してみましょう.

```
In[15]:= 岩見家`朝食 = {"食パン", "ハム", "コーヒー", "サラダ"}
Out[15]= {食パン, ハム, コーヒー, サラダ}

In[16]:= 岩見家`昼食 = {"素麺", "サラダ"}
Out[16]= {素麺, サラダ}
```

ついでにおやつや夕食に関するグループも定義しておきましょう. *Mathematica*にグループを覚えてもらう場合，これまで*Mathematica*は復唱してくれていました. 覚えた内容を次の行に表示してくれていたと思います. 復唱がいらない場合は，行の最後にセミコロン（「;」）を付けてください. *Mathematica*はグループの内容を記憶しますが，その内容を復唱することはありません.

```
In[17]:= 長坂家`おやつ = {"ショートケーキ", "コーヒー"};

In[18]:= 岩見家`おやつ = {"梨", "コーヒー"};
```

【複合式と出力の抑制】 *Mathematica*では，セミコロン（「;」）で区切ることで同時に複数の指示を行うことが出来ます. 行の最後にセミコロンだけを書くことは，次に実行すべき指示を空欄，つまりは何もしない，と指示していることに他なりません. 従って，何も結果が表示されません. Version 6よりも前の版では，出力が異様に大きい場合も全てを表示しようとするので，セミコロンの利用が必須でしたが，Version 6からは巨大な出力を抑制してくれるので非常に便利になりました.

```
In[19]:= 長坂家`夕食 = {"肉じゃが", "生姜焼", "味噌汁", "ごはん", "漬物"}
Out[19]= {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[20]:= 岩見家`夕食 = {"焼肉", "味噌汁", "サラダ", "ごはん"}

Out[20]= {焼肉, 味噌汁, サラダ, ごはん}
```

- **どんなグループだったか問い合わせてみる**

実際に長坂家と岩見家のそれぞれの食事の内容を覚えているのかを確認してみましょう．確認する際には，コンテキスト名である「長坂家」や「岩見家」とグループ名の間に始まりを表す単一引用符（「`」）を忘れずに付けてください．

```
In[21]:= 長坂家`朝食

Out[21]= {ごはん, 味噌汁, 焼き魚}
```

```
In[22]:= 岩見家`朝食

Out[22]= {食パン, ハム, コーヒー, サラダ}
```

コンテキスト名を付け忘れると次のように河合家の朝食グループが出てきてしまいます．これは，一番最後に定義した（コンテキスト名を指定しない）朝食グループが河合家のものだったからです．

```
In[23]:= 朝食

Out[23]= {ごはん, 納豆, 味噌汁, 漬物}
```

実は明示的にコンテキスト名を指定しなくても「Global」というコンテキスト名が指定されていると*Mathematica*は解釈します．長坂家の話をしている最中に，わざわざ「長坂家の」という指定はしないのと同じで，判りきっていることは省略できるようになっています．いま使われているコンテキストは，$Contextという名前で管理されています．この出力の「Global」は特定のコンテキストでない，即ち汎用のコンテキストという意味になります．

```
In[24]:= $Context

Out[24]= Global`
```

長坂家と岩見家に関するコンテキストで管理されているグループを問い合わせてみましょう．「長坂家のコンテキストで管理されているグループ名は何ですか？」と*Mathematica*に聞くわけです．すると，次のように「長坂家」のコンテキストには朝食，昼食，おやつ，夕食のグループが存在していることがわかります．

```
In[25]:= ? 長坂家`*
```

▼ 長坂家`

夕食	昼食	朝食	おやつ

同様に岩見家についても問い合わせてみます．先ほど，定義したグループ名が存在していることがわかります．気になるグループ名をマウスでクリックしますと，そのグループ（リストですね）の要素を確認することが出来ます．

```
In[26]:= ?岩見家`*
```

▼ 岩見家`

夕食	昼食	朝食	おやつ

ところで，朝食に関するグループを3つ（長坂家，岩見家，河合家）定義しましたが，どのようにしたら，コンテキスト名を入力しなくても長坂家であったり，岩見家であったりの朝食グループを確認することが出来るでしょうか．暗にどのようなコンテキストが使われてるかは，次の変数（$ContextPath）を確認することでわかります．

```
In[27]:= $ContextPath

Out[27]= {PacletManager`, WebServices`, System`, Global`}
```

このグループの定義を次のように変更することで，長坂家のコンテキストが優先されるようになります．この場合，ただの「朝食」でも長坂家の朝食グループを確認することが出来ます．

```
In[28]:= $ContextPath = {"長坂家`", "PacletManager`",
           "WebServices`", "System`", "Global`"}

Out[28]= {長坂家`, PacletManager`, WebServices`, System`, Global`}

In[29]:= 朝食

Out[29]= {ごはん, 味噌汁, 焼き魚}
```

もちろん，このグループの定義を元に戻すことで，ただの「朝食」では河合家の朝食グループしか参照されない状態に戻すことが出来ます．

```
In[30]:= $ContextPath = {"PacletManager`", "WebServices`",
           "System`", "Global`"}

Out[30]= {PacletManager`, WebServices`, System`, Global`}

In[31]:= 朝食

Out[31]= {ごはん, 納豆, 味噌汁, 漬物}
```

【カスタマイズと定義済み変数】*Mathematica*には「$ContextPath」のように定義済みの変数がたくさんあります．それらの変数の値を操作することで，自分好みの*Mathematica*にカスタマイズすることが出来ます．より簡単に変更できるよう，見た目

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

や編集操作に関することも含めて，編集メニューの「環境設定...」でカスタマイズが出来るようになっています．

グループ（リスト）の性質を調べたり操作をしよう

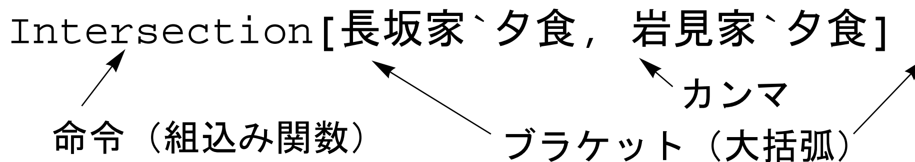
■ 長坂家と岩見家に共通するもの

長坂家と岩見家という二つの家庭の食事に共通するものを調べることにしましょう．既に，朝食，昼食，おやつ，夕食のグループをそれぞれの家庭について定義しています．これらのグループごとに共通するものを調べましょう．

朝食（長坂家）	ごはん，味噌汁，焼き魚
朝食（岩見家）	食パン，ハム，コーヒー，サラダ
昼食（長坂家）	サンドイッチ，サラダ，コーヒー
昼食（岩見家）	素麺，サラダ
おやつ（長坂家）	ショートケーキ，コーヒー
おやつ（岩見家）	梨，コーヒー
夕食（長坂家）	肉じゃが，生姜焼，味噌汁，ごはん，漬物
夕食（岩見家）	焼肉，味噌汁，サラダ，ごはん

これまでに定義した長坂家と岩見家のグループの内容

*Mathematica*に何か頼む場合（今回はグループに共通するものを調べさせる場合），組込み関数と呼ばれる命令を使って指示する必要があります．*Mathematica*の母国語は英語なので，基本的にすべての支持は英語で出す必要もあります．下図は，基本的な指示の出し方を説明したものです．



*Mathematica*への指示の出し方

ここで「Intersection」は「グループに共通するものを調べよ」という指示になります．比較すべきグループ名はカンマで区切る必要がありますし，ブラケットが使われていることにも注意してください．これらのブラケットやカンマは，*Mathematica*が指示を理解するために必要なものです．実際に，長坂家と岩見家の夕食に共通する内容を調べてみましょう．

```
In[32]:= Intersection[長坂家`夕食, 岩見家`夕食]

Out[32]= {ごはん, 味噌汁}
```

All Rights Re:

グループ名の部分を変更するだけで，朝食，昼食，おやつについても調べることが出来ます．

```
In[33]:= Intersection[長坂家`朝食, 岩見家`朝食]

Out[33]= { }

In[34]:= Intersection[長坂家`昼食, 岩見家`昼食]

Out[34]= {サラダ}

In[35]:= Intersection[長坂家`おやつ, 岩見家`おやつ]

Out[35]= {コーヒー}
```

【中身のないグループ】*Mathematica*では「{}」のように，中括弧の間に何も含まれていないグループもあります．これはグループを集合と考えた場合の空集合にあたるものです．長坂家と岩見家の朝食の共通部分の結果が「{}」となっている意味は，双方に共通するものは存在しない，ということになります．この他，*Mathematica*の使い方や出力には，数学的なものの表現が多数あります．

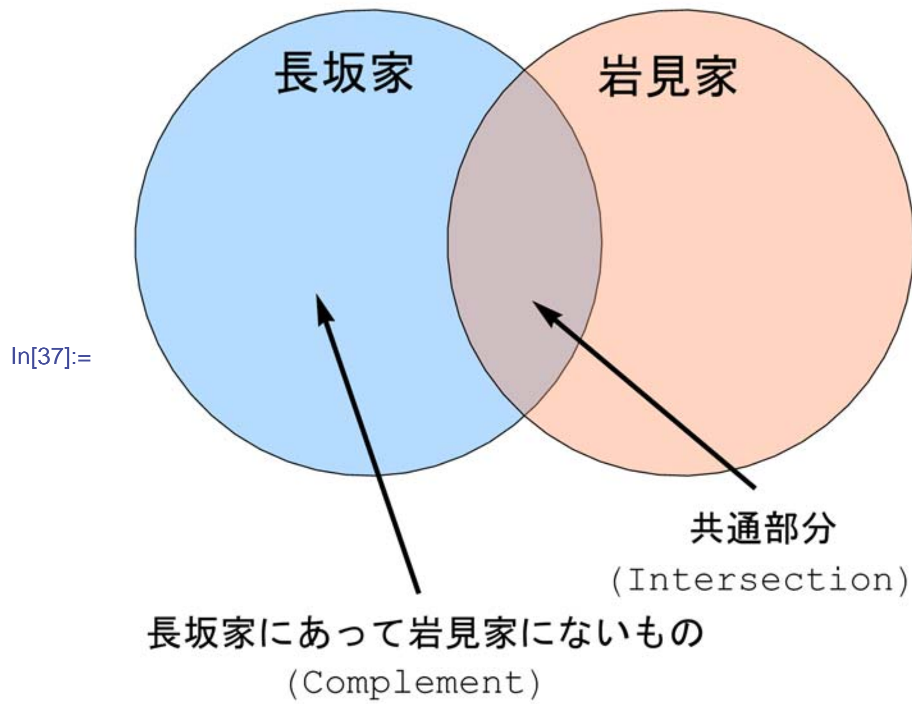
- **長坂家と岩見家に共通しないもの**

一方，長坂家の食事の内容には含まれているのに，岩見家の食事には含まれていないものを調べるには，異なる命令（異なる組込み関数）を使う必要があります．そのために使う指示は「`Complement`」になります．これは「ある集まりから別の集まりに含まれているものを取り除きなさい」という指示になります．従って，次のように*Mathematica*に命令することで，長坂家の夕食には含まれているのに，岩見家の夕食には含まれていないものを調べられます．

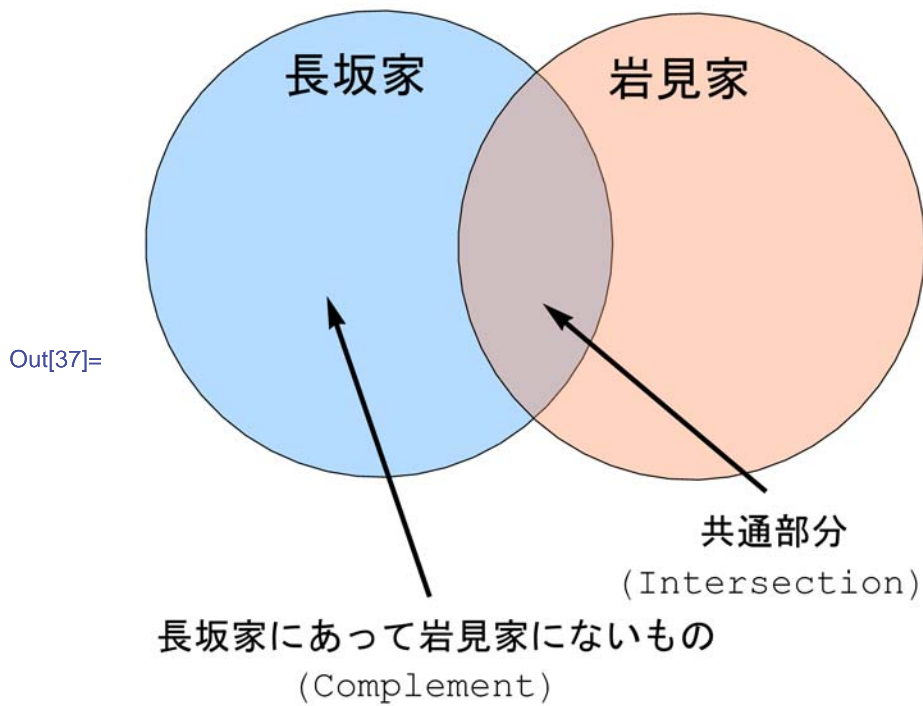
```
In[36]:= Complement[長坂家`夕食, 岩見家`夕食]

Out[36]= {漬物, 生姜焼, 肉じゃが}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[37]:=



Out[37]=



IntersectionとComplementの図解

他の食事についても同様に調べてみましょう.

In[38]:= **Complement[長坂家`朝食, 岩見家`朝食]**

Out[38]= {ごはん, 味噌汁, 焼き魚}

```
In[39]:= Complement[長坂家`昼食, 岩見家`昼食]
Out[39]= {コーヒー, サンドイッチ}

In[40]:= Complement[長坂家`おやつ, 岩見家`おやつ]
Out[40]= {ショートケーキ}
```

- **グループの吸収合併（新しい食事の組合せの発見）**

長坂家と岩見家が一緒に食事をしたとき，食事はどのようになるかを調べましょう．つまり，長坂家の食事に岩見家の食事の内容を加えたものを*Mathematica*に調べてもらいます．このような場合，「Union」という指示で「重複しないように双方の中身を併せなさい」という作業を行わせます．

```
In[41]:= Union[長坂家`夕食, 岩見家`夕食]
Out[41]= {漬物, 焼肉, ごはん, サラダ, 味噌汁, 生姜焼, 肉じゃが}
```

これを応用することで，長坂家の一日の全ての食事の構成も調べられます．

```
In[42]:= Union[長坂家`朝食, 長坂家`昼食, 長坂家`おやつ, 長坂家`夕食]
Out[42]= {漬物, ごはん, サラダ, 味噌汁, 焼き魚, 生姜焼,
  コーヒー, 肉じゃが, サンドイッチ, ショートケーキ}
```

岩見家についても調べてみましょう．

```
In[43]:= Union[岩見家`朝食, 岩見家`昼食, 岩見家`おやつ, 岩見家`夕食]
Out[43]= {梨, ハム, 焼肉, 素麺, ごはん, サラダ, 味噌汁, 食パン, コーヒー}
```

なお，*Mathematica*では共通部分「Intersection」と和集合「Union」という横文字の指示の代わりに，数学で用いられる記号を使っても同じことを調べさせることができます．

```
In[44]:= 長坂家`朝食 ⋂ 長坂家`夕食
Out[44]= {ごはん, 味噌汁}

In[45]:= 岩見家`朝食 ⋃ 岩見家`昼食
Out[45]= {ハム, 素麺, サラダ, 食パン, コーヒー}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

- **グループへの新しい要素の追加**

これまで長坂家と岩見家の食事の内容についてリストを作成して共通点などを調べてきました．ところが実は長坂家の朝食には「漬物」も出ていたことが判明しました．もう一度最初から操作し直さなければいけないでしょうか．大丈夫です．*Mathematica*では，既に定義したリストに新たな項目を追加する機能が備わっています．「指定したグループに新しい仲間を加えなさい」という意味の指示は「AppendTo」になります．

現在の長坂家の朝食の内容を確認し，夕食との共通点を調べてみます．

```
In[46]:= 長坂家`朝食

Out[46]= {ごはん, 味噌汁, 焼き魚}

In[47]:= Intersection[長坂家`朝食, 長坂家`夕食]

Out[47]= {ごはん, 味噌汁}
```

実際に「AppendTo」を使って，長坂家の朝食に漬物を加えてみましょう．夕食との共通点を調べると新たに漬物が共通していることがわかります．

```
In[48]:= AppendTo[長坂家`朝食, "漬物"]

Out[48]= {ごはん, 味噌汁, 焼き魚, 漬物}

In[49]:= Intersection[長坂家`朝食, 長坂家`夕食]

Out[49]= {漬物, ごはん, 味噌汁}
```

同じように，長坂家と岩見家のおやつにそれぞれ新しい食品を加えるには次のように指示を出します．

```
In[50]:= AppendTo[長坂家`おやつ, "チーズケーキ"]

Out[50]= {ショートケーキ, コーヒー, チーズケーキ}

In[51]:= AppendTo[岩見家`おやつ, "ヨーグルト"]

Out[51]= {梨, コーヒー, ヨーグルト}
```

All Rights Re:

- **グループに含まれているかを確認する**

膨大なデータを含むグループに特定の項目が含まれているかを目視で確認することは大変な作業です．例えば，長坂家の夕食に「漬物」が含まれていたかを確認するなどの作業です．そのような調べごとは*Mathematica*に任せてしまいましょう．「MemberQ」という組込み関数を使うことで，「あるグループに指定した項目が含まれているか」を調べさせることができます．実際に，長坂家の夕食に漬物が含まれているかを調べてみましょう．

```
In[52]:= MemberQ[長坂家`夕食, "漬物"]

Out[52]= True
```

「True」と表示されました．日本語に直せば「はい」です．今回の場合，食事の内容が少ないので目視でも，次のように長坂家の夕食を確認すれば，実際に漬物が含まれていることがわかります．

```
In[53]:= 長坂家`夕食

Out[53]= {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物}
```

それでは，長坂家と岩見家の朝食，昼食，おやつ，夕食を通して「うどん」が含まれているかを確認するにはどうすれば良いでしょうか．漬物を確認したように8回*Mathematica*に調べさせてみましょう．

```
In[54]:= MemberQ[長坂家`朝食, "うどん"]

Out[54]= False

In[55]:= MemberQ[長坂家`昼食, "うどん"]

Out[55]= False

In[56]:= MemberQ[長坂家`おやつ, "うどん"]

Out[56]= False

In[57]:= MemberQ[長坂家`夕食, "うどん"]

Out[57]= False

In[58]:= MemberQ[岩見家`朝食, "うどん"]

Out[58]= False
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[59]:= MemberQ[岩見家`昼食, "うどん"]

Out[59]= False

In[60]:= MemberQ[岩見家`おやつ, "うどん"]

Out[60]= False

In[61]:= MemberQ[岩見家`夕食, "うどん"]

Out[61]= False
```

結果は「False」，即ち日本語に直せば「いいえ」です．しかし，8回も命令するのは面倒です．このような場合はグループの吸収合併で使用した「Union」を併用します．即ち，両家併せて8回分の食事の内容を併せたものの中に「うどん」が含まれているかを確認させます．

```
In[62]:= MemberQ[Union[長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
           長坂家`夕食, 岩見家`朝食, 岩見家`昼食, 岩見家`おやつ,
           岩見家`夕食], "うどん"]

Out[62]= False
```

このように，*Mathematica*への指示は複数個組み合わせることもできます．もしくは，両家の食事内容に新しく名前を付けてから確認させるということもできます．

```
In[63]:= 両家の食事 = Union[長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
           長坂家`夕食, 岩見家`朝食, 岩見家`昼食, 岩見家`おやつ,
           岩見家`夕食]

Out[63]= {梨, ハム, 漬物, 焼肉, 素麺, ごはん, サラダ,
          味噌汁, 焼き魚, 生姜焼, 食パン, コーヒー, 肉じゃが,
          ヨーグルト, サンドイッチ, チーズケーキ, ショートケーキ}

In[64]:= MemberQ[両家の食事, "うどん"]

Out[64]= False
```

【括弧の対応付け】*Mathematica*では括弧や引用符の対応付けが正しくない場合，通常とは異なる色で表示するようになっています．例えば，中括弧の左側「{」に対応する右側「}」がないような場合，紫色で括弧が表示されます．これにより間違った指示を出さないように気を付けることができます．色が黒色でない場合は，もう一度見直すようにしましょう．

All Rights Re:

グループ（リスト）のグループを作ろう

■ **グループのグループ**

長坂家の朝食，昼食，おやつ，夕食と見てきましたが，これらを長坂家の食事としてグループに出来ると便利そうです．リストに含まれる項目をまとめるには，Unionという組込み関数を使いました．従って，次のように指示を出すことで長坂家の全ての食事の内容をまとめることは出来ます．

```
In[65]:= 長坂家の食事 = Union[長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
           長坂家`夕食]

Out[65]= {漬物, ごはん, サラダ, 味噌汁, 焼き魚, 生姜焼, コーヒー,
          肉じゃが, サンドイッチ, チーズケーキ, ショートケーキ}
```

しかしながら，このようにまとめてしまうと，どの項目が朝食で食べられたものか，どの項目が重複して含まれていたのか，という情報が失われてしまいます．そこで出てくるのが「グループのグループ」ないしは「リストのリスト」です．これまでのリストを使った方法に倣い，次のようにリストのリストを作ってみました．

```
In[66]:= 長坂家の食事 = {長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
           長坂家`夕食}

Out[66]= {{ごはん, 味噌汁, 焼き魚, 漬物}, {サンドイッチ, サラダ, コーヒー},
          {ショートケーキ, コーヒー, チーズケーキ},
          {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物}}
```

結果として得られたものは，リストの各要素（グループの各項目）が再度リスト（グループ）になっています．*Mathematica*では繰り返して括弧を使うことで，グループのグループを表現することが出来るようになっています．

グループのグループにも共通するものがないかのチェックを「Intersection」で行うことが出来ます．例えば，河合家の食事の内容が次のようなものだったとして，長坂家の食事との共通点を探してみましょう．

```
In[67]:= 河合家の食事 = {{"サンドイッチ", "サラダ", "コーヒー"},
           {"素麺", "漬物", "煎茶"},
           {"焼き魚", "味噌汁", "ごはん", "漬物", "惣菜"}}

Out[67]= {{サンドイッチ, サラダ, コーヒー},
          {素麺, 漬物, 煎茶}, {焼き魚, 味噌汁, ごはん, 漬物, 惣菜}}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[68]:= **Intersection[長坂家の食事, 河合家の食事]**

Out[68]= {{サンドイッチ, サラダ, コーヒー}}

長坂家の昼食にあたるものが河合家の食事にも含まれていることが確認できました.

【リストのネスト】*Mathematica*では「リストのリスト」のように同じ構造を繰り返して使うことが良くあります. また, 同じ構造を繰り返して入れ子状に使うことを「ネストする」と表現したりします. 括弧の多さに嫌気がさすこともあるかもしれませんが, データ構造を表す重要なものです. 注意深くネストされた情報を確認しましょう.

- **コピーとリンクの違い**

長坂家の食事というグループを作ることが出来ましたが, 実は長坂家の朝食には「納豆」もあったことに気がつきました. そこで, 長坂家の朝食の内容に次のように納豆を追加しました.

In[69]:= **AppendTo[長坂家`朝食, "納豆"]**

Out[69]= {ごはん, 味噌汁, 焼き魚, 漬物, 納豆}

しかしながら, 長坂家の食事を確認してみると「納豆」が追加されていません.

In[70]:= **長坂家の食事**

Out[70]= {{ごはん, 味噌汁, 焼き魚, 漬物}, {サンドイッチ, サラダ, コーヒー},
{ショートケーキ, コーヒー, チーズケーキ},
{肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物}}

長坂家の食事は次のように定義したので, 長坂家の朝食の内容が変化すれば, それに応じて長坂家の食事も変化するように思えます. が, その考え方は正しくありません. グループ名を定義するときに「=」を使うと, 定義内容が「コピー」されてグループの一員になります. 既にコピーされているので, コピー前の原本を編集してもコピー済みのものは変化しないのです.

**長坂家の食事 = {長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
長坂家`夕食}**

このようなことを防ぐために, ショートカットやリンクという機能がファイルマネージャには備わっていると思います. *Mathematica*のグループ定義で同じような機能を使うためには, 「=」の代わりに「:=」を使います（コロンとイコール）. 例えば, 長坂家の食事をコピーではなく, 朝食や昼食などの内容にリンクする場合は次のようにします.

All Rights Re:

