人間の世界では「1+1=2」

コンピュータの世界では「1+1=10」となる

これはそれぞれの世界の数値の表記法が異なるため

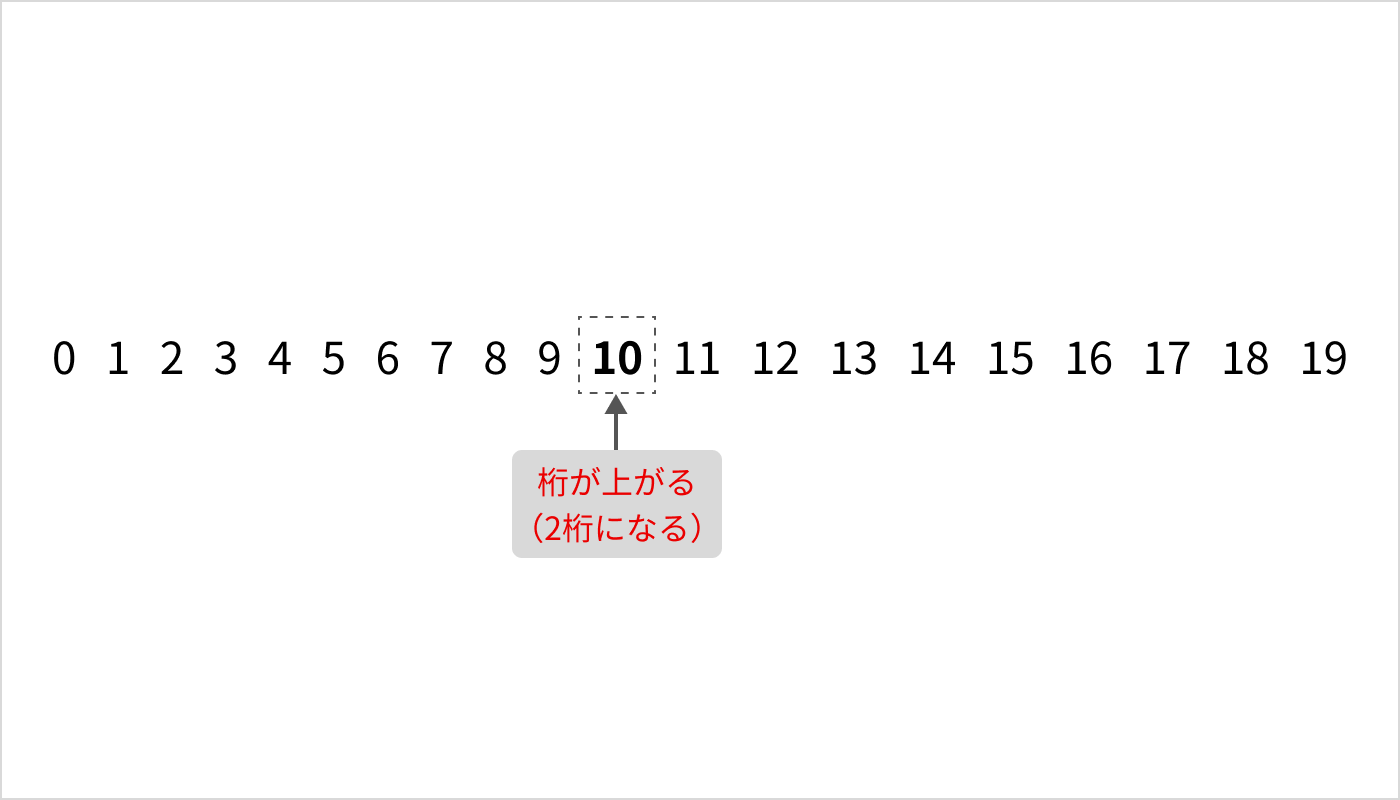
**１数値の考え方**

人間の世界で利用している数値の表記は10進数で

コンピュータの世界で使用する表記は「２進数」と「１６進数」になる

１０進数　２進数　１６進数

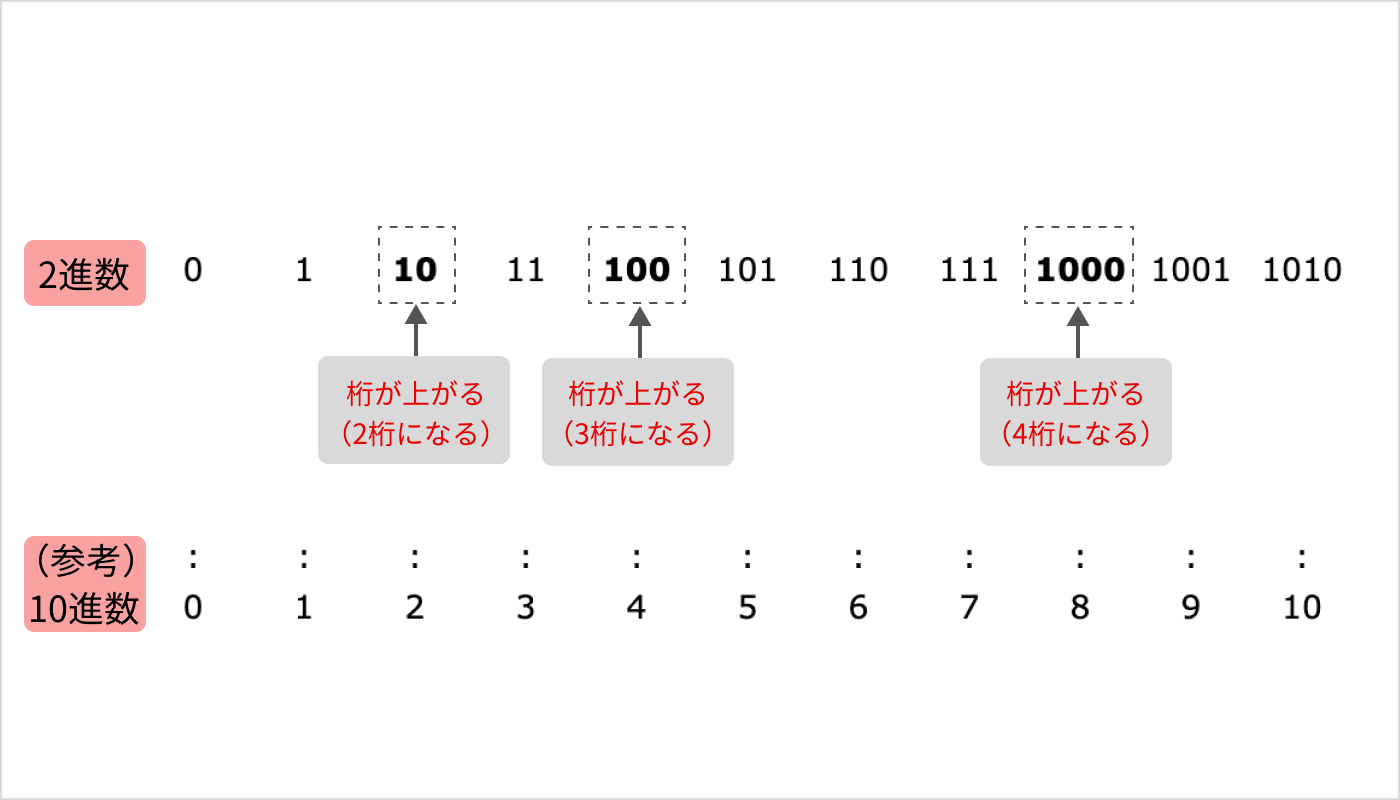
１０進数、値が「１０」になったときに桁を一つあげる表記法



**２進数**

値が「２のｎ乗」になったときに桁を一つ上げる表記法

コンピュータの世界で使われる表記法になる



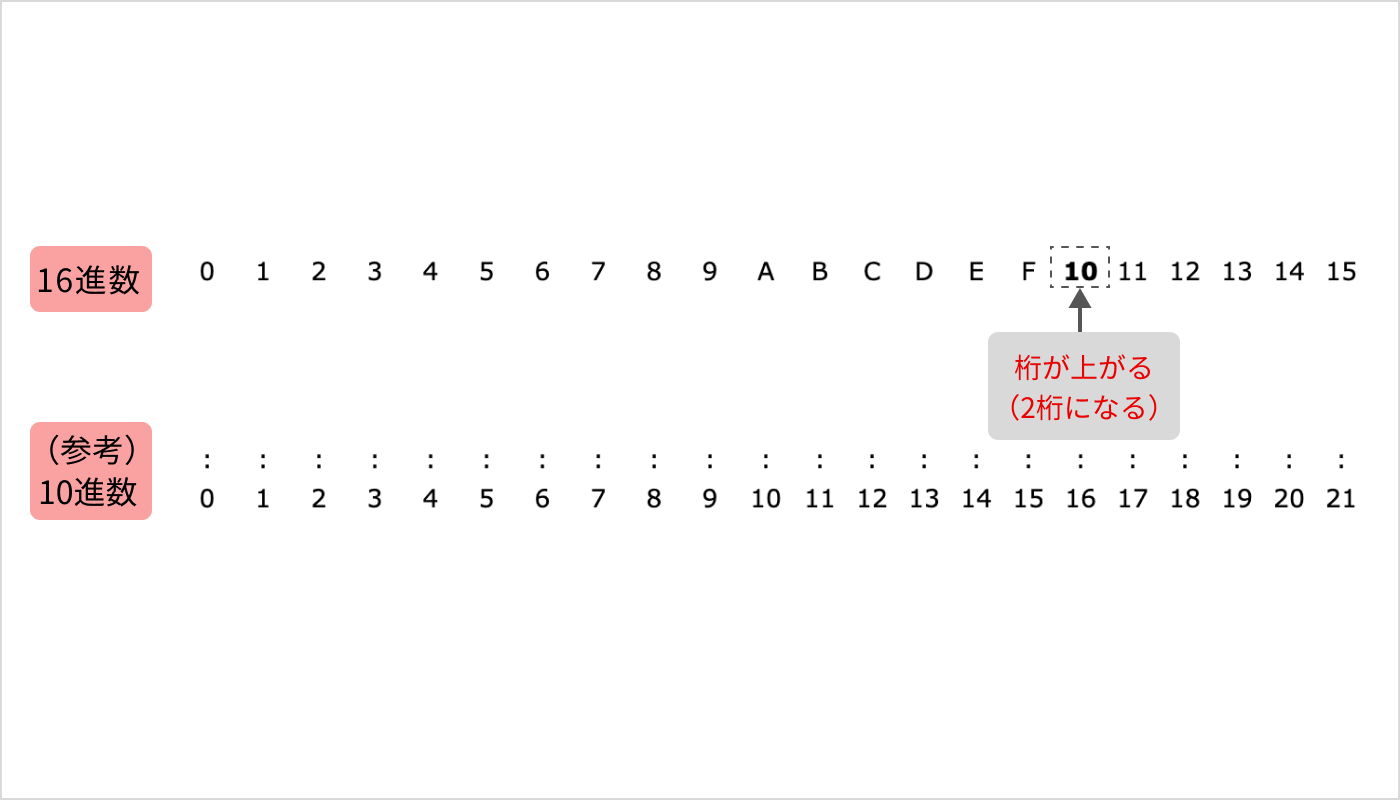
０から１までは１桁で表記する、１も次は２ではなく桁を１つ上げて１０と表現するのが特徴、つまり１０進数の２は２進数で表記すると１０となる

コンピューターで扱うのは２進数で、コンピューターの世界を２進数で表現すると、桁が多くなり人間にとっては扱いにくくなる、そこで１６進数を使用する

**１６進数**

値が１６になったときに桁を１つあげる表記法、０～９とＡ～Ｆの１６文字を使用する

コンピュータで扱うのは２進数で、大きな数値を２進数で表現すると桁が多くなり人間にとって扱いにくくなるから１６進数を使う



０から１５までは１桁で表記するため９以降はAからFを使用する

すなわちAは１０進数の１０でFは１０進数の１５になる

１６進数ではFの次は桁を1つ上げて１０になる

１６進数の１０は１０進数の１６のこと

**基数と基数変換**

基数とは桁上がりの基準となる数のこと、例えば１０進数は１０になるときに桁があがるので１０進数の基数は「１０」となる

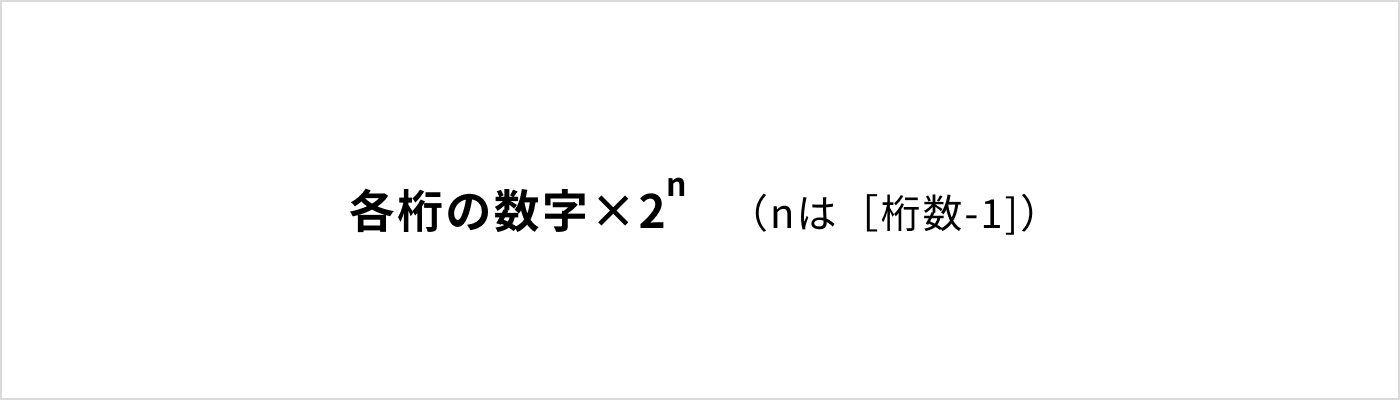
２進数の基数２、１６進数の基数は１６

| **2進数** | **10進数** | **16進数** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 10 | 2 | 2 |
| 11 | 3 | 3 |
| 100 | 4 | 4 |
| 101 | 5 | 5 |
| 110 | 6 | 6 |
| 111 | 7 | 7 |
| 1000 | 8 | 8 |
| 1001 | 9 | 9 |
| 1010 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

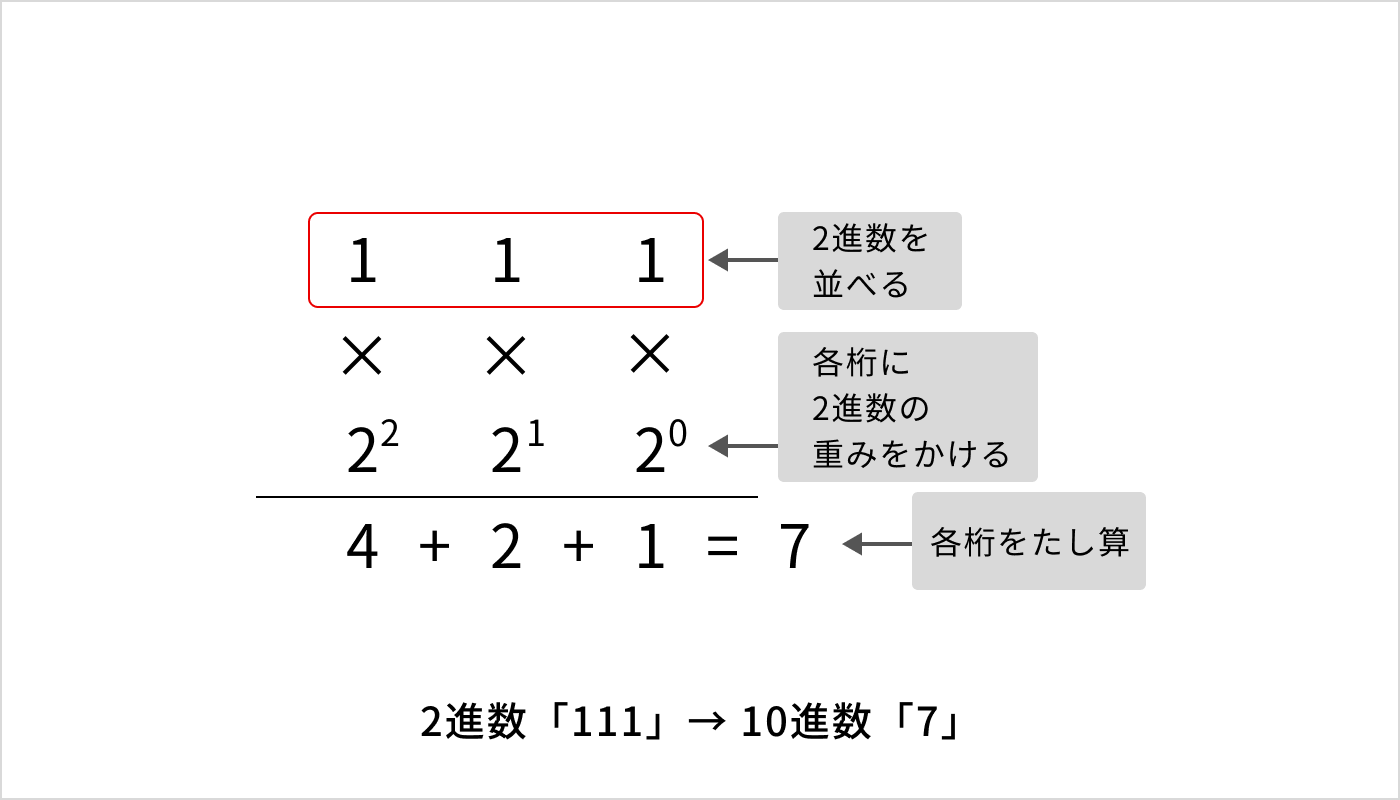
基数変換のやり方

#### **2進数→10進数**

2進数から10進数への変換は、まずは下の計算を行い、次にかけ算した各桁の数字をたし算します。



例えば、2進数「111」を10進数に変換する計算は、次のとおりです。

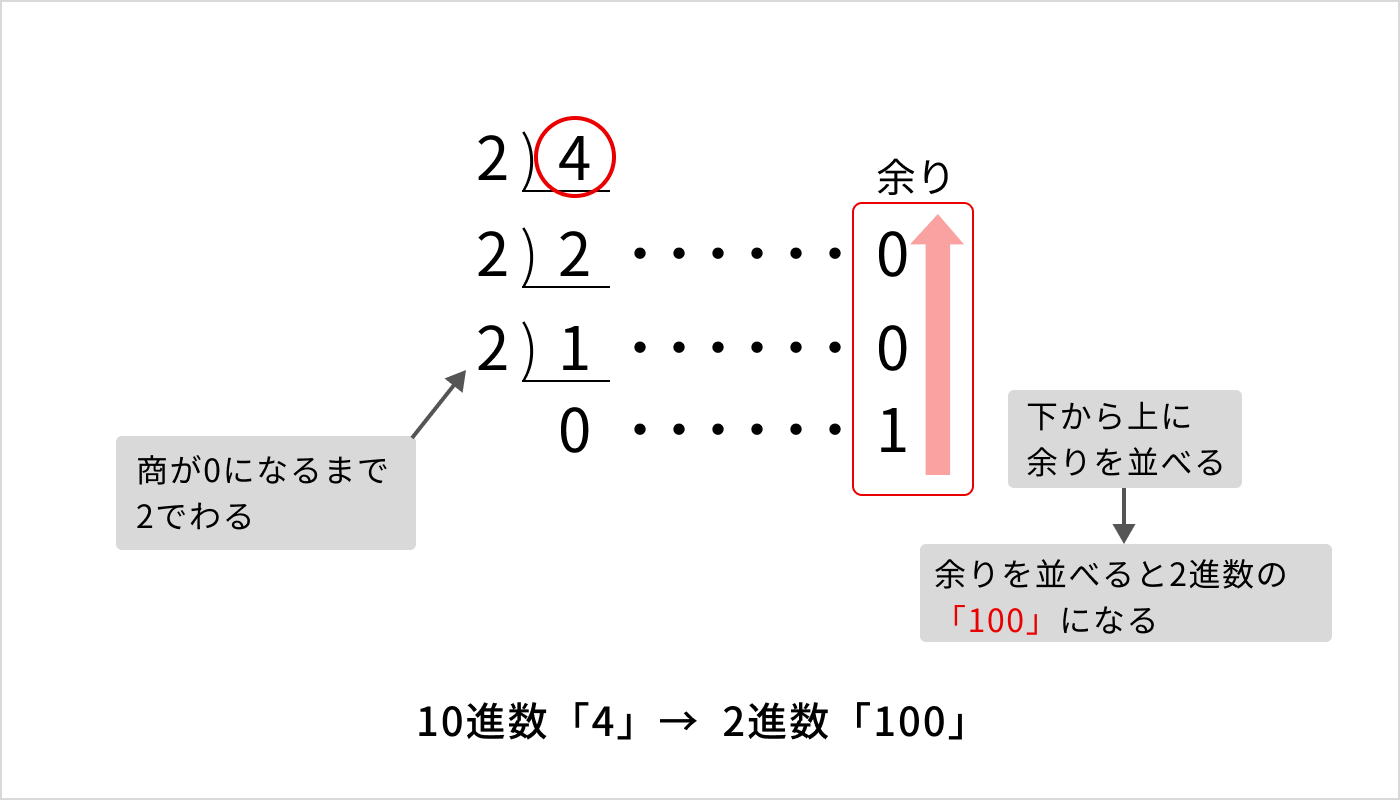


2進数「111」は、10進数の「7」になります。

#### **10進数→2進数**

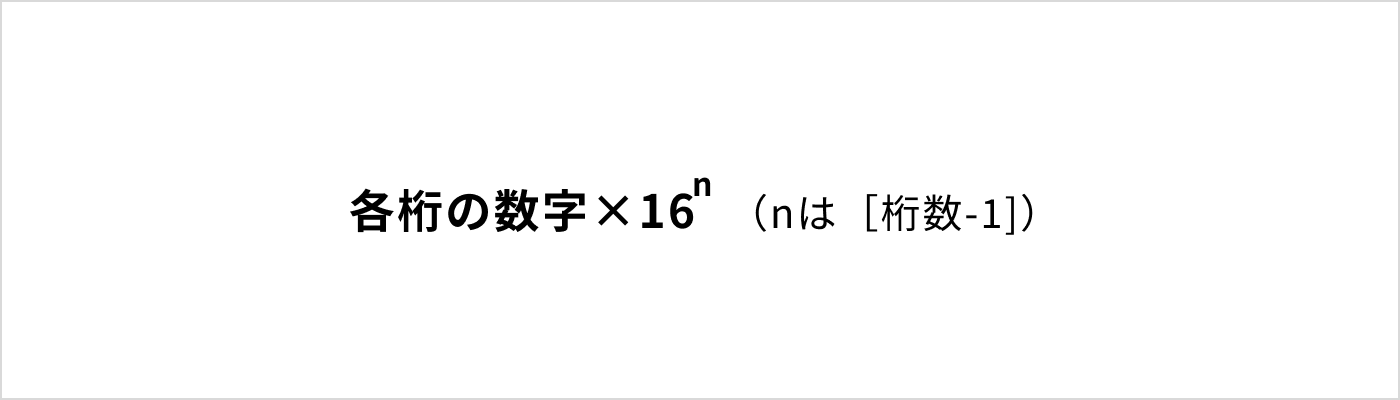
10進数から2進数への変換は、10進数の数値を2でわり算して「商」が0になるまで計算を行い、次に「余り」を下から並べます。