```
In[71]:= 長坂家の食事 := {長坂家`朝食, 長坂家`昼食, 長坂家`おやつ,
           長坂家`夕食}
```

実際にリンクされていることを確認しましょう．次のように，長坂家の夕食に「ビール」を追加すると，長坂家の食事の内容が変化していることがわかると思います．

```
In[72]:= 長坂家の食事

Out[72]= {{ごはん, 味噌汁, 焼き魚, 漬物, 納豆},
          {サンドイッチ, サラダ, コーヒー},
          {ショートケーキ, コーヒー, チーズケーキ},
          {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物}}

In[73]:= AppendTo[長坂家`夕食, "ビール"]

Out[73]= {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物, ビール}

In[74]:= 長坂家の食事

Out[74]= {{ごはん, 味噌汁, 焼き魚, 漬物, 納豆},
          {サンドイッチ, サラダ, コーヒー},
          {ショートケーキ, コーヒー, チーズケーキ},
          {肉じゃが, 生姜焼, 味噌汁, ごはん, 漬物, ビール}}
```

【即時的な定義と遅延的な定義】*Mathematica*では，定義の内容を参照するタイミングによりファイルマネージャにおける「コピー」と「リンク」のような機能を提供しています．コピーのように，操作した時点で「=」の右側にあるものの内容を確認してしまうことを「即時的な定義」と呼びます．リンクのように，操作した時点では「:=」の右側にあるものの内容を確認せずに，実際に必要になったときに改めて確認することを「遅延的な定義」と呼びます．これらは非常に重要な概念になります．

- **コピーとリンクの違いを調べる**

長坂家の食事はリンク（遅延的な定義）として*Mathematica*に覚えてもらったことになります．一方で，河合家の食事はリストのリストとして定義しましたから，これはコピー（即時的な定義）として*Mathematica*に覚えてもらったことになります．

あるグループ名（変数）の定義が即時的に行われているか，それとも遅延的に行われているかを確認するには，以前にも使用した「?」を使います．次のようにそれぞれの内容を問い合わせることで，即時的（=）なのか遅延的（:=）なのかを確認することが出来ます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[75]:= **? 長坂家の食事**

Global`長坂家の食事

長坂家の食事 := {長坂家`朝食, 長坂家`昼食, 長坂家`おやつ, 長坂家`夕食}

In[76]:= **? 河合家の食事**

Global`河合家の食事

河合家の食事 = {{サンドイッチ, サラダ, コーヒー}, {素麺, 漬物, 煎茶}, {焼き魚, 味噌汁, ごはん, 漬物, 惣菜}}

実際，遅延的な定義を行った長坂家の食事においては，朝食や昼食などへのリンクがあるだけで，その内容まではコピーされていないことがわかると思います．河合家の食事の方では即時的な定義を行っているので，既にリストのリストになっています（もちろん，河合家の食事はそのように最初から定義したのではありますが）．

グループ（リスト）をわかり易く表示しよう

- **リスト表示の仕方を少し変える**

これまで*Mathematica*に覚えさせたものを確認するには，次のように単純にその変数名を呼び出すだけでした．しかし，この方法では見た目がパッとしないと思う方もいるでしょう．そこで今回は，リストの内容の確認方法（画面への表示方法）について取り上げます．

In[77]:= **長坂家`朝食**

Out[77]= {ごはん, 味噌汁, 焼き魚, 漬物, 納豆}

最初に取り上げるのは，「列として表示しなさい」という指示になる「Column」です．この指示で長坂家の朝食の内容を確認してみましょう．次のように列として食事が表示されます．

In[78]:= **Column[長坂家`朝食]**

Out[78]=
ごはん
味噌汁
焼き魚
漬物
納豆

All Rights Re:

一方，長坂家のすべての食事の内容をリストのリストとして表現していた変数を表示させると次のようになります．Columnという命令は単純なリストを列として縦に表示するだけなので，リストのリストを表示させると，各項目のリストがそのまま表示されることになります．

```
In[79]:= Column[長坂家の食事]

Out[79]= {ごはん，味噌汁，焼き魚，漬物，納豆}
         {サンドイッチ，サラダ，コーヒー}
         {ショートケーキ，コーヒー，チーズケーキ}
         {肉じゃが，生姜焼，味噌汁，ごはん，漬物，ビール}
```

なお，列ではなく「行として表示しなさい」という指示「Row」もあります！

```
In[80]:= Row[長坂家`朝食]

Out[80]= ごはん味噌汁焼き魚漬物納豆
```

- **表としてリストのリストを表示する**

変数の内容にもよりますが，食事の内容をわかりやすい方法で確認したいのであれば，列として表示（Column）や行として表示（Row）よりも，普通の表で表示させたいのではないでしょうか．

リストのリストで表現されているデータを表形式で表示する命令には種類があります．まず，ColumnとRowの延長線上にある「格子状に並べて表示しなさい」という指示の「Grid」です．実際に，長坂家の食事（遅延的に定義されたリストのリスト）を表示してみましょう．

```
In[81]:= Grid[長坂家の食事]

Out[81]=   ごはん        味噌汁      焼き魚      漬物  納豆
         サンドイッチ     サラダ      コーヒー
         ショートケーキ  コーヒー  チーズケーキ
           肉じゃが       生姜焼      味噌汁     ごはん  漬物  ビール
```

より汎用性が高く，関数名も判りやすいのが「TableForm」という命令で「結果が長方形の表になるようにフォーマットしなさい」という指示になります．この組込み関数は，リストであればネストの度合い（リスト形式がどれだけ入れ子になっているか）に関係なく利用することが出来ます．一方，先ほどのGridはリストのリストにしか使用することが出来ません．同じように，長坂家の食事を表示してみましょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[82]:= **TableForm[長坂家の食事]**

Out[82]//TableForm=

ごはん	味噌汁	焼き魚	漬物	納豆	
サンドイッチ	サラダ	コーヒー			
ショートケーキ	コーヒー	チーズケーキ			
肉じゃが	生姜焼	味噌汁	ごはん	漬物	ビー

表形式でリストがネストしているデータを表示すると，比較もしやすいです．例えば，次のように河合家の食事の内容も`TableForm`で表示することで，長坂家の食事の内容との比較が簡単になります．

In[83]:= **TableForm[河合家の食事]**

Out[83]//TableForm=

サンドイッチ	サラダ	コーヒー		
素麺	漬物	煎茶		
焼き魚	味噌汁	ごはん	漬物	惣菜

- **より表らしく表示するには**

表計算ソフトなどを使うと，とても綺麗な表を作ることが出来ると思います．同じような装飾は*Mathematica*でも行うことが可能です．まずは，簡単なところからはじめましょう．次のように命令の最後に「`Frame->All`」という補足指示を付け加えることで，表に枠を付けることが出来ます．なお，キーボードから入力された「`->`」は自動的に「→」に変換されます．

In[84]:= **Grid[長坂家の食事, Frame → All]**

Out[84]=

ごはん	味噌汁	焼き魚	漬物	納豆	
サンドイッチ	サラダ	コーヒー			
ショートケーキ	コーヒー	チーズケーキ			
肉じゃが	生姜焼	味噌汁	ごはん	漬物	ビール

`TableForm`を使った場合，次のように項目名を付け加えることが出来ます．項目名もリストで指示しているように，リストという仕組みは*Mathematica*の様々なところで使用されています．このような命令の最後に付け加える補足的な指示のことをオプションと呼びます．

All Rights Re:

```
In[85]:= TableForm[長坂家の食事,
  TableHeadings → {{"朝食", "昼食", "おやつ", "夕食"},
    {"食事の内容"}}]
```

Out[85]//TableForm=

	食事の内容				
朝食	ごはん	味噌汁	焼き魚	漬物	納
昼食	サンドイッチ	サラダ	コーヒー		
おやつ	ショートケーキ	コーヒー	チーズケーキ		
夕食	肉じゃが	生姜焼	味噌汁	ごはん	漬

これらのオプションは，次のような構造になっていることに注意してください．

*Mathematica*への補足的な指示の出し方

【オプション】*Mathematica*では，右矢印を使って補足的な指示を付け加えることで様々な効果を生み出していきます．オプションという機能があることで，覚えるべき組込み関数の数を少なくすると共に，非常にバラエティに富んだ機能を*Mathematica*は提供しています．その関数にどのようなオプションがあるかは，ドキュメントセンターで確認するか，単純に「Options[関数名]」を評価することで確認します（例えば，Options[Grid]など）．

- **エラーメッセージとその対処方**

*Mathematica*は頭の固い計算機なので，私たちの出した指示について，非常に細かい駄目出しをしてきます．人間同士であれば融通を利かせてもらえることもありますが，*Mathematica*は人間でないので愛想がありません．

絶対の不文律「全ての命令は，*Mathematica*の作法に従う必要があります」

しかしながら，*Mathematica*も愛想は悪いながらもエラーメッセージという形で，我々の間違いを正そうと試みてくれます．例えば，次のように閉じる括弧（]）を付けずに評価すると，右側のほうにオレンジ色のマークが表示されます．このマークは「何かおかしいぞ」という意味です．

```
Grid[長坂家`朝食
```

この時点で修正すべき間違いに気がつけなくても，オレンジ色のマークをクリックすることで「どう修正すべきか」という提案を行ってくれます．例えば今回の場合，閉じる括弧（]）が不足していることを指摘してくれます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
Grid[長坂家`朝食
```

Syntax::bktmcp : 式"Grid[長坂家`朝食"には閉じる"]"がありません.

Syntax::sntxi : 式は不完全です. 追加入力しなければなりません.

*Mathematica*の指摘に基づいて修正を行い，再度同じセルを評価することで正しい出力を得ることが出来ます．「*Mathematica*を使う」ことが「*Mathematica*と対話を行う」ことであることを実感してもらえるのではないでしょうか．

```
In[86]:= Grid[長坂家`朝食]

Out[86]= Grid[{ごはん, 味噌汁, 焼き魚, 漬物, 納豆}]
```

また，次の例で確認できるように，適切な関数名については色が黒になりますが，そうでない場合は青色のままなので，「これはもしかして何か間違っている？」と察することが出来れば評価の前に修正することも出来ます．

```
TableForm, Tableform, tableform
```

【エラーメッセージ】*Mathematica* 6では，スペルミス（大文字と小文字を間違えるなど）が文字の色分けで判断が付くようになっています．そのため，スペルミスに関するエラーメッセージを従来の*Mathematica*とは異なり，標準の状態では出力しないように設定されています．この機能を有効にしたい場合は「`On[General::spell1, General::spell]`」を評価します．これにより，*Mathematica*の受け取った指示にスペルミスの可能性がある場合，エラーメッセージが表示されます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

文字列：文章を扱い，自動作文をさせてみよう

文字列という基本となるデータ形式を学びましょう．

ワープロの編集機能を真似てみよう

■ 文章に含まれる単語の数を調べてみる

リストの話を思い出してください．文字や文字列（文字が複数連なったもの）を*Mathematica*に指示する場合，二重引用符（ダブルクオーテーション：「"」）で囲む必要がありました．同じように，食品の名前という単語だけでなく，文章を扱う場合にも二重引用符を使う必要があります．例えば，早口言葉を*Mathematica*に覚えさせる場合も次のように二重引用符を使う必要があります．

```
In[1]:= 早口言葉`桃 = "スモモも桃も桃のうち桃もスモモもものうち"
Out[1]= スモモも桃も桃のうち桃もスモモもものうち
```

この文章の文字数を*Mathematica*に調べさせるには「次の文字列の文字数を調べよ」という意味を持つ「StringLength」なる命令を使います．次のように，この早口言葉の文字数は21文字であることを*Mathematica*は教えてくれます．

```
In[2]:= StringLength[早口言葉`桃]
Out[2]= 21
```

この早口言葉は「も」や「桃」がたくさん出てくるので，それぞれ何回ずつ出てくるのか知りたくなりますね．そのような場合には「StringCount」という命令を使います．これは「次の文字列の中に指定した文字列が何回出てくるか調べよ」という意味の指示になります．実際に次のように*Mathematica*に問い合わせることで，「も」が6回，「桃」が3回使われていることがわかります．

```
In[3]:= StringCount[早口言葉`桃, "も"]
Out[3]= 6
```

```
In[4]:= StringCount[早口言葉`桃, "桃"]
Out[4]= 3
```

このStringCountは，一文字だけでなく単語や文章についても出現回数を調べる命令になっています．従って，次のように「もも」が何度出てくるかも調べることが出来ます．この早口言葉には「も」が6回も出てくるのに「もも」は1回しか出てこないことがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[5]:= **StringCount[早口言葉`桃, "もも"]**

Out[5]= 1

【キーボード入力の補完】*Mathematica*に指示を出す組込み関数は，英単語から構成されているので覚え易いものです．しかしながら，キーボードで「StringCount」と11文字も入力するのは面倒です．そのため*Mathematica*には「式の補完」という機能が備わっています．この機能は，WindowsとUnix(Linux)ではCTRLを押しながら「k」を押すことで，MacintoshではCMDを押しながら「k」を押すことで利用できます．例えば，StringCountを入力するには「StringCoCTRLk」でも良いですし，「StrCTRL」で表示される一覧表の中から選択することも出来ます．

- **文章に含まれる単語を置き換えてみる**

ワープロソフトでは「置換」という機能が備わっていることは多く，文章中の特定の言葉を別の言葉に置き換えることが出来たりします．同じような*Mathematica*の機能を使って，先ほどの早口言葉の漢字をひらがなに直してみましょう．置き換えに用いる組込み関数は「StringReplace」で「次の規則に従って，文字列に含まれる単語を置き換えなさい」という意味になります．「→」は「->」（マイナス記号と大なり記号）として入力します．

In[6]:= **StringReplace[早口言葉`桃, "桃" → "もも"]**

Out[6]= スモモももももものうちもももスモモもものうち

【InputAutoReplacements】*Mathematica*のノートブックに入力された式は，見易くなるように自動的に変換されます．これには例えば次のようなものがあります．

キーボードからの入力	*Mathematica* の自動変換先
->	→
:>	:→
<=	≤
>=	≥
!=	≠
==	==

非常に読みづらい文章になってしまいましたが，加えて，カタカナもひらがなに直してみましょう．カタカナは「ス」と「モ」が使われています．これらを個別にひらがなにするには「桃」の場合と同様に次のように指示を行います．

In[7]:= **StringReplace[早口言葉`桃, "ス" → "す"]**

Out[7]= すモモも桃も桃のうち桃もすモモももものうち

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[8]:= StringReplace[早口言葉`桃, "モ" → "も"]
```

```
Out[8]= スもも桃も桃のうち桃もスもももものうち
```

．．．意味がないですね．「桃，ス，モ」の全てを同時にひらがなにしたいですね．このような場合，置き換えの規則（"桃"→"もも", "ス"→"す", "モ"→"も"）をリストで一括指定します．このように，*Mathematica*への指示では単一のものを複数組み合わせるときにもリストが活躍します．

```
In[9]:= StringReplace[早口言葉`桃,
          {"桃" → "もも", "ス" → "す", "モ" → "も"}]
```

```
Out[9]= すもももももももものうちもももすもももものうち
```

この状態で先ほど使った組込み関数で文字の数を調べると，音としての「も」の回数がわかります．*Mathematica*が調べた結果を次の指示で使うためには，「*Mathematica*が直前に報告した結果」を意味する「%」を使うと非常に便利です．もちろん，「%」の代わりに直接「StringReplace[早口言葉`桃,{"桃"→"もも", "ス"→"す", "モ"→"も"}]」を使うことも出来ます．

```
In[10]:= StringCount[% , "も"]
```

```
Out[10]= 16
```

```
In[11]:= StringLength[早口言葉`桃]
```

```
Out[11]= 21
```

このようにして調べると，この早口言葉は21文字中16文字も同じ「も」が出てくることがわかります．面白いですね．

> 【前の結果の引用】*Mathematica*を使うことは「*Mathematica*との会話」です．人間同士の会話から代名詞がなくなったら非常に不便なのと同様に，*Mathematica*を使う上でも代名詞は非常に重要です．今回登場した「%」は「直前の結果」を表すものですが，これ以外にも「%%」で「二つ前の結果」，「%%%」で「三つ前の結果」を表します．即ち，パーセント記号を並べただけ前の結果を遡って引用することが出来ます．

- **文章に含まれる似通った単語の数を調べてみる**

ワープロの置換機能に似た方法で音として「も」がたくさん入っていることがわかりました．しかし，私たちがいま使っているのはハイテクの塊とも言える*Mathematica*なのに，少しローテク（地味で時代後れ）な方法のように思えませんか．実は，もっとスマートな方法があります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
早口言葉`桃 = "スモモも桃も桃のうち桃もスモモもものうち"
```

いま取り上げている早口言葉の定義

*Mathematica*の組込み関数「StringCases」は「次の文字列の中から，指定したパターンにマッチする単語（部分文字列）を取り出せ」という意味の指示になります．パターンとは例えば，「の」で始まって「桃」で終わる文字列，のような条件のことを指します．この場合，*Mathematica*への命令は次のようになります．

```
In[12]:= StringCases[早口言葉`桃, "の" ~~ ___ ~~ "桃"]

Out[12]= {のうち桃}
```

最初は「の」→ "の"~~___~~"桃" ← 最後は「桃」

チルダ2つ「~~」は「次に続く」の意味

下線3つ「___」は「何でもOK」の意味

文字列パターン：「の」で始まって「桃」で終わる文字列

他のパターンでも調べてみましょう．次の命令では，「桃」または「もも」または「モモ」にマッチする文字列，を調べさせています．この早口言葉には音として「もも」を6個も含んでいることがわかります．縦線（「|」）を「または」と読むことで，文字列パターンは日本語での意味と同じになることがわかると思います．

```
In[13]:= StringCases[早口言葉`桃, "桃" | "もも" | "モモ"]

Out[13]= {モモ, 桃, 桃, 桃, モモ, もも}
```

*Mathematica*の調査結果と早口言葉を比較することで，「ももも」の部分が1つ分の「もも」としか数えられていないことがわかると思います．*Mathematica*に重複している部分も数えるように補足指示（オプション）を出すことも出来ます．命令の最後に

```
In[14]:= StringCases[早口言葉`桃, "桃" | "もも" | "モモ",
          Overlaps → True]

Out[14]= {モモ, 桃, 桃, 桃, モモ, もも, もも}
```

音として「もも」になるようなカタカナとひらがなの組合せも重複して調べさせることも可能です．この場合，次のように文字列パターンは長くなってしまいます．

```
In[15]:= StringCases[早口言葉`桃, "桃" | "もも" | "モモ" | "もモ" | "モも",
          Overlaps → True]

Out[15]= {モモ, モも, 桃, 桃, 桃, モモ, モも, もも, もも}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

カタカナとひらがなの組合せの「もも」を効率良く*Mathematica*に指示するには，または（「|」）と次に続く「（~~）」を組み合わせて指示を出します．丸括弧（「(」と「)」）は「または」の対象を明確に示すために必要です．この丸括弧がないと，1文字目が「桃」か「も」か「モ」で，2文字目が「も」か「モ」の単語，という意味になってしまいます．*Mathematica*での丸括弧は，このように物事の優先順位を定めるために使われます．

```
In[16]:= StringCases[早口言葉`桃, "桃" | ("も" | "モ" ~~ "も" | "モ"),
          Overlaps → True]

Out[16]= {モモ, モも, 桃, 桃, 桃, モモ, モも, もも, もも}
```

より正確に「もも」という音が何回出てくるかを調べるには，次のように複雑な文字列パターンを使わなければいけませんが，このパターンの意味は自分で考えてみてください．「リストの要素数を調べよ」という意味の命令「Length」を使って，13個の

```
In[17]:= StringCases[早口言葉`桃,
          "桃" | ({"も", "モ"} ~~ {"も", "モ"}) |
           ({"も", "モ"} ~~ {"桃"}) | ({"桃"} ~~ {"も", "モ"}),
          Overlaps → All]

Out[17]= {モモ, モも, も桃, 桃, 桃も,
          も桃, 桃, 桃, 桃も, モモ, モも, もも, もも}

In[18]:= Length[%]

Out[18]= 13
```

なお，オプションの「Overlaps→All」は「桃」と「桃も」のようにマッチする文字列が他のマッチする文字列に含まれる場合も別のものとして扱うという追加指示になります．これがないと13個ではなく11個しか検出されません．

【文字列パターン】*Mathematica*では，様々な種類のパターンを利用することが出来ます．早口言葉の件で扱っている文字列パターンはそのひとつです．文字列パターンについての詳しい情報はドキュメントセンターで「StringExpression」を調べると書いてあります．なお，文字列パターンは「StringCount」とも組み合わせることが出来ます．「StringCases」の代わりに「StringCount」を使うと，同時に「Length」も使ったのと同じように，パターンにマッチする単語の個数を調べてくれます．ぜひ試してみてください．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

お猿さんはシェークスピアの話を書けるか？

■ **言葉（ひらなが，カタカナ，記号）のグループを作ろう**

*Mathematica*で言葉のリストを作るのには「CharacterRange」という命令を使います．この命令の意味は「次の2つの文字の間にある全ての文字を調べよ」になります．例えば，「な」と「の」の間にある全ての文字を調べさせるには次のように命令します．結果はリストで報告されますが，当然「なにぬねの」です．

```
In[19]:= CharacterRange["な", "の"]

Out[19]= {な, に, ぬ, ね, の}
```

この機能を使うと，ひらがなのリストを作ることが出来ます．「ひらがな」という名前でひらがなのリストを覚えさせておきましょう．なお，下記の「ぁ」は小さい「あ」であって，大きな「あ」ではないので注意してください．

```
In[20]:= ひらがな = CharacterRange["ぁ", "ん"]

Out[20]= {ぁ, あ, ぃ, い, ぅ, う, ぇ, え, ぉ, お, か, が, き, ぎ, く, ぐ, け,
 げ, こ, ご, さ, ざ, し, じ, す, ず, せ, ぜ, そ, ぞ, た, だ, ち, ぢ,
 っ, つ, づ, て, で, と, ど, な, に, ぬ, ね, の, は, ば, ぱ, ひ, び,
 ぴ, ふ, ぶ, ぷ, へ, べ, ぺ, ほ, ぼ, ぽ, ま, み, む, め, も, ゃ,
 や, ゅ, ゆ, ょ, よ, ら, り, る, れ, ろ, ゎ, わ, ゐ, ゑ, を, ん}
```

同様に，カタカナのリストも「カタカナ」という名前で覚えさせておきましょう．「ヴ」が最後に来ていますが，これは「ン」でも良いと思います．なお，ひらがなのときと同じで，下記の「ァ」は小さい「ア」であって，大きな「ア」ではないので注意してください．

```
In[21]:= カタカナ = CharacterRange["ァ", "ヴ"]

Out[21]= {ァ, ア, ィ, イ, ゥ, ウ, ェ, エ, ォ, オ, カ, ガ, キ, ギ,
 ク, グ, ケ, ゲ, コ, ゴ, サ, ザ, シ, ジ, ス, ズ, セ, ゼ,
 ソ, ゾ, タ, ダ, チ, ヂ, ッ, ツ, ヅ, テ, デ, ト, ド, ナ,
 ニ, ヌ, ネ, ノ, ハ, バ, パ, ヒ, ビ, ピ, フ, ブ, プ, ヘ,
 ベ, ペ, ホ, ボ, ポ, マ, ミ, ム, メ, モ, ャ, ヤ, ュ, ユ,
 ョ, ヨ, ラ, リ, ル, レ, ロ, ヮ, ワ, ヰ, ヱ, ヲ, ン, ヴ}
```

記号については，順序が複雑で一括して作り出すことが難しいので，リスト（グループ）として必要な記号を「記号」という名前で覚えさせておきます．

```
In[22]:= 記号 = {"，", "．", "「", "」", "（", "）"}

Out[22]= {，, ．, 「, 」, （, ）}
```

ちなみに半角のアルファベットも次のように生成することが出来ます．

```
In[23]:= CharacterRange["a", "z"]

Out[23]= {a, b, c, d, e, f, g, h, i, j, k, l,
          m, n, o, p, q, r, s, t, u, v, w, x, y, z}

In[24]:= CharacterRange["A", "Z"]

Out[24]= {A, B, C, D, E, F, G, H, I, J, K, L,
          M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}
```

■ **言葉のグループからランダムな単語を作ってみよう**

この章では，お猿さんがキーボードを使ってシェークスピアの文章を入力する可能性は皆無ではない，という有名な話を実際に確かめてみようとしています．言い替えれば，ランダムに選ばれた語から構成される文章が意味を持つことがあるか，ということですね．もっとも，我々人間に確認可能な短い時間で，お猿さんがシェークスピアを入力することは非常に難しいと思いますが．．．

この実験にちょうど良い命令として「RandomChoice」があります．意味は「次のグループの中から無作為に要素を取り出し，取り出した要素を元に戻す．これを指示された回数実験せよ」です．ひらがなのリストからランダム（無作為）に3個のひらがなを取り出すには，次のように命令することになります．3個のひらがなはランダムに選ばれるため，*Mathematica*に命令を出すたび（評価を行うごと）に結果は変化します．

```
In[25]:= RandomChoice[ひらがな, 3]

Out[25]= {じ, ぉ, じ}
```

このままでは「語のリスト」であって「単語」になっていません．そこで，*Mathematica*に対して「次の語のリストを単語にまとめなさい」という意味の「StringJoin」なる命令も併せて実行させることにします．次の例のように，ひらがな5つから構成される単語が評価毎に作られていきます．

```
In[26]:= StringJoin[RandomChoice[ひらがな, 5]]

Out[26]= いほぞぅだ
```

同様に，カタカナの単語，記号のみからなる文字列なども作り出すことが出来ます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[27]:= `StringJoin[RandomChoice[カタカナ, 5]]`

Out[27]= メシオナケ

In[28]:= `StringJoin[RandomChoice[記号, 5]]`

Out[28]= ））（（,

ちなみに，似たような命令に「RandomSample」があります．この意味は「次のグループの中から無作為に要素を取り出す（元には戻さない）．これを指示された回数実験せよ」です．従って，結果は非常に良く似ていますが，いくら繰り返しても「きつつき」，「とまと」，「とうきょう」などは出てきません．

In[29]:= `StringJoin[RandomSample[ひらがな, 3]]`

Out[29]= ぴとど

【ランダムな選択】*Mathematica*には最初のバージョンから疑似乱数を作り出す命令「Random」が組み込まれていました．最新の*Mathematica* 6では，より使い易く目的に応じた6つの命令「RandomChoice, RandomComplex, RandomInteger, RandomPrime, RandomReal, RandomSample」に進化しています．今回使用したものは，以前のバージョンになかった組込み関数になります．

- **言葉のグループからランダムな文章を作ってみよう**

漢字はありませんが，ひらがな，カタカナ，記号だけでも文章を作るには十分です．例えば，冒頭で出てきた早口言葉で漢字を使わない場合，次のようになりますが十分理解できます（分かり易いかは別問題とすれば）．

In[30]:= `StringReplace[早口言葉`桃, "桃" → "もも"]`

Out[30]= スモモもももももものうちもももスモモもものうち

以前に使った組込み関数Unionで，文章の構成要素となるリスト「言葉のもと」を作っておけば，単語を作るようにRandomChoiceとStringJoinにより任意の長さの文章を作り出すことが出来ます．

In[31]:= `言葉のもと = Union[ひらがな, カタカナ, 記号];`

In[32]:= `StringJoin[RandomChoice[言葉のもと, 30]]`

Out[32]= ゅボゆプぽわゃづペクゲュご．ぅナむュぷヘザデヂゎのゆウほチク

しかし，これでは文章には程遠いですね．そこで，文章はひらがなの塊とカタカナの塊と記号から構成されていると仮定しましょう．塊の長さ（1文字から5文字程度）もランダムに決める必要があります．文字数は整数（$\ldots, -2, -1, 0, 1, 2, \ldots$）なので，単

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

語作りに使った組込み関数に似た命令の「RandomInteger」により「次の範囲から整数を無作為に取り出せ」という指示を*Mathematica*に出すことが出来ます．次の例は，−5から5の整数（−5, −4, −3, −2, −1, 0, 1, 2, 3, 4, 5）から無作為にひとつ選ばれます．

```
In[33]:= RandomInteger[{-5, 5}]

Out[33]= 1
```

この組込み関数を使って「塊の長さ」を覚えさせておきましょう．まず，長さとしては1文字から5文字とし，RandomIntegerへの指示は{1,5}となります．塊の長さは評価するたびに変化する可能性（無作為に整数が選ばれるため）があるので，ショートカットやリンクに準ずる遅延的な定義（コロンとイコール）を使って定義します．

```
In[34]:= 塊の長さ := RandomInteger[{1, 5}]

In[35]:= 塊の長さ

Out[35]= 3

In[36]:= 塊の長さ

Out[36]= 2
```

定義した塊の長さを使って，ひらがなの塊も定義しましょう．基本的にはひらがなの単語を作ったときと同じですが，無作為に取り出す回数を「塊の長さ」とすることと，遅延的な定義（コロンとイコール）を使うこと点で異なります．同じようにカタカナの塊も定義しておきますが，記号については常に1文字になるように定義しておきます．実際に，ひらながの塊やカタカナの塊を評価すると，無作為な言葉の塊が出てくることを確認できます．

```
In[37]:= ひらがなの塊 := StringJoin[RandomChoice[ひらがな, 塊の長さ]]

In[38]:= カタカナの塊 := StringJoin[RandomChoice[カタカナ, 塊の長さ]]

In[39]:= 記号の欠片 := StringJoin[RandomChoice[記号, 1]]

In[40]:= ひらがなの塊

Out[40]= ふき

In[41]:= カタカナの塊

Out[41]= ル
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

さて，これらの塊を使って文章を作るのですが，ひらがな，カタカナ，記号をどのような割合いでどのような順番で組み合わせるかという問題が残っています．特に，普通の文章に含まれる記号の割合いは著しく低いですし，カタカナもひらがなよりは低いでしょう．これを反映させなければいけません．何度も使っている*Mathematica*の組込み関数「RandomChoice」では，選ばれる確率を指定することも出来ます．例えば，次の例では「言葉のもとからひとつ」を遅延的に定義していますが，ひらがなの塊が70%程度，カタカナの塊が20%程度，記号の欠片が10%程度の割合いで選択されるように指示しています．

```
In[42]:= 言葉のもとからひとつ :=
          RandomChoice[
            {0.7, 0.2, 0.1} → {ひらがなの塊, カタカナの塊, 記号の欠片}]

In[43]:= 言葉のもとからひとつ

Out[43]= ゑ
```

文章にするには，この「言葉のもとからひとつ」をいくつも組み合わせることになります．そのためには「次の指示を指定された回数だけ繰り返せ」という意味の「Table」を使うことになります．この組込み関数は非常に便利なので，今後何度も使うことになります．例えば，言葉のもとからひとつを10回繰り返したい場合，次のように「{10}」という指示を出します．この数字を変えれば繰り返す回数を変化させることが可能です．

```
In[44]:= Table[言葉のもとからひとつ, {10}]

Out[44]= {」, てたりたみ, よるよ, にゅが, は, ）, ぢぃき, , , で, オ}
```

文章にするには，全体をStringJoinで囲めば良いので，最終的に次のような命令でランダムな文章を作ることが出来るようになります．何度も評価を繰り返すと，普段の会話のような文章が表れるかも，しれません．．．

```
In[45]:= StringJoin[Table[言葉のもとからひとつ, {10}]]

Out[45]= ルネポヅデフアャラコオんいとわぃヒスチぶおオクもつじぽなろひび
```

【遅延的な定義の本質】*Mathematica*での「即時的な定義」と「遅延的な定義」の違いは，ファイルマネージャにおける「コピー」と「ショートカット」の違いに近い，という話をしました．より正確に表現すると，イコール（=）だけを使った即時的な定義では，右辺の評価結果を覚えますが，コロンとイコール（:=）を使った遅延的な定義では，右辺の評価はせずに右辺をそのまま覚えます．従って，無作為に何を選択する組込み関数と即時的な定義を組み合わせても，毎回同じ結果しか表示されませんが，遅延的な定義を組み合わせれば，毎回違うランダムな結果が表示されることになります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

小学校で習う漢字も追加しよう

やはり漢字がないと寂しいので，小学校の最初に習う漢字を追加しましょう．文部科学省のウェブサイトでは，各種の指導要領が公開されています．その中から小学校学習指導要領の国語に関するところに掲載されている「学年別漢字配当表」から，第一学年のものを写したのが次の文字列です．

```
In[46]:= 小学校学習指導要領一学年漢字 =
           "一右雨円王音下火花貝学気九休玉金空月犬見五口校左三山子四糸字耳
             七車手十出女小上森人水正生青夕石赤千川先早草足村大男竹中虫
             町天田土二日入年白八百文木本名目立力林六";
```

ひらがなやカタカナなどは一語ずつのリストになっていましたので，この漢字についても一語ずつのリストに直す必要があります．手作業で行うことも可能ですが，*Mathematica*の「Characters」という命令を使うことで，次のように簡単に1文字ずつに分割できます．この命令は「次の文字列を一語ずつの文字のリストに分解せよ」という意味なのは明らかでしょう．

```
In[47]:= 小学校一学年 = Characters[小学校学習指導要領一学年漢字]

Out[47]= {一, 右, 雨, 円, 王, 音, 下, 火, 花, 貝, 学, 気, 九, 休, 玉, 金,
          空, 月, 犬, 見, 五, 口, 校, 左, 三, 山, 子, 四, 糸, 字, 耳, 七,
          車, 手, 十, 出, 女, 小, 上, 森, 人, 水, 正, 生, 青, 夕, 石, 赤,
          千, 川, 先, 早, 草, 足, 村, 大, 男, 竹, 中, 虫, 町, 天, 田, 土,
          二, 日, 入, 年, 白, 八, 百, 文, 木, 本, 名, 目, 立, 力, 林, 六}
```

あとは先ほどと同様に「小学校一学年の塊」を遅延的な方法で定義しておきます．

```
In[48]:= 小学校一学年の塊 :=
          StringJoin[RandomChoice[小学校一学年, 塊の長さ]]
```

「言葉のもとからひとつ」の定義にも漢字を入れて定義し直しましょう．割合いは，ひらがなが40%，漢字が30%，カタカナが20%，記号が10%にしてみました．結果を見ると，多少は文章らしくなってきたのではないでしょうか．

```
In[49]:= 言葉のもとからひとつ :=
          RandomChoice[{0.4, 0.3, 0.2, 0.1} →
            {ひらがなの塊, 小学校一学年の塊, カタカナの塊, 記号の欠片}]

In[50]:= StringJoin[Table[言葉のもとからひとつ, {10}]]

Out[50]= 女空川正ほぇ名三小足ネ上タタパぎゆよざわぃけいじっれ青水学川
```

文章として意味のあるものになることは非常に稀ですが，漢字の塊だけを次のように20組程度作ると，中には意味のあるものが作られたり，意味がありそうな不思議な熟

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

語になっていたりします．もしかすると，本当に猿もシェークスピアが書けてしまうかもしれませんね．

```
In[51]:= Table[小学校一学年の塊, {20}]

Out[51]= {学男, ロ入学, 車生日, 木田早, 八, 先貝火,
  気白, 見貝大, 草, 左校先川土, 村, 木白男九, 雨八空,
  目, 十女目, 見正, 名水, 立見九王, 赤大見手, 校}
```

ランダム作文で洒落た短文を書いてもらおう

- **いつ，誰が，どうした？**

キーボードを無作為に押していくことでシェークスピアを書くのは無謀だということがわかりました．ランダムな熟語を作るという試みとしては面白かったと思いますが，現実的な作文にはなっていませんでした．そこで，よりシンプルに短い文を作ることにしましょう．「いつ，誰が，どうした」というシンプルな作文です．まずは，それぞれの候補をリストで定義しておきます．

```
In[52]:= いつ = {"昨日の朝，", "昨日の昼間，", "昨日の夕方，",
    "昨日の夜，", "今日の朝，", "今しがた，"};

In[53]:= 誰が = {"私は", "あなたは", "先輩は", "後輩は", "知合いは"};

In[54]:= どうした = {"寝坊した．", "遅刻した．", "宝くじに当たった．",
    "宝くじに外れた．", "ベッドから落ちた．", "ごはんを食べた．"};
```

それぞれの候補の中から無作為に選択する命令は既に使ったRandomChoiceで行えます．前回使用したときとは異なり，取り出す要素の個数は1個なので，わざわざ「RandomChoice[いつ,1]」と*Mathematica*に指示を出す必要はありません．

```
In[55]:= RandomChoice[いつ]

Out[55]= 昨日の夜，
```

文章を構成するには，残りも併せて3つの要素をそれぞれ無作為に選択すれば良いことがわかります．これを遅延的な定義で次のように覚えさせておきましょう．扱う部品が3つあるのでリストでまとめてあることに注意してください．遅延的な定義なので，評価するたびにランダムな文章が作られるようになります．

```
In[56]:= いつ誰がどうしたリスト :=
   {RandomChoice[いつ], RandomChoice[誰が],
    RandomChoice[どうした]}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[57]:= いつ誰がどうしたリスト

Out[57]= {今日の朝，，あなたは，ごはんを食べた．}

In[58]:= いつ誰がどうしたリスト

Out[58]= {昨日の夜，，知合いは，宝くじに外れた．}

このままでは1つの文章（文字列）になっていないので，前回も使用したStringJoinを使って結合することにしましょう．これも遅延的な定義を使って覚えさせておくことで，評価するたびにランダムに文章が作られるようになります．

```
In[59]:= いつ誰がどうした := StringJoin[いつ誰がどうしたリスト]
```

In[60]:= いつ誰がどうした

Out[60]= 昨日の昼間，あなたはごはんを食べた．

In[61]:= いつ誰がどうした

Out[61]= 今しがた，先輩は宝くじに外れた．

*Mathematica*には様々な省略表記があり，StringJoinという指示は「<>」（小なり記号，大なり記号）を使っても同じ意味になります．従って，先ほど定義した「いつ誰がどうした」という名前のショートカット（後述しますが，本来は変数ないしは関数と呼ぶ）は次のように定義することも可能です．

```
In[62]:= いつ誰がどうした := RandomChoice[いつ] <> RandomChoice[誰が] <>
           RandomChoice[どうした]
```

In[63]:= いつ誰がどうした

Out[63]= 今しがた，私はベッドから落ちた．

【マルチパラダイム】このように，*Mathematica*では同じ意味の指示を出すのに複数の具体的な命令が存在します．書き方によって，*Mathematica*が指示を実行するために必要な時間が大きく変化するため，科学技術計算などの速度が求められる場合には注意深く記述する必要があります．一方，これまでに取り組んできた内容のように高速に計算される必要がない場合は，ユーザが理解しやすい方法で柔軟に書くことが出来るという大きなメリットになります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

- **いつ，誰が，何を，どうして，どうなった？**

いつ，誰が，どうした，では短すぎて面白みが少ないかもしれません．もっと長い文章が生成されるように，いつ，誰が，何を，どうして，どうなった，という構成要素に拡張しましょう．まずは，先ほどと同じように各グループ毎に名前をつけて定義しておきます．各グループの要素が変化することはありませんので，即時的なもの（コピー）で大丈夫です．

```
In[64]:= いつ = {"昨日の朝，", "昨日の昼間，", "昨日の夕方，",
           "昨日の夜，", "今日の朝，", "今しがた，"};

In[65]:= 誰が = {"私は", "あなたは", "先輩は", "後輩は", "知合いは"};

In[66]:= 何を = {"パソコンを", "財布を", "ケーキを", "コーヒーを",
           "歯磨き粉を", "かばんを"};

In[67]:= どうして = {"壊して", "失くして", "食べて", "こぼして",
           "搾って", "背負って"};

In[68]:= どうなった = {"非常に困った．", "交番に届け出た．",
           "幸せだった．", "火傷した．", "歯磨きをした．",
           "買い物に行った．"};
```

簡単のため，冗長な方法で「いつ誰が何をどうしてどうなった」を遅延的に定義しましょう．前回と同じく`RandomChoice`と`StringJoin`を使っています．繰り返しになりますが，毎回，異なる言葉が無作為に選択されないといけませんから，遅延的な定義にする必要があることに注意してください．

```
In[69]:= いつ誰が何をどうしてどうなったリスト :=
          {RandomChoice[いつ], RandomChoice[誰が],
           RandomChoice[何を], RandomChoice[どうして],
           RandomChoice[どうなった]}

In[70]:= いつ誰が何をどうしてどうなった :=
          StringJoin[いつ誰が何をどうしてどうなったリスト]
```

実際に「いつ誰が何をどうしてどうなった」を評価すると，次のように面白い文章が毎回ランダムに生成されることがわかります．

```
In[71]:= いつ誰が何をどうしてどうなった

Out[71]= 昨日の昼間，あなたはパソコンを背負って交番に届け出た．
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[72]:= **いつ誰が何をどうしてどうなった**

Out[72]= 昨日の夜，あなたはかばんを壊して火傷した．

言葉を選んでくるグループが5個なのでRandomChoiceを5回記述して，それをStringJoinでまとめるという定義を行っていますが，これが10個になれば10回記述する必要が出てきます．*Mathematica*は計算機なので面倒とは思わないかもしれませんが，私たちは人間ですから同じことを何度もするのは面倒です．そのため，このような同じ操作を繰り返すことを簡単に指示できる仕組みが準備されています．まずは「いつ誰が何をどうしてどうなったリスト」の定義を確認して，共通部分がどこにあるのか，何を冗長に何度も繰り返しているかを確認しましょう．

?いつ誰が何をどうしてどうなったリスト

Global`いつ誰が何をどうしてどうなったリスト

```
いつ誰が何をどうしてどうなったリスト :=
 {RandomChoice[いつ], RandomChoice[誰が], RandomChoice[何を],
  RandomChoice[どうして], RandomChoice[どうなった]}
```

赤字で強調表示している部分が共通している部分です．いつ，誰が，何を，どうして，どうなった，のそれぞれに対してRandomChoiceの指示を出しています．これと同じ指示を組込み関数「Map」を使うと簡単に書くことが出来ます．この命令の意味は「次のリストのそれぞれに対して同じ命令を実行せよ」です．この関数を使うと「いつ誰が何をどうしてどうなったリスト」は次のように書き直すことが出来ます．

In[73]:= **Map[RandomChoice, {いつ, 誰が, 何を, どうして, どうなった}]**

Out[73]= {昨日の朝，，先輩は，ケーキを，搾って，交番に届け出た．}

Map[RandomChoice, {いつ,誰が,何を}]

どんな操作を実行させたいか　　指示の対象を記したリスト

リストの一括処理：いくつものリストに同じ操作を実行

この命令を使って「いつ誰が何をどうしてどうなった」を再度定義しなおすと次のように短い表現になります．わかりやすくなったはずですが，新しい命令であるMapを理解できていないと難しく感じるかもしれません．この命令は非常に重要で，今後も何度も出てきます．何度も読み返して便利さを確認してください．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[74]:= いつ誰が何をどうしてどうなった :=
          StringJoin[Map[RandomChoice,
             {いつ, 誰が, 何を, どうして, どうなった}]]

In[75]:= いつ誰が何をどうしてどうなった

Out[75]= 昨日の夕方，あなたはコーヒーを搾って幸せだった．
```

【Mapによるリスト処理】*Mathematica*はLispの流れを汲む言語と考えるのが自然で，リストで表現されたデータに対する一括処理に秀でています．ここでは単純に命令を短くする目的だけにMapを使用していますが，後の章では計算速度を速くするためにも用います．前の方の章で何度も断っていますが，*Mathematica*を使う上でリストは非常に重要な概念ですので，この一括操作も非常に重要な概念になります．始めにリストありき，という考え方を身につけることは*Mathematica*を覚える近道です．

- **「いつ誰がどうした？」で時制もランダムにしたい**

前項まで取り上げた作文では，時制や助詞も含めてある言葉から無作為に選択していました．これでは，時制や助詞の分だけ全てを想定した言葉を事前にリストで定義しておかなければいけません．もっと手軽に時制を扱える方法がないかを考えてみたいと思います．まずは，時制に関係のない部分だけを*Mathematica*に覚えておいてもらいましょう．

```
In[76]:= 誰が = {"私", "彼", "彼女", "友達"};

In[77]:= 助詞 = {"が", "は", "も"};

In[78]:= どうした = {"勉強", "食事", "カラオケ", "寝坊"};
```

次に，時制の部分が除かれた無作為な文章を生成する「誰がどうした」を定義しましょう．毎回異なる文章を生成させるには，遅延的な定義になることに注意してください．

```
In[79]:= 誰がどうした :=
          StringJoin[Map[RandomChoice, {誰が, 助詞, どうした}]]
```

実際に「誰がどうした」を*Mathematica*に問い合わせることで，時制と述語が不足している文章がランダムに返ってきます．あとは，この文章にランダムに時制と述語を付けられれば何も問題ありませんが，時制と述語は「昨日，した」と「明日，する」とペアになるので，各々を勝手に述語を決める訳にもいきません．どうすれば良いでしょうか．

```
誰がどうした

彼は食事
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

誰がどうした

私が寝坊

まずは，その出来事が昨日のことであるかを判定する命令「昨日のこと？」を作りましょう（はてなマークは全角にしてください）．*Mathematica*では「True」という言葉で「その通り！」を意味する「真」を，「False」で「そうじゃない！」を意味する「偽」を表します．従って，真偽を無作為に選択する「昨日のこと？」は次のような遅延的な定義になります．何度も出てきているRandomChoiceを使っていることに注意してください．

```
In[80]:= 昨日のこと？ := RandomChoice[{True, False}]
```

実際に*Mathematica*へ問い合わせてみると，評価するたびに真と偽が無作為に報告されるのがわかると思います．

```
昨日のこと？
False
```

```
昨日のこと？
True
```

不足している時制と述語のペアを文章に追加するには，「昨日のこと？」が真であるか偽であるかという状況に応じて*Mathematica*がすべきことを変化させる必要があります．これには，「次の条件が満たされたら，指定の命令を実行せよ」という意味の「If」を使います．昨日のことであれば「昨日，」と「した．」を追加し，そうでなければ「明日，」と「する．」を追加しています．

```
In[81]:= If[昨日のこと？, StringJoin["昨日， ", 誰がどうした, "した． "],
          StringJoin["明日， ", 誰がどうした, "する． "]]

Out[81]= 昨日，私も食事した．
```

If[条件，成立の場合，不成立の場合]