例えば、10進数「4」を2進数に変換する計算は、次のとおりです。



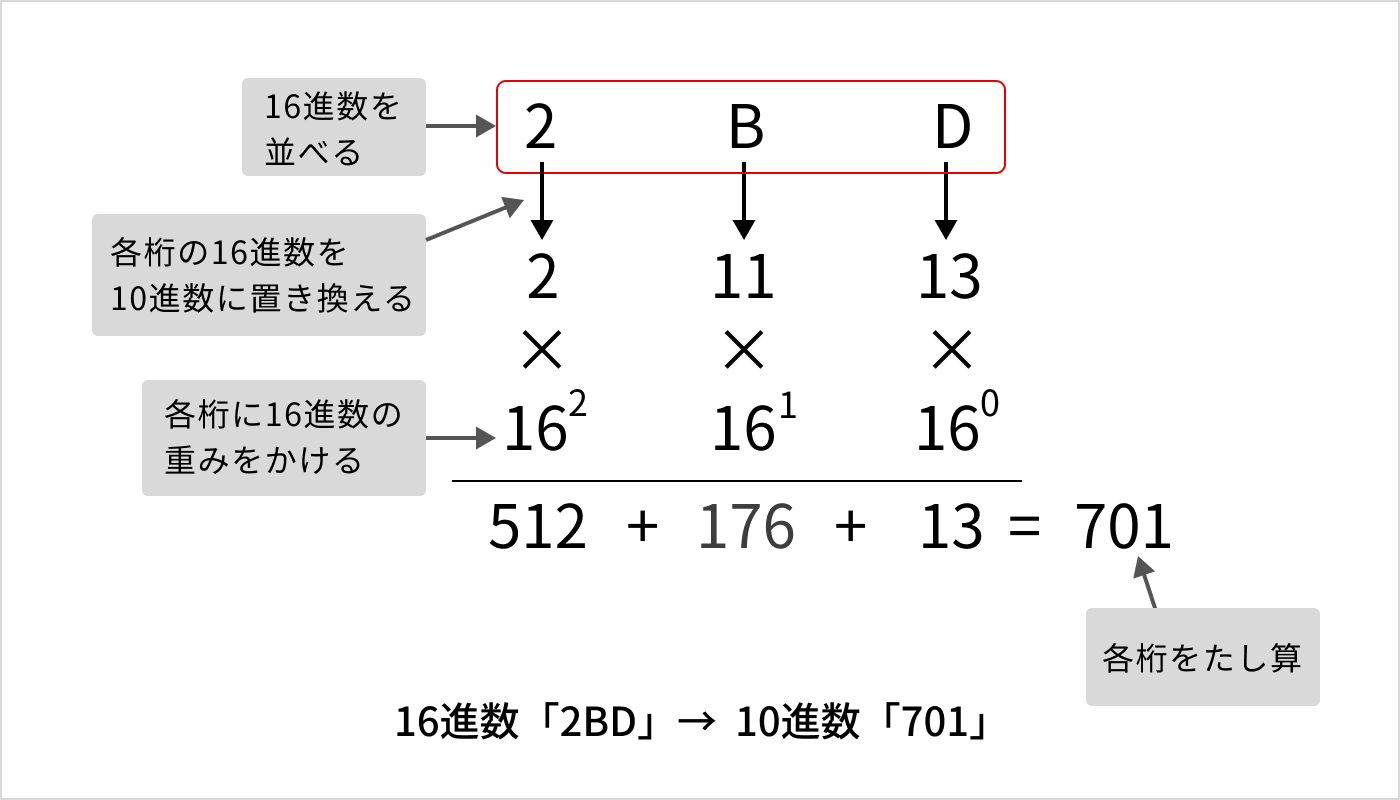
#### **16進数→10進数**

16進数から10進数への変換は、まずは下の計算を行い、次にかけ算した各桁の数字をたし算します。2進数から10進数への変換式に似た形です。



16進数から10進数に変換するときは、**16進数のアルファベットを10進数に変換してから計算**します。

例えば、16進数「2BD」を10進数に変換する計算は、次のとおりです。



#### **10進数→16進数**

10進数から16進数への変換は、10進数の数値を16でわり算して「商」が0になるまで計算を行い、次に「余り」を下から並べます。

**「余り」の「10〜15」は、「A～F」のアルファベットに変換**する必要があります。

例えば、10進数「701」を16進数に変換する計算は、次のとおりです。

#### **10進数→16進数**

10進数から16進数への変換は、10進数の数値を16でわり算して「商」が0になるまで計算を行い、次に「余り」を下から並べます。

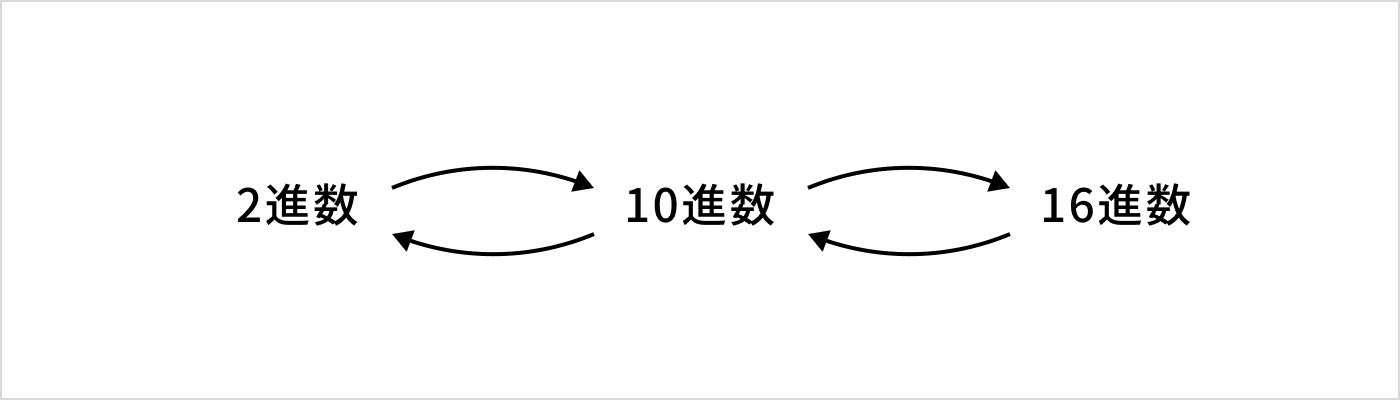
**「余り」の「10〜15」は、「A～F」のアルファベットに変換**する必要があります。

例えば、10進数「701」を16進数に変換する計算は、次のとおりです。

最後に、2進数と16進数の基数変換について補足します。

##### **2進数と16進数の変換**

「2進数→16進数」「16進数→2進数」の基数変換は、10進数を経由したほうが簡単です。



「2進数→16進数」の場合、「2進数→10進数→16進数」の流れになります。「16進数→2進数」の場合は、流れを反対にするだけです。

* 2進数→10進数
* 10進数→16進数
* 16進数→10進数
* 10進数→2進数

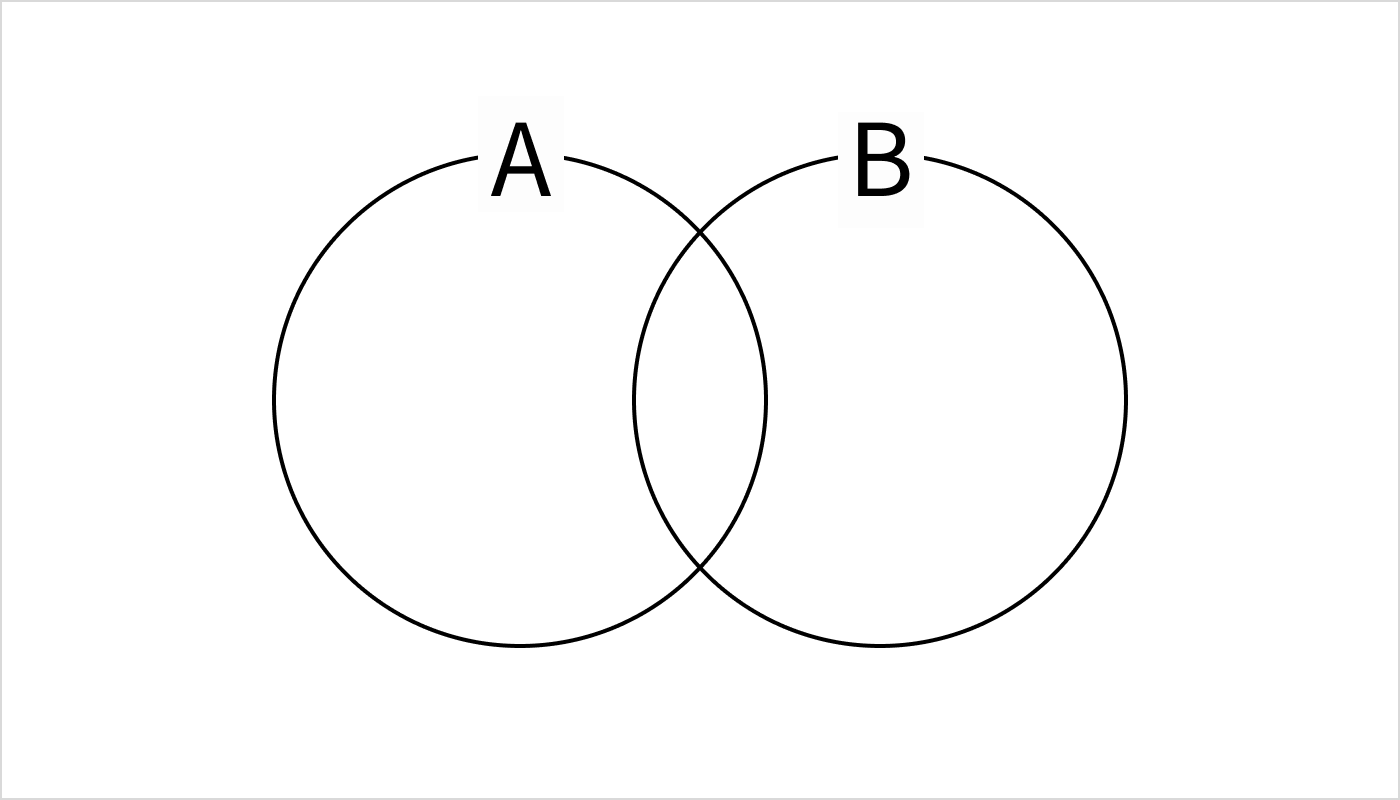
集合とベン図

集合とは、何かしらの条件でグループ分けされたモノの集まり

集合を構成する一つ一つのものを要素という

また、集合の関係を図にしたものをベン図という

ベン図は以下の通りで、集合を円で表す

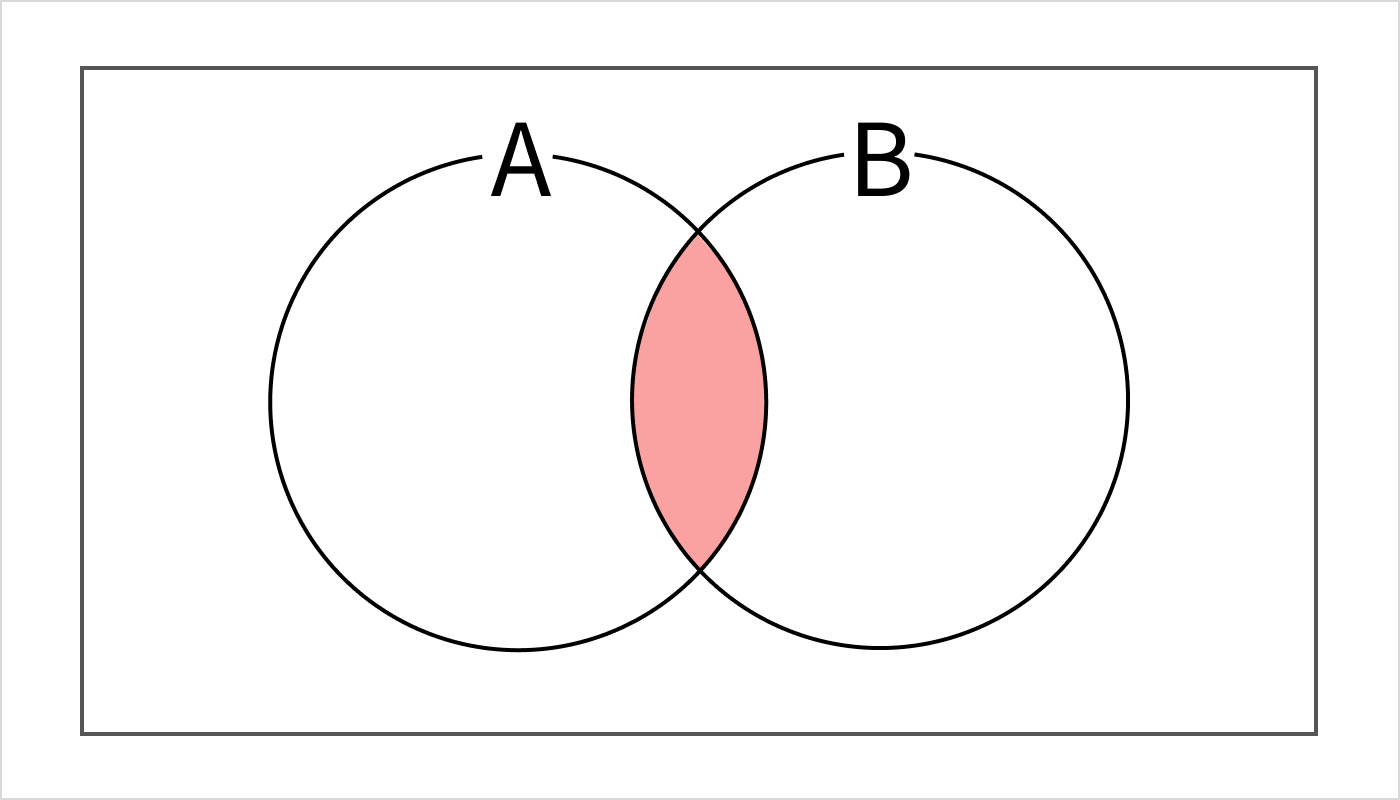


∩と∪

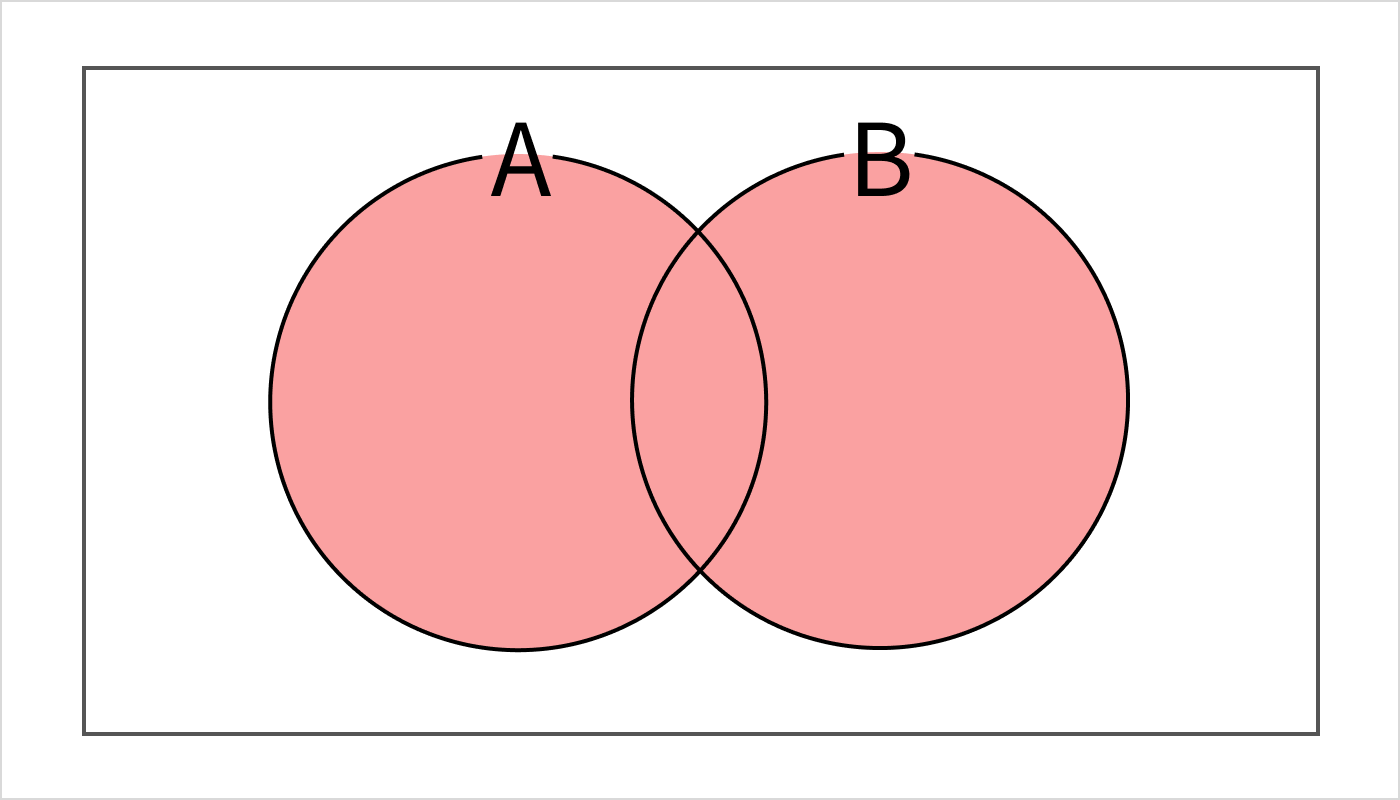
複数の集合は要素の表す記号を「∩（キャップ）」と「∪（カップ）」という

∩は「かつ」で∪は「または」という意味

A∩B（AかつB）　呼び方：積集合



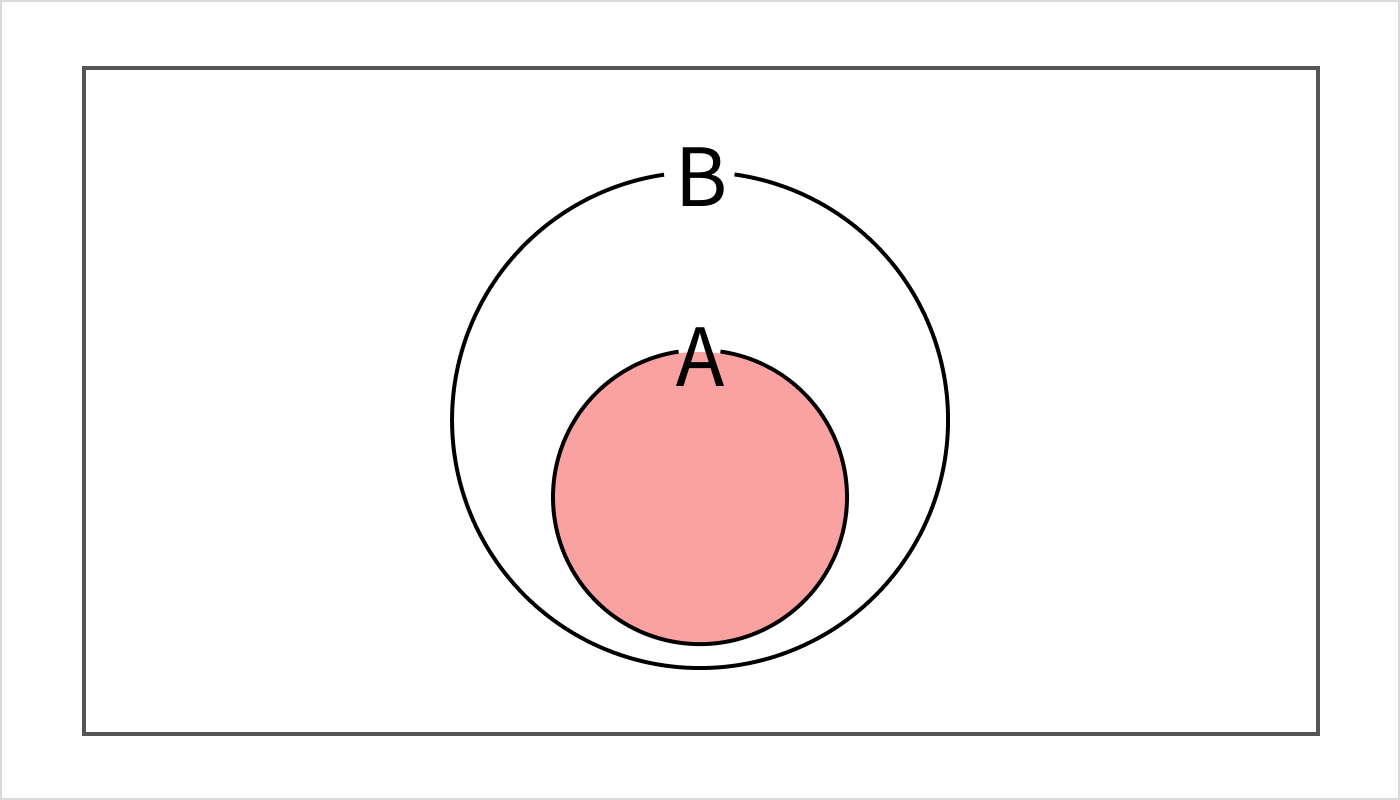
A∪B（AまたはB） 呼び方：和集合



部分集合

ある集合が別の集合の一部となっている状態

下図の場合は「集合Aは集合Bの部分集合である」という

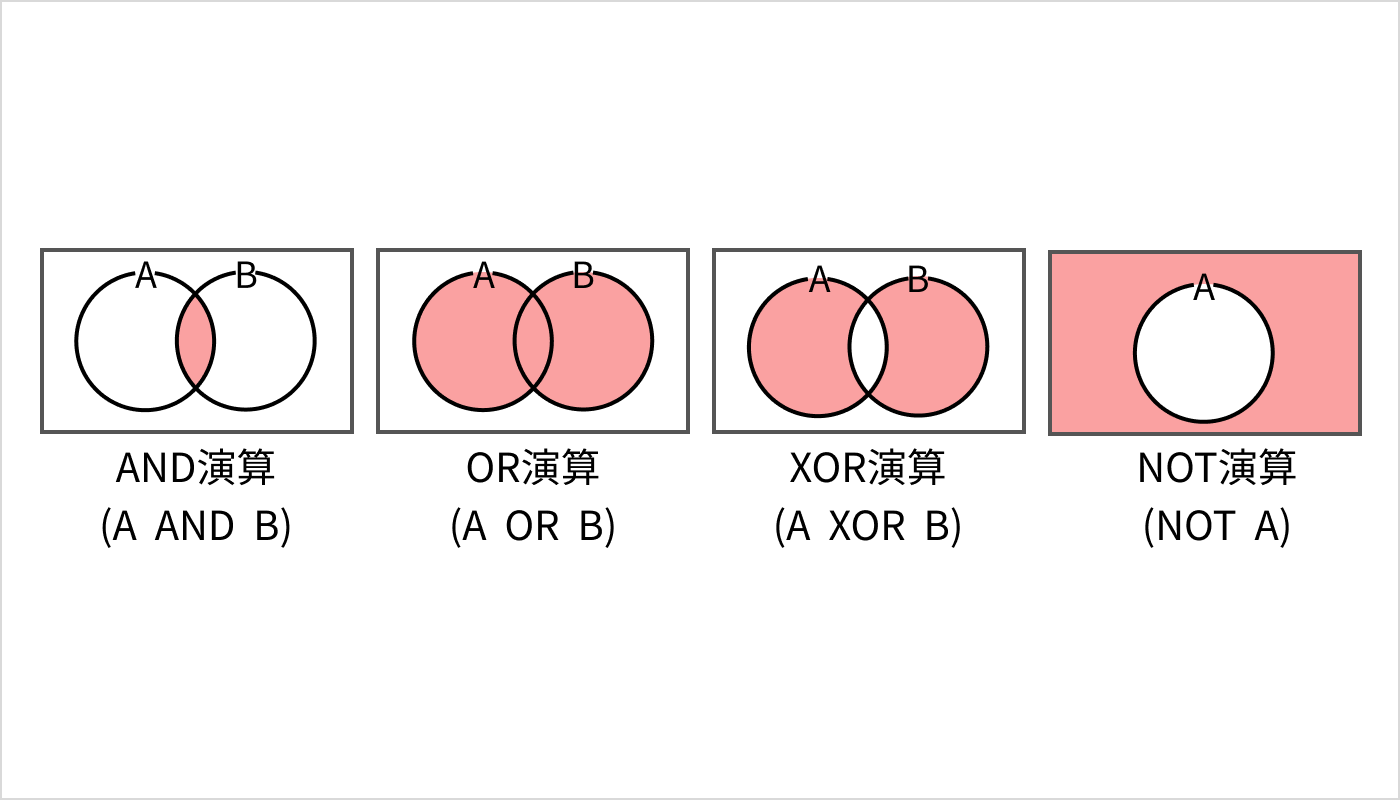


**論理演算**  
論理演算を簡単にまとめると「コンピューターやロボットなどの機械が情報を正確に処理し、適切な動作を行うために必要な方法」  
論理演算を使うと、機械がある情報を真か偽を判断し、それに応じた動作をする

論理演算とは「真（true）」は「条件が成立」  
「偽（false）」は「条件が成立しない」という意味  
**論理演算では「１（真）」と「０（偽）」だけで演算を行い、計算結果も「１」か「０」になる**

例えば、ドアが開いていたらドアを閉める機械を作るとして  
・ドアが開いている、という情報が「真」  
・ドアが閉まっている、という情報が「偽」  
この情報をもとにドアが開いている場合には「ドアを閉める」、ドアが閉  
まっている場合には何もしないというように、機械が判断して動作を行う  
以上の通り、論理演算は機械が情報を正確に処理し、適切な動作を行うために必要な方法

**演算子と真理値表**  
前述のドアの例のように、様々な処理を実行しようと考えたときに、さまざまな条件を指定する必要がある。  
そこで使うのが、４種類の演算子ＡＮＤ、ＯＲ、ＸＯＲ、ＮＯＴ

１を赤色、０を白色とした場合、論理演算の結果をベン図で表すと下図になる  
  
また下表では、各演算子の説明と演算結果をまとめている、論理演算の結果をまとめた表のことを「真理値表」という  
資格取得の際は「演算子、ベン図、真理値表」はセットで暗記する

| **演算子** | **説明** |
| --- | --- |
| **論理積** **（AND：アンド）** 論理積のベン図 | AとBが**両方とも「1（真）」** の場合、結果が「1（真）」になる それ以外の場合、結果が「0（偽）」になる AND演算の真理値表 |
| **論理和** **（OR：オア）** 論理和のベン図 | AとBの**少なくともどちらか一方が「1（真）」** の場合、結果が「1（真）」になる それ以外の場合、結果が「0（偽）」になる OR演算の真理値表 |
| **排他的論理和** **（XOR：エックスオア）** 排他的論理和のベン図 | AとBが**異なる**場合、結果が「1（真）」 AとBが**同じ**の場合、結果が「0（偽）」 XOR演算の真理値表 |
| **否定** **（NOT：ノット）** 否定のベン図 | 入力した値の**反対の値**が結果 NOT演算の真理値表 |

ITパスポート試験では、上記の基本的な4種類の演算子以外にも「NAND」と「NOR」が出題されることがあります。

NANDとNOR

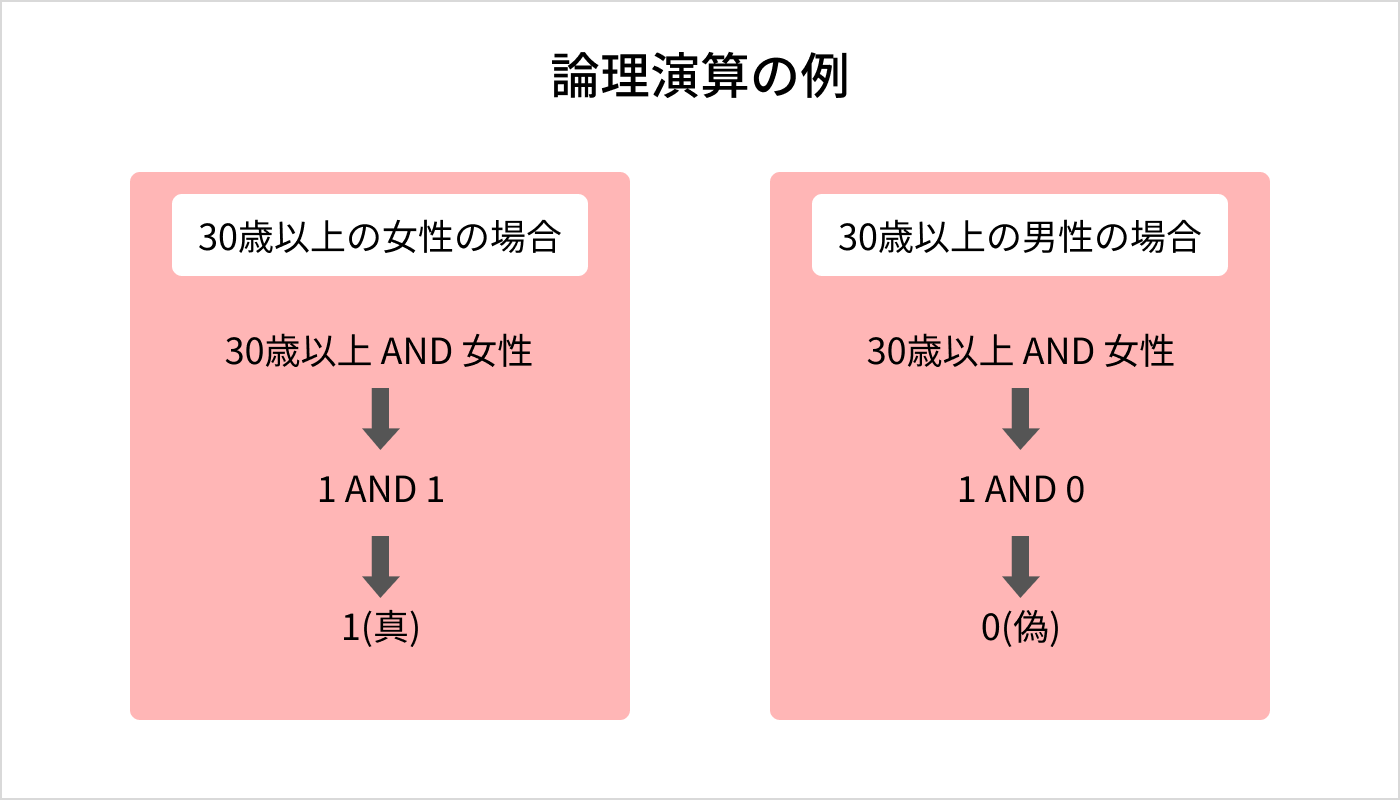
| **演算子** | **説明** |
| --- | --- |
| **否定論理積** **（NAND：ナンド）** 否定論理積のベン図 | **論理積（AND）と否定（NOT）を組み合わせた演算**を行う 計算方法は、AND演算の結果を反転するだけ 反転とは「0」を「1」に、「1」を「0」にすること NAND演算の真理値表 |
| **否定論理和** **（NOR：ノア）** 否定論理和のベン図 | **論理和（OR）と否定（NOT）を組み合わせた演算**を行う 計算方法は、OR演算の結果を反転するだけ NOR演算の真理値表 |

**論理演算の具体例**  
論理演算で使う値は「０」と「１」のみ、しかし「０」を偽「１」を真に置き換えると様々な計算ができる  
  
例えば、次のような演算式



この演算式は「３０以上」と「女性」のAND演算となっている  
計算結果が「１（真）」になるのは「３０歳以上の女性」のみ  
  
・３０歳以上「１」（真）の女性「１」（真）  
計算結果が「０」（偽）となるのは以下のパターン

* 30歳以下「0」（偽）の女性「1」（真）
* 30歳以上「1」（真）の男性「0」（偽）
* 30歳以下「0」（真）の男性「0」（偽）

**括弧の扱い**  
論理演算では、優先的に計算を行う括弧「（）」を使用できる  
たし算や掛け算などの四則演算と使い方は同じ

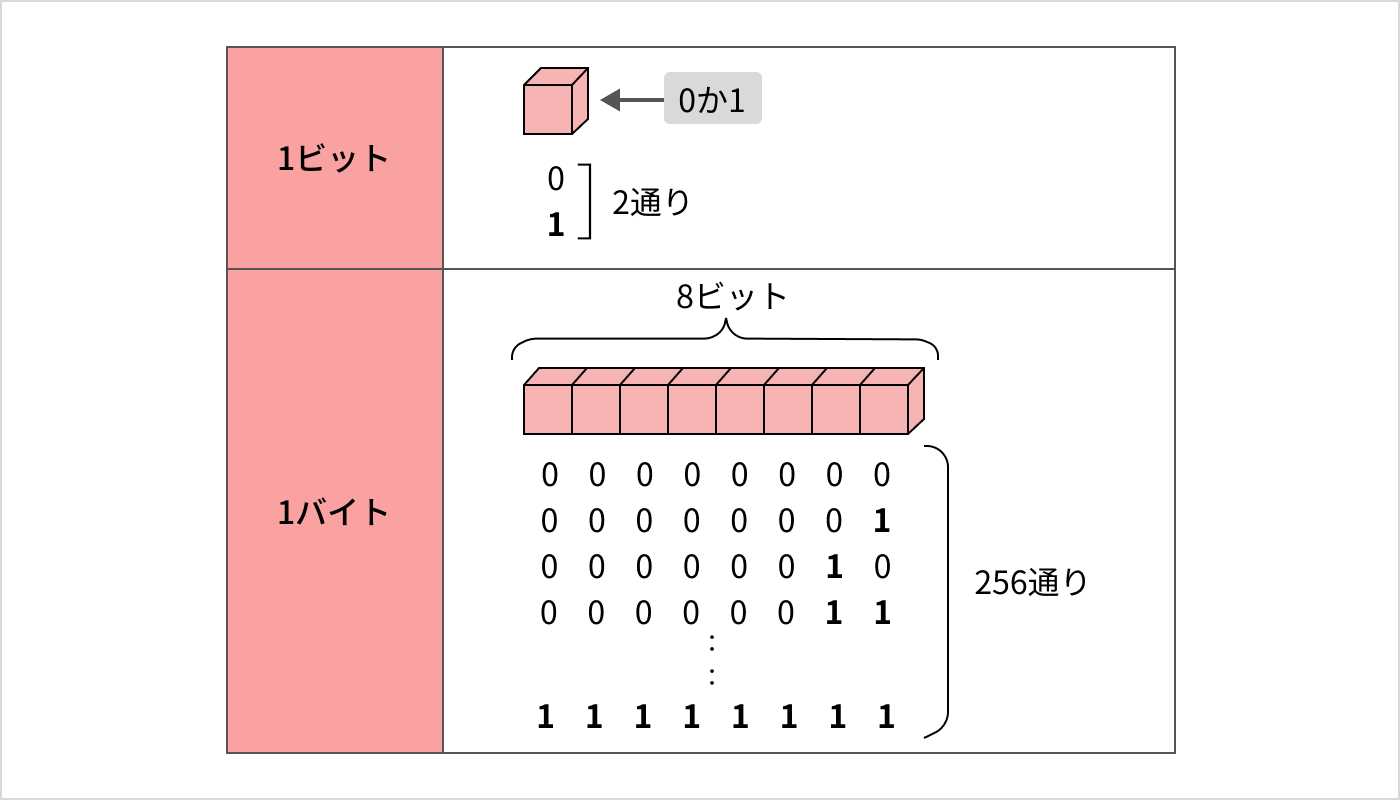


上記の演算式では、括弧内の「女性OR主婦」が先に計算される  
「女性」または「主婦」であれば「１」（真）  
括弧内の演算をした後は、括弧外の演算を行う  
この演算式は「３０歳以上の女性」または「３０歳以上の主婦」の場合、計算結果が「１」真

* 30歳以上「1」（真）の女性「1」（真）
* 30歳以上「1」（真）の主婦「1」（真）

**データ単位**

**ビットとバイト**  
ビットはコンピュータが扱うデータの「最小単位」のこと  
１ビットには「０」または「１」の２通りの状態がある  
そのため、１ビットで表せるデータ量は、２進数の１桁で表せるデータ量と同じ  
  
**バイトとはビットが８個集まったもの**  
１バイト＝８ビット　になる、１ビットには「０」か「１」の２通りの状態があるため「２の８乗」となり、１バイトは「２５６通りの状態」と表せる



**様々なデータの単位**



**まとめ**

・集合とは何かしらの条件でグループ分けされた「モノ」の集まりのこと  
・ベン図とは、集合の関係を視覚的に表した図のこと  
・論理演算とは「真の値」「偽の値」という２つの値を用いて行う演算  
・ビットとはコンピュータが扱うデータの最小単位  
・バイトとはビットが８個集まったもの

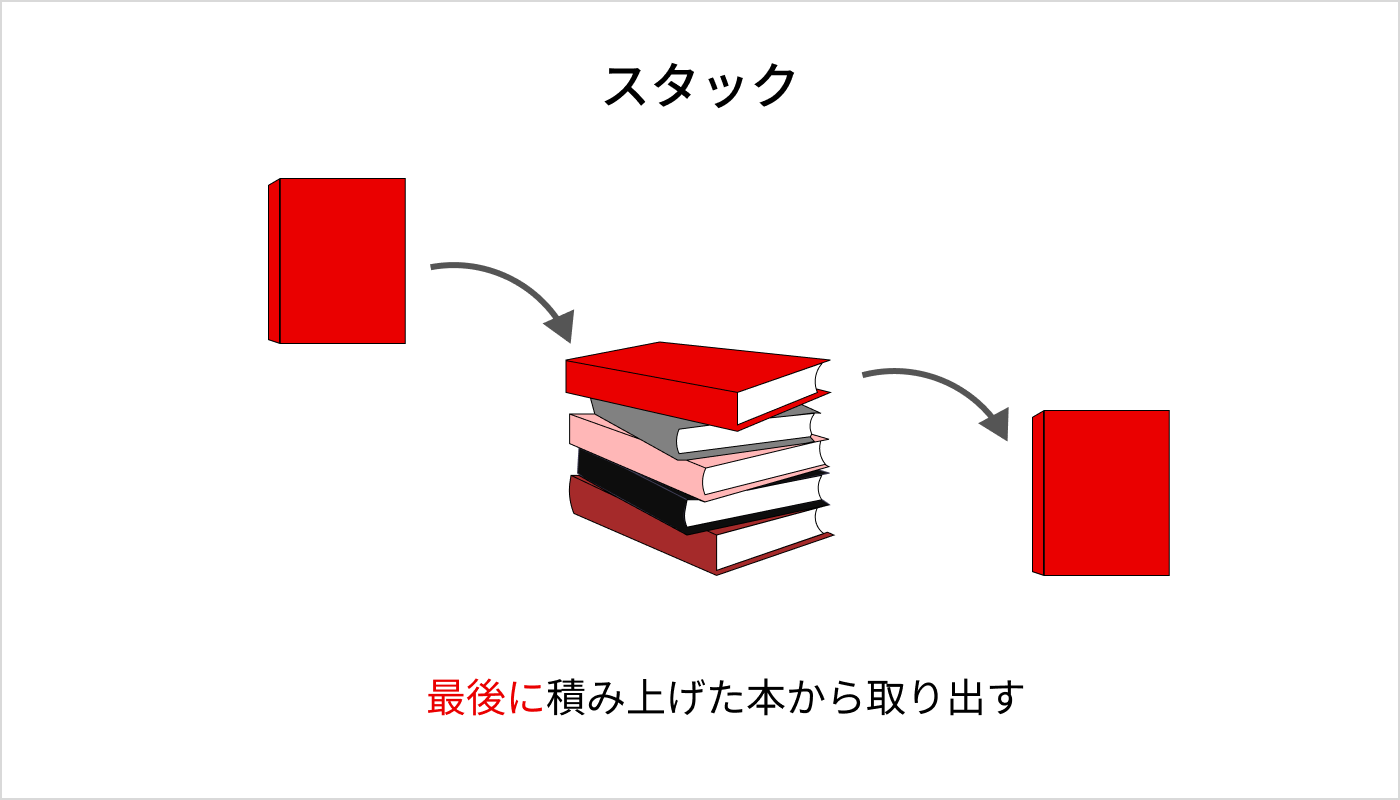
データ構造

データとはコンピュータが扱う情報のこと、コンピュータ内にあるメモリといわれる記憶装置に保存される  
また、メモリにおけるデータの並べ方を「データ構造」という

スタックとキュー  
  
スタック  
最後に格納したデータから取り出すデータ構造のこと

「積み上げられた本」をイメージすると分かりやすい  
本を積み上げていき、一番上にある本から順番に取っていく、途中にある本は取り出すことはできない

積み上げる（Stack）のことから、スタックと呼ばれている

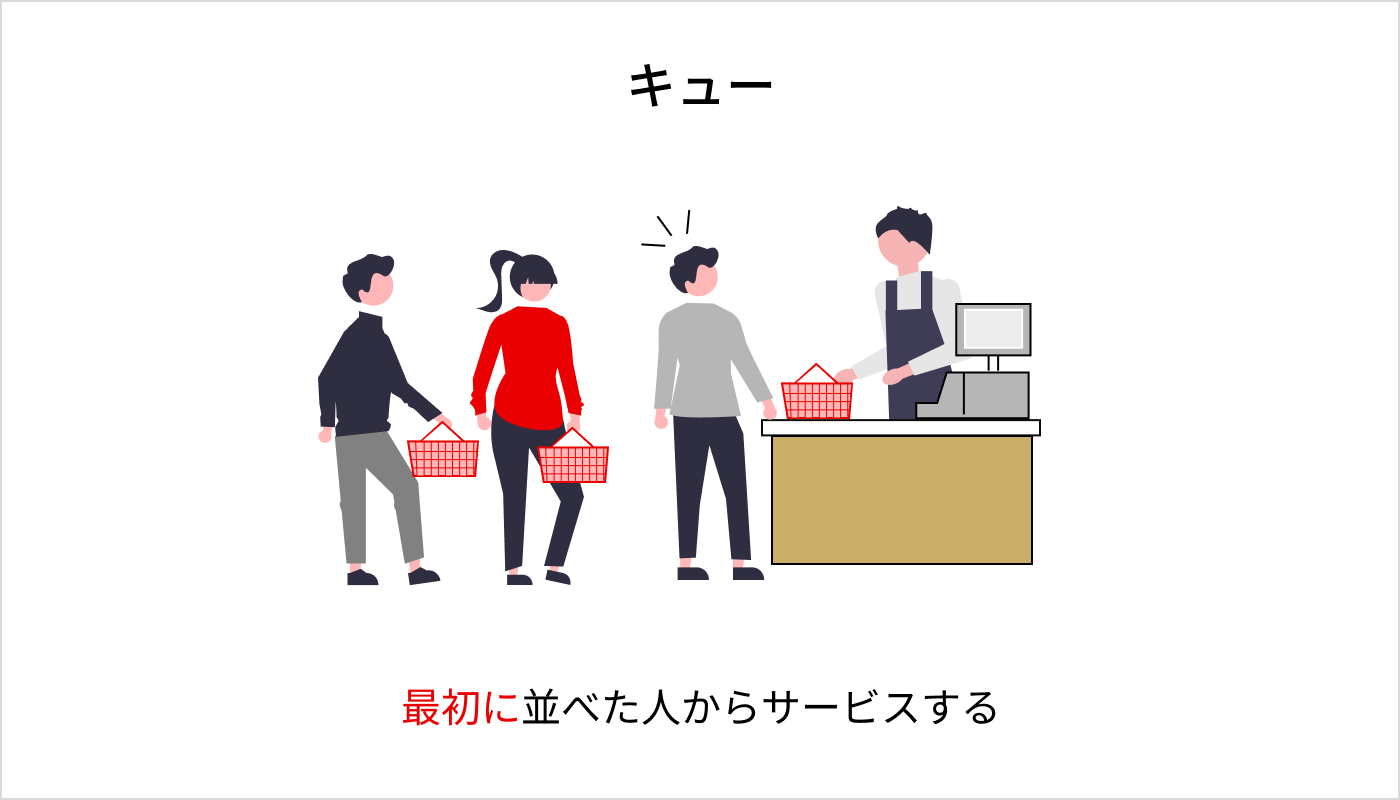
スタックというデータ構造を使い「行った処理」の過程を保存すれば、コンピュータはいつでも元の処理に戻すことが可能にある  
理由は、保存されている処理を１つずつ戻すことで、元の処理にたどり着けるから  
この仕組みこそ、スタックのメリットでもある  
  