例：昨日のことかを確認　　例：昨日の時制　　例：明日の時制

条件分岐：内容によって命令を変える

最終的に，時制も無作為に選択される「いつ誰がどうした」は次のように遅延的な定義で表現されることになります．実際に評価してみると，いくつか面白い文章が作られることがわかると思います．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[82]:= いつ誰がどうした :=
  If[昨日のこと？, StringJoin["昨日，", 誰がどうした, "した．"],
   StringJoin["明日，", 誰がどうした, "する．"]]
```

いつ誰がどうした

明日，彼が寝坊する．

いつ誰がどうした

昨日，友達が寝坊した．

> 【TrueとFalse】*Mathematica*では条件件が満たされていることを「True」で表し，条件が満たされていないことを「False」で表します．いわゆる真偽値と呼ばれるもので，今回のような条件分岐の他，以前にも出てきたような判定問題（リストに指定した要素が含まれているか）などでの*Mathematica*からの報告にも使われます．今後も頻繁に出てきますので，良く覚えておいてください．

■ 「いつ誰が何をどうしてどうなった？」で時制もランダムにしたい

前項よりもバリエーションを増やして文章も長くすることを考えましょう．まずは，前項と同じように時制を除いた「誰が何をどうしてどうなった」をこれまでと同じように定義しましょう．

```
In[83]:= 誰が = {"私", "彼", "彼女", "友達"};

In[84]:= 助詞 = {"が", "は", "も"};

In[85]:= 何を = {"数学を", "運動会を", "十八番を", "携帯電話を", "卵を"};

In[86]:= どうして = {"勉強して", "食べて", "歌って", "寝過ごして",
    "壊して", "落として"};

In[87]:= 何が = {"成績を", "調子を", "懐を", "雰囲気を", "気分を"};

In[88]:= どうなった = {"良く", "悪く", "寂しく", "悲しく"};

In[89]:= 誰が何をどうしてどうなった :=
  StringJoin[Map[RandomChoice,
    {誰が, 助詞, 何を, どうして, 何が, どうなった}]]
```

この時点で「誰が何をどうしてどうなった」を*Mathematica*に評価させると，これまでと同じように，時制の部分を除いた文章がランダムに返ってきます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[90]:= 誰が何をどうしてどうなった

Out[90]= 彼女も携帯電話を寝過ごして調子を悪く
```

時制と述語を加えて，私達としては「明日，（本文）する．」とか「昨日，（本文）した．」という文章にしたいですね．そこで，このような時制と述語のペアをリストで定義しておきましょう．リストには「時制」という名前を付けて*Mathematica*に覚えておいてもらいます．

```
In[91]:= 時制 = {"昔，（本文）したものだ．", "昨日，（本文）した．",
            "今日，（本文）したところだ．", "明日，（本文）する．",
            "将来，（本文）したい．"};
```

いつの文章であるかをランダムにしたいので，時制をランダムに選択するように「時制の選択」も定義しておきます．そうすると，次のように時制と述語のペアが無作為に返ってくることがわかると思います．遅延的な定義を使うのにも慣れてきたでしょうか．

```
In[92]:= 時制の選択 := RandomChoice[時制]

In[93]:= 時制の選択

Out[93]= 今日，（本文）したところだ．

In[94]:= 時制の選択

Out[94]= 明日，（本文）する．
```

「（本文）」の部分が例えば「私は数学を勉強して成績を良く」になれば，作りたかった文章に近づきます．そこで，時制がランダムに選択されるようにしたままで，（本文）を置き換えるよう*Mathematica*に指示を出してみます．次のように，以前に利用した命令`StringReplace`を使うことで，簡単に置き換えは行えます．

```
In[95]:= StringReplace[時制の選択,
           "（本文）" → "私は数学を勉強して成績を良く"]

Out[95]= 将来，私は数学を勉強して成績を良くしたい．
```

このままでは，本文のところがランダムに選択されないので，置き換え先をランダムに本文を生成する「誰が何をどうしてどうなった」に置き換えてみます．すると，次のように時制と本文が無作為に選択された文章が作られるようになります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[96]:= StringReplace[時制の選択,
          "（本文）" → 誰が何をどうしてどうなった]

Out[96]= 明日，友達は携帯電話を落として気分を悪くする.
```

あとは遅延的な定義で「いつ誰が何をどうしてどうなった」を次のように定義することで，時制も本文も無作為に選択された文章をつくり出すことが出来ます.

```
In[97]:= いつ誰が何をどうしてどうなった :=
          StringReplace[時制の選択,
            "（本文）" → 誰が何をどうしてどうなった]

In[98]:= いつ誰が何をどうしてどうなった

Out[98]= 今日，友達は運動会を寝過ごして成績を悪くしたところだ.
```

文豪の作品を小学生になって読んでみよう

- **作品に使われている文字を調べてみよう**

今回の目的は文豪の作品を調べることです．例題として夏目漱石の「吾輩は猫である」から最初の段落を取り上げたいと思います．自分で試してみる際には，青空文庫（http://www.aozora.gr.jp）などからお気に入りの作品を探してみることをお薦めします．まずは，文章を分かり易い名前で*Mathematica*に覚えさせましょう．このような文章の定義の際などは，最後にセミコロン（;）を付けておくと復唱（結果出力）がないのでスッキリするので，忘れずに付けておきましょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[99]:= 吾輩は猫である =
  "吾輩は猫である。名前はまだ無い。どこで生れたかとんと見当がつか
    ぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だ
    けは記憶している。吾輩はここで始めて人間というものを見た。
    しかもあとで聞くとそれは書生という人間中で一番獰悪な種族で
    あったそうだ。この書生というのは時々我々を捕えて煮て食うと
    いう話である。しかしその当時は何という考もなかったから別段
    恐しいとも思わなかった。ただ彼の掌に載せられてスーと持ち上
    げられた時何だかフワフワした感じがあったばかりである。掌の
    上で少し落ちついて書生の顔を見たのがいわゆる人間というもの
    の見始であろう。この時妙なものだと思った感じが今でも残って
    いる。第一毛をもって装飾されべきはずの顔がつるつるしてまる
    で薬缶だ。その後猫にもだいぶ逢ったがこんな片輪には一度も出
    会わした事がない。のみならず顔の真中があまりに突起している
    。そうしてその穴の中から時々ぷうぷうと煙を吹く。どうも咽せ
    ぽくて実に弱った。これが人間の飲む煙草というものである事は
    ようやくこの頃知った。";
```

この文章で使われている文字一覧は，既に使ったことのある`Characters`と`Union`を使うことで，次のように*Mathematica*に調べさせることが出来ます．直前の出力を表すパーセント記号（%）も使って，使われている文字の種類（今回の場合，文字一覧のリストに含まれる要素の個数）がいくつあるかも調べさせています．

```
In[100]:= 使用文字の一覧 = Union[Characters[吾輩は猫である]]

Out[100]= {。，々，あ，い，う，え，か，が，き，く，け，げ，こ，さ，し，じ，
  ず，せ，そ，た，だ，ち，っ，つ，て，で，と，ど，な，に，ぬ，の，
  は，ば，ぶ，ぷ，べ，ぽ，ま，み，む，め，も，や，ゆ，よ，ら，り，
  る，れ，ろ，わ，を，ん，ス，ニ，フ，ャ，ワ，ー，一，上，中，
  事，人，今，会，何，出，別，前，名，吹，吾，咽，妙，始，実，
  少，度，弱，当，彼，後，思，恐，悪，感，憶，我，所，持，捕，
  掌，族，時，暗，書，残，段，毛，泣，無，煙，煮，片，猫，獰，
  生，番，真，知，種，穴，突，第，缶，考，聞，草，落，薄，薬，
  装，見，記，話，起，載，輩，輪，逢，間，頃，顔，食，飲，飾}

In[101]:= Length[%]

Out[101]= 138
```

文章の文字数は，次のように448文字ですから，かなり同じ文字（そのほとんどは平仮名と思います）が繰り返して使われていることがわかります．`Length`と

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

StringLengthの使い分けをきちんと出来るか大丈夫でしょうか．Lengthはリストに含まれる要素の個数で，StringLengthは文字列の文字の長さです．

```
In[102]:= StringLength[吾輩は猫である]

Out[102]= 448
```

【青空文庫】本文中の「吾輩は猫である」は，夏目漱石全集を底本とする青空文庫のXHTMLファイルから引用しています．引用にあたり，括弧書きのルビは削除してあります．この場を借りて，青空文庫のような素晴らしい取り組みをされている方々に感謝します．貴重な電子データをありがとうございます！

■ **作品に使われている文字の種類を調べてみよう**

もっと詳しく調べていくことにしましょう．まずは文字を種類ごとにリストとして定義しましょう．ひらがなとカタカナは以前の通り，次のように定義します（結果を確認する必要がない場合はセミコロンを付けましょう）．

```
In[103]:= ひらがな = CharacterRange["ぁ", "ん"];

In[104]:= カタカナ = CharacterRange["ァ", "ヴ"];
```

記号は非常に多くの種類があるので，いくつか主要なものだけを含めて「漢字以外」というグループも定義しておきます．ここでは全角スペース（　）からカタカナの伸ばす記号（ー）までとして定義しています．どのような内容が含まれるかを確認したい場合は，次のようにセミコロンを付けずに実行しましょう．文字が表示されていないところは，機種依存文字であったり，もともと文字が未定義になっている区画になります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[105]:= 漢字以外 = CharacterRange["　", "ー"]

Out[105]= {　, 、, 。, 〃, 〄, 々, 〆, 〇, 〈, 〉, 《, 》, 「, 」, 『, 』, 【,
  】, 〒, 〓, 〔, 〕, 〖, 〗, 〘, 〙, 〚, 〛, 〜, 〝, 〞, 〟, 〠,
  〡, 〢, 〣, 〤, 〥, 〦, 〧, 〨, 〩,  ,  ,  ,  ,  ,  , 〰, 〱,
  〲, 〳, 〴, 〵, 〶, 〷,  ,  ,  ,  ,  ,  ,  ,  ,  , ぁ, あ, ぃ,
  い, ぅ, う, ぇ, え, ぉ, お, か, が, き, ぎ, く, ぐ, け, げ,
  こ, ご, さ, ざ, し, じ, す, ず, せ, ぜ, そ, ぞ, た, だ, ち,
  ぢ, っ, つ, づ, て, で, と, ど, な, に, ぬ, ね, の, は, ば,
  ぱ, ひ, び, ぴ, ふ, ぶ, ぷ, へ, べ, ぺ, ほ, ぼ, ぽ, ま, み,
  む, め, も, ゃ, や, ゅ, ゆ, ょ, よ, ら, り, る, れ, ろ, ゎ,
  わ, ゐ, ゑ, を, ん, ゔ,  ,  ,  ,  ,  ,  , ゛, ゜, ゝ, ゞ,  ,
   , ァ, ア, ィ, イ, ゥ, ウ, ェ, エ, ォ, オ, カ, ガ, キ, ギ, ク,
  グ, ケ, ゲ, コ, ゴ, サ, ザ, シ, ジ, ス, ズ, セ, ゼ, ソ, ゾ, タ,
  ダ, チ, ヂ, ッ, ツ, ヅ, テ, デ, ト, ド, ナ, ニ, ヌ, ネ, ノ,
  ハ, バ, パ, ヒ, ビ, ピ, フ, ブ, プ, ヘ, ベ, ペ, ホ, ボ, ポ,
  マ, ミ, ム, メ, モ, ャ, ヤ, ュ, ユ, ョ, ヨ, ラ, リ, ル, レ,
  ロ, ヮ, ワ, ヰ, ヱ, ヲ, ン, ヴ, ヵ, ヶ, ヷ, ヸ, ヹ, ヺ, ・, ー}
```

グループについての章で使った，`Intersection`，`Complement`，`Union`などの組込み関数を使うことで，文章で使われている文字を種類ごとに分類することが出来ます．復讐しておくと，共通部分が`Intersection`で，補集合（指定したもの以外の集合）が`Complement`で，和集合が`Union`です．

従って，吾輩は猫であるの冒頭の一段落に含まれるひらがなの種類は次のように命令します．`Length`と`%`を使うことで，使われているひらがなの種類が52個であることもわかります．

```
In[106]:= Intersection[使用文字の一覧, ひらがな]

Out[106]= {あ, い, う, え, か, が, き, く, け, げ, こ, さ,
  し, じ, ず, せ, そ, た, だ, ち, っ, つ, て, で, と,
  ど, な, に, ぬ, の, は, ば, ぶ, ぷ, べ, ぽ, ま, み,
  む, め, も, や, ゆ, よ, ら, り, る, れ, ろ, わ, を, ん}

In[107]:= Length[%]

Out[107]= 52
```

カタカナについても同様です．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[108]:= Intersection[使用文字の一覧, カタカナ]
Out[108]= {ス, ニ, フ, ャ, ワ}
```

使われている漢字を調べるには，先ほど準備した漢字以外の補集合（漢字以外に含まれていない要素の集合）を*Mathematica*に調べてもらいます．漢字以外には，ひらがなとカタカナ，いくつかの記号が含まれているので，次のように文章に使われている漢字だけを含むリストが得られます．冒頭の段落だけなのに，78種類もの漢字が使われているとは驚きですね．

```
In[109]:= Complement[使用文字の一覧, 漢字以外]
Out[109]= {一, 上, 中, 事, 人, 今, 会, 何, 出, 別, 前, 名, 吹, 吾, 咽, 妙,
  始, 実, 少, 度, 弱, 当, 彼, 後, 思, 恐, 悪, 感, 憶, 我, 所, 持,
  捕, 掌, 族, 時, 暗, 書, 残, 段, 毛, 泣, 無, 煙, 煮, 片, 猫, 獰,
  生, 番, 真, 知, 種, 穴, 突, 第, 缶, 考, 聞, 草, 落, 薄, 薬,
  装, 見, 記, 話, 起, 載, 輩, 輪, 逢, 間, 頃, 顔, 食, 飲, 飾}
```

```
In[110]:= Length[%]
Out[110]= 78
```

記号を調べさせるときの命令は少し複雑になります．まず，ひらがなとカタカナと記号だけを使われている文字一覧から抜き出してから，ひらがなとカタカナを取り除きます．`Intersection`と`Complement`を組み合わしているので少し難しいかもしれません．

```
In[111]:= Complement[Intersection[使用文字の一覧, 漢字以外],
  ひらがな, カタカナ]
Out[111]= {。, 々, ー}
```

「`Complement`」という命令は正確には「最初のリストから，二番目以降のリストに含まれている要素を全て取り除け」という意味になるので，共通部分を抜き出しているところを分解してみれば難しくないことがわかると思います．

```
In[112]:= 使用文字の一覧のうち記号とひらがなとカタカナ =
   Intersection[使用文字の一覧, 漢字以外];
```

```
In[113]:= Complement[使用文字の一覧のうち記号とひらがなとカタカナ,
  ひらがな, カタカナ]
Out[113]= {。, 々, ー}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

- **作品を小学生のつもりになって読んでみよう**

以前に小学校第一学年の漢字リストを作ったように，文部科学省のウェブサイトの小学校学習指導要領の国語に関するところに掲載されている「学年別漢字配当表」から，各学年に配当されている漢字のリストを作っておきます．

```
In[114]:= 小学校1学年 =
   Characters[
    "一右雨円王音下火花貝学気九休玉金空月犬見五口校左三山子四糸字耳七車手十出女小上森人水正生青夕石赤千川先早草足村大男竹中虫町天田土二日入年白八百文木本名目立力林六"];
```

```
In[115]:= 小学校2学年 =
   Characters[
    "引羽雲園遠何科夏家歌画回会海絵外角楽活間丸岩顔汽記帰弓牛魚京強教近兄形計元言原戸古午後語工公広交光考行高黄合谷国黒今才細作算止市矢姉思紙寺自時室社弱首秋週春書少場色食心新親図数西声星晴切雪船線前組走多太体台地池知茶昼長鳥朝直通弟店点電刀冬当東答頭同道読内南肉馬売買麦半番父風分聞米歩母方北毎妹万明鳴毛門夜野友用曜来里理話"];
```

```
In[116]:= 小学校3学年 =
   Characters[
    "悪安暗医委意育員院飲運泳駅央横屋温化荷界開階寒感漢館岸起期客究急級宮球去橋業曲局銀区苦具君係軽血決研県庫湖向幸港号根祭皿仕死使始指歯詩次事持式実写者主守取酒受州拾終習集住重宿所暑助昭消商章勝乗植申身神真深進世整昔全相送想息速族他打対待代第題炭短談着注柱丁帳調追定庭笛鉄転都度投豆島湯登等動童農波配倍箱畑発反坂板皮悲美鼻筆氷表秒病品負部服福物平返勉放味命面問役薬由油有遊予羊洋葉陽様落流旅両緑礼列練路和"];
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[117]:= 小学校4学年 =
  Characters[
   "愛案以衣位囲胃印英栄塩億加果貨課芽改械害街各覚完官管関観願希
     季紀喜旗器機議求泣救給挙漁共協鏡競極訓軍郡径型景芸欠結
     建健験固功好候航康告差菜最材昨札刷殺察参産散残士氏史司
     試児治辞失借種周祝順初松笑唱焼象照賞臣信成省清静席積折
     節説浅戦選然争倉巣束側続卒孫帯隊達単置仲貯兆腸低底停的
     典伝徒努灯堂働特得毒熱念敗梅博飯飛費必票標不夫付府副粉
     兵別辺変便包法望牧末満未脈民無約勇要養浴利陸良料量輪類
     令冷例歴連老労録"];
```

```
In[118]:= 小学校5学年 =
  Characters[
   "圧移因永営衛易益液演応往桜恩可仮価河過賀快解格確額刊幹慣眼基
     寄規技義逆久旧居許境均禁句群経潔件券険検限現減故個護効
     厚耕鉱構興講混査再災妻採際在財罪雑酸賛支志枝師資飼示似
     識質舎謝授修述術準序招承証条状常情織職制性政勢精製税責
     績接設舌絶銭祖素総造像増則測属率損退貸態団断築張提程適
     敵統銅導徳独任燃能破犯判版比肥非備俵評貧布婦富武復複仏
     編弁保墓報豊防貿暴務夢迷綿輸余預容略留領"];
```

```
In[119]:= 小学校6学年 =
  Characters[
   "異遺域宇映延沿我灰拡革閣割株干巻看簡危机揮貴疑吸供胸郷勤筋系
     敬警劇激穴絹権憲源厳己呼誤后孝皇紅降鋼刻穀骨困砂座済裁
     策冊蚕至私姿視詞誌磁射捨尺若樹収宗就衆従縦縮熟純処署諸
     除将傷障城蒸針仁垂推寸盛聖誠宣専泉洗染善奏窓創装層操蔵
     臓存尊宅担探誕段暖値宙忠著庁頂潮賃痛展討党糖届難乳認納
     脳派拝背肺俳班晩否批秘腹奮並陛閉片補暮宝訪亡忘棒枚幕密
     盟模訳郵優幼欲翌乱卵覧裏律臨朗論"];
```

面倒なので，小学校の各学年に配当された全ての漢字を含むリスト「小学校配当漢字」も定義しておきます．ここでは「複数のリストをひとつのリストにまとめなさい」という意味を持つ「Join」という命令を使っていますが，Unionでも同じ結果に

```
In[120]:= 小学校配当漢字 = Join[小学校1学年, 小学校2学年, 小学校3学年,
    小学校4学年, 小学校5学年, 小学校6学年];
```

前回に使った「使用文字の一覧」と「漢字以外」を組み合わせることで，次のように小学校配当漢字に含まれない漢字のみを抜き出すことが出来ます（これを未配当漢字として定義しておきましょう）．そんなに難しそうな漢字を含んでいるようには見え

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

ませんし，わずか一段落ですが，21種類もの小学校に配当されていない漢字を含んでいることがわかります．

```
In[121]:= 未配当漢字 = Complement[使用文字の一覧, 漢字以外, 小学校配当漢字]

Out[121]= {吹, 吾, 咽, 妙, 彼, 恐, 憶, 捕, 掌,
           煙, 煮, 猫, 獰, 突, 缶, 薄, 載, 輩, 逢, 頃, 飾}

In[122]:= Length[%]

Out[122]= 21
```

早口言葉のところで何度も使ったStringReplaceを使うと，未配当漢字を　マークなどに置き換えることで，小学生になった気分で作品を読むことが出来ます．次の例は実際に「猫」という文字を「　」に置き換えてみたものです．出力された作品中で「猫」が「　」に変わっているのがわかると思います．

```
In[123]:= StringReplace[吾輩は猫である, "猫" → "　"]

Out[123]= 吾輩は　である。名前はまだ無い。どこで生れたかとんと見当がつかぬ。
    何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶
    している。吾輩はここで始めて人間というものを見た。しかもあとで
    聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。こ
    の書生というのは時々我々を捕えて煮て食うという話である。しかし
    その当時は何という考もなかったから別段恐しいとも思わなかった。
    ただ彼の掌に載せられてスーと持ち上げられた時何だかフワフワした
    感じがあったばかりである。掌の上で少し落ちついて書生の顔を見た
    のがいわゆる人間というものの見始であろう。この時妙なものだと思
    った感じが今でも残っている。第一毛をもって装飾されべきはずの顔
    がつるつるしてまるで薬缶だ。その後　にもだいぶ逢ったがこんな片
    輪には一度も出会わした事がない。のみならず顔の真中があまりに突
    起している。そうしてその穴の中から時々ぷうぷうと煙を吹く。どう
    も咽せぽくて実に弱った。これが人間の飲む煙草というものである事
    はようやくこの頃知った。
```

この作業を全ての未配当漢字に対して繰り返せば，確かに小学生の気持ちになって作品を読むことは出来ます．しかしながら，余りにも冗長で計算機のパワーを，即ち*Mathematica*のパワーを使っていません．冗長な作業は*Mathematica*に任せることが重要です．

*Mathematica*における文字の置き換え規則には，「リストの中のどれか」という意味を持つ「Alternatives」という命令が使えます．従って，次のような置き換え規則を使うことで，未配当漢字の全てを「　」に置き換えることが出来ます．当り前のこと

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

かもしれませんが，小学生に「吾輩は猫である」は少し難しいかもしれないことがわかります．

```
In[124]:= StringReplace[吾輩は猫である, Alternatives[未配当漢字] → " "]
```

```
Out[124]= は である。名前はまだ無い。どこで生れたかとんと見当がつかぬ。何
 でも 暗いじめじめした所でニャーニャー泣いていた事だけは記 して
 いる。 はここで始めて人間というものを見た。しかもあとで聞くと
 それは書生という人間中で一番 悪な種族であったそうだ。この書生
 というのは時々我々を えて て食うという話である。しかしその当時
 は何という考もなかったから別段 しいとも思わなかった。ただ の
 に せられてスーと持ち上げられた時何だかフワフワした感じがあっ
 たばかりである。 の上で少し落ちついて書生の顔を見たのがいわゆ
 る人間というものの見始であろう。この時 なものだと思った感じが
 今でも残っている。第一毛をもって装 されべきはずの顔がつるつる
 してまるで薬 だ。その後 にもだいぶ ったがこんな片輪には一度も
 出会わした事がない。のみならず顔の真中があまりに 起している。
 そうしてその穴の中から時々ぷうぷうと を く。どうも せぽくて実
 に弱った。これが人間の飲む 草というものである事はようやくこの
 知った。
```

■ **学年別配当漢字を使って細かく調べてみよう**

この作品をもう少し詳しく調べてみることにしましょう．*Mathematica*に調べさせるのに必要な組込み関数は既に使ったものばかりです．文字列の長さを調べさせるStringLength，該当する文字の個数を調べさせるStringCount，リストの要素のどれかに該当という意味を持つAlternativesです．まずは，対象の文章の長さを確認しておきましょう．

```
In[125]:= 文章の長さ = StringLength[吾輩は猫である]