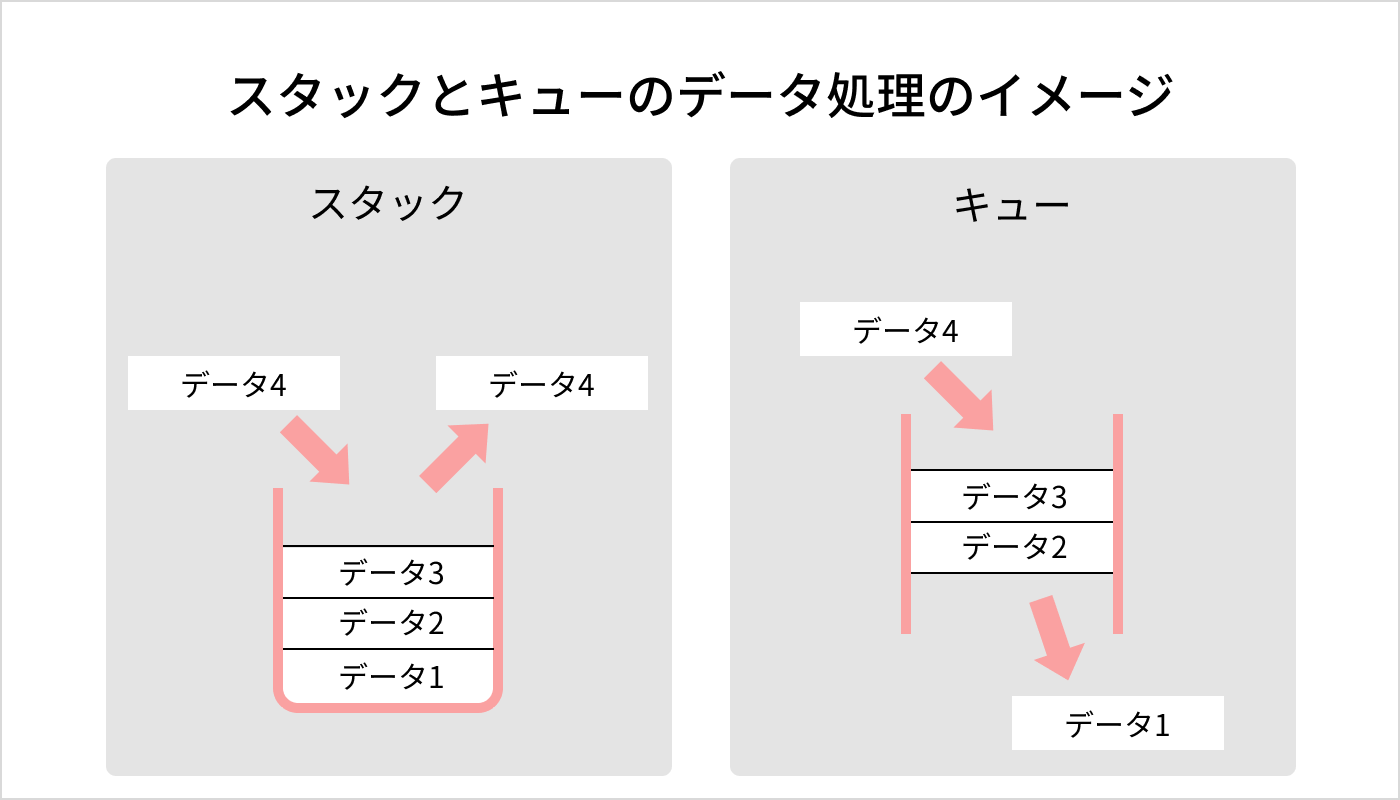
キュー

最初に格納したデータから取り出すデータ構造のこと

「レジ待ちの人の列」をイメージすると分かりやすい  
客をデータに例えて、客は列の最後尾に並ぶ、最初に並んだ客が、最初にレジを済ませる、列の途中に割り込むことはできない

命令を入力した順番どおりの処理が可能となる点がキューのメリット





アルゴリズムとフローチャート  
問題を解決したり、目標を達成したりするための手順を文章で表現したものが「アルゴリズム」

図にしたものは「フローチャート」

アルゴリズム

問題を解決するために必要な手順のこと

アルゴリズムを使うことで、問題を解決するために必要な手順を明確にし、効率的に問題を解決できる

コンピュータやロボットなどの機械はアルゴリズムに従って指示された手順を実行することで、複雑な計算や問題を解決できる

例えば、ロボットにカレーを作ってもらう（問題）ケースを考える  
ロボットはカレーの作り方を知らないが、調理手順（アルゴリズム）を教えればカレーを作れる

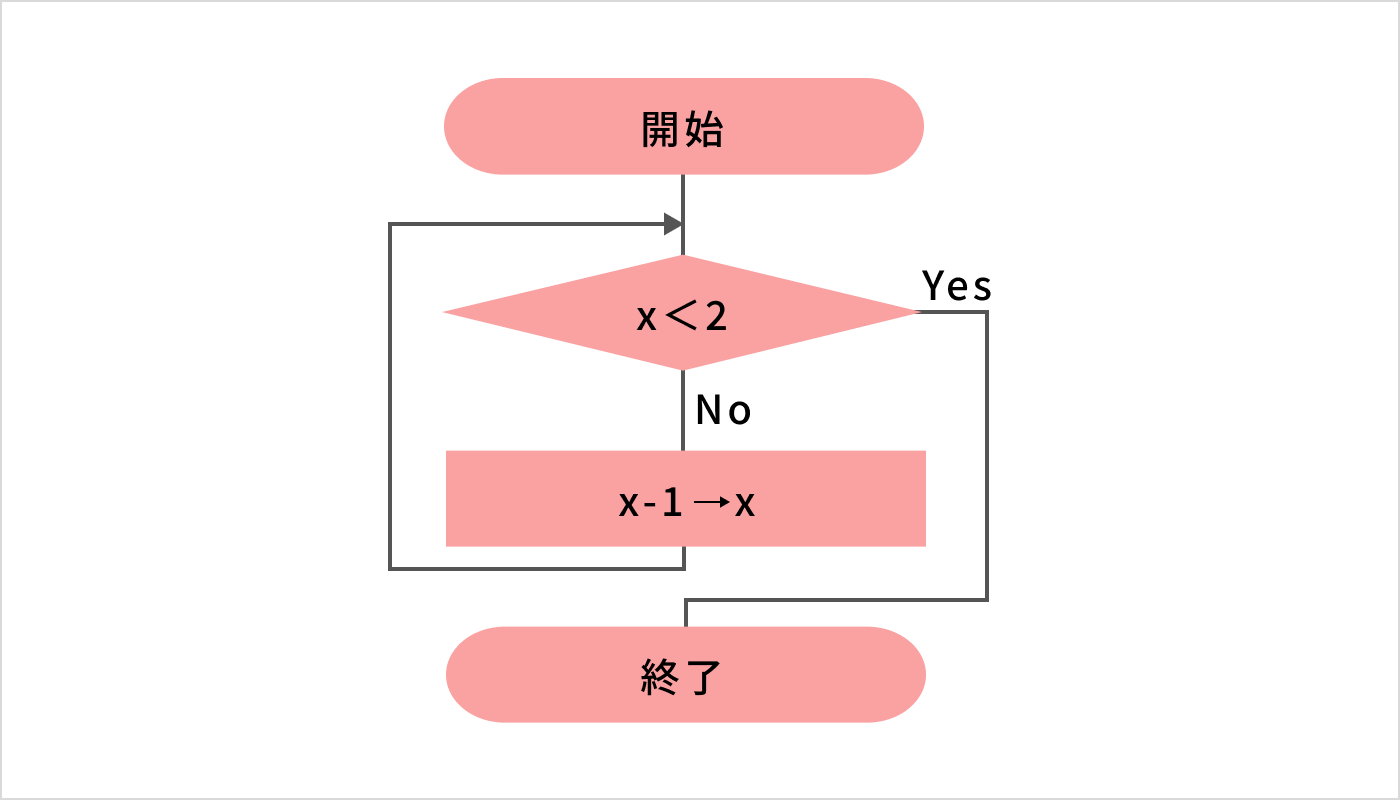
また、一つの問題に対して、アルゴリズムは一つとは限らない  
カレーを作る問題といっても、先に肉を炒めるのか、野菜を最初に切るかなどの調理手順（アルゴリズム）はさまざまになる

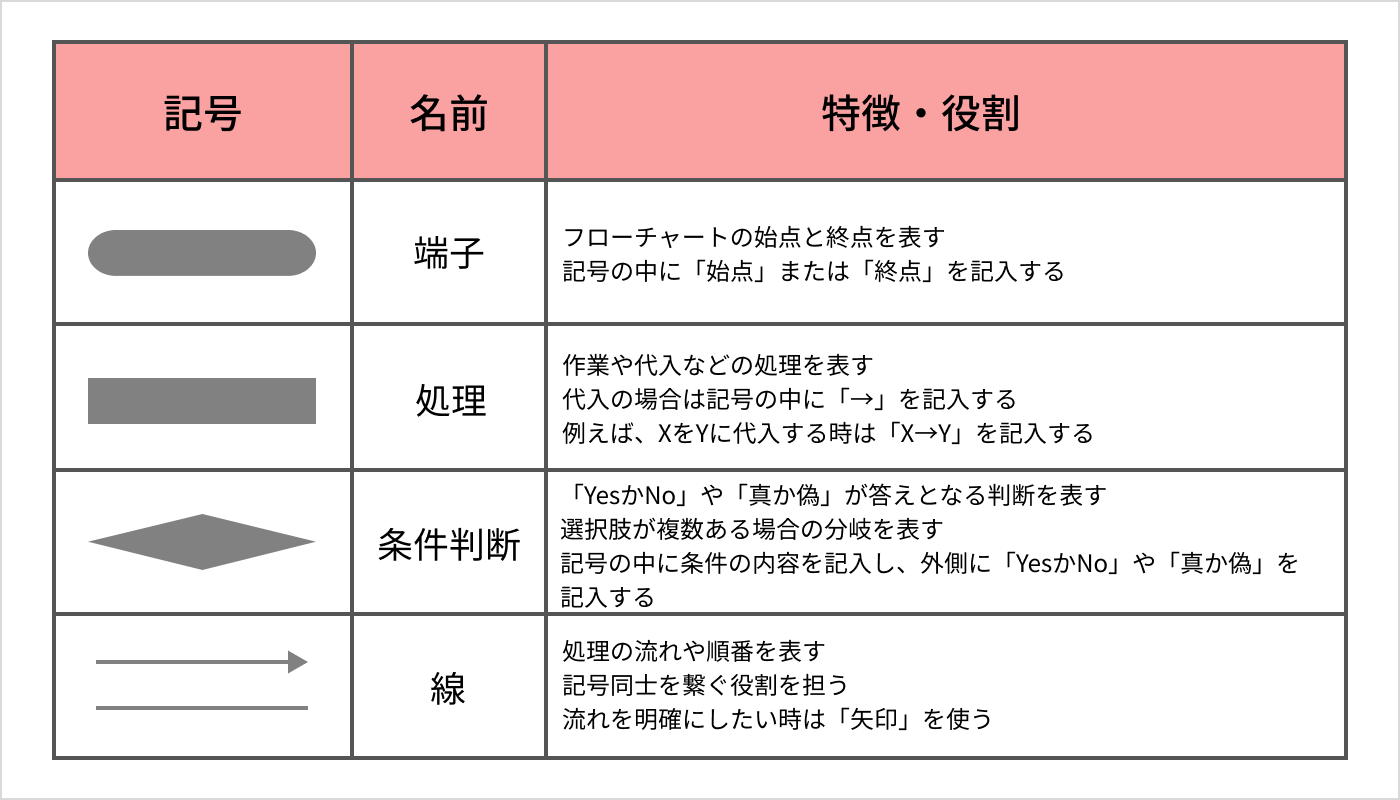
フローチャート

アルゴリズムの内容を視覚的にわかりやすく表した図のこと

「流れ図」ともいう

フローチャートの例と記号



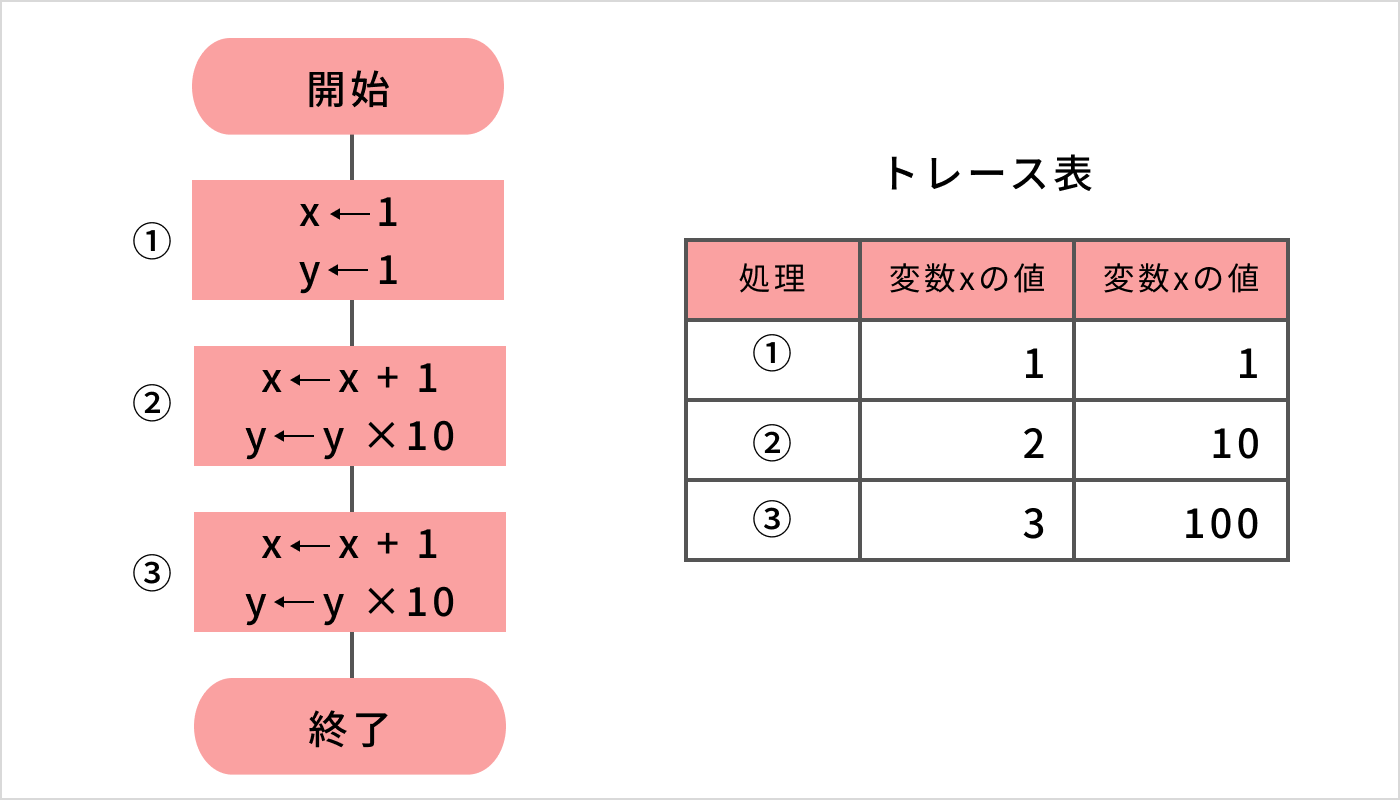


トレース表

フローチャートどおりに計算を行い、それぞれの処理を実行するときの「変数の値」を書きだした表のこと

ITパスポートにはほとんど出題されない

トレース表が書ければアルゴリズムやフローチャートの問題が解ける



フローチャートの①〜③は、**処理記号**のため「矢印」で代入を表しています。そのため、処理①から順に代入するだけでトレース表を書き出せます。

コンピュータ言語

言語にはさまざまな種類があるが、ITパスポートでは「HTML」と「XML」が出題されやすい

HTML

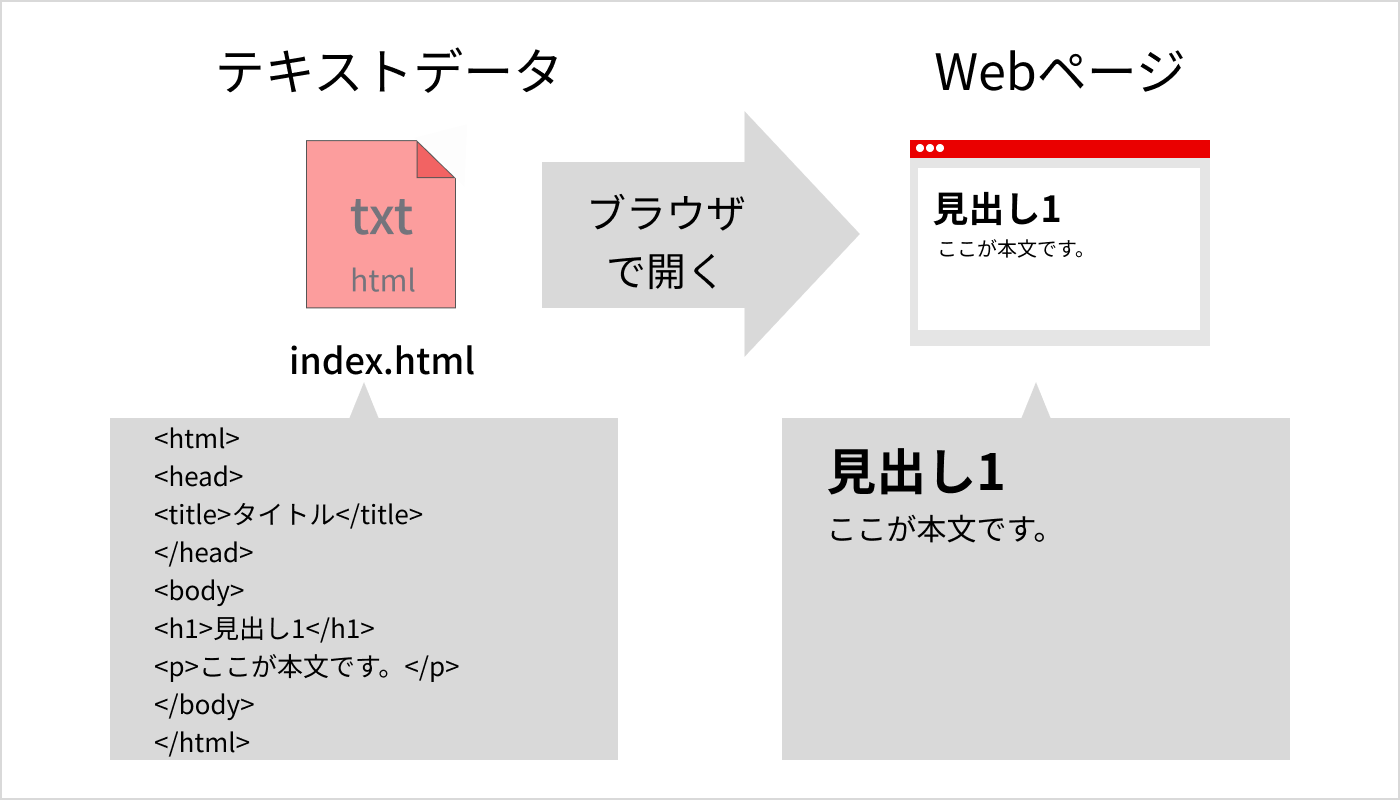
Webページを作成するときに使用するマークアップ言語

マークアップ言語とはタグを使用してコンピュータが扱う文書を記述する言語のこと

HTMLではタグを使用してWebページの論理構造や文字要素を指定する

Webページの論理構造とは、どの要素があって、どのような関係性をもっているのか示すもの

HTMLの例は、次のとおり



<title></title>で囲まれている文字がブラウザのタイトル部分に表示される

<body></body>に囲まれている文字がブラウザの本文に表示される

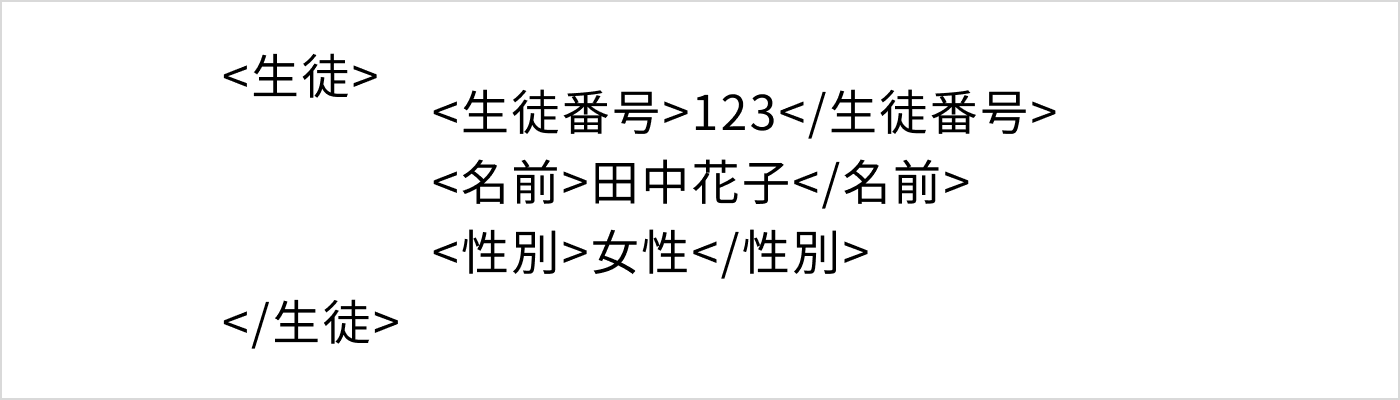
このようにHTMLタグを使うことで、Webページの文字要素や論理構造を指定できる

XML

独自で定義したタグを使用してデータを記述するマークアップ言語

「EXtensible」は拡張可能という意味、独自のタグを定義できる（拡張可能）という特徴がある

XMLの例は次のとおり



XMLタグを使用してデータを記述する、タグを利用することでWeb上で「データのやり取り」が可能になる

ＨＴＭＬの用途は「Ｗｅｂページを作成すること」

ＸＭＬ「Ｗｅｂ上でデータを交換すること」

ＸＭＬのメリットはコンピュータと人間の両方が読みやすい形式であること

ＨＴＭＬとＸＭＬの違い

| **言語** | **記述対象** | **記述方法** | **用途** | **独自タグ** |
| --- | --- | --- | --- | --- |
| HTML | 論理構造 | タグ | Webページ作成 | 使用不可 |
| XML | 論理構造 | タグ | データ交換 | 使用可 |

CSS

Webページのレイアウトやデザインを指定する言語  
HTMLで文章を作成して、CSSで文書に装飾をする

まとめ

スタック　最後に格納したデータから取り出すデータ構造

キュー　最初に格納したデータから取り出すデータ構造

アルゴリズム　問題を解決したり目標を達成したりするための手順

フローチャート　アルゴリズムの内容を視覚的にわかりやすくした図

HTML　Webページを作成するときに使用するマークアップ言語

XML　独自で定義したタグを使用してデータを記述するマークアップ言語

プログラミング基礎知識

プログラミングとは、**コンピュータへの命令を書き出す作業**です。書き出した命令のことを **「プログラム」** といいます。

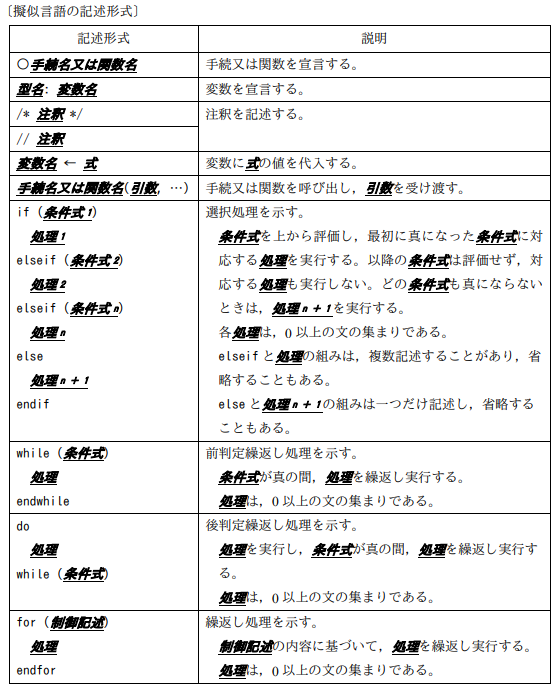
プログラム言語は、プログラムを記述するための言語です。「コンピュータに命令するための言語」ともいえます。

プログラム言語には「Java（ジャバ）」「Python（パイソン）」など、数多くの種類が存在しています。

擬似言語とは、試験問題用に考え出された「**架空のプログラム言語**」です。ITパスポート試験では、疑似言語が使用されます。

疑似言語の文法は、試験問題の中に記載されています。そのため、**疑似言語の文法を覚える必要はありません**。

疑似言語の文法例は、以下のような内容です。



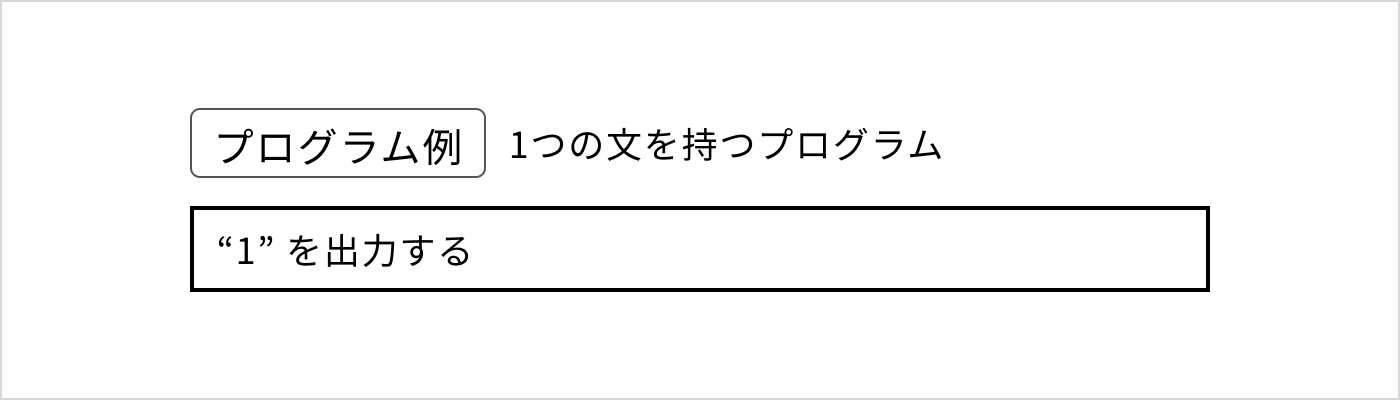
プログラミングの基本的な用語は、以下の4つです。

| **用語** | **説明** |
| --- | --- |
| 文 | プログラムの中の1つの命令 |
| 式 | コンピュータに計算の命令を出す文 |
| 条件式 | 計算結果が「真」か「偽」になる式 |
| 注釈 | プログラム内の「メモ書き」や「解説文」 |

#### **文**

文とは、**プログラムの中の1つの命令**のことです。基本的に、プログラムは複数の文で構成されています。プログラムに書かれた文を、コンピュータは上から順番に実行します。

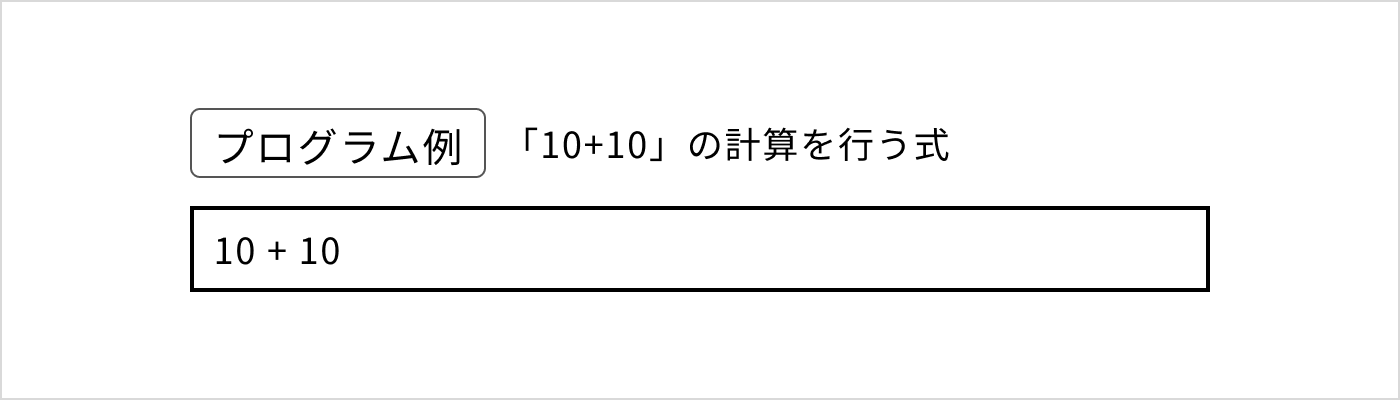
文の例は、以下のとおりです。ディスプレイやプリンタといった**出力装置に「1」という数字を出力させる命令**になります。



#### **式**

式とは、**コンピュータに計算の命令を出す文**のことです。

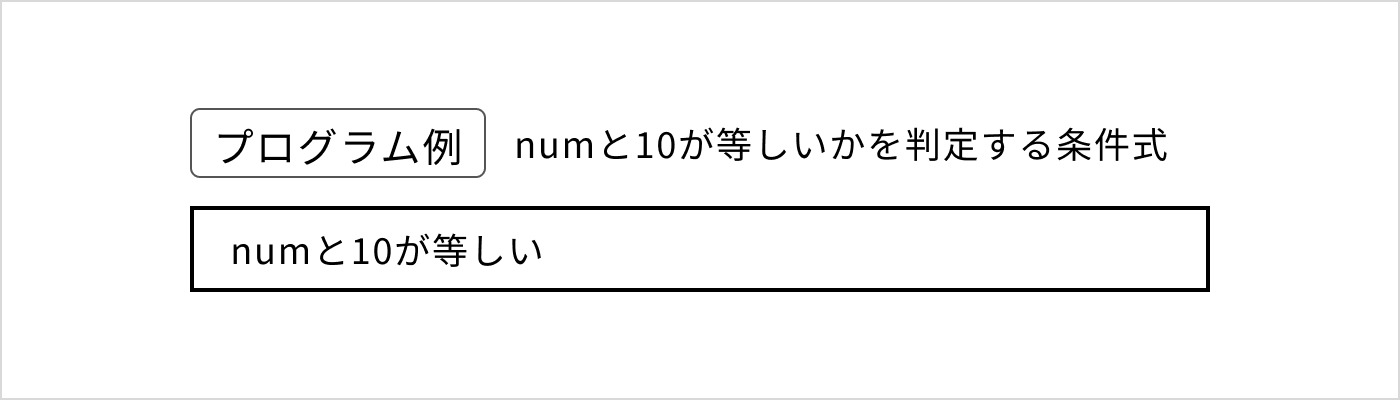
式の例は、以下のとおりです。「10+ 10」の計算を行う式になります。



#### **条件式**

条件式とは、**計算結果が「真」か「偽」になる式**です。式の一種になります。

条件式の例は、以下のとおりです。「num」と「10」が等しいか、等しくないかを判定する条件式となります。「num」は「number」の略語です。

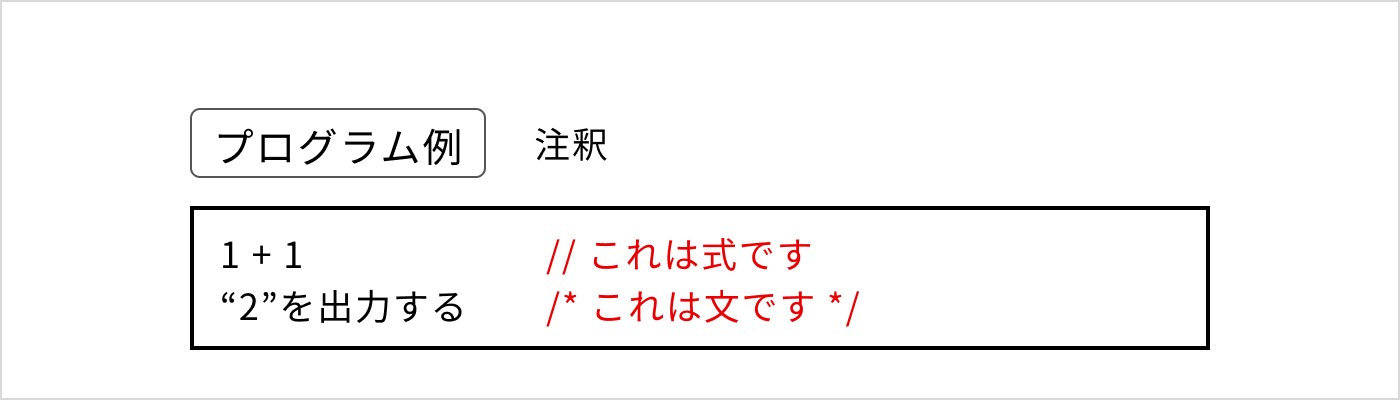


ITパスポート試験では模擬言語を使用するので、上記のように、条件式は文章で書きます。プログラミング経験者には違和感があるかもしれませんが、この形式に慣れることが大切です。

#### **注釈**

注釈とは、**プログラム内の「メモ書き」や「解説文」** のことです。注釈の対象は人間なので、**コンピュータは注釈を無視**します。

注釈の例は、以下のとおりです。色つきの部分が注釈となります。



注釈の書き方には、基本的な2パターンがあります。

* 注釈の前に「//」を書く方法。それ以降の文字が注釈となる
* 「/\*」と「\*/」で囲む方法。囲まれた部分が注釈になる。複数行をまとめて注釈にしたいときに便利

プログラミングの基礎用語が、ITパスポート試験で直接出題されることはありません。しかし、プログラミングには欠かせない知識なので、ここで確実に理解しましょう。

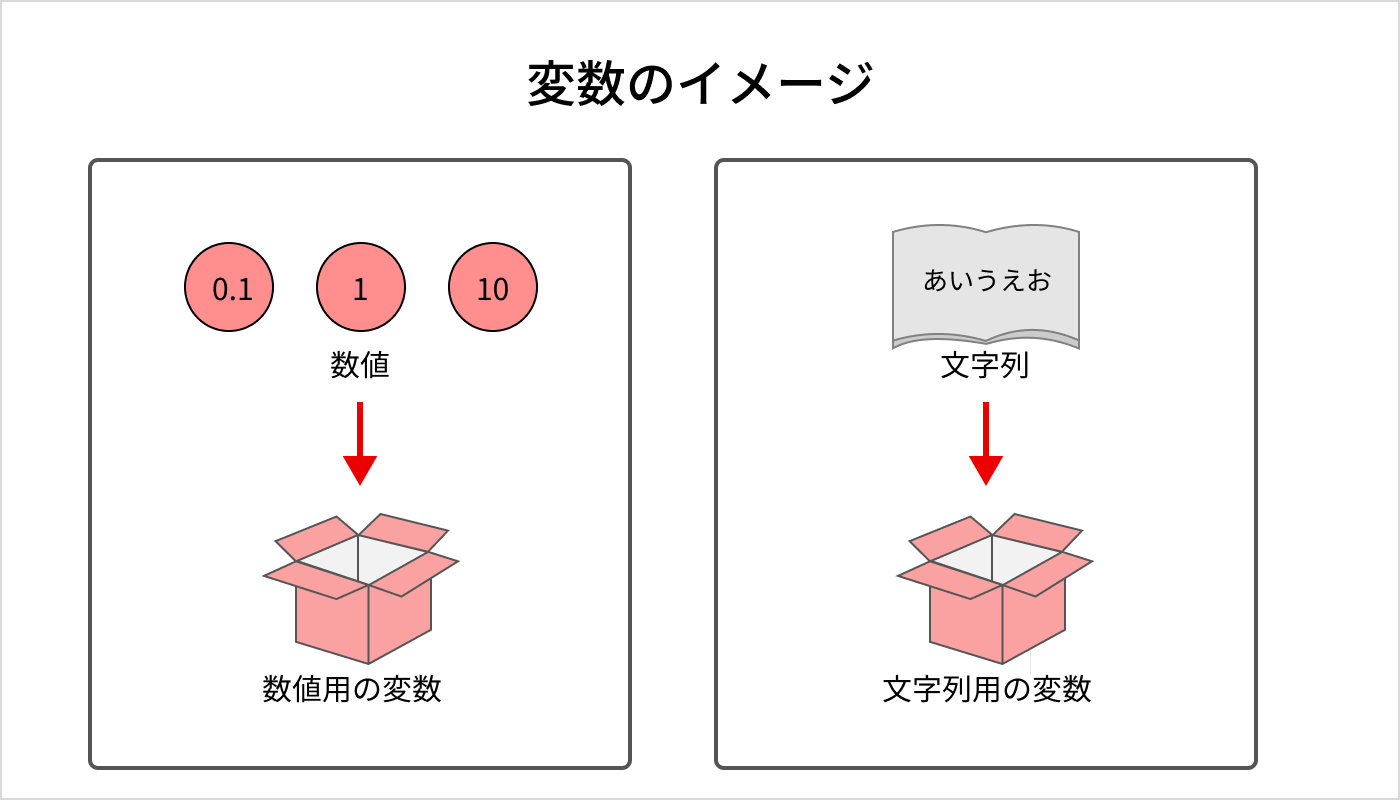
次に、**プログラミングにおいて最も重要な「変数」** について説明します。

### **変数**

変数とは、**数値や文字列などの「値」をコンピュータの主記憶装置に格納する機能**です。

イメージは、コンピュータ内に用意された「値を入れる箱」です。

プログラムで扱う「値」を変数に入れることで、**「値を一時的に保存できる」「他のプログラムに渡せる」** ことが可能です。

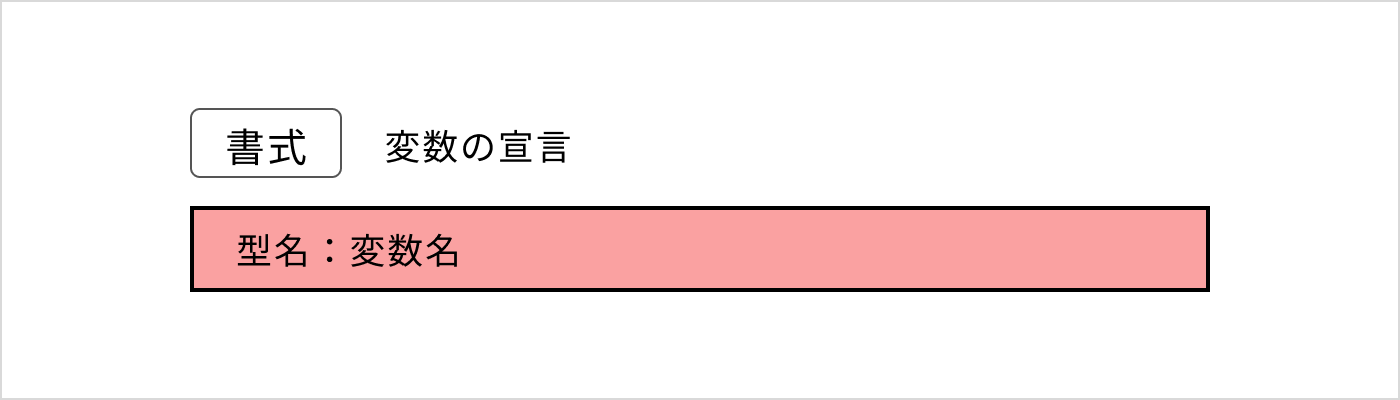


#### **変数の宣言**

変数を利用する場合、**変数の種類（整数や文字列）と「名前」をコンピュータに伝える必要**があります。これが「**変数の宣言**」です。

変数の宣言の書式は、以下のとおりです。

* 最初に型名（種類）を書く
* 次に「：」（コロン）で区切り、変数名（名前）を書く



コンピュータに変数（箱）の名前を知らせることで、コンピュータは他の変数（箱）と区別できるようになります。

下の図は、「整数型のnumという名前」の変数の宣言です。

「型」については、次項で詳しく解説します。繰り返しになりますが、「num」は「number」の略語です。

変数名はプログラムを書く人が自由に決められますが、**型名はプログラミング言語ごとに決められています。**



型とは、**整数や文字列といった値の「種類」** です。

変数を宣言するときに「型」を指定することで、指定した値だけ格納できるようになります。

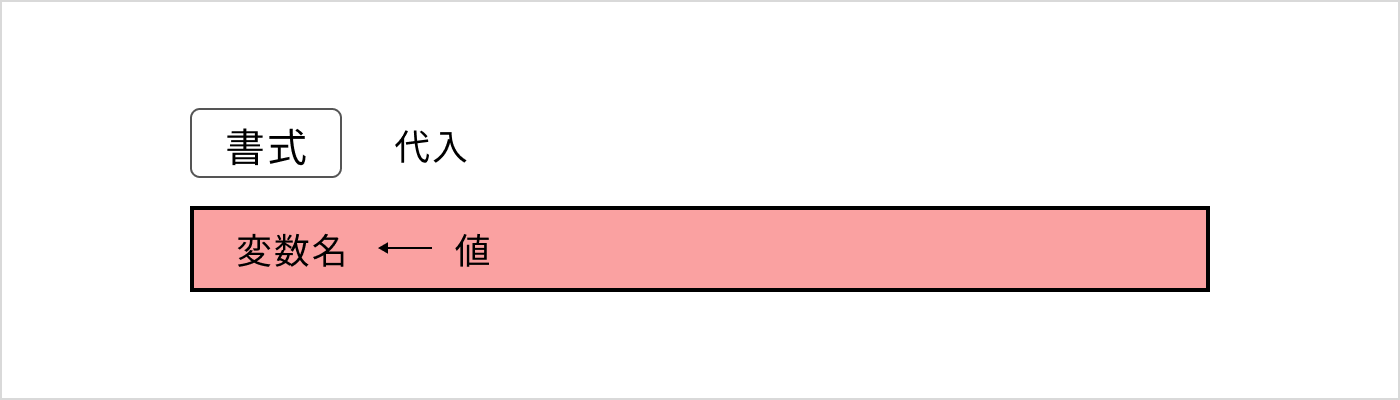
主な「型」の種類は、以下のとおりです。

| **型** | **説明** | **具体例** |
| --- | --- | --- |
| **整数型** | 整数の値 | -1、0、1 |
| **実数型** | 小数点を含む値 | 1.23、-0.1 |
| **文字列型** | 文字を並べた値 値を「”」（ダブルクォート）で囲んで格納する | “あいうえお” |