Out[125]= 448
```

小学校の各学年に配当されている漢字の個数を確認していきましょう．StringCountとAlternativesで使われている漢字の個数を求められます．各学年ごとに6回繰り返すことで，それぞれの配当漢字の個数がわかります．

```
In[126]:= StringCount[吾輩は猫である, Alternatives[小学校1学年]]

Out[126]= 23
```

```
In[127]:= StringCount[吾輩は猫である, Alternatives[小学校2学年]]

Out[127]= 36
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[128]:= StringCount[吾輩は猫である, Alternatives[小学校3学年]]

Out[128]= 20

In[129]:= StringCount[吾輩は猫である, Alternatives[小学校4学年]]

Out[129]= 6

In[130]:= StringCount[吾輩は猫である, Alternatives[小学校5学年]]

Out[130]= 0

In[131]:= StringCount[吾輩は猫である, Alternatives[小学校6学年]]

Out[131]= 5
```

文字が赤くなっていますが，この赤字で強調表示している部分は，それぞれの命令に共通している部分です．変化しているのは学年別の配当漢字を覚えさせたリストの名前だけです．このように共通する部分が多い場合，以前にも出てきましたが，これと同じ指示を組込み関数「Map」を使うと簡単に書くことが出来ます．この命令の意味は「次のリストのそれぞれに対して同じ命令を実行せよ」です．

今回の場合，以前に比べて共通する部分が多いので，次のように少し複雑になります．赤い文字の部分は，個別に命令した場合に共通していた部分を表しています．共通しなかった部分が「シャープ（#）」になり，最後に「アンパサンド（&）」が付け加わっているのが変更部分です．以前のMapの例では，#と&が省略されていたのですが，Mapを使うときは基本的に，共通しない部分を#に，共通する部分の最後に&を付け加えなければいけません．

```
In[132]:= Map[StringCount[吾輩は猫である, Alternatives[#]] &,
            {小学校1学年, 小学校2学年, 小学校3学年, 小学校4学年,
             小学校5学年, 小学校6学年}]

Out[132]= {23, 36, 20, 6, 0, 5}
```

```
Map[StringCount[文章,Alternatives[#]]&, {学年1,学年2}]
```

どんな操作か　　共通部分は#，最後に&　　指示の対象を記したリスト

リストの一括処理：いくつものリストに同じ操作を実行

ひらがなとカタカナの個数も同時に調べるには，次のように，ひらがなとカタカナも追加するだけで，300個と9個ということがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[133]:= Map[StringCount[吾輩は猫である, Alternatives[#]] &,
  {小学校1学年, 小学校2学年, 小学校3学年, 小学校4学年,
   小学校5学年, 小学校6学年, ひらがな, カタカナ}]

Out[133]= {23, 36, 20, 6, 0, 5, 300, 9}

In[134]:= % / 文章の長さ * 100.0

Out[134]= {5.13393, 8.03571, 4.46429,
  1.33929, 0, 1.11607, 66.9643, 2.00893}
```

次章で扱いますが，求めた文字の個数を「文章の長さ」で割って100倍すれば使用率がパーセントで求められます．わり算は「スラッシュ（/）」で，かけ算は「アスタリスク（*）」で命令します．これによれば，小学校1学年の配当漢字は5%，2学年が8%，3学年が4%，4学年が1%，5学年が0%，6学年が1%，ひらがなが67%，カタカナが2%使われていることがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

数と式：数に隠された秘密を少し覗いてみよう

数と式，計算をする上で基本となる仕組みを学びましょう．

四則演算で面白い数を見付けてみよう

これまで*Mathematica*にリストや文字列などの数学とはかけ離れた内容を扱わせてきましたが，*Mathematica*には難しい計算をさせることも出来ます．ここでは，基本的な四則演算（足す，引く，掛ける，割る）について取り上げていきたいと思います．

■ 足し算（+）で足して10になる数を探そう

*Mathematica*に足し算を行わせるには，算数や数学でも使われるプラス記号「+」を使います．教科書などにあるように，そのまま命令として入力すれば，*Mathematica*は計算結果を教えてくれます．例えば，単純な「1+2+3+4」もそのまま入力するだけで十分です．答えが「10」であると教えてくれます．

```
In[1]:= 1 + 2 + 3 + 4

Out[1]= 10
```

これまで組込み関数と呼ばれる*Mathematica*へ指示するための命令をいくつも習ってきました．実は，足し算にもプラス記号とは別に，「次の数を足し合わせよ」という意味を持つ「Plus」という命令も用意されています．例えば，この命令を使って先ほどの単純な「1+2+3+4」を計算するには次のようにします．もちろん，答えは「10」になります．

```
In[2]:= Plus[1, 2, 3, 4]

Out[2]= 10
```

では，日本で余り見ないタイプの問題「3+□=10」を考えましょう．日常的には良くある状況を表していて，例えば，パーティの参加者は10人なのにコップが3人分しかない，あといくつコップを用意すれば良いでしょうか，という問題になります．

普通の考え方とは違いますが，用意しなければならないコップの数を直接求めるのではなく，新たにいくつコップを追加すると10人分になるのかを考えましょう．そこで，追加するコップの個数として，5個，6個，7個，8個，9個，10個の場合を考えます．分かりやすくするために，追加する可能性のあるコップの個数を*Mathematica*に覚えさせておきます．

```
In[3]:= 追加するコップの個数 = {5, 6, 7, 8, 9, 10}

Out[3]= {5, 6, 7, 8, 9, 10}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

問題は，既にある3個のコップに加えると10個になるコップの個数ですから，この数のそれぞれに3を加えて10になるかを確認すれば良いですね．プラス記号「+」を使って確認しようとすると，6回も足し算を命令する必要があります．非常に面倒ですね．*Mathematica*では面倒なことをしなくても良いように，もっと便利な方法を用意しています．それは．．．

『リストに足し算を行ってしまう』

です．*Mathematica*では「数とリストの足し算」を，「リストの各要素ごとに同じ足し算をしたいのだろう」と解釈してくれます．従って，次のように単純に3をリストに加えることで一度に全ての足し算を指示することが出来ます．

```
In[4]:= 3 + 追加するコップの個数

Out[4]= {8, 9, 10, 11, 12, 13}
```

3+{1,2,3,4,5} ⇒ {3+1,3+2,3+3,3+4,3+5}

リストと四則演算　　各要素との四則演算に展開される

リストと四則演算：各要素に対しての同じ四則演算が行われる

実際には次のように，以前に出てきたTableFormをオプション（追加指示）TableHeadingsと組み合わせて使うことで，よりわかりやすい見た目となります．

```
In[5]:= TableForm[{追加するコップの個数, 3 + 追加するコップの個数},
         TableHeadings → {{"追加するコップの個数", "合計"}, None}]
```

Out[5]//TableForm=

追加するコップの個数	5	6	7	8	9	10
合計	8	9	10	11	12	13

もちろん，同じ操作を何度も繰り返す，という視点からは文字列操作で出てきたMapを使うことも出来ますね．足し算の2種類の書き方のそれぞれに対して，実際にMapを使ってみたものが次になります．直接リストと足し算した方が簡単で便利だということがわかると思います．

```
In[6]:= Map[Plus[3, #] &, 追加するコップの個数]

Out[6]= {8, 9, 10, 11, 12, 13}

In[7]:= Map[(3 + #) &, 追加するコップの個数]

Out[7]= {8, 9, 10, 11, 12, 13}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

引き算（-）で足して10になることを確認しよう

*Mathematica*に引き算を行わせるには，算数や数学でも使われるマイナス記号「-」を使います．教科書などにあるように，そのまま命令として入力すれば，*Mathematica*は計算結果を教えてくれます．例えば，単純な「20-4-3-2-1」もそのまま入力するだけで十分です．答えが「10」であると教えてくれます．

```
In[8]:= 20 - 4 - 3 - 2 - 1

Out[8]= 10
```

引き算にも，足し算と同じように，「次の2数の引き算をせよ」という意味を持つ「Subtract」という命令も用意されています．ただし，足し算と異なり2数に対してしか使えません．例えば，この命令を使って単純な「20-10」を計算するには次のようにします．もちろん，答えは「10」になります．

```
In[9]:= Subtract[20, 10]

Out[9]= 10
```

では，先ほどと同じ問題「3+⬚=10」を考えましょう．これを「10-⬚=3」と変形して解きましょう．即ち，10から何を引いたら3になるのか，を考えましょう．そこで，引くコップの個数として，5個，6個，7個，8個，9個，10個の場合を考えます．先ほど同じように，引き算するコップの個数を*Mathematica*に覚えさせておきます．

```
In[10]:= 引くコップの個数 = {5, 6, 7, 8, 9, 10}

Out[10]= {5, 6, 7, 8, 9, 10}
```

引き算についても*Mathematica*では「数とリストの引き算」を，「リストの各要素ごとに同じ引き算をしたいのだろう」と解釈してくれます．従って，次のように単純に10からリストを引くことで一度に全ての引き算を指示することが出来ます．

```
In[11]:= 10 - 引くコップの個数

Out[11]= {5, 4, 3, 2, 1, 0}
```

実際には次のように，以前に出てきたTableFormをオプション（追加指示）TableHeadingsと組み合わせて使うことで，よりわかりやすい見た目となります．

```
In[12]:= TableForm[{引くコップの個数, 10 - 引くコップの個数},
           TableHeadings → {{"引くコップの個数", "残り"}, None}]
```

Out[12]//TableForm=

引くコップの個数	5	6	7	8	9	10
残り	5	4	3	2	1	0

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

もちろん，同じくMapを使うことも出来ます．

```
In[13]:= Map[Subtract[10, #] &, 引くコップの個数]
Out[13]= {5, 4, 3, 2, 1, 0}
```

```
In[14]:= Map[(10 - #) &, 引くコップの個数]
Out[14]= {5, 4, 3, 2, 1, 0}
```

【数学的表記と四則演算】*Mathematica*での計算に関する指示は，なるべく算数や数学の慣用的な表現方法を踏襲するようになっています．従って，教科書等にある計算問題はそのまま入力すれば，結果を*Mathematica*が求めて教えてくれます．実際に，この項では足し算と引き算を習っていますが，他の演算や括弧なども含め，慣用的な表現がそのまま利用できるようになっています．

- **掛け算（×）して規則正しい数を探してみよう**

*Mathematica*に掛け算を行わせるには，算数や数学でも使われる記号「×」を使います．教科書などにあるように，そのまま命令として入力すれば，*Mathematica*は計算結果を教えてくれます．例えば，単純な「1×2×3×4」もそのまま入力するだけで十分です．答えが「24」であると教えてくれます．なお，記号「×」は「ESC*ESC」で入力できます．

```
In[15]:= 1 × 2 × 3 × 4
Out[15]= 24
```

かけ算の記号には，「×」の他，アスタリスク「*」やスペース「　」も使えます．このうちスペースについては，*Mathematica*が自動的に「×」を補って表示してくれますので，入力簡単で表示もわかりやすいのでお薦めです．

```
In[16]:= 1 * 2 * 3 * 4
Out[16]= 24
```

```
In[17]:= 1 × 2 × 3 × 4
Out[17]= 24
```

更に掛け算にも，足し算や引き算と同じように，「次の数を掛け合わせよ」という意味を持つ「Times」という命令も用意されています．例えば，この命令を使って先ほどの単純な「1×2×3×4」を計算するには次のようにします．もちろん，答えは「24」になります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[18]:= Times[1, 2, 3, 4]

Out[18]= 24
```

次の2つの掛け算を見てください.

```
In[19]:= 110 011 * 101

Out[19]= 11 111 111
```

```
In[20]:= 110 011 * 202

Out[20]= 22 222 222
```

*Mathematica*では算数や数学で習う計算順序（掛け算を足し算や引き算よりも先に行う）に従って計算をします．従って，何も考えずに教科書にあるような計算式をそのまま指示するだけで答えが得られます．

```
In[21]:= 110 011 * 202 - 110 011 * 101

Out[21]= 11 111 111
```

また，括弧による計算順序の指定も可能です．このとき使う括弧は必ず丸括弧（「(」と「)」）にしてください．算数や数学で使用することもある「{ }」や「[ ]」は，*Mathematica*では別のに意味で使われますので，何重になっても計算順序の指定には丸括弧だけを使います．

```
In[22]:= (10 000 + 100 + 1) * 11

Out[22]= 111 111
```

```
In[23]:= 10 101 * 11

Out[23]= 111 111
```

足し算や引き算と同じく，*Mathematica*では「数とリストの掛け算」を，「リストの各要素ごとに同じ掛け算をしたいのだろう」と解釈してくれます．この機能をうまく使って，「12345679」に何を掛けたら「111111」のような規則正しい数になるかを考えましょう．試しに，1から5のリストと積をとってみます．規則正しそうな数は表れません．

```
In[24]:= 12 345 679 * {1, 2, 3, 4, 5}

Out[24]= {12 345 679, 24 691 358, 37 037 037, 49 382 716, 61 728 395}
```

もっと大きな数，もっとたくさんの数との積を計算させないと見付かりそうにありません．でも，そんなに長いリストを手で入力するのは面倒です．*Mathematica*には「次

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

の数までの整数のリストを作成せよ」という意味の「Range」なる命令があります．これを使うことで，次のように簡単にリストを作ることが出来ます．

```
In[25]:= Range[20]

Out[25]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
          11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

従って，次のように使うことで，9のときと18のときに「111111」のような規則正しい数になることがわかります．

```
In[26]:= 12 345 679 * Range[20]

Out[26]= {12 345 679, 24 691 358, 37 037 037, 49 382 716, 61 728 395,
          74 074 074, 86 419 753, 98 765 432, 111 111 111, 123 456 790,
          135 802 469, 148 148 148, 160 493 827, 172 839 506, 185 185 185,
          197 530 864, 209 876 543, 222 222 222, 234 567 901, 246 913 580}
```

TableFormに「行方向に表形式で表示せよ」というオプション「TableDirections → Row」を付けて指示を出すことで，次のようにわかりやすい結果も得られます．

```
In[27]:= TableForm[{Range[20], 12 345 679 * Range[20]},
          TableDirections → Row]

Out[27]//TableForm=
          1    12 345 679
          2    24 691 358
          3    37 037 037
          4    49 382 716
          5    61 728 395
          6    74 074 074
          7    86 419 753
          8    98 765 432
          9    111 111 111
          10   123 456 790
          11   135 802 469
          12   148 148 148
          13   160 493 827
          14   172 839 506
          15   185 185 185
          16   197 530 864
          17   209 876 543
          18   222 222 222
          19   234 567 901
          20   246 913 580
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

【数のブロック表示】*Mathematica*では次のように比較的大きな数を入力すると，3桁ずつまとめて表示を行います．日本式では4桁ずつの方がわかりやすいかもしれません．4桁ずつに変更したい場合は，編集メニューの「環境設定...」で表示されるダイアログにて，「外観」のタブの中の「数」のタブの中の「桁ブロックのサイズ」を変更してください．このダイアログでは様々な設定を変更できるようになっています．いろいろと変更してみるのも面白いかもしれません．「デフォルト値に戻す」ボタンで元に戻せるので安心して変更してみてください．

■ **割り算（/）も使って分数の計算をしてみよう**

*Mathematica*に割り算を行わせるには，算数や数学でも使われる記号「÷」を使います．教科書などにあるように，そのまま命令として入力すれば，*Mathematica*は計算結果を教えてくれます．例えば，単純な「20÷2」もそのまま入力するだけで十分です．答えが「10」であると教えてくれます．なお，記号「÷」は「ESCdivESC」で入力できます．

```
In[28]:= 20 ÷ 2

Out[28]= 10
```

割り算の記号には，「÷」の他，スラッシュ「/」も使えます．

```
In[29]:= 20 / 2

Out[29]= 10
```

更に割り算にも，他の四則演算と同じように，「次の2数の割り算を求めよ」という意味を持つ「Divide」という命令も用意されています．例えば，この命令を使って先ほどの単純な「20÷2」を計算するには次のようにします．もちろん，答えは「10」になります．

```
In[30]:= Divide[20, 2]

Out[30]= 10
```

割り切れない場合，*Mathematica*は結果を分数で教えてくれます．

```
In[31]:= 1 / 2
```

Out[31]= $\frac{1}{2}$

他の四則演算と同じく，*Mathematica*では「数とリストの割り算」を，「リストの各要素ごとに同じ割り算をしたいのだろう」と解釈してくれます．この機能をうまく使って，「3」で割り切れる数字を1から20までの整数から探してみましょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[32]:= `Range[20] / 3`

Out[32]= $\left\{\frac{1}{3}, \frac{2}{3}, 1, \frac{4}{3}, \frac{5}{3}, 2, \frac{7}{3}, \frac{8}{3}, 3, \frac{10}{3}, \frac{11}{3}, 4, \frac{13}{3}, \frac{14}{3}, 5, \frac{16}{3}, \frac{17}{3}, 6, \frac{19}{3}, \frac{20}{3}\right\}$

In[33]:= `TableForm[{Range[20], Range[20] / 3}]`

Out[33]//TableForm=

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\frac{1}{3}$	$\frac{2}{3}$	1	$\frac{4}{3}$	$\frac{5}{3}$	2	$\frac{7}{3}$	$\frac{8}{3}$	3	$\frac{10}{3}$	$\frac{11}{3}$	4	$\frac{13}{3}$	$\frac{14}{3}$	5	$\frac{16}{3}$	$\frac{17}{3}$	6	$\frac{19}{3}$	$\frac{20}{3}$

割り算の順番を変えることで，「30」を割り切る数を1から20までの整数から探すことも出来ます．

In[34]:= `TableForm[{Range[20], 30 / Range[20]}]`

Out[34]//TableForm=

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	1
30	15	10	$\frac{15}{2}$	6	5	$\frac{30}{7}$	$\frac{15}{4}$	$\frac{10}{3}$	3	$\frac{30}{11}$	$\frac{5}{2}$	$\frac{30}{13}$	$\frac{15}{7}$	2	$\frac{15}{8}$	$\frac{30}{17}$	$\frac{5}{3}$	[illegible]

分数同士の四則演算もそのまま行えます．

In[35]:= `1 / 2 + 1 / 3`

Out[35]= $\frac{5}{6}$

より分数らしく入力したい場合は，CTRLを押しながら「/」を入力します．例えば，次のような分数の計算式を入力するには，「1」を入力，CTRLを押しながら「/」を入力，「2」を入力，カーソルキーの右，「+」を入力，「1」を入力，CTRLを押しながら「/」を入力，「3」を入力します．カーソルキーの右は，CTRLを押しながらスペースを入力でも代用できます．

In[36]:= $\frac{1}{2} + \frac{1}{3}$

Out[36]= $\frac{5}{6}$

【帯分数と仮分数】*Mathematica*は分数を仮分数で表現します．帯分数では掛け算との区別が難しいためです．区別が難しいため利用はお薦めしませんが，どうしても帯分数で入力を行いたい場合は，暗黙の加算である「ESC+ESC」を使ってください．「1ESC+ESC$\frac{1}{3}$」で「$1\frac{1}{3}$」と入力でき，「$\frac{4}{3}$」を表します．

累乗（^）を使って複利計算をしてみよう

*Mathematica*に累乗（冪乗，指数計算）を行わせるには，算数や数学での表現とは異なる記号「^」を使います．教科書などにあるような表現（数の右肩に小さく指数を書く方法）にしたい場合は，パレット（□^□）を使うか後述するキーボードショートカットを利用してください．例えば，記号「^」を使う場合，単純な「2^{10}」を入力するには「2^10」と入力します．答えが「1024」であると教えてくれます．

```
In[37]:= 2^10

Out[37]= 1024
```

累乗にも四則演算（加減乗除）と同じように，「次の累乗を求めよ」という意味の「Power」という命令も用意されています．例えば，この命令を使って先ほどの単純な「2^10」を計算するには次のようにします．もちろん，答えは「1024」になります．

```
In[38]:= Power[2, 10]

Out[38]= 1024
```

四則演算と同じく，*Mathematica*では「数とリストの累乗」を，「リストの各要素ごとに同じ累乗をしたいのだろう」と解釈してくれます．この機能をうまく使って，「3」を何度掛けたら1000に近くなるか，1から10までの整数から探してみましょう．結果をTableFormで表示すると，3を6回掛けると1000に近いことがわかります．

```
In[39]:= 3^Range[10]

Out[39]= {3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049}

In[40]:= TableForm[{Range[10], 3^Range[10]}]

Out[40]//TableForm=
```

1	2	3	4	5	6	7	8	9	10
3	9	27	81	243	729	2187	6561	19683	59049

より指数らしく入力したい場合は，CTRLを押しながら「^」を入力します．例えば，次のような累乗の計算式を入力するには，「2」を入力，CTRLを押しながら「^」を入力，「10」を入力，カーソルキーの右，「+」を入力，「3」を入力，CTRLを押しながら「^」を入力，「10」を入力します．カーソルキーの右は，CTRLを押しながらスペースを入力でも代用できます．

In[41]:= $2^{10} + 3^{10}$

Out[41]= 60073

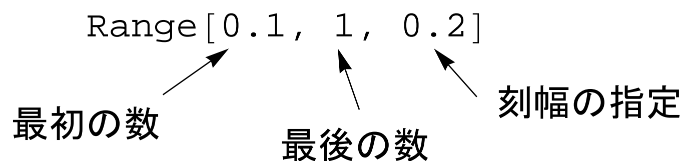
All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

累乗の計算が出来ると複利計算が簡単に出来ます．複利計算は，年利を小数表示に直したものに1を加えた数を，年数分だけ累乗するだけです．例えば，0.5%の年利の場合は1.005を年数分だけ累乗すれば良いことになります．10年の場合の計算をしてみます．1.05114ということは，10年間での利率は約5.1%ということになります．

```
In[42]:= 1.005^10

Out[42]= 1.05114
```

0.1%から1%まで0.2%刻みの年利で，10年後にどの程度違うかを計算してみましょう．まずは，Rangeを使って年利のリストを作りましょう．Rangeでは次のように使うことで，0.1から1まで0.2刻みの数を作ることが出来ます．

```
In[43]:= 年利 = Range[0.1, 1, 0.2]

Out[43]= {0.1, 0.3, 0.5, 0.7, 0.9}
```



リストの作成：刻幅を指定して数のリストを作る

年利をパーセントから小数に直します．

```
In[44]:= 年利の割合 = 年利 / 100

Out[44]= {0.001, 0.003, 0.005, 0.007, 0.009}
```

年利の割合に1を加えます．1は元本に対応しています．

```
In[45]:= 元本込の割合 = 年利の割合 + 1

Out[45]= {1.001, 1.003, 1.005, 1.007, 1.009}
```

複利計算で10年後の元本比を計算し，計算結果から1を引き，100を掛けて小数からパーセントに直すことで10年間での利率を計算します．結果を見ると，かなり違うことがわかりますね．なお，何ら計算結果を保証するものではありませんので，あしからず．

```
In[46]:= TableForm[{年利, ((元本込の割合^10) - 1) * 100},
          TableHeadings → {{"年利", "10年間での利率"}, None}]
```

Out[46]//TableForm=

年利	0.1	0.3	0.5	0.7	0.9
10年間での利率	1.00451	3.04083	5.11401	7.22467	9.37339

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

数のリストから目当てのものを捜し出そう

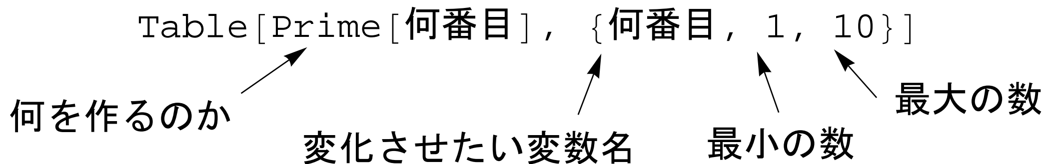
■ 素数を探そう

素数とは「1とその数自身以外に正の約数を持たない （つまり1とその数以外のどんな自然数によっても割り切れない），1より大きな自然数」のことです．具体的に素数を探してみましょう．*Mathematica*では「n番目の素数を教えなさい」という意味の命令「Prime」を使うことで，簡単に素数を知ることが出来ます．例えば，小さい順に10番目の素数は次のように求められます．