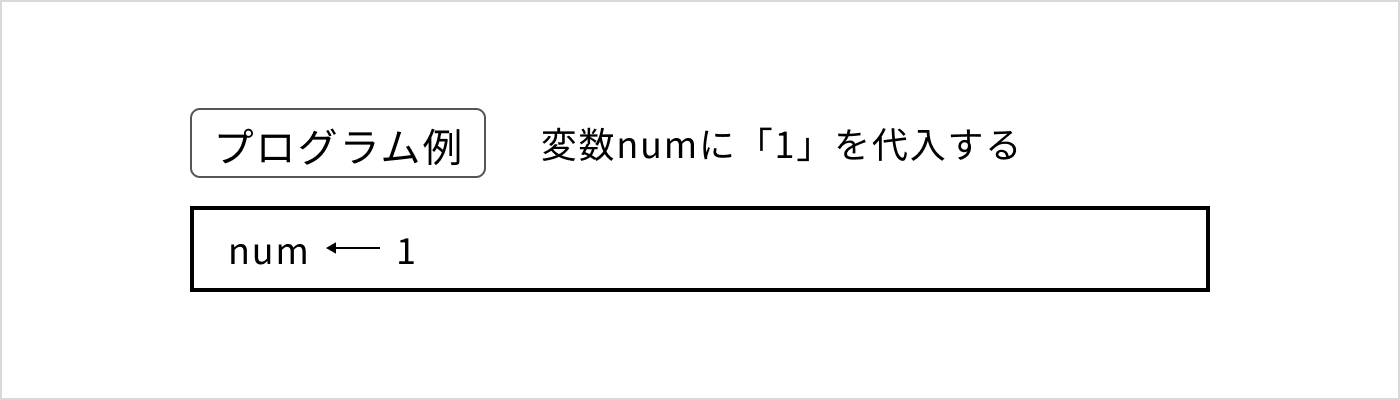
代入とは、**変数に値を入れること**です。変数は宣言するだけだと意味がなく、代入することで役に立ちます。

変数に値を代入する書式は、以下のとおりです。**代入には矢印（←）を使用します**。

* 最初に「値」を格納する「変数の名前」を書く
* 次に、左向きの矢印「**←**」を書いて、右側に「代入する値」を書く



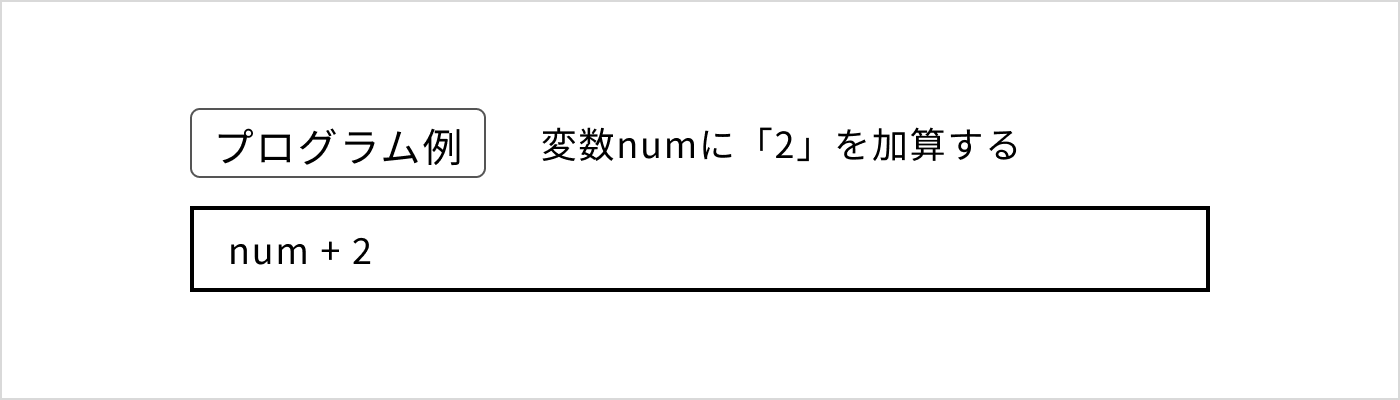
下の図は、「変数numに1を代入する」プログラムです。



**変数に代入された値は、コンピュータの主記憶装置に保存されます。** そのため、以降のプログラムでは、変数「num」を何度も使用可能です。

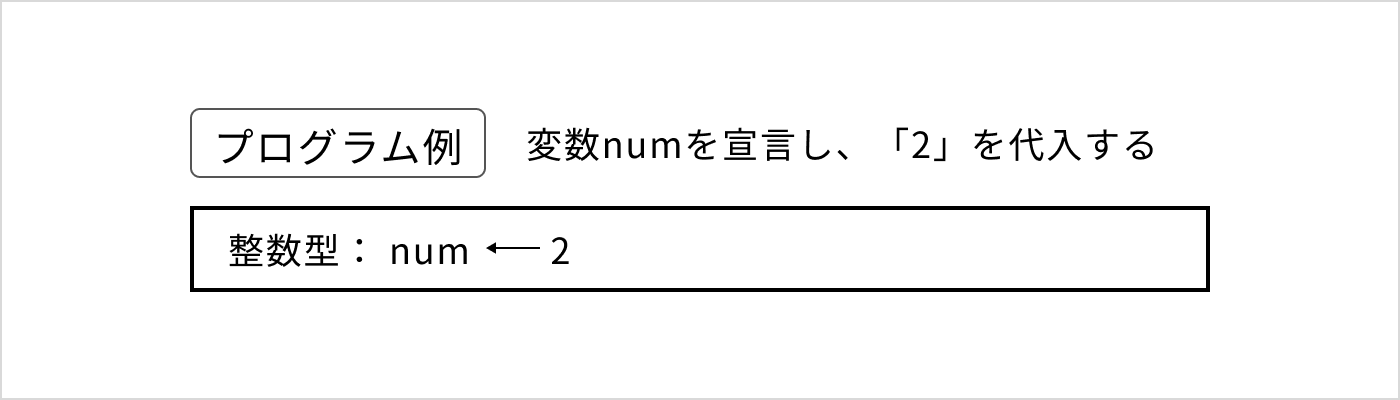
#### **他の値との組み合わせ**

**変数は、他の値と組み合わせることが可能**です。例えば、「変数num」に「2」を加算するときの書き方は、以下のとおりです。



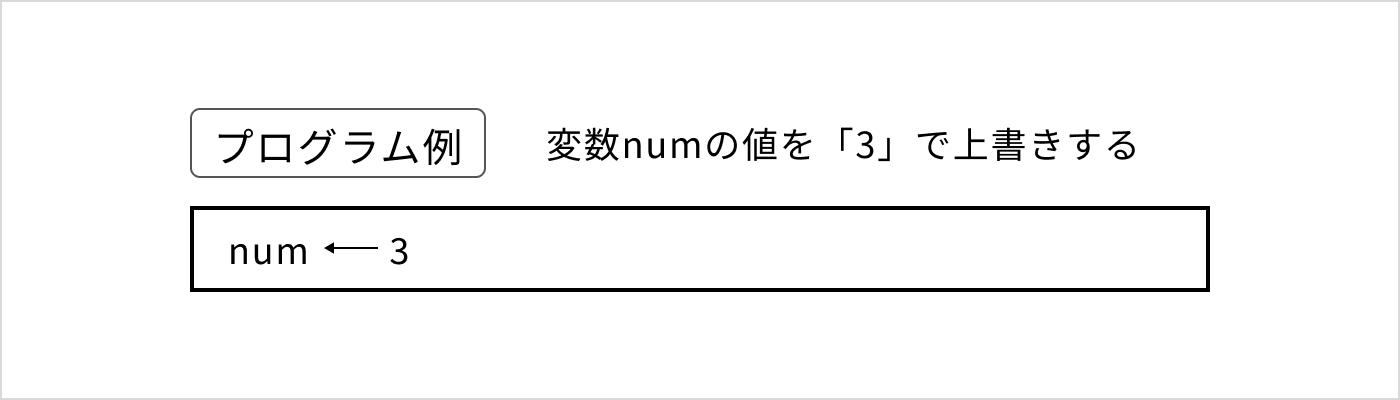
#### **初期化と上書き**

「変数の宣言」と「値の代入」は、同時に行えます。ITパスポート試験で出題されやすい書き方です。



上記のように、**値がまだ何もない変数に「初期値」を代入すること**を「初期化」といいます。

また、**すでに値がある変数に「別の値」を代入すること**を「上書き」といいます。上書きすると、元の値はなくなります。



| 文 | プログラムの中の1つの命令 |
| --- | --- |
| 式 | コンピュータに計算の命令を出す文 |
| 条件式 | 計算結果が「真」か「偽」になる式 |
| 注釈 | プログラム内の「メモ書き」や「解説文」 |
| 変数 | 数値や文字列などの「値」をコンピュータの主記憶装置に格納する機能 |

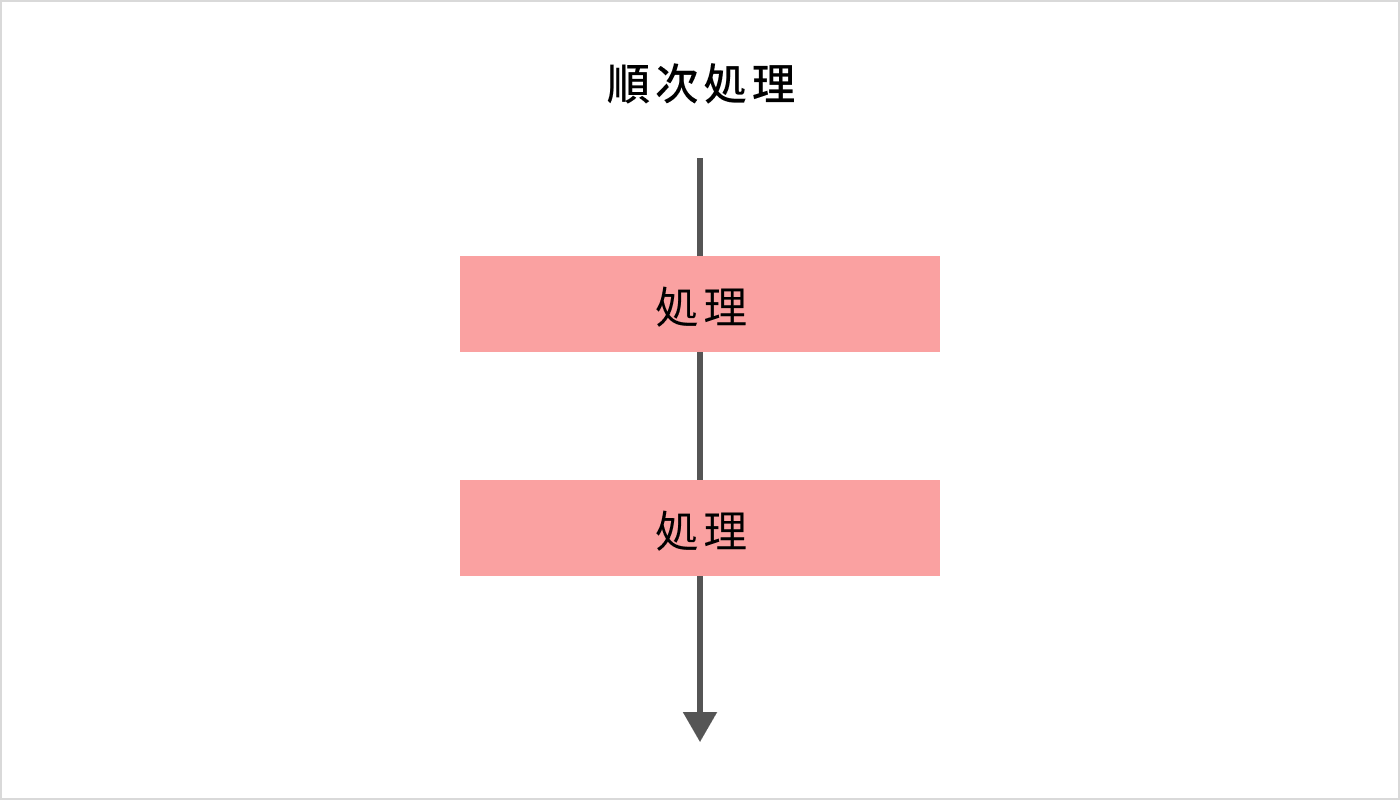
プログラムの処理の流れは、「順次処理」「選択処理」「繰り返し処理」の3種類です。

| **処理の流れ** | **説明** |
| --- | --- |
| 順次処理 | プログラムを**上から順に実行**する |
| 選択処理 | 指定した条件に**一致**した場合のみ、**処理を実行**する |
| 繰り返し処理 | 指定した条件に**一致**した場合のみ、**処理を繰り返し実行**する |

表の説明を見て「意外に簡単」と思われた方も多いのではないでしょうか。

それでは、「順次処理」から順に説明します。

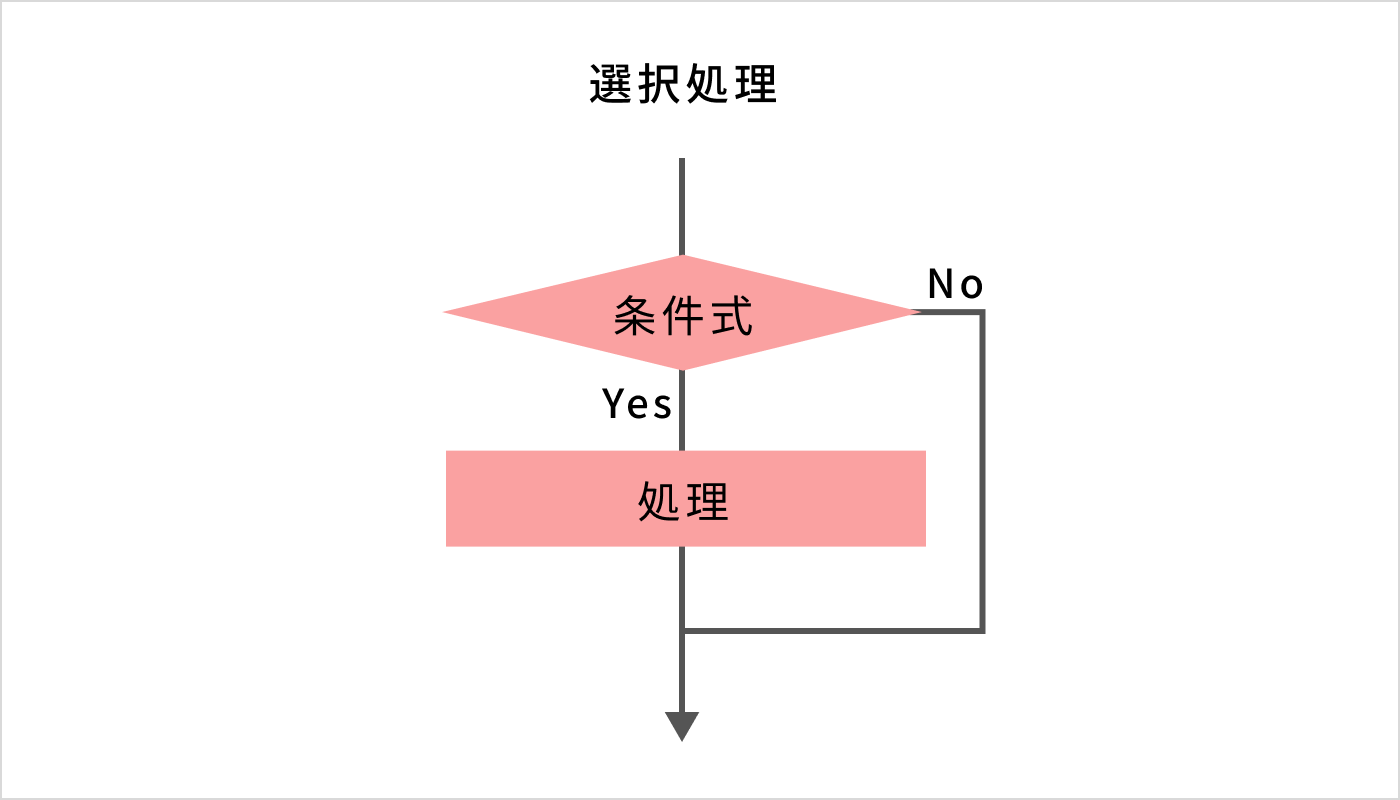
順次処理とは、**コンピュータがプログラムを上から順に実行していくこと**です。プログラムは基本的に上から下に書かれているので、上から順に実行します。

「選択処理」や「繰り返し処理」を指定しない場合は、順次処理が実行されます。

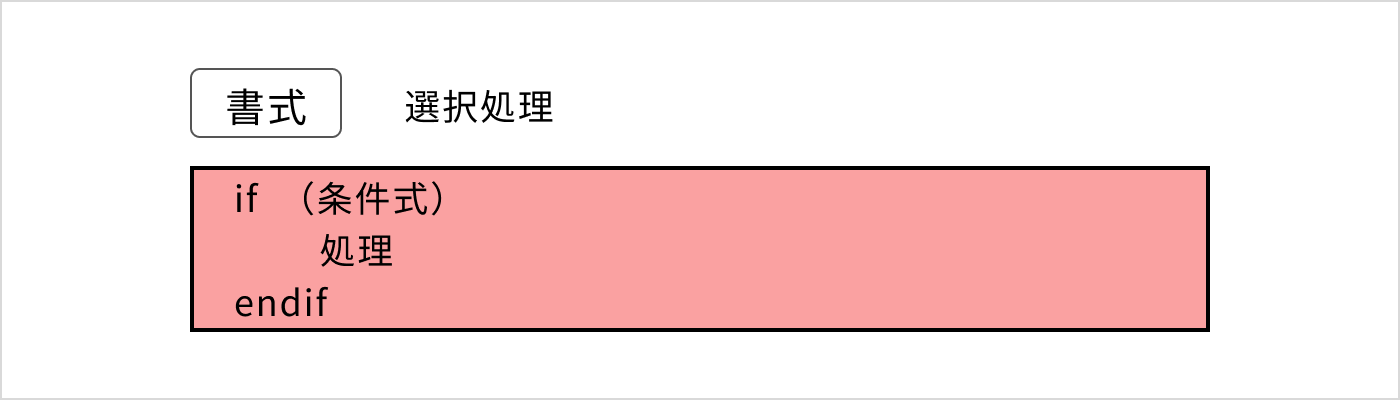
選択処理とは、**指定した条件に一致した場合のみ、処理を実行すること**です。条件に一致しない場合、処理は実行されません。

選択処理を、カレーライスの調理に例えます。

* 具材を煮るときに「あくが出たらあく取りをする」という**条件を指定**
* 「あくが出た」という**条件に一致した場合**、「あく取り」という**処理を実行**
* あくが出ない場合、**条件に一致しない**ので「あく取り」は**実行されない**



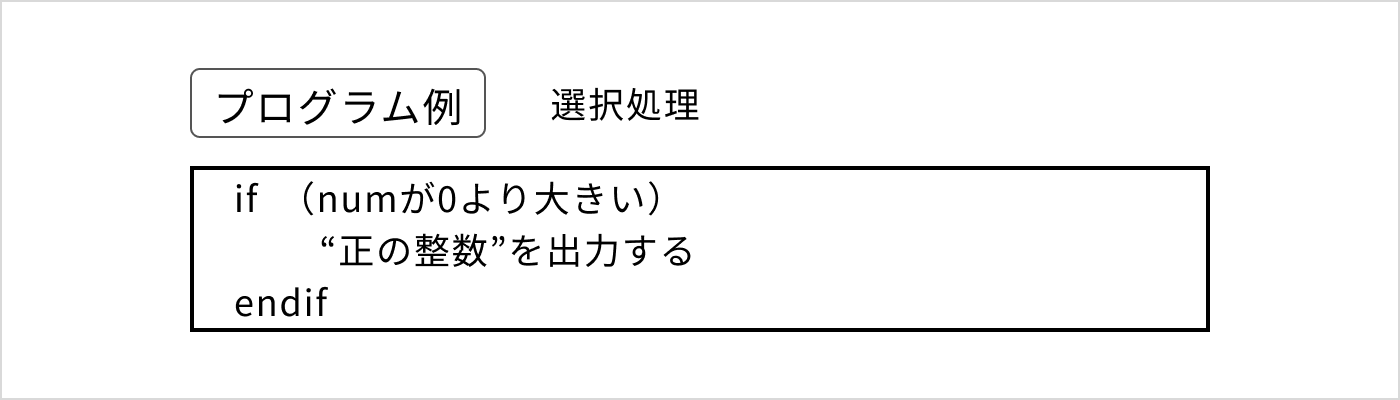
選択処理では、「**if**」を使用します。「if」を使うことで、**処理を分岐**させることが可能です。条件に一致している場合に実行する処理を「if」と「endif」の間に記述します。



* まずは「if」を書き、右に「（）」（丸括弧）を書く
* （）の中に「条件式」を書く
* 次に「処理」を書き、最後に「endif」を書く

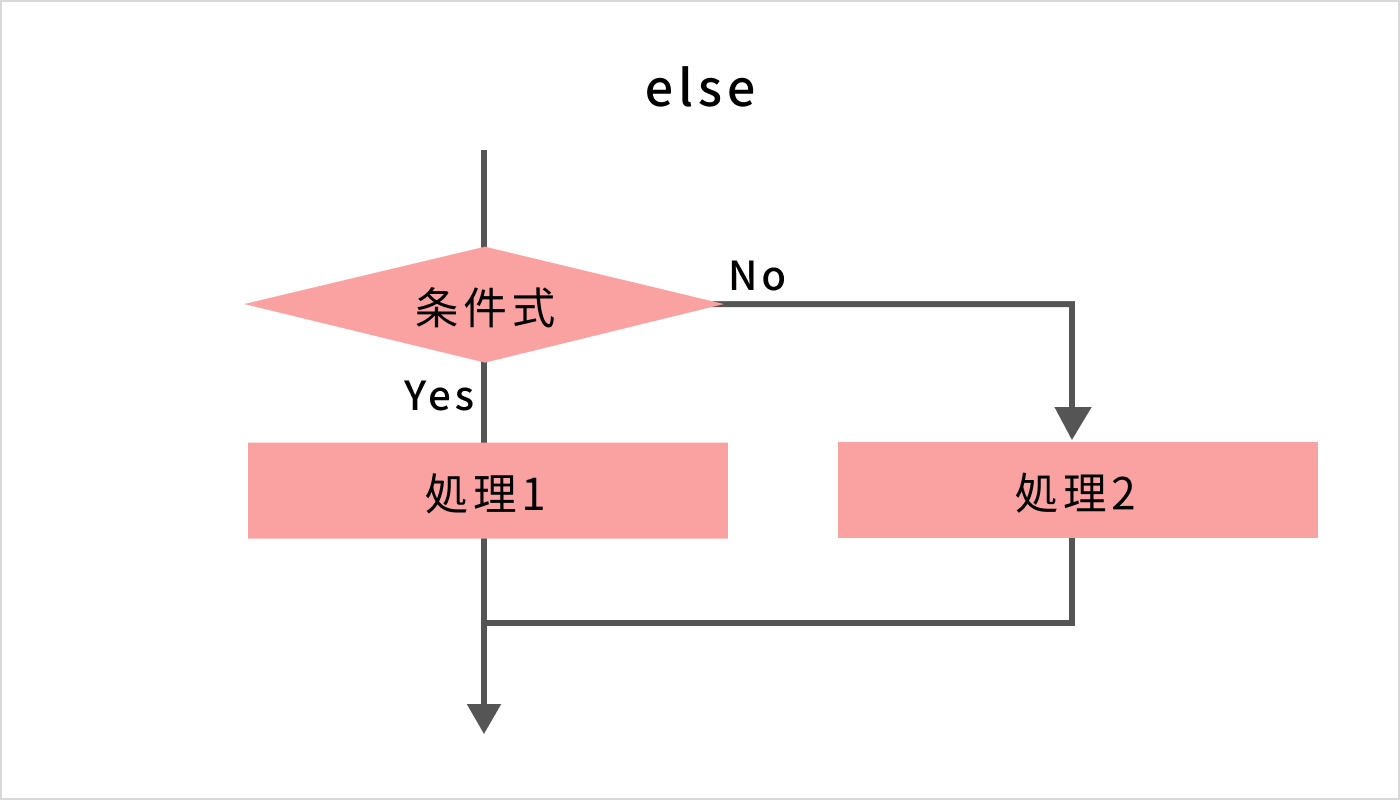
条件式の結果が「真」の場合のみ、「if」と「endif」の間に書かれた処理を実行します。

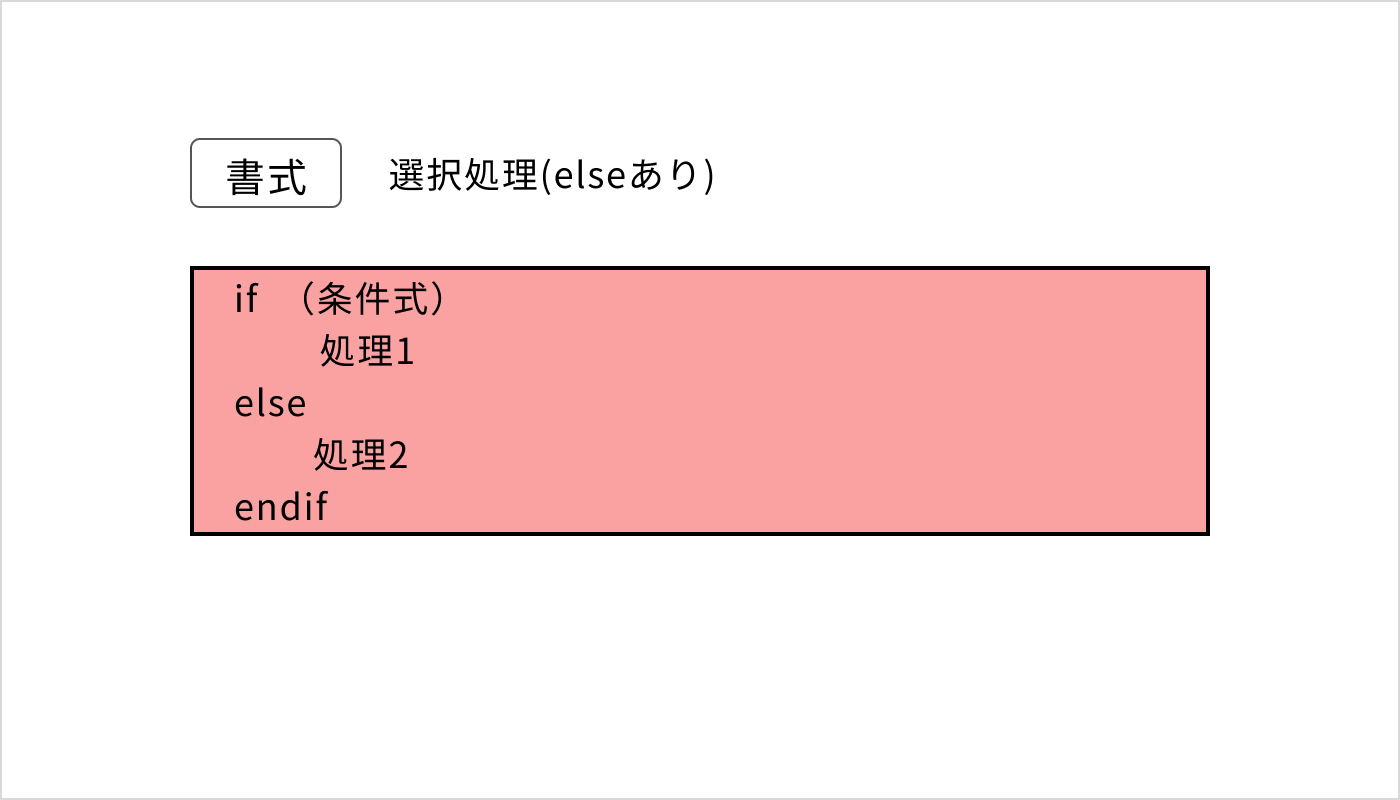
以下の例は、変数numの値が「0」よりも大きい場合のみ、「正の整数」と出力するプログラムです



「else」は、**指定した条件に一致しない場合にも、処理を実行するときに使用**します。

「else」を使用することで、条件式の結果によって、違う処理を実行できます。



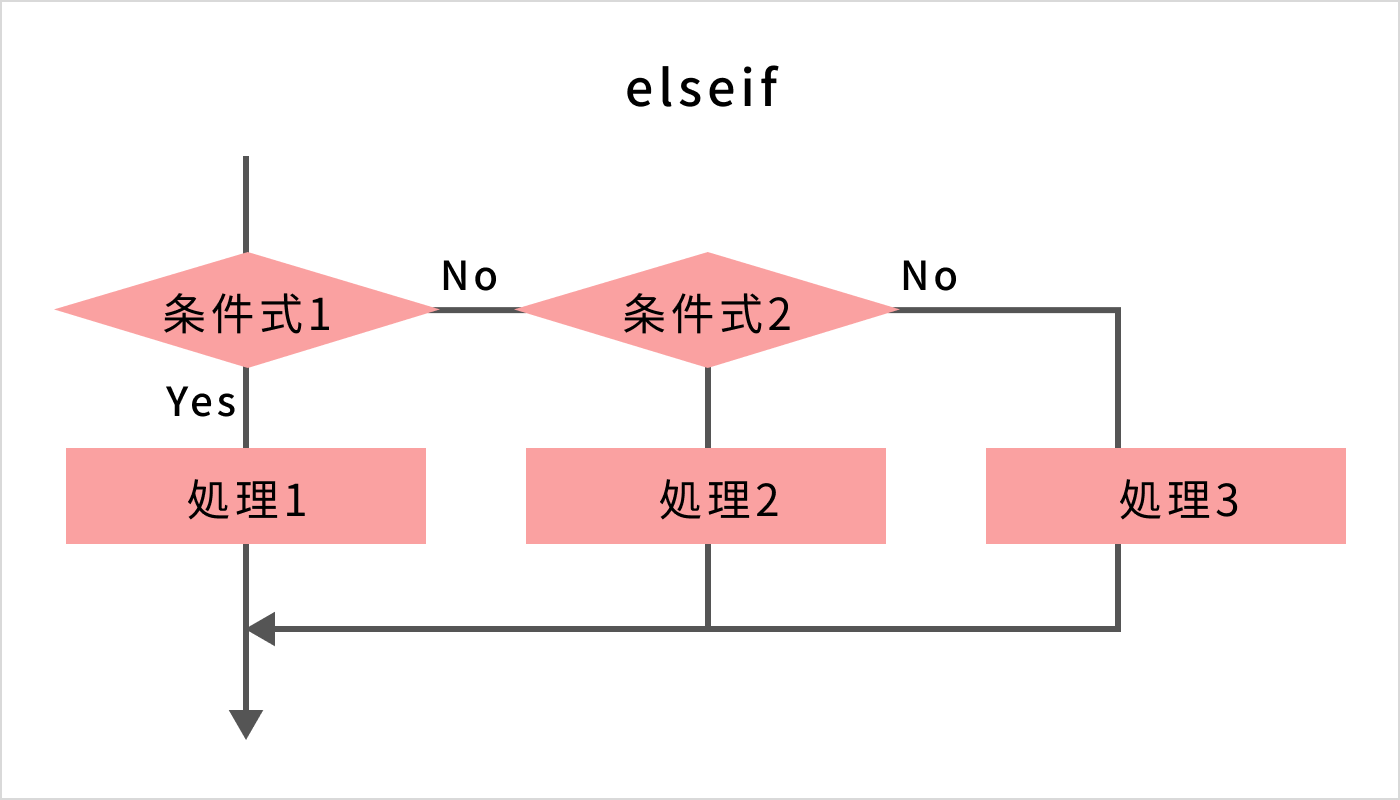


以下の例では、処理を2つに分岐させています。

* 変数numの値が「0」よりも大きい場合は「正の整数」と出力
* それ以外の場合は「負の整数か0」と出力



「elseif」は、**複数の条件を設定**するときに使用します。

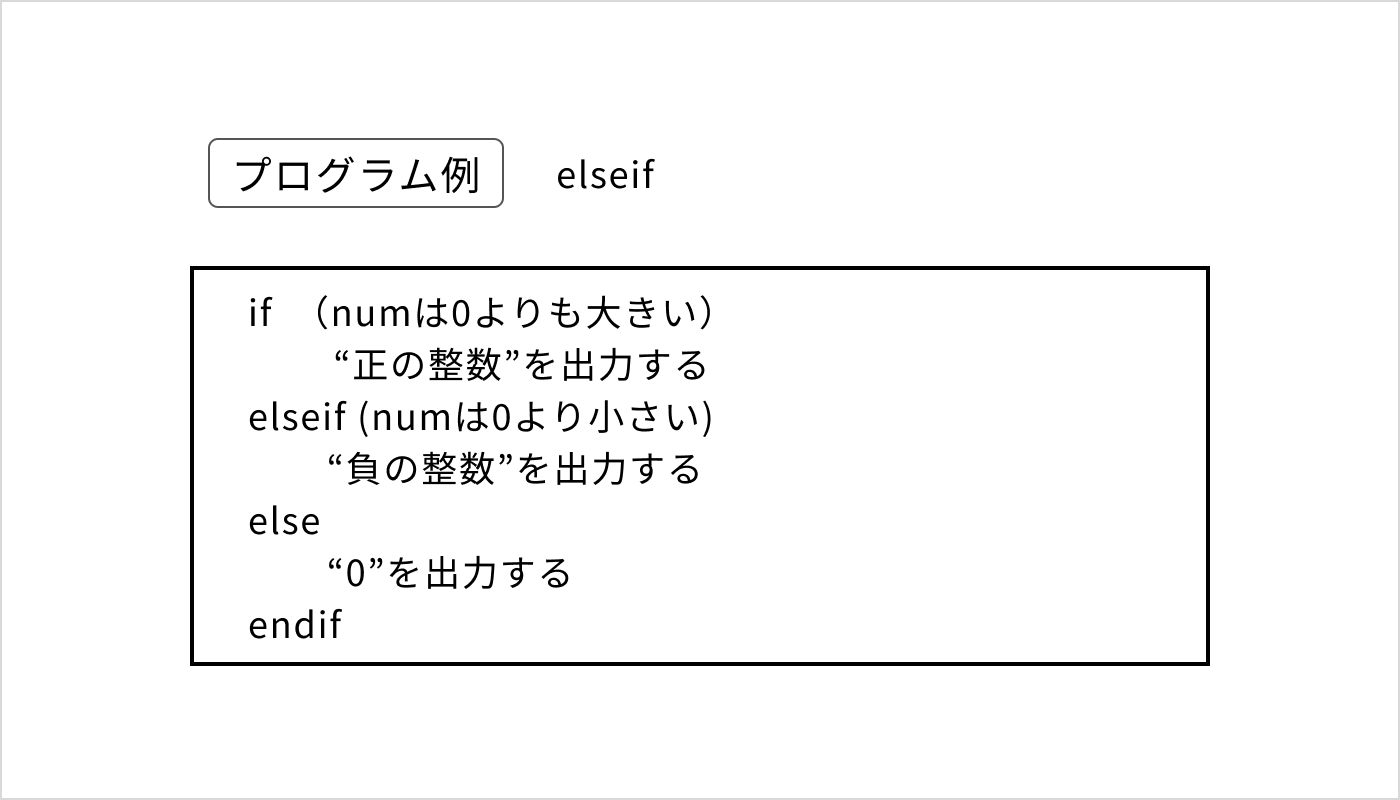


「elseif」を使用した選択処理の書式は、以下のとおりです。



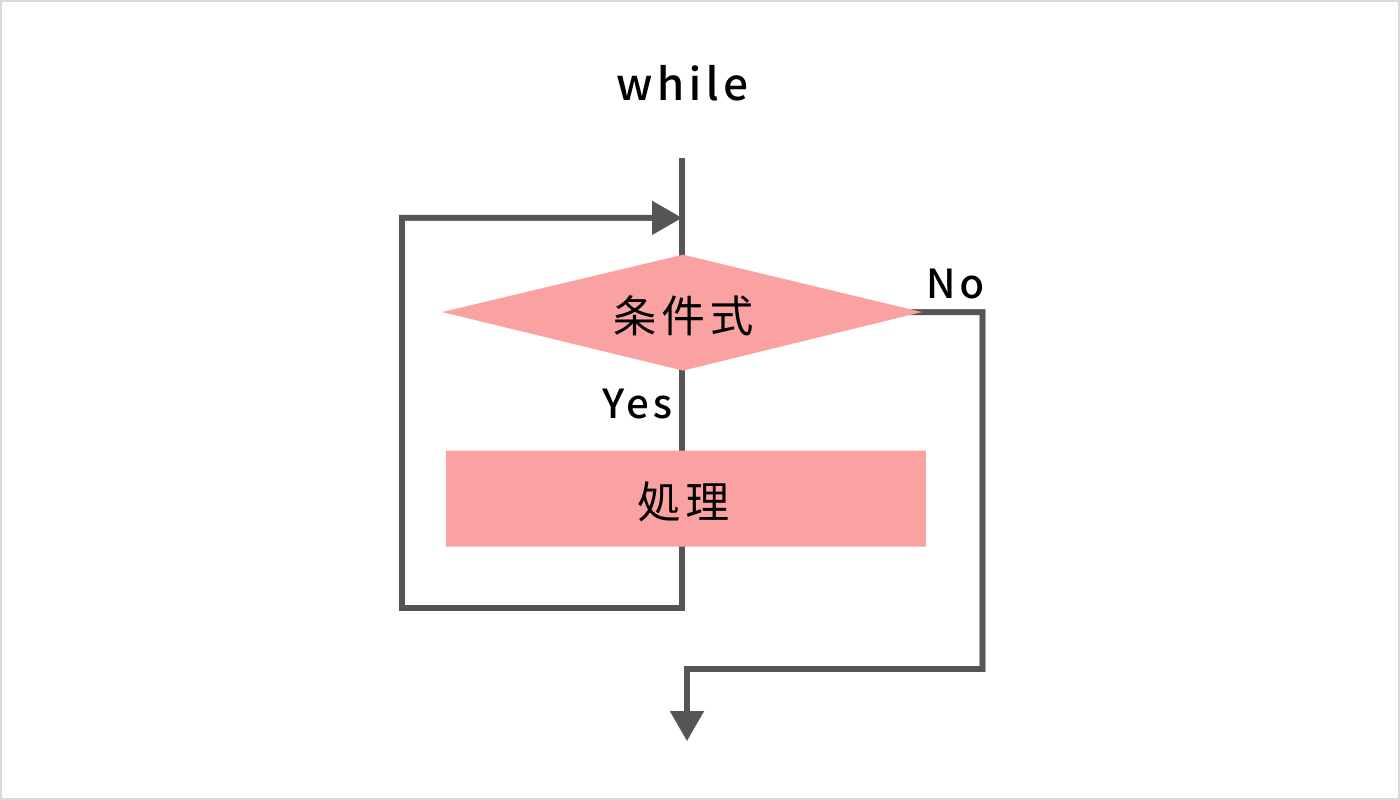
下の例では、3つの処理に分岐させています。

* 変数numの値が「0」よりも大きい場合は「正の整数」と出力する
* 変数numの値が「0」よりも小さい場合は「負の整数」と出力する
* それ以外の場合は「0」と出力する

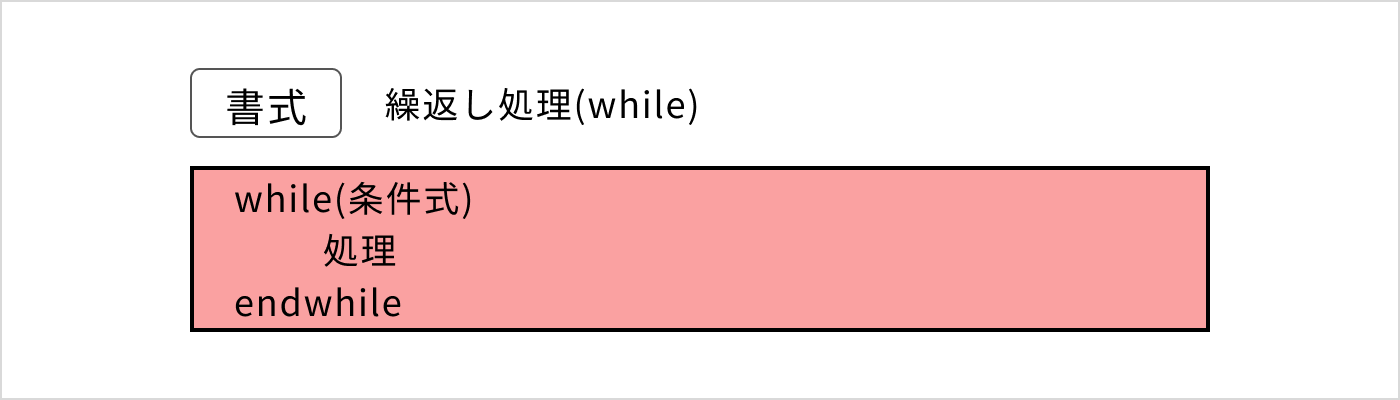
「elseif」を使用し、変数numの値によって3つに分岐させていることがわかります。