```
In[47]:= Prime[10]
Out[47]= 29
```

作文のときに使った組込み関数のTableを使うことで，より簡単に素数を調べることが出来ます．以前とは異なり，Primeには何番目の素数を知りたいのかを指定する必要があるので，Tableの使い方も以前とは変わってきます．例えば，1番目から10番目までの素数を知りたい場合は，次のように，何番目という変数を1から10まで変化させつつ「Prime[何番目]」を評価することが必要になります．

```
In[48]:= Table[Prime[何番目], {何番目, 1, 10}]
Out[48]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```



リストの作成：同じ仕組みでたくさん作るには

もちろん，いままでに登場した他の組込み関数を使っても同じことは可能です．例えば，次の例はMapとRangeを組み合わせた方法です．

```
In[49]:= Map[Prime, Range[10]]
Out[49]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

```
In[50]:= Map[Prime[#] &, Range[10]]
Out[50]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

*Mathematica*にはもっと便利な方法も用意されています．まずは，Primeという命令の属性を「関数の属性を調べよ」という意味の「Attributes」なる命令で調べましょう．すると，「リストを自動展開して，個々の要素に対して関数を適用する」という属性である「Litable」をPrimeが持っていることがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[51]:= **Attributes[Prime]**

Out[51]= {Listable, Protected}

即ち，次のように命令しても，1番目から10番目までの素数を知ることが出来ます．

In[52]:= **Prime[Range[10]]**

Out[52]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}

ところで本当に「29」は素数でしょうか．素数であれば「1とその数以外のどんな自然数によっても割り切れない」はずですから，本当に割り切れないか調べましょう．29よりも大きな数で割り切れないことは明らかですから，1から29までの整数で割ってみましょう．確かに，1と29以外の数では割り切れていません．

In[53]:= **29 / Range[29]**

Out[53]= $\left\{29, \frac{29}{2}, \frac{29}{3}, \frac{29}{4}, \frac{29}{5}, \frac{29}{6}, \frac{29}{7}, \frac{29}{8}, \frac{29}{9}, \frac{29}{10}, \frac{29}{11}, \frac{29}{12}, \frac{29}{13}, \frac{29}{14}, \frac{29}{15}, \frac{29}{16}, \frac{29}{17}, \frac{29}{18}, \frac{29}{19}, \frac{29}{20}, \frac{29}{21}, \frac{29}{22}, \frac{29}{23}, \frac{29}{24}, \frac{29}{25}, \frac{29}{26}, \frac{29}{27}, \frac{29}{28}, 1\right\}$

では「14351」はどうでしょうか．全部で14351個も数が表示されることになるので，*Mathematica*が気を利かせて，一部分だけの出力をしてくれています．表示を増やしながら割り切れているかを確認しても良いのですが，非常に面倒です．

In[54]:= **14 351 / Range[14 351]**

Out[54]=

非常に大きな出力が生成されました．以下はそのサンプルです．

$\left\{14\,351, \frac{14\,351}{2}, \frac{14\,351}{3}, \frac{14\,351}{4}, \frac{14\,351}{5}, \frac{14\,351}{6}, \frac{14\,351}{7}, \frac{14\,351}{8}, \frac{14\,351}{9}, \frac{14\,351}{10}, \frac{14\,351}{11}, \ll 14\,329 \gg, \frac{14\,351}{14\,341}, \frac{14\,351}{14\,342}, \frac{14\,351}{14\,343}, \frac{14\,351}{14\,344}, \frac{14\,351}{14\,345}, \frac{14\,351}{14\,346}, \frac{14\,351}{14\,347}, \frac{14\,351}{14\,348}, \frac{14\,351}{14\,349}, \frac{14\,351}{14\,350}, 1\right\}$

表示を少なく | もっと表示 | 全出力を表示 | 大きさ制限の設定...

このような場合は，「14351/Range[14351]」の結果であるリストから知りたい情報だけを抜き出して表示させた方が便利です．割り切れていれば整数で，割り切れていなければ分数ですから，整数だけを抜き出すことにしましょう．

整数だけの抜き出しには，「次の条件を満たすものだけをリストから抜き出しなさい」という意味の「Select」なる関数と，「整数かどうか調べなさい」という意味の

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

「IntegerQ」を組み合わせます．結果を見ると，1と14351以外に127と113でも割り切れていますから，14351は素数でないことがわかります．

```
In[55]:= Select[14 351 / Range[14 351], IntegerQ]

Out[55]= {14 351, 127, 113, 1}
```

Select[調べたいリスト，抽出したい条件]

リストの名称やリストそのもの

主な例としては NumberQ NumericQ

リストからの選出：条件を満たす要素だけを取り出す

条件として使える関数には様々なものがあります．必ずしも使えるとは限りませんが，命令の最後に「Q」が付いているものは，*Mathematica*に何かを問い合わせる命令なので，条件として使えるも野が多くあります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[56]:= `? System`*Q`

▼ System`

AlgebraicIntegerQ	MatrixQ
AlgebraicUnitQ	MemberQ
ArgumentCountQ	NameQ
ArrayQ	NumberQ
AtomQ	NumericQ
CoprimeQ	OddQ
DigitQ	OptionQ
DistributionDomainQ	OrderedQ
DistributionParameterQ	PartitionsQ
EllipticNomeQ	PolynomialQ
EvenQ	PositiveDefiniteMatrixQ
ExactNumberQ	PossibleZeroQ
FreeQ	PrimePowerQ
HermitianMatrixQ	PrimeQ
HypergeometricPFQ	QuadraticIrrationalQ
InexactNumberQ	RootOfUnityQ
IntegerQ	SameQ
IntervalMemberQ	SquareFreeQ
InverseEllipticNomeQ	StringFreeQ
LegendreQ	StringMatchQ
LetterQ	StringQ
LinkConnectedQ	SyntaxQ
LinkReadyQ	TensorQ
ListQ	TrueQ
LowerCaseQ	UnsameQ
MachineNumberQ	UpperCaseQ
MatchLocalNameQ	ValueQ
MatchQ	VectorQ

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

双子素数，三つ子素数，四つ子素数を探そう

素数についてもっと調べていきましょう．素数の中には中の良い双子のような数もあり，「差が2の二つの素数の組」のことを双子素数といいます．双子素数の例としては，「3と5」や「11と13」などがあります．

*Mathematica*を使って双子素数を捜し出す方法はいろいろとありますが，ここでは比較的単純で簡単な方法を使ってみます（もっと良い方法もありますが，もっと詳しくなってからにしましょう）．まずは，双子素数の候補となる隣り合った素数のペアを作らなければいけませんが，次のように「素数同士の差」も含めたグループにしておきます．

```
In[57]:= 差と素数のペア =
          Table[{Prime[n + 1] - Prime[n], Prime[n], Prime[n + 1]},
           {n, 1, 20}]

Out[57]= {{1, 2, 3}, {2, 3, 5}, {2, 5, 7}, {4, 7, 11},
          {2, 11, 13}, {4, 13, 17}, {2, 17, 19}, {4, 19, 23},
          {6, 23, 29}, {2, 29, 31}, {6, 31, 37}, {4, 37, 41},
          {2, 41, 43}, {4, 43, 47}, {6, 47, 53}, {6, 53, 59},
          {2, 59, 61}, {6, 61, 67}, {4, 67, 71}, {2, 71, 73}}
```

双子素数であれば，赤字の部分で求めさせた「素数同士の差」は2になっているはずです．そこで，「パターンにマッチする要素を取り出しなさい」という意味の「Cases」なる命令で，双子素数を含むグループのみを取り出します．Casesは以前に出てきた文字列用の同じような関数StringCasesの拡張バージョンと思ってください．

```
In[58]:= 差と双子素数のグループ = Cases[差と素数のペア, {2, ___}]

Out[58]= {{2, 3, 5}, {2, 5, 7}, {2, 11, 13}, {2, 17, 19},
          {2, 29, 31}, {2, 41, 43}, {2, 59, 61}, {2, 71, 73}}
```

{2, ___}

「{」でリストの開始

「}」でリストの終了

最初の要素は「2」

下線3つ「___」は「何でもOK」の意味

リストのパターン：最初の要素が「2」のリスト

この段階で，Casesにより双子素数を含むグループだけを選別できましたが，いまとなっては余計な「素数同士の差」も含まれてしまっています．これを取り除かないと双子素数のペアにはなりません．要素の削除には「Drop」という命令を使います．この意味は「リストの最初の要素から指定した個数分だけ削除しなさい」です．例えば，「{2,3,5}」から最初の1つを削除し「{3,5}」にするには次のようにします．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[59]:= Drop[{2, 3, 5}, 1]

Out[59]= {3, 5}
```

Drop[リスト, n]

要素の一部を削除する対象のリスト

最初のいくつを削除するか

1なら最初のひとつ
2なら最初のふたつ

リストから要素を削除：最初のいくつかを削除

従って，DropをMapと組み合わせて使うことで，個々のグループから余計な部分を削ぎ落とした，次のような双子素数のペアのリストを捜し出すことが出来ます．この結果を見る限りでは，双子素数はたくさんありますね．

```
In[60]:= Map[Drop[#, 1] &, 差と双子素数のグループ]

Out[60]= {{3, 5}, {5, 7}, {11, 13}, {17, 19},
          {29, 31}, {41, 43}, {59, 61}, {71, 73}}
```

双子素数を探した方法を少し変えることで，三つ子素数と呼ばれる「差が2である3つの素数の組」も見付けることが出来ます．三つ子素数の例としては「3と5と7」です．双子素数の場合と異なるのは，赤字の部分の差を隣り合ったふたつのペアそれぞれで計算していること，三つ子なので素数のトリオになっていることです．

```
In[61]:= 差と素数のトリオ =
          Table[{Prime[n + 2] - Prime[n + 1], Prime[n + 1] - Prime[n],
            Prime[n], Prime[n + 1], Prime[n + 2]}, {n, 1, 20}]

Out[61]= {{2, 1, 2, 3, 5}, {2, 2, 3, 5, 7}, {4, 2, 5, 7, 11},
          {2, 4, 7, 11, 13}, {4, 2, 11, 13, 17}, {2, 4, 13, 17, 19},
          {4, 2, 17, 19, 23}, {6, 4, 19, 23, 29},
          {2, 6, 23, 29, 31}, {6, 2, 29, 31, 37}, {4, 6, 31, 37, 41},
          {2, 4, 37, 41, 43}, {4, 2, 41, 43, 47}, {6, 4, 43, 47, 53},
          {6, 6, 47, 53, 59}, {2, 6, 53, 59, 61}, {6, 2, 59, 61, 67},
          {4, 6, 61, 67, 71}, {2, 4, 67, 71, 73}, {6, 2, 71, 73, 79}}
```

この個々のリストの中から，最初の2つの差がどちらも2になっているグループを捜し出せば良いので，Casesで指定するパターンは2がひとつ増え，DropとMapで削ぎ落とす余計な部分は最初の2つに変わります．結果は，「3と5と7」の組しか見付かりませんでしたが，Tableの20を増やしてもこれしか見付かりません．実は，三つ子素数はこの一組しか存在しないのです．不思議ですね．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[62]:= Map[Drop[#, 2] &, Cases[差と素数のトリオ, {2, 2, ___}]]

Out[62]= {{3, 5, 7}}
```

同じように，四つ子素数と呼ばれる「n，n+2，n+6，n+8がすべて素数であるような数の組」も見付けることが出来ます．これまでと異なるのは，赤字の部分の差を隣り合った三つのペアそれぞれで計算していること，四つ子なので素数のカルテットになっていることです．

```
In[63]:= 差と素数のカルテット =
          Table[{Prime[n + 3] - Prime[n + 2],
            Prime[n + 2] - Prime[n + 1], Prime[n + 1] - Prime[n],
            Prime[n], Prime[n + 1], Prime[n + 2], Prime[n + 3]},
           {n, 1, 20}]

Out[63]= {{2, 2, 1, 2, 3, 5, 7}, {4, 2, 2, 3, 5, 7, 11},
          {2, 4, 2, 5, 7, 11, 13}, {4, 2, 4, 7, 11, 13, 17},
          {2, 4, 2, 11, 13, 17, 19}, {4, 2, 4, 13, 17, 19, 23},
          {6, 4, 2, 17, 19, 23, 29}, {2, 6, 4, 19, 23, 29, 31},
          {6, 2, 6, 23, 29, 31, 37}, {4, 6, 2, 29, 31, 37, 41},
          {2, 4, 6, 31, 37, 41, 43}, {4, 2, 4, 37, 41, 43, 47},
          {6, 4, 2, 41, 43, 47, 53}, {6, 6, 4, 43, 47, 53, 59},
          {2, 6, 6, 47, 53, 59, 61}, {6, 2, 6, 53, 59, 61, 67},
          {4, 6, 2, 59, 61, 67, 71}, {2, 4, 6, 61, 67, 71, 73},
          {6, 2, 4, 67, 71, 73, 79}, {4, 6, 2, 71, 73, 79, 83}}
```

この個々のリストの中から，最初の3つの差が2,4,2になっているグループを捜し出せば良いので，Casesで指定するパターンは「{2,4,2,___}」となり，DropとMapで削ぎ落とす余計な部分は最初の3つに変わります．

```
In[64]:= Map[Drop[#, 3] &, Cases[差と素数のカルテット, {2, 4, 2, ___}]]

Out[64]= {{5, 7, 11, 13}, {11, 13, 17, 19}}
```

> 【BlankNullSequence（___）】*Mathematica*ではパターンマッチが非常に重要な役割を担っています．今回のように特定の形にマッチするものを抜き出したり，以前の例のように特定のパターンの文字列だけを検出することは，多くの実際的な作業において頻出します．その中でもアンダースコアを複数組み合わせたパターンは重要で，いわゆるワイルドカード，何にでもマッチするジョーカーです．詳細はまだ説明しませんが，アンダースコアの数は重要な意味を持つので，入力間違いには注意してください．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

- **完全数や友愛数を探そう**

素数に限らず整数には関係の深いペアが存在します．ここでは非常に有名な「完全数」と「友愛数」について取り上げたいと思います．完全数とは「その数自身を除く約数の和が，その数自身と等しい自然数」のことを指します．例えば，6の約数は1と2と3と6で，それ自身である6を除いて足すと1+2+3=6となるので，6は完全数です．

*Mathematica*に約数を求めさせるには「Divisors」という命令を使います．この意味は「指示した数の約数を全て求めなさい」です．実際に，6の約数を求めてみましょう．きちんと，1と2と3と6をリストで報告してくれます．

```
In[65]:= Divisors[6]

Out[65]= {1, 2, 3, 6}
```

この約数のリストからその数自身を除いて和を計算しなければ，完全数であるかの確認が出来ません．その数自身は常にリストの最後尾に出てくる（大きさの順に並んでいるので，最後になる）ので，双子素数のところで使った「Drop」を応用して，最後の要素だけを削除します．Dropに指示する整数が負の数の場合，リストの最後尾から指示した個数だけ要素を削除できます．最後尾だけを削除する場合は「-1」を指定します．

```
In[66]:= Drop[Divisors[6], -1]

Out[66]= {1, 2, 3}
```

Drop[リスト, -n]

要素の一部を削除する対象のリスト

最後のいくつを削除するか

-1なら最後のひとつ
-2なら最後のふたつ

リストから要素を削除：最後尾のいくつかを削除

その数自身をリストから削除できたので，あとはこのリストに含まれる数の総和が元の数になっていれば，完全数ということになります．これには「リストに含まれる要素の合計を計算しなさい」という意味をもつ「Total」なる関数を使います．すると，次のように元の数である6になるので，6は完全数であることを確認出来ます．

```
In[67]:= Total[Drop[Divisors[6], -1]]

Out[67]= 6
```

それでは完全数を探してみましょう．まずは，次のように整数とその約数の和をペアにして，完全数の候補のリストを作ります．1が完全数でないことは明らかですから，1は外して2から30までを候補として作っています．もちろん，30よりも大きな数まで作っても良いでしょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[68]:= 整数と約数の和 = Table[{n, Total[Drop[Divisors[n], -1]]},
           {n, 2, 30}]

Out[68]= {{2, 1}, {3, 1}, {4, 3}, {5, 1}, {6, 6}, {7, 1}, {8, 7},
          {9, 4}, {10, 8}, {11, 1}, {12, 16}, {13, 1}, {14, 10},
          {15, 9}, {16, 15}, {17, 1}, {18, 21}, {19, 1}, {20, 22},
          {21, 11}, {22, 14}, {23, 1}, {24, 36}, {25, 6},
          {26, 16}, {27, 13}, {28, 28}, {29, 1}, {30, 42}}
```

もちろん，この操作はMapを使っても同じです．Tableとの違いを見比べると理解がより深まります．

```
In[69]:= 整数と約数の和 = Map[{#, Total[Drop[Divisors[#], -1]]} &,
            Range[2, 30]];
```

さて，完全数であるならば，整数とその約数の和は同じになっているはずです．そこで，ペアを構成する数が同じものをCasesで取り出したいのですが，これまでとは異なるパターンが必要になります．まず，アンダースコア1つ「_」で「何でもOKだが，ひとつだけ」という意味を持ちます．アンダースコア3つ「___」の場合と異なり，ひとつにしかマッチしないことに注意してください．

さらに，今回の場合，ペアの数が同じになっていなければいけません．パターンを「{_,_}」としてしまうと，同じ数でなくてもマッチしてしまうことになります．そこで，*Mathematica*に追加指示として，各アンダースコアに名前を付けることにします．すると，同じ名前のアンダースコアは同じものを表すことになるので，「{m_,m_}」は「何だかわからないけど，同じものふたつからなるリスト」を表します．従って，次のようにすることで完全数のペアを取り出すことが出来ます．

```
In[70]:= Cases[整数と約数の和, {m_, m_}]

Out[70]= {{6, 6}, {28, 28}}
```

「{」でリストの開始 → {m_, m_} ← 「}」でリストの終了

同じ名前は同じもの

下線1つ「_」は「何でもOKだけど，1つだけ」

リストのパターン：最初の要素が「2」のリスト

より洗練された結果が欲しい場合は，「リストの最初の要素だけを取り出しなさい」という意味の「First」を使うことで，重複のない完全数のみのリストに出来ます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[71]:= Map[First, Cases[整数と約数の和, {m_, m_}]]

Out[71]= {6, 28}
```

完全数を探す方法を少しだけ工夫することで，友愛数も見付けることが出来ます．友愛数は親和数とも呼ばれ，「異なる2つの自然数の自分自身を除いた約数の和が，互いに他方と等しくなるような数」のことを指します．例えば，220と284は友愛数です．まずは，これを確認してみましょう．

220の約数の和を求めさせるには，完全数のときと同じくDivisors，Drop，Totalを使えば良いので，次のようになります．そして結果の「約数の和」に対して同じ操作をして220に戻れば良いので，引き続いてその操作を行っています．結果は220となっており，220と284は友愛数であることを確認できました．

```
In[72]:= 約数の和 = Total[Drop[Divisors[220], -1]]

Out[72]= 284

In[73]:= Total[Drop[Divisors[約数の和], -1]]

Out[73]= 220
```

なお，この操作を一度で済まそうとすると，次のようになりますね．

```
In[74]:= Total[Drop[Divisors[
            Total[Drop[Divisors[220], -1]]
          ], -1]]

Out[74]= 220
```

では実際に友愛数を見付けてみましょう．完全数のときは，整数と約数の和のペアのリストを作って，そこから整数と約数の和が同じものを選んできました．今回も同じ方法でアプローチしてみます．整数，その約数の和，さらに和の約数の和という3つの数から構成されるリストを作ります．友愛数であれば，最初の整数と最後の和が同じになっているはずです．友愛数はそう頻出しないので，2から30000までの整数を対象にします．セミコロンを付けて出力を抑制した方が良いかもしれませんね．

```
In[75]:= 整数と約数の和とその約数の和 =
           Table[{n, Total[Drop[Divisors[n], -1]],
             Total[Drop[Divisors[Total[Drop[Divisors[n], -1]]],
               -1]]}, {n, 2, 30000}];
```

完全数のときと同じように，友愛数の候補を「{m_, _, m_}」というパターンで取り出します．完全数のときとは異なり，リストが長くなっている分，パターンの間にアンダースコアがひとつ挿入しています．

```
In[76]:= 友愛数候補 = Cases[整数と約数の和とその約数の和, {m_, _, m_}]

Out[76]= {{6, 6, 6}, {28, 28, 28}, {220, 284, 220},
  {284, 220, 284}, {496, 496, 496}, {1184, 1210, 1184},
  {1210, 1184, 1210}, {2620, 2924, 2620},
  {2924, 2620, 2924}, {5020, 5564, 5020},
  {5564, 5020, 5564}, {6232, 6368, 6232},
  {6368, 6232, 6368}, {8128, 8128, 8128},
  {10744, 10856, 10744}, {10856, 10744, 10856},
  {12285, 14595, 12285}, {14595, 12285, 14595},
  {17296, 18416, 17296}, {18416, 17296, 18416}}
```

と，良く見ると完全数が混ざってしまっています．完全数の場合，その約数の和も，その和の約数の和も，元の数になりますから友愛数候補として検出されてしまっているのです．そこで，Casesの逆操作を行う命令「DeleteCases」を使い余計な完全数を排除してしまいましょう．この命令の意味は「パターンにマッチする要素をリストから削除しなさい」です．指定するパターンは「全て同じ要素」を意味する「{m_, m_, m_}」です．

```
In[77]:= 友愛数の真の候補 = DeleteCases[友愛数候補, {m_, m_, m_}]

Out[77]= {{220, 284, 220}, {284, 220, 284},
  {1184, 1210, 1184}, {1210, 1184, 1210},
  {2620, 2924, 2620}, {2924, 2620, 2924},
  {5020, 5564, 5020}, {5564, 5020, 5564},
  {6232, 6368, 6232}, {6368, 6232, 6368},
  {10744, 10856, 10744}, {10856, 10744, 10856},
  {12285, 14595, 12285}, {14595, 12285, 14595},
  {17296, 18416, 17296}, {18416, 17296, 18416}}
```

このままでは少し見づらいので，個々の要素数が3のリストをUnionを使って重複をなくして，友愛数のペアにしましょう（赤字の部分の命令に相応）．さらに，友愛数のペアのそれぞれに対して友愛数のペアが作られてしまっているので，この重複もUnionを使ってなくすと，次のように友愛数のペアが8個できあがります．

```
In[78]:= Union[Map[Union, 友愛数の真の候補]]

Out[78]= {{220, 284}, {1184, 1210}, {2620, 2924},
  {5020, 5564}, {6232, 6368}, {10744, 10856},
  {12285, 14595}, {17296, 18416}}
```

【TableとRangeとMap】*Mathematica*では，ある特定の性質を満たすリスト（今回はたくさんのそういうリストを作りました）を準備する場合，いろいろな方法が選べます．大抵の場合，TableとRangeとMapのどの方法でも同じものを作れますので，自

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

分の分かり易い方法を選ぶようにしましょう．ただ，今後はどれかでしか出来ない操作も出てきますので，なるべく比較検討しておく方が勉強にはなります．

■ **円周率の中に誕生日を捜し出そう**

素数の他にも不思議な数はたくさんあります．ここでは，円周率πと自然対数の底eを扱いたいと思います．突然ですが，円周率を何桁まで覚えているでしょうか．もしくは何桁まで覚えたいでしょうか．*Mathematica*に「指定した桁数まで小数に直しなさい」という意味を持つ命令「N」を使うことで，円周率を何桁まででも知ることが出来ます．例えば，100桁まで円周率を知りたい場合は次のように命令します．円周率は「Pi」または「ESCpiESC」と入力するか，パレットから選択して入力します．

```
In[79]:= N[Pi, 100]

Out[79]= 3.14159265358979323846264338327950288419716939937510582\
         0974944592307816406286208998628034825342117068
```

同じように，自然対数の底についてもNを使うことで，何桁まででも知ることが出来ます．同様に，100桁まで自然対数の底を知りたい場合は次のように命令します．自然対数の底は「E」または「ESCeeESC」と入力するか，パレットから選択して入力します．

```
In[80]:= N[E, 100]

Out[80]= 2.71828182845904523536028747135266249775724709369995957\
         4966967627724076630353547594571382178525166427
```

この組込み関数「N」は非常に便利な命令で，次のように様々な厳密な数を，より直感的に大きさを把握することのできる小数に変換してくれます．

In[81]:= $N\left[\left\{\frac{12}{29}, \frac{12}{65}, \frac{40}{41}, \frac{83}{208}, \frac{5}{34}, \frac{18}{41}, \frac{7}{153}, \frac{17}{58}, \frac{12}{35}, \frac{16}{27}\right\}\right]$

```
Out[81]= {0.413793, 0.184615, 0.97561, 0.399038, 0.147059,
          0.439024, 0.0457516, 0.293103, 0.342857, 0.592593}
```

「N」の逆変換（小数を分数に変換）させるための命令も用意されており，英語で有理化を意味する「Rationalize」という命令になります．

```
In[82]:= Rationalize[0.5]
```

Out[82]= $\frac{1}{2}$

少し横道にそれましたが，円周率も自然対数の底も無限小数（小数点以下，延々と無限に桁が続く小数）です．実に様々な数字が小数点以下に表れてきます．もしかすると誕生日と同じ数字の並びがあるかもしれません．それを探してみましょう．

数字の並びを探すには，文字列として捜し出す方が簡単です．まずは「次の数を文字列に変換しなさい」という意味をもつ「ToString」で，1万桁程度の円周率を文字列

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

にしておきましょう．1万桁分の円周率を表示するには何ページか必要なので，ここではセミコロンを付けて出力を抑制しています．

```
In[83]:= 円周率の文字列 = ToString[N[Pi, 10 000]];
```

文字列の中から特定の文字列を探し出すには，早口言葉のところで使った，StringCount，StringCasesと似ている命令「StringPosition」を使います．この命令の意味は「次の文字列は何文字目から何文字目までに表れるか調べなさい」です．従って，2月29日生まれの人の場合，次のようにすることで円周率の中に誕生日と同じ数字「0229」が表れるかを調べることが出来ます．

```
In[84]:= StringPosition[円周率の文字列, "0229"]

Out[84]= {{6708, 6711}, {9227, 9230}}
```

結果は「{6708, 6711}」と「{9227, 9230}」になっています．円周率の文字列の最初には「3.14」と小数点以下でない文字が2文字含まれているので，2月29日生まれの誕生日は，円周率の小数点以下6706桁目から6709桁目と，小数点以下9225桁目から9228桁目に表れることがわかります．

同じように自然対数の底についても調べてみると，1万桁分の中には存在しないので，3万桁分まで調べたところ，小数点以下22438桁目から22441桁目に表れることがわかります．他にもいろいろな数字が隠れているかもしれません．面白いですね．

```
In[85]:= StringPosition[ToString[N[E, 10 000]], "0229"]

Out[85]= {}

In[86]:= StringPosition[ToString[N[E, 30 000]], "0229"]

Out[86]= {{22 440, 22 443}}
```

【厳密数と近似数】*Mathematica*では，小数の形で表される数のことを「近似数」，整数や分数のように小数の形でないものを「厳密数」と分類しています．円周率は組込み関数Nで小数に変換しなければ厳密数ですが，有限な小数で表現してしまうと正確には円周率とは言えない近似数になります．*Mathematica*は，厳密数同士の計算結果は厳密数で，近似数との計算結果は近似数で報告してきます．正確さか，分かり易さか，必要に応じて選んで計算するようにしてください．

切符の番号で10を作る遊びを再現してみよう

- **切符の番号で10を作る答えを探そう**

鉄道の切符には大抵4桁の番号が振ってあり，その数の四則演算で10を作るという遊びがあります．例えば，「3142」であれば「3-1+4×2」と演算子を補うことで10に出来ます．この遊びには様々なローカルルールがあるようなので，本書では次のようなルールを定めておきます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

数の順番を入れ換えてはいけない
2. 使ってよい演算は，加減乗除の4種類（累乗は禁止）
3. 同じ演算を何度使っても良い
4. 括弧は使用禁止
5. 使える演算子の数は3個（最初の文字の前には付けられない）

このようなルールの元，*Mathematica*に次の問題を解いてもらいましょう．

```
In[87]:= 問題 = {4, 3, 6, 2}

Out[87]= {4, 3, 6, 2}
```

*Mathematica*は計算機なので，全ての演算子の組合せに対して10になるかを確認してもらうという力業が出来ます．そこで，ルールに基づいて4種類から重複を許した3個の演算の組合せを全て求めます．「Tuples」という命令を使うと非常に簡単に作れます．この命令の意味は「リストの要素から重複を許して指定した個数になるような組合せを全て求めなさい」です．

従って，次のようにTuplesを使うことで演算の全ての組合せを求められます．なお，演算子を文字列として入力しないと，足し算やら引き算やらを実行しようとしてしまいエラーになってしまうので注意してください．

```
In[88]:= 演算の組合せ = Tuples[{"+", "-", "*", "/"}, 3]

Out[88]= {{+, +, +}, {+, +, -}, {+, +, *}, {+, +, /},
  {+, -, +}, {+, -, -}, {+, -, *}, {+, -, /}, {+, *, +},
  {+, *, -}, {+, *, *}, {+, *, /}, {+, /, +}, {+, /, -},
  {+, /, *}, {+, /, /}, {-, +, +}, {-, +, -}, {-, +, *},
  {-, +, /}, {-, -, +}, {-, -, -}, {-, -, *}, {-, -, /},
  {-, *, +}, {-, *, -}, {-, *, *}, {-, *, /}, {-, /, +},
  {-, /, -}, {-, /, *}, {-, /, /}, {*, +, +}, {*, +, -},
  {*, +, *}, {*, +, /}, {*, -, +}, {*, -, -}, {*, -, *},
  {*, -, /}, {*, *, +}, {*, *, -}, {*, *, *}, {*, *, /},
  {*, /, +}, {*, /, -}, {*, /, *}, {*, /, /}, {/, +, +},
  {/, +, -}, {/, +, *}, {/, +, /}, {/, -, +}, {/, -, -},
  {/, -, *}, {/, -, /}, {/, *, +}, {/, *, -}, {/, *, *},
  {/, *, /}, {/, /, +}, {/, /, -}, {/, /, *}, {/, /, /}}
```

演算子を文字列で表現しているので，問題に含まれる数も文字列に変換しておきます．文字列は文字列同士，数は数同士，揃えて扱わないと*Mathematica*も混乱してしまいます．

```
In[89]:= 問題文字列 = Map[ToString, 問題]

Out[89]= {4, 3, 6, 2}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

問題文字列と演算の組合せから，計算式を構成します．これには「要素を交互に組み合わせて1つのリストにしなさい」という意味をもつ「Riffle」なる命令が非常に便利です．例えば，演算の組合せ「{"+", "-", "*"}」に対しては次のようなリストが得られるので，StringJoinで1つの文字列にまとめると，1つの計算式が出来ることになります．

```
In[90]:= Riffle[{"4", "3", "6", "2"}, {"+", "-", "*"}]

Out[90]= {4, +, 3, -, 6, *, 2}
```

```
In[91]:= StringJoin[Riffle[{"4", "3", "6", "2"}, {"+", "-", "*"}]]

Out[91]= 4+3-6*2
```

このような変換を全ての演算の組合せに対して*Mathematica*にさせるには，何度も出てきているMapを次のように使えば良いことはすぐにわかると思います．

```
In[92]:= 計算式 = Map[StringJoin[Riffle[問題文字列, #]] &, 演算の組合せ]

Out[92]= {4+3+6+2, 4+3+6-2, 4+3+6*2, 4+3+6/2, 4+3-6+2,
  4+3-6-2, 4+3-6*2, 4+3-6/2, 4+3*6+2, 4+3*6-2,
  4+3*6*2, 4+3*6/2, 4+3/6+2, 4+3/6-2, 4+3/6*2, 4+3/6/2,
  4-3+6+2, 4-3+6-2, 4-3+6*2, 4-3+6/2, 4-3-6+2, 4-3-6-2,
  4-3-6*2, 4-3-6/2, 4-3*6+2, 4-3*6-2, 4-3*6*2, 4-3*6/2,
  4-3/6+2, 4-3/6-2, 4-3/6*2, 4-3/6/2, 4*3+6+2, 4*3+6-2,
  4*3+6*2, 4*3+6/2, 4*3-6+2, 4*3-6-2, 4*3-6*2, 4*3-6/2,
  4*3*6+2, 4*3*6-2, 4*3*6*2, 4*3*6/2, 4*3/6+2, 4*3/6-2,
  4*3/6*2, 4*3/6/2, 4/3+6+2, 4/3+6-2, 4/3+6*2, 4/3+6/2,
  4/3-6+2, 4/3-6-2, 4/3-6*2, 4/3-6/2, 4/3*6+2, 4/3*6-2,
  4/3*6*2, 4/3*6/2, 4/3/6+2, 4/3/6-2, 4/3/6*2, 4/3/6/2}
```

このままでは10になるか分からないので，*Mathematica*に計算させましょう．文字列になっている計算式を計算させるには，「文字列を数式と解釈して，計算を行いなさい」という意味の「ToExpression」を使います．計算式はリストで与えられているので，この場合もMapを次のように使います．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[93]:= **計算結果 = Map[ToExpression, 計算式]**

Out[93]= $\{15, 11, 19, 10, 3, -1, -5, 4, 24, 20, 40, 13, \frac{13}{2}, \frac{5}{2}, 5, \frac{17}{4}, 9, 5, 13, 4, -3, -7, -11, -2, -12, -16, -32, -5, \frac{11}{2}, \frac{3}{2}, 3, \frac{15}{4}, 20, 16, 24, 15, 8, 4, 0, 9, 74, 70, 144, 36, 4, 0, 4, 1, \frac{28}{3}, \frac{16}{3}, \frac{40}{3}, \frac{13}{3}, -\frac{8}{3}, -\frac{20}{3}, -\frac{32}{3}, -\frac{5}{3}, 10, 6, 16, 4, \frac{20}{9}, -\frac{16}{9}, \frac{4}{9}, \frac{1}{9}\}$

この中に10があれば問題を解く演算の組合せがあることになります．リストの中から10を探すには，StringPositionのリスト版である「Position」を使います．Positionの結果を「Extract」に指定することで，計算結果が10になった計算式を取り出すことが出来ます．「Extract」は「リストから指定した部分を取り出しなさい」という意味を持つ命令です．「{{4}, {57}}」は「リストの4番目と57番目」という部分指定になります．

In[94]:= **Position[計算結果, 10]**

Out[94]= {{4}, {57}}

In[95]:= **Extract[計算式, Position[計算結果, 10]]**

Out[95]= {4+3+6/2, 4/3*6+2}

まとめると，4362という問題は「4+3+6/2」ないしは「4/3*6+2」で10に出来ることがわかります．

> 【文字列と数式】*Mathematica*は，入力された数式を自動的に計算して見易く報告してくれます．従って，数式を計算せずにそのままの状態で扱うには，文字列として扱ったり，明示的に「評価を行うな」という補足指示を出す必要があります．この補足指示に興味のある方は，ドキュメントセンターで「Hold」を調べてみてください．

■ **切符の番号で10を作る問題を作ってみよう**

今度は逆に「10に出来る4桁の数字」を作ってみましょう．切符の番号の中には，どんなに頑張っても10に出来ない数（例えば，1000や2345）が存在しますが，事前に*Mathematica*で確認すれば，解の存在する問題を簡単に作ることが出来ます．

まず，演算の組合せは必要なので，前項に引き続いて定義しておきましょう．

In[96]:= **演算の組合せ = Tuples[{"+", "-", "*", "/"}, 3];**

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

問題の解き方を調べさせるときに準備した各種のリスト（問題候補，問題候補文字列，問題文，計算結果）を一括して作成できるように，遅延的に定義しておきます．問題候補自体は，何度も使っているRandomIntegerを使うことで簡単に作れます．ここでは，演算として割り算を使えるルールになっているので，意図的に問題候補から0を除外しています（0では割り算できないため）．

```
In[97]:= 下準備 := (問題候補 = RandomInteger[{1, 9}, 4];
   問題候補文字列 = Map[ToString, 問題候補];
   問題文 = Map[StringJoin[Riffle[問題候補文字列, #]] &,
     演算の組合せ];
   計算結果 = Map[ToExpression, 問題文])
```

この命令のように，*Mathematica*に出す指示はセミコロンで区切れば，複数の指示を組み合わせることが出来ます．全体を丸括弧で囲んでいるのは，この4つの命令を1つのまとまりとして「下準備」と名付けなさい，と*Mathematica*に指示するためです．括弧がないと，最初の命令のみが下準備になってしまいます．

次に，問題候補が解くことの出来る問題であるかをチェックする遅延的な定義を行いましょう．計算結果の中に10があるかをMemberQで問い合わせるだけでチェックが出来ます．結果は，真偽値であるTrue（解ける）かFalse（解けない）になります．

```
In[98]:= 問題候補は解ける？ := MemberQ[計算結果, 10]
```

問題候補が解ける場合に，問題文を抽出する遅延的な定義も行っておきましょう．

```
In[99]:= 問題の解き方 := Extract[問題文, Position[計算結果, 10]]
```

これらをまとめると，次のように問題作成を行ってくれる遅延的な定義を作れます．問題作成を呼び出すたびに，10にすることの出来る問題が解き方と共に報告されます．「If」という命令は以前にも出てきましたが，今回の場合，条件が満たされていれば「{問題候補，　　問題の解き方}」を，満たされていなければ「問題作成」を*Mathematica*は実行します．即ち，問題候補が解ける状態になるまで，延々と「遅延的な定義の問題作成」が何度も呼び出されることになります．

```
In[100]:= 問題作成 := (下準備;
   If[問題候補は解ける？, {問題候補, 問題の解き方}, 問題作成])

In[101]:= 問題作成

Out[101]= {{2, 8, 7, 7}, {2+8+7-7, 2+8-7+7, 2+8*7/7, 2+8/7*7}}

In[102]:= 問題作成

Out[102]= {{4, 4, 2, 2}, {4+4*2-2, 4*4/2+2}}
```

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

変数と式を使って数学の問題を解いてみよう

■ **未知の何かを見つけ出す**

中等教育で未知数について習うときに頻出する問題があります．それは「地球の半径が1m長くなったら，地球の周の長さは何m長くなるか」という問です．周の長さは「直径×円周率」ですから，地球の半径を「地球の半径」という言葉で表現することにすると，地球の周の長さは次のように書くことが出来ます．`Pi`は既に出てきていますが，円周率を表しています．もちろん，「ESCpiESC」により「π」と入力されても構いません．

```
In[103]:= 地球の周の長さ = 地球の半径 * 2 * Pi
Out[103]= 2 地球の半径 π
```

続いて，地球の半径が1m長くなった場合の周の長さも同様に計算してみましょう．丸括弧を使うことで次のように書くことが出来ますね．ここでも地球の半径は未知のままで，「地球の半径」という言葉で表現しているだけです．

```
In[104]:= 地球の半径が1m長くなった場合の周の長さ = (地球の半径 + 1) * 2 * Pi
Out[104]= 2 (1 + 地球の半径) π
```

どれだけ長くなるかを知るためには，「地球の半径が1m長くなった場合の周の長さ」から「地球の周の長さ」を引いて差を求める必要があります．実際に差を計算してみましょう．

```
In[105]:= 差 = 地球の半径が1m長くなった場合の周の長さ - 地球の周の長さ
Out[105]= -2 地球の半径 π + 2 (1 + 地球の半径) π
```

何やら複雑な式が出てきました．やはり，地球の半径がわからないと何m長くなるかを知ることは出来ないのでしょうか．そうではありません．*Mathematica*は人間から与えられた命令を忠実に実行する機械なので，数式を勝手に展開しないだけです．場合によっては展開してしまうと見づらくなることもあるからです．展開させたい場合には，「`Expand`」という命令を使います．この意味は「次の数式を出来るだけ展開しなさい」です．

```
In[106]:= Expand[差]
Out[106]= 2 π
```

結局，半径が1m長くなると周の長さは「2 π」m長くなることがわかりました．「2π」では具体的にどれくらいの長さなのか把握しづらい場合は，以前に出てきた組込み関数Nを使うことで，次のように約6.28mであることを知ることも出来ます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[107]:= `N[Expand[差]]`

Out[107]= 6.28319

この計算例に出てくる「地球の半径」のように，未知の何かを表すもの，等号を使って*Mathematica*に値を覚えさせるもの，を一般に「変数」と呼んでいます．一方，`Expand`や`N`のように*Mathematica*への命令のことを「関数」と呼びます．*Mathematica*では変数と関数に大きな違いはなく，どちらも「シンボル」と呼ぶこともあります．これらのシンボルの名前には，どのような名前も（英語でも，日本語でも，何文字でも）付けられますが，次のような制限があります．

1. シンボル名は半角の数字から始めてはいけない（シンボルの前にある数字はスカラー倍として認識されるため．例えば，「$2x$」など）
2. シンボル名に演算子やスペースなど，他に意味のある記号を使ってはいけない（それぞれの記号は別に用途があり名前とは認識されないため）
3. シンボル名を半角英数字にするときは，大文字から始めないことが望ましい（*Mathematica*の組込み関数は大文字から始まっており混乱を避けるため）

これらを踏まえて，似た問題を解いてみましょう．それは「太陽の周を1m長くするには，太陽の半径を何m長くすれば良いか」という問です．太陽の半径の長さは「太陽の周の長さ÷円周率÷2」ですから，太陽の周の長さを「太陽の周の長さ」という言葉で表現することにすると，太陽の半径は次のように書くことが出来ます

In[108]:= `太陽の半径 = 太陽の周の長さ / Pi / 2`

Out[108]= $\dfrac{\text{太陽の周の長さ}}{2\pi}$

太陽の周を1m長くした場合の太陽の半径も同様に計算してみましょう．

In[109]:= `太陽の周を1m長くした太陽の半径 = (太陽の周の長さ + 1) / Pi / 2`

Out[109]= $\dfrac{1+\text{太陽の周の長さ}}{2\pi}$

どれだけ長くする必要があるかを知るためには，「太陽の周を1m長くした太陽の半径」から「太陽の半径」を引いて差を求める必要があります．実際に差を計算してみましょう．

In[110]:= `差 = 太陽の周を1m長くした太陽の半径 - 太陽の半径`

Out[110]= $-\dfrac{\text{太陽の周の長さ}}{2\pi}+\dfrac{1+\text{太陽の周の長さ}}{2\pi}$

今回も同様に`Expand`で展開させると，太陽の周を1m長くするには，太陽の半径を「$\frac{1}{2\pi}$」m長くする必要があることがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[111]:= **Expand[差]**

Out[111]= $\frac{1}{2\pi}$

具体的な数でないと大きさが把握できないので，今回もNで小数に直しましょう．すると，約16cm半径を長くすると周の長さが1m長くなることがわかりました．

In[112]:= **N[Expand[差]]**

Out[112]= 0.159155

- **結局のところそれは何？，方程式を解いてみよう**

今度は次のような問題を考えてみよう．それは「1000円だけ財布に入れ，阪急六甲駅から神戸大学までバスで2往復しました．途中，缶珈琲が100円だったので2本買ったところ，ちょうど1000円を使いきりました．バスの片道運賃はいくらでしょうか」です．

バスの片道運賃を表すシンボル「バスの片道運賃」を未知数として方程式を立ててみましょう．使ったお金の合計を計算すると次のようになります．

バスの片道運賃 × 2 × 2 + 100 × 2

この金額が1000円に等しいので，方程式は次のようになります．

バスの片道運賃 × 2 × 2 + 100 × 2 = 1000

しかし，*Mathematica*にとって「=」は，リストや数に名前を付けて覚えておく「シンボルへの値の割り当て」の意味を持つため，左辺と右辺が等しい「方程式」として認識してくれません．*Mathematica*に方程式だと認識させる場合には，左辺と右辺が数学的に等しいことを表す「==」を使います（等号を2つ並べます）．従って，この問題の未知数が満たす関係式は次のように表すことになります．

バスの片道運賃 × 2 × 2 + 100 × 2 == 1000

では，バスの片道運賃はいくらでしょうか．せっかくですから，*Mathematica*に解いてもらいましょう．「次の方程式を解いて未知数を求めよ」という意味の命令は「Solve」になります．次のように指示することで，バスの片道運賃が200円であることがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[113]:= `Solve[バスの片道運賃 × 2 × 2 + 100 × 2 == 1000, バスの片道運賃]`

Out[113]= `{{バスの片道運賃 → 200}}`

Solve[左辺==右辺, 未知数]

解きたい方程式　　答えを知りたい未知数

方程式を解く：未知数の値を求めるには

Solveは係数にパラメータを持つ場合も答えを求めてくれますので，次のように解の公式なども調べることが出来ます．3次や4次の場合の解の公式も同様に求めることが出来ます（スペースの関係でここでは取り扱いませんが）．

In[114]:= `Solve[a x^2 + b x + c == 0, x]`

Out[114]= $\left\{\left\{x \to \frac{-b-\sqrt{b^2-4ac}}{2a}\right\}, \left\{x \to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$

では，もっと有名な鶴亀算も*Mathematica*に解いてもらいましょう．問題は「鶴と亀があわせて10匹，足の数があわせて28本であるとき，鶴と亀は何匹いるか．ただし，鶴の足は2本，亀の足は4本とする」です．

「鶴」と「亀」を未知数として次の連立方程式を作ることになりますね．

$$\begin{cases} \text{鶴} + \text{亀} = 10 \\ \text{鶴} \times 2 + \text{亀} \times 4 = 28 \end{cases}$$

まず，*Mathematica*に方程式だと認識させる場合には，左辺と右辺が数学的に等しいことを表す「==」を使わなければいけません．そして，連立方程式であることを示すには，方程式をリストで与える必要があります．従って，この鶴亀算の連立方程式は次のように指示をしなければいけません．

`{鶴 + 亀 == 10, 鶴 × 2 + 亀 × 4 == 28}`

あとは先ほどと同じく組込み関数のSolveで解いてもらえば良いのですが，今回は未知数が「鶴」と「亀」の2つあるので，未知数もリストで指定する必要があります．従って，次のように指示することで，鶴が6匹，亀が4匹であることがわかります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

In[115]:= `Solve[{鶴 + 亀 == 10, 鶴 × 2 + 亀 × 4 == 28}, {鶴, 亀}]`

Out[115]= `{{鶴 → 6, 亀 → 4}}`

In[116]:=

Solve[{左辺==右辺, ...}, {未知数, ...}]

解きたい連立方程式をリストで　　答えを知りたい未知数もリストで

Out[116]=

Solve[{左辺==右辺, ...}, {未知数, ...}]

解きたい連立方程式をリストで　　答えを知りたい未知数もリ

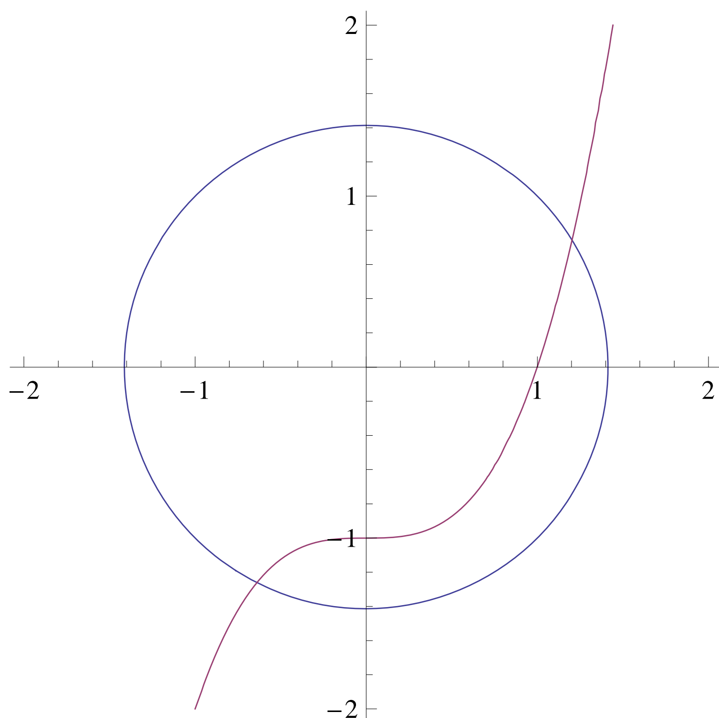
方程式を解く：複数の未知数の値を求めるには

> 【=と==と===】*Mathematica*では，等号を複数組み合わせて3つの異なる意味を持たせています．等号1つ「=」で「シンボルの定義」，等号2つ「==」で「数学的に等しい関係」，そして等号3つ「===」で「見た目に等しい関係」です．

- **結局のところそれは何？，もっと難しい方程式を解いてみよう**

今度は少し恣意的ですが，半径が2の円と3次曲線$y = x^3 - 1$の交点を求めてみましょう．即ち，次のような連立方程式を解かなければいけません．

$$\begin{cases} x^2 + y^2 = 2 \\ y = x^3 - 1 \end{cases}$$



そこで，前項で習ったSolveを使い，次のように*Mathematica*に解かせると，何やら見たことのない結果が表示されました．これらは何を意味するものでしょうか．

```
In[117]:= Solve[{x^2 + y^2 == 2, y == x^3 - 1}, {x, y}]

Out[117]= {{y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 1]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 1]},
   {y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 2]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 2]},
   {y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 3]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 3]},
   {y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 4]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 4]},
   {y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 5]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 5]},
   {y → -1 + Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 6]^3,
    x → Root[-1 + #1^2 - 2 #1^3 + #1^6 &, 6]}}
```

実は，ある程度次数の大きな方程式の解を，数学的な厳密さを失わずに表示するためには，代数的数と呼ばれる難しい概念を導入する必要があります．解の中に含まれる「Root」というのは，それを表現しているものです．厳密さを失っても良いのでわかりやすい形で解を知りたい場合は，Nで数値化させます．

```
In[118]:= N[%]

Out[118]= {{y → -1.26141, x → -0.639404}, {y → 0.742818, x → 1.20342},
   {y → 1.74449 - 0.524192 ⅈ, x → -0.779525 - 1.17308 ⅈ},
   {y → 1.74449 + 0.524192 ⅈ, x → -0.779525 + 1.17308 ⅈ},
   {y → -1.48519 - 0.213862 ⅈ, x → 0.497517 - 0.638423 ⅈ},
   {y → -1.48519 + 0.213862 ⅈ, x → 0.497517 + 0.638423 ⅈ}}
```

Solveで解を求めてからNで数値化（小数化）するのは面倒ですね．でも安心してください．*Mathematica*への命令には「NSolve」というのもあり，これは「数学的な厳密さは多少失っても良いから，わかりやすい小数の形で解を求めなさい」という意味なので，次のように非常にわかりやすい解を直接求めてくれます．なお，結果を「解」という名前で覚えさせておきましょう．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[119]:= 解 = NSolve[{x^2 + y^2 == 2, y == x^3 - 1}, {x, y}]

Out[119]= {{x → -0.779525 + 1.17308 ⅈ, y → 1.74449 + 0.524192 ⅈ},
  {x → -0.779525 - 1.17308 ⅈ, y → 1.74449 - 0.524192 ⅈ},
  {x → 0.497517 + 0.638423 ⅈ, y → -1.48519 + 0.213862 ⅈ},
  {x → 0.497517 - 0.638423 ⅈ, y → -1.48519 - 0.213862 ⅈ},
  {x → -0.639404, y → -1.26141}, {x → 1.20342, y → 0.742818}}
```

この解を改めて良く見ると，右矢印（→）が多用されています．この右矢印はオプション（捕捉指示）のときにも利用しましたが，*Mathematica*は「左側のものを右側に対応させる」という意味を持つ「置換規則（変換規則）」として認識します．

なぜ，*Mathematica*は方程式の解を置換規則で報告してくるのでしょうか．このことを理解するためには，「/.」という記号で行える操作「左の式に含まれる変数を，右側の置換規則で置き換えなさい」を理解する必要があります．

「$x+y$」という式の「x」を「3」に置き換えたい（xに3を代入したい）場合，次のように「/.」の左に式を，右側に置換規則「$x \to 3$」を書きます．*Mathematica*は右側の置換規則に基づいて，左側に含まれている変数を全て置き換えます．

```
In[120]:= x + y /. x → 3

Out[120]= 3 + y
```

同時に「y」も「z」に置き換えたい場合は，置換規則をリストで指定します．

```
In[121]:= x + y /. {x → 3, y → z}

Out[121]= 3 + z
```

置換規則をリストのリストで与えると，次のように最初の置換規則の組の代入結果と，次の置換規則の組の代入結果をリストで返してきます．

```
In[122]:= x + y /. {{x → 3, y → z}, {x → 4, y → w}}

Out[122]= {3 + z, 4 + w}
```

このような置換規則の仕組みを理解した上で，方程式の解を確認してみると，置換規則のリストのリストになっていることがわかると思います．すなわち，解の組ごとにリストになっているので，次のように「/.」を使って変数の置き換えをすることで，解の数のみからなるリストを得ることが出来ます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[123]:= 解のリスト = {x, y} /. 解

Out[123]= {{-0.779525 + 1.17308 i, 1.74449 + 0.524192 i},
           {-0.779525 - 1.17308 i, 1.74449 - 0.524192 i},
           {0.497517 + 0.638423 i, -1.48519 + 0.213862 i},
           {0.497517 - 0.638423 i, -1.48519 - 0.213862 i},
           {-0.639404, -1.26141}, {1.20342, 0.742818}}
```

元の問題ですが，グラフの交点は実数でなければいけないので，解のうち虚部が0でない複素数は関係ありません．従って，解のリストの中から実数だけを取り出す必要があります．このような場合，双子素数などの検出に使った組込み関数Casesを使って，実数のペアになっているリストだけを取り出します．

```
In[124]:= Cases[解のリスト, {_Real, _Real}]

Out[124]= {{-0.639404, -1.26141}, {1.20342, 0.742818}}
```

このパターン「{_Real, _Real}」の意味はなんでしょうか．アンダースコアの後ろにはヘッド（頭部）を指定することができ，「Real」とは「実数」を意味します．これ以外にも次のように頭部を調べることで，「Integer」で「整数」を，「Rational」で「分数」を，「Complex」で「複素数」（虚数単位は「I」で表す）を，「Symbol」で「シンボル（変数）」を表すことがわかります．

```
In[125]:= Map[Head, {1, 1.5, 2 / 3, 1 + I, Pi}]

Out[125]= {Integer, Real, Rational, Complex, Symbol}
```

【SolveとNSolve】*Mathematica*の組込み関数は，英語の単語になっている他，いくつかの機能の組合せになっている場合（今回のSolveとNSolveのように），単純に組み合わせたものが新たな機能の名称になっていることが多いです．ドキュメントセンターを確認したりしなくても，ある程度機能の内容や，必要な関数の名称がわかったりします．

- **使えるお金はいくら？，不等式を解いてみよう**

今度はもっと現実的な鶴亀算を解いてみましょう．例えば「鶴と亀があわせて9匹か10匹，足の数があわせて28本から32本くらいであるとき，鶴と亀は何匹いるか．ただし，鶴の足は2本，亀の足は4本とする」です．曖昧なので不等式になります．

「鶴」と「亀」を未知数として次の連立不等式を作ることになりますね．

$$\begin{cases} 9 \leq \text{鶴} + \text{亀} \leq 10 \\ 28 \leq \text{鶴} \times 2 + \text{亀} \times 4 \leq 32 \end{cases}$$

*Mathematica*に連立不等式であることを示すには，連立方程式と同じくリストで与える必要があります．従って，この鶴亀算の連立不等式は次のように指示をしなければいけません．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

```
In[126]:= {9 ≤ 鶴 + 亀 ≤ 10, 28 ≤ 鶴×2 + 亀×4 ≤ 32}

Out[126]= {9 ≤ 亀 + 鶴 ≤ 10, 28 ≤ 4 亀 + 2 鶴 ≤ 32}
```

不等式はSolveやNSolveを使って指示を出しても方程式でないので*Mathematica*は解いてくれません．不等式を解かせるには，「Reduce」という命令を使います．この命令の意味は「方程式や不等式を解き，変数に関する条件を簡約しなさい」です．使い方は基本的にSolveと同じです．

```
In[127]:= Reduce[{9 ≤ 鶴 + 亀 ≤ 10, 28 ≤ 鶴×2 + 亀×4 ≤ 32}, {鶴, 亀}]
```

$$\text{Out[127]= } (鶴 == 2 \;\&\&\; 亀 == 7) \;||\; \left(2 < 鶴 \le 4 \;\&\&\; 9 - 鶴 \le 亀 \le \frac{16 - 鶴}{2}\right) \;||\; \left(4 < 鶴 < 6 \;\&\&\; \frac{14 - 鶴}{2} \le 亀 \le 10 - 鶴\right) \;||\; (鶴 == 6 \;\&\&\; 亀 == 4)$$

この報告を見ると，鶴が2匹で亀が7匹，鶴が6匹で亀が4匹という答えは分かりますが，他にも何やら複雑な式が出ていて良く分かりません．このような場合は，Reduceの最後に「鶴」と「亀」は整数だと思って解きなさい，という付加的な命令「Integers」を付け加えます．そうすると，次のように全ての可能性のある解を得ることが出来ます．

```
In[128]:= Reduce[{9 ≤ 鶴 + 亀 ≤ 10, 28 ≤ 鶴×2 + 亀×4 ≤ 32}, {鶴, 亀},
           Integers]

Out[128]= (鶴 == 2 && 亀 == 7) || (鶴 == 3 && 亀 == 6) || (鶴 == 4 && 亀 == 5) ||
           (鶴 == 4 && 亀 == 6) || (鶴 == 5 && 亀 == 5) || (鶴 == 6 && 亀 == 4)
```

SolveやNSolveとは異なり結果が置換規則でないことに注意してください．Reduceは指定されたシンボルの満たす条件を簡約するための命令であり，結果が規則でなく条件になっています．

【論理演算】*Mathematica*でも他のプログラミング言語と同じく，アンパサンド2つ（&&）で論理積（かつ）を，縦線2つ（||）で論理和（または）を表します．これらの論理演算は，通常の演算記号（+，-，*，/，^）と同じく*Mathematica*の至る所で使うことが出来ます．特に，条件判定などで役に立つと思います．

- **最大値と最小値，もっとも得な値を探し出す**

今度はもっと現実的な問題を解いてみましょう．例えば「私の貯金は10万円あります．この中から光熱費と食費，そして旅行費用を出すとした場合，旅行には最大いくら使えるでしょうか．なお，先月の金額から推定すると，光熱費は5,000円から6,500円，食費は25,000円から32,500円必要になります」のようなものです．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

最大いくら？という問題を*Mathematica*に解かせるためには，「Maximize」という命令を使います．これは「制約条件下で指定したものを最大化するにはどうすれば良いか調べよ」という意味です．今回の問題の場合，次のような制約条件下で「旅行費用」を最大化したいわけです．

$$\begin{cases} \text{光熱費} + \text{食費} + \text{旅行費用} = 100000 \\ 5000 \le \text{光熱費} \le 6500 \\ 25000 \le \text{食費} \le 32500 \end{cases}$$

Maximizeを使った指示の出し方は，SolveやReduceと同じです．最初にリストで制約条件などを書き，最後に変数のリストを書きます．ただし，制約条件のリストの最初の要素は，それらの制約下で最大にしたい式を書きます（今回の場合は旅行費用です）．従って，次のように*Mathematica*に指示を出すことになります．

```
In[129]:= Maximize[{旅行費用, 光熱費 + 食費 + 旅行費用 == 100 000,
             5000 ≤ 光熱費 ≤ 6500, 25 000 ≤ 食費 ≤ 32 500},
            {旅行費用, 光熱費, 食費}]

Out[129]= {70 000, {旅行費用 → 70 000, 光熱費 → 5000, 食費 → 25 000}}
```

結果を見ると，光熱費を5000円に，食費を25000円に抑えれば，旅行費用として70000円を出せることがわかりました．簡単な関係なので，わざわざ*Mathematica*に計算してもらわなくても暗算できるかもしれませんが，難しい問題では役に立つでしょう．

Maximize[{最大化したい式， 制約条件， ...}， {変数,...}]

方程式や不等式での制約条件

最大値を求めたい式なので等号や不等号は含まない

最大値を求める：式を最大化する変数の値は？

一方，逆に旅行費用を最小化したい場合は，「Minimize」という命令を使います．この命令はMaximizeの逆で，リストの最初の要素を最小化する変数の値を求めてくれます．

```
In[130]:= Minimize[{旅行費用, 光熱費 + 食費 + 旅行費用 == 100 000,
             5000 ≤ 光熱費 ≤ 6500, 25 000 ≤ 食費 ≤ 32 500,
             68 000 ≤ 旅行費用 ≤ 85 000}, {旅行費用, 光熱費, 食費}]

Out[130]= {68 000, {旅行費用 → 68 000, 光熱費 → 6500, 食費 → 25 500}}
```

この他，Solveに対するNSolveなどのように，NMaximizeやNMinimizeなどもあります．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

この章のまとめ

この章で学んだことを箇条書きでまとめておきます.

リスト：仲間に分けたり，一緒にまとめたり

1. リストには，中括弧（「{」と「}」）を使います.
2. リストの要素ごとの区切りには，カンマ（「,」）を使います.
3. 文字列は，二重引用符（ダブルクオーテーション：「"」）で囲みます.
4. SHIFTを押しながらENTERで指示を*Mathematica*に実行（評価）させます.
5. 等号（「=」）を使って*Mathematica*にシンボルの値を覚えさせます.
6. コンテキスト（文脈）を表すには，単一引用符（「`」）を使います.
7. 行の最後にセミコロン（「;」）を付けると結果が表示されません.
8. 疑問符（「?」）をシンボルに付けると知っていることを返します.
9. 「Intersection」は「グループに共通するものを調べよ」
10. 「Complement」は「リストから次のリストの要素を取り除きなさい」
11. 「Union」は「重複しないように双方の中身を併せなさい」
12. 共通部分「Intersection」と和集合「Union」は「⋂」と「⋃」でも可
13. 「AppendTo」は「指定したグループに新しい仲間を加えなさい」
14. 「MemberQ」は「あるグループに指定した項目が含まれているか」
15. 「はい」が「True」で，「いいえ」が「False」
16. リストはネストさせて，リストのリストなどを作ることが出来ます.
17. 「遅延的な定義」は「ファイルマネージャでのショートカットやリンク」
18. 「:=」が「遅延的な定義」で，「=」が「即時的な定義」
19. 「Column」は「リストを列として表示しなさい」
20. 「Row」は「リストを行として表示しなさい」
21. 「Grid」は「リストを格子状に並べて表示しなさい」
22. 「TableForm」は「結果が長方形の表になるようにフォーマットしなさい」
23. 命令の最後に付け加える補足的な指示のことを「オプション」と呼びます.
24. オプションで使う「→」は「->」をキーボード入力すれば自動変換されます.

文字列：文章を扱い，自動作文をさせてみよう

1. 「StringLength」は「次の文字列の文字数を調べよ」
2. 「StringCount」は「次の文字列の中に指定した文字列が何回出てくるか」
3. WindowsとUnix(Linux)ではCTRLを押しながら「k」で「式の補完」
4. MacintoshではCMDを押しながら「k」で「式の補完」
5. 「StringReplace」は「規則に従って，文字列の単語を置き換えなさい」
6. 「%」は「*Mathematica*が直前に報告した結果」
7. 「%%」で「二つ前の結果」，「%%%」で「三つ前の結果」を表します.

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

「StringCases」は「次の文字列の中から，指定した文字列パターン（StringExpression）にマッチする単語（部分文字列）を取り出せ」

9. 文字列パターンの「~~」は「次に続く」の意味です．
10. 文字列パターンの「___」（下線3つ）は「何でもOK」の意味です．
11. 「Length」は「リストの要素数を調べよ」
12. 「CharacterRange」は「次の2つの文字の間にある全ての文字を調べよ」
13. 「RandomChoice」は「次のグループの中から無作為に要素を取り出し，取り出した要素を元に戻す．これを指示された回数実験せよ」
14. 「StringJoin」は「次の語のリストを単語にまとめなさい」
15. 「RandomSample」は「次のグループの中から無作為に要素を取り出す（元には戻さない）．これを指示された回数実験せよ」
16. 「RandomInteger」は「次の範囲から整数を無作為に取り出せ」
17. 「Table」は「次の指示を指定された回数だけ繰り返せ」
18. 「Characters」は「次の文字列を一語ずつの文字のリストに分解せよ」
19. 「StringJoin」は「<>」（小なり記号，大なり記号）を使っても同じ．
20. 「Map」は「次のリストのそれぞれに対して同じ命令を実行せよ」
21. 「If」は「次の条件が満たされたら，指定の命令を実行せよ」
22. 「Join」は「複数のリストをひとつのリストにまとめなさい」
23. 「Alternatives」は「リストの中のどれか」という意味です．
24. 「Map」を使うときは基本的に，共通しない部分を「シャープ（#）」に，共通する部分の最後に「アンパサンド（&）」を付け加えなければいけません．

数と式：数に隠された秘密を少し覗いてみよう

1. 足し算を行わせるには，プラス記号「+」を使います．
2. 「Plus」は「次の数を足し合わせよ」
3. *Mathematica*では「数とリストの足し算」を，「リストの各要素ごとに同じ足し算をしたいのだろう」と解釈してくれます．
4. 引き算を行わせるには，マイナス記号「-」を使います．
5. 「Subtract」は「次の2数の引き算をせよ」
6. *Mathematica*では「数とリストの引き算」を，「リストの各要素ごとに同じ引き算をしたいのだろう」と解釈してくれます．
7. 掛け算には，記号「×」，アスタリスク「*」やスペース「 」を使います．
8. 記号「×」は「␛*␛」で入力できます．
9. 「Times」は「次の数を掛け合わせよ」
10. *Mathematica*では「数とリストの掛け算」を，「リストの各要素ごとに同じ掛け算をしたいのだろう」と解釈してくれます．
11. 何重になっても計算順序の指定には丸括弧（「(」と「)」）だけを使います．
12. 数のブロック表示の変更は，編集メニューの「環境設定...」で表示されるダイアログにて，「外観/数/桁ブロックのサイズ」を変更してください．
13. 割り算を行わせるには，記号「÷」やスラッシュ「/」を使います．
14. 記号「÷」は「␛div␛」で入力できます．
15. 「Divide」は「次の2数の割り算を求めよ」

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

*Mathematica*では「数とリストの割り算」を，「リストの各要素ごとに同じ割り算をしたいのだろう」と解釈してくれます．

17. 分数らしく入力したい場合は，CTRLを押しながら「/」を入力します．
18. 累乗（冪乗，指数計算）を行わせるには，記号「^」を使います．
19. 「Power」は「次の累乗を求めよ」
20. *Mathematica*では「数とリストの累乗」を，「リストの各要素ごとに同じ累乗をしたいのだろう」と解釈してくれます．
21. 指数らしく入力したい場合は，CTRLを押しながら「^」を入力します．
22. 「Prime」は「n番目の素数を教えなさい」
23. 「Attributes」は「次の関数の属性を調べよ」
24. 「Litable」は「リストを自動展開して，各要素に関数を適用する」属性
25. 「Select」は「次の条件を満たすものだけをリストから抜き出しなさい」
26. 「IntegerQ」は「整数かどうか調べなさい」
27. 「Cases」は「パターンにマッチする要素をリストから取り出しなさい」
28. 「Drop」は「リストの最初の要素から指定した個数分だけ削除しなさい」
29. 「Divisors」は「指示した数の約数を全て求めなさい」
30. 「Drop」に指示する整数が負の数の場合，リストの最後尾から指示した個数だけ要素を削除できます．
31. 「Total」は「リストに含まれる要素の合計を計算しなさい」
32. 下線1つ「_」で「何でもOKだが，ひとつだけ」という意味のパターンです．
33. 同じ名前の付いた下線（アンダースコア）は同じものを表すパターンです．
34. 「First」は「リストの最初の要素だけを取り出しなさい」
35. 「DeleteCases」は「パターンにマッチする要素を削除しなさい」
36. 「N」は「指定した桁数まで小数に直しなさい」
37. 円周率は「Pi」または「ESCpiESC」と入力します．
38. 自然対数の底は「E」または「ESCeeESC」と入力します．
39. 「Rationalize」は「小数を分数に変換しなさい」
40. 「ToString」は「次の数を文字列に変換しなさい」
41. 「StringPosition」は「次の文字列が表れる場所を調べなさい」
42. 「Tuples」は「リストの要素から重複を許して指定した個数になるような組合せを全て求めなさい」
43. 「Riffle」は「要素を交互に組み合わせて1つのリストにしなさい」
44. 「ToExpression」は「文字列を数式と解釈して，計算を行いなさい」
45. 「Position」は「リストの中で次のものが表れる場所を調べなさい」
46. 「Extract」は「リストから指定した部分を取り出しなさい」
47. 「Expand」は「次の数式を出来るだけ展開しなさい」
48. *Mathematica*では変数と関数に違いはなく，どちらも「シンボル」です．
49. 「=」は，リストや数に名前を付けて覚えておく「シンボルへの値の割り当て」の意味を持ちます．
50. 「==」（等号を2つ並べます）は，左辺と右辺が数学的に等しいことを表し，*Mathematica*に方程式だと認識させます．

All Rights Reserved, Copyright (C) Kosaku Nagasaka 2007

「Solve」は「次の方程式を解いて未知数を求めよ」

52. 連立方程式であることを示すには，方程式をリストで与えます．

53. 等号1つ「=」で「シンボルの定義」，等号2つ「==」で「数学的に等しい関係」，そして等号3つ「===」で「見た目に等しい関係」です．

54. 「NSolve」は「数学的な厳密さは多少失っても良いから，わかりやすい小数の形で解を求めなさい」

55. 「/.」は「左の式に含まれる変数を，右側の置換規則で置き換えなさい」

56. 右矢印（→）は「左側のものを右側に対応させる置換規則（変換規則）」

57. パターンにおいて，「Real」で「実数」を，「Integer」で「整数」を，「Ratic
数単位は「I」で表す）を，「Symbol」で「シンボル（変数）」を表します．

58. 「Reduce」は「方程式や不等式を解き，変数に関する条件を簡約しなさい」

59. *Mathematica*でも，アンパサンド2つ（&&）で論理積（かつ）を，縦線2つ（||）で論理和（または）を表します．

60. 「Maximize」は「制約条件下で指定した式を最大化する変数の値を求めよ」