繰り返し処理とは、**ある条件に一致した場合のみ、処理を繰り返し実行すること**です。繰り返し処理では「**while**」を使用します。

下図のように、**条件が一致している間は、ずっと同じ処理を繰り返し実行**します。条件に一致しない場合は、処理を行いません。

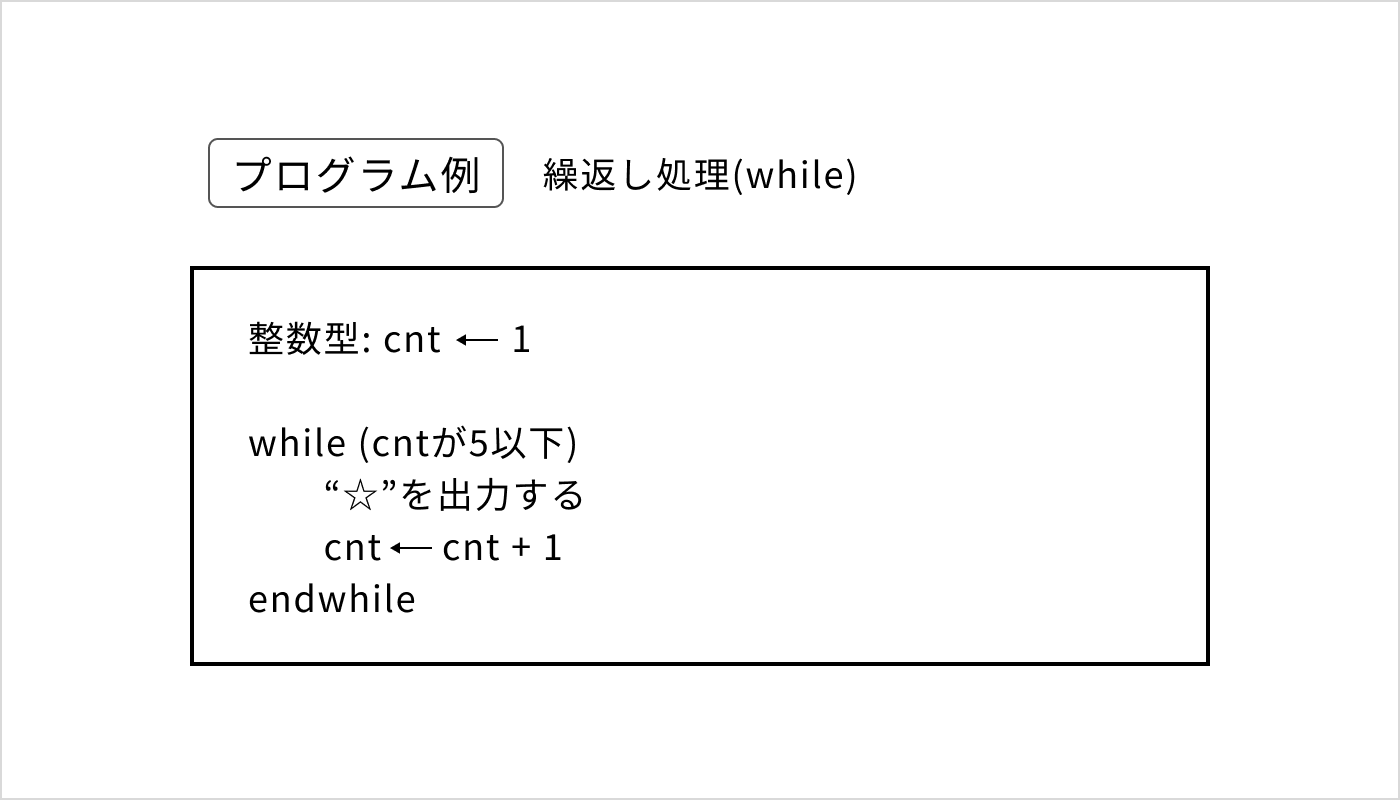


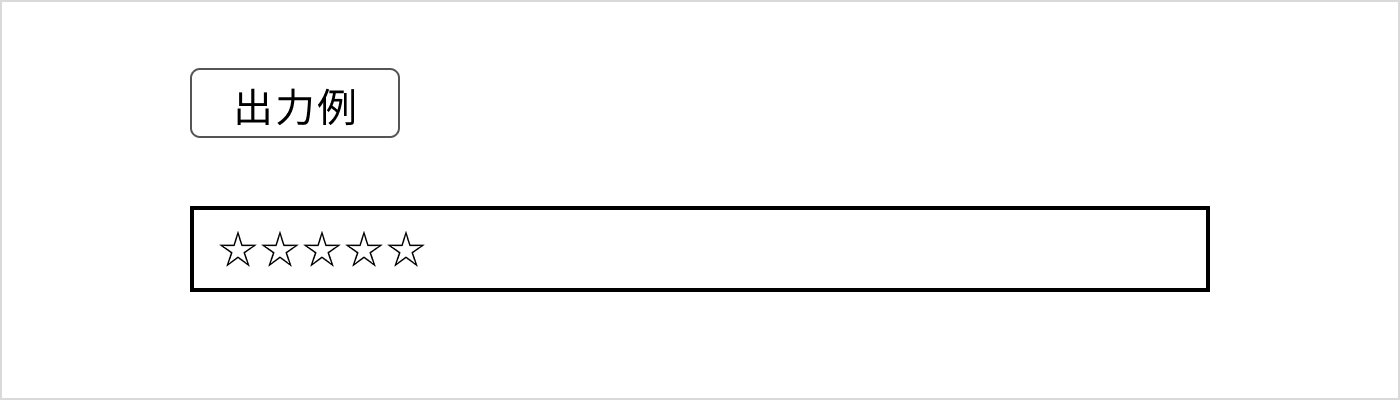
「**while**」の書式は、以下のとおりです。



条件式の結果が「真」であれば、「whileとendwhile」の処理をずっと繰り返し実行します。

以下の例は、「☆マーク」を5個出力するプログラムです。5回の繰り返し処理をしています。





下表にて、プログラムの内容を解説します。

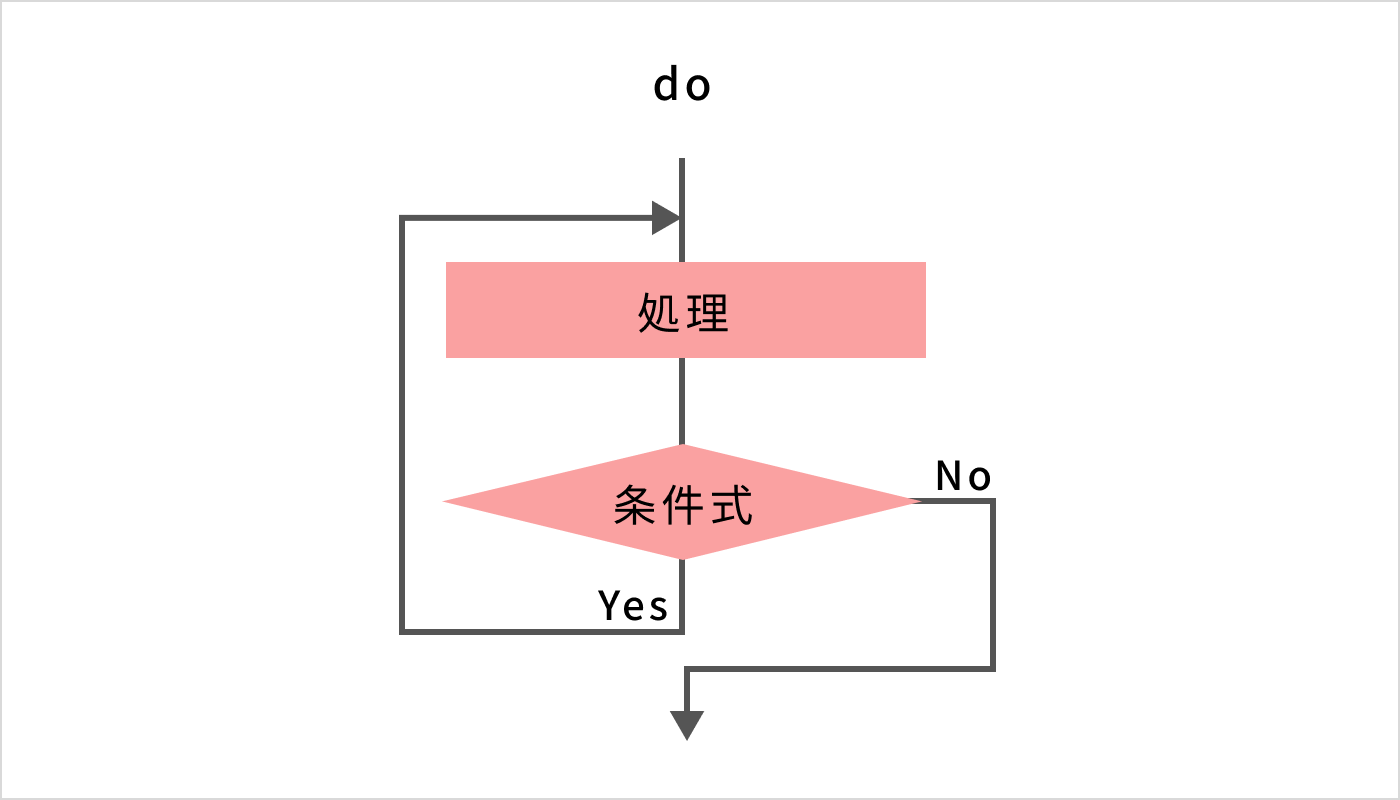
| **プログラム** | **説明** |
| --- | --- |
| 整数型：cnt ← 1 | 整数型の変数cntを宣言して「1」で初期化 |
| while (cntが5以下) | 変数cntの値が「5以下」の場合、処理を繰り返し実行する |
| “☆”を出力する | 「☆マーク」を出力する処理 |
| cnt ← cnt + 1 | 変数cntに「1」を加算し、加算した値で上書き 例えば、変数cntが2の場合、2に1を加算し、変数cntを3で上書き |
| endwhile | whileの処理が終わる |

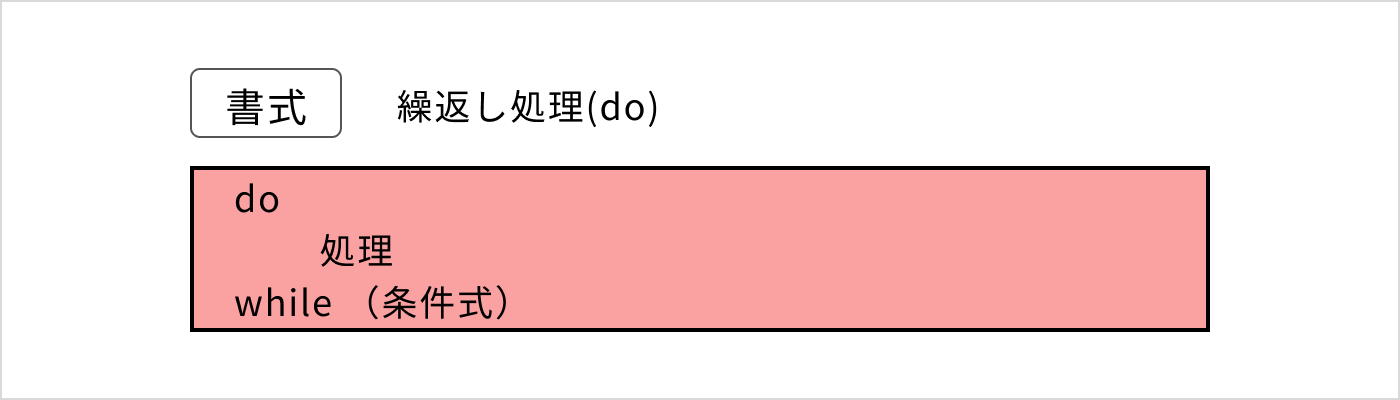
変数cntの値が5になると、条件式（cntが5以下）に一致しなくなるので、繰り返し処理が終わります。

次は、whileに関連する「do」について解説します。

「do」は、先に処理を行い、その後に**条件を判断するときに**使用します。「do」も「while」と同じく、繰り返し処理です。

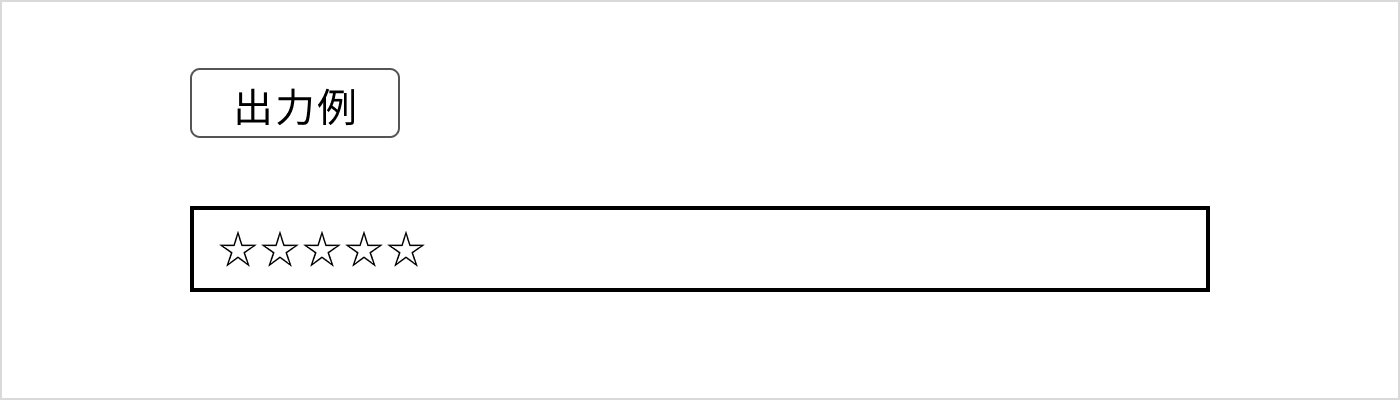
「while」と「do」は、「処理」と「条件式」の位置が逆になります。





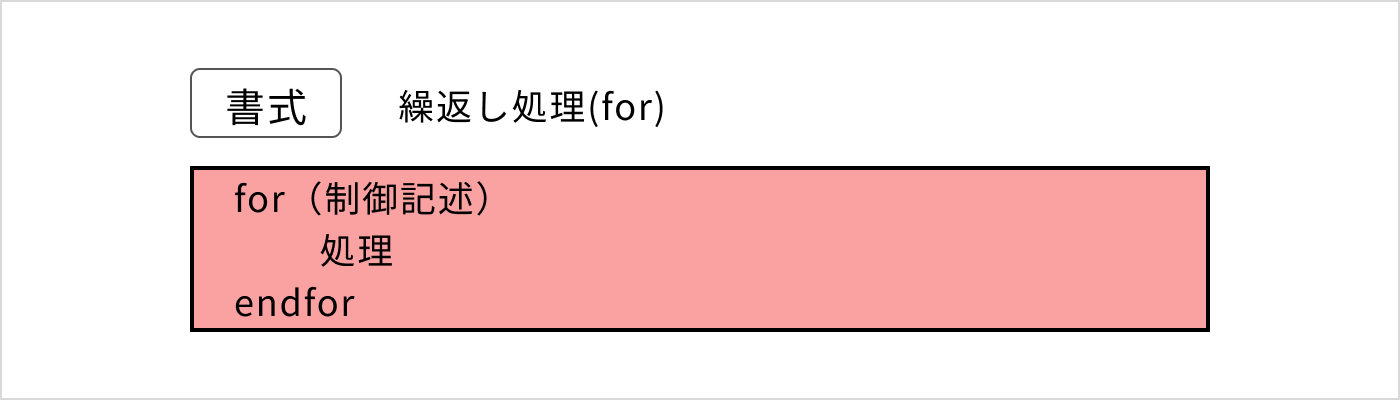
以下の例は、「while」の例と同じように「☆マーク」を5個出力するプログラムです。出力結果も、「while」の例と同じです。





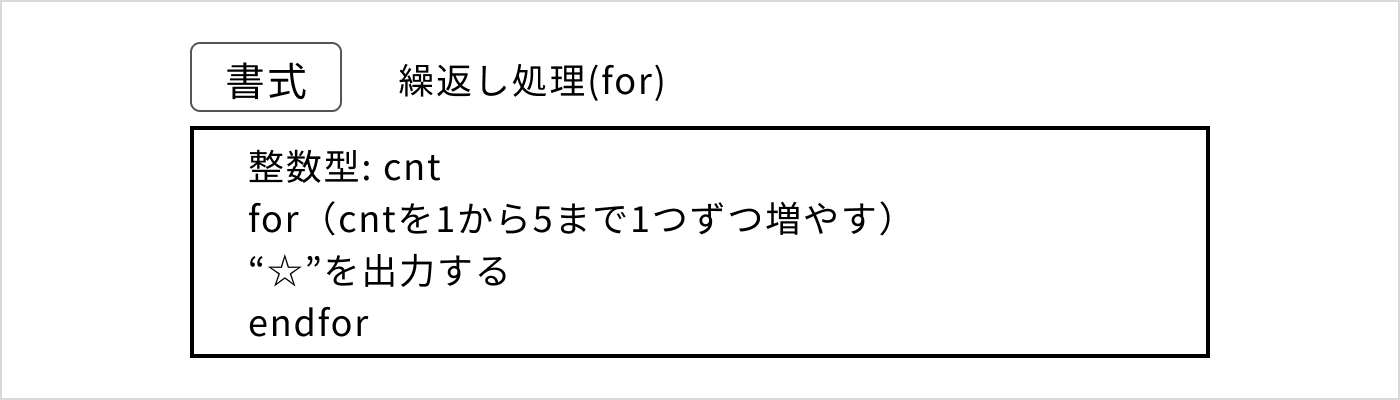
「while」と「do」は、**処理の中**で変数の値を指定します。

「for」では、**条件式の中**で、変数の値を指定できます。



制御記述に、**繰り返し処理を行う条件**を書きます。

例は、「while」と「do」と同じく「☆マーク」を5個出力するプログラムです。もちろん、出力結果も同じです。

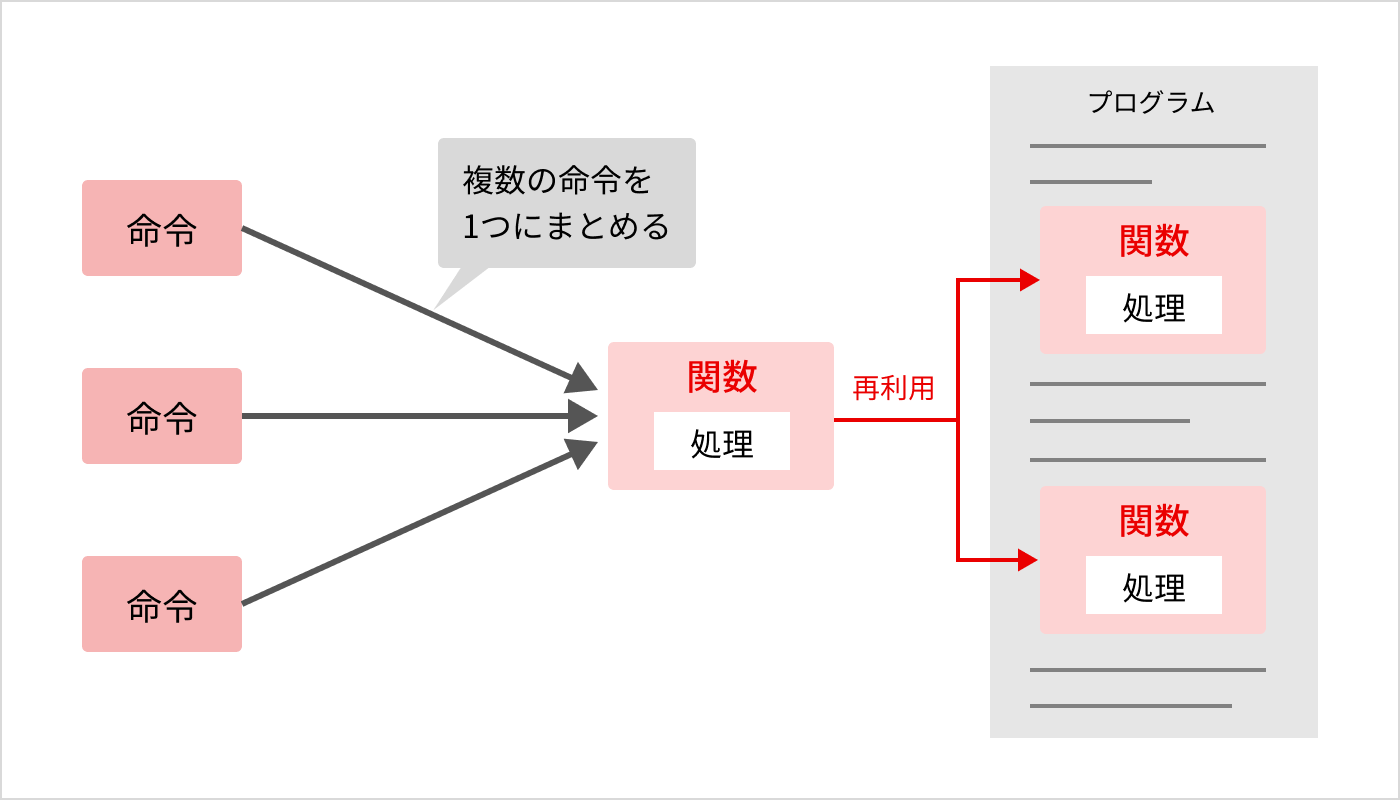


下表にて、プログラムの内容を解説します。

| **プログラム** | **説明** |
| --- | --- |
| 整数型：cnt | 整数型の変数cntを宣言 |
| for (cntを1から5まで1つずつ増やす) | 変数cntの値に「1」を加算し、値が5になるまで処理を繰返す |
| “☆”を出力する | 「☆マーク」を画面に出力する |
| endfor | forの処理が終わる |

繰り返しになりますが、「関数」とは、**複数の命令を1つにまとめて処理入力できるツール**です。命令の内容によって、関数の処理は異なります。

関数のイメージを図にすると以下のとおりです。

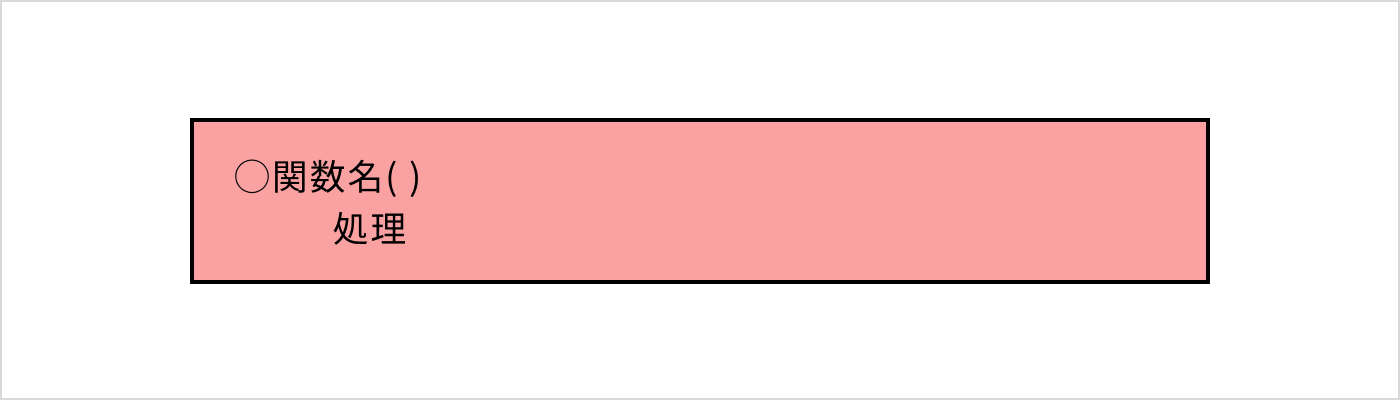


関数を使用する時は、2つのステップ「**関数の定義**」と「**関数の呼び出し**」が必要です。「関数の定義」から順に説明します。

関数を作成する1つ目のステップは、関数の定義です。**複数の命令を1つにまとめるステップになります。**

関数を定義するときは、最初に「**〇**」を使用します。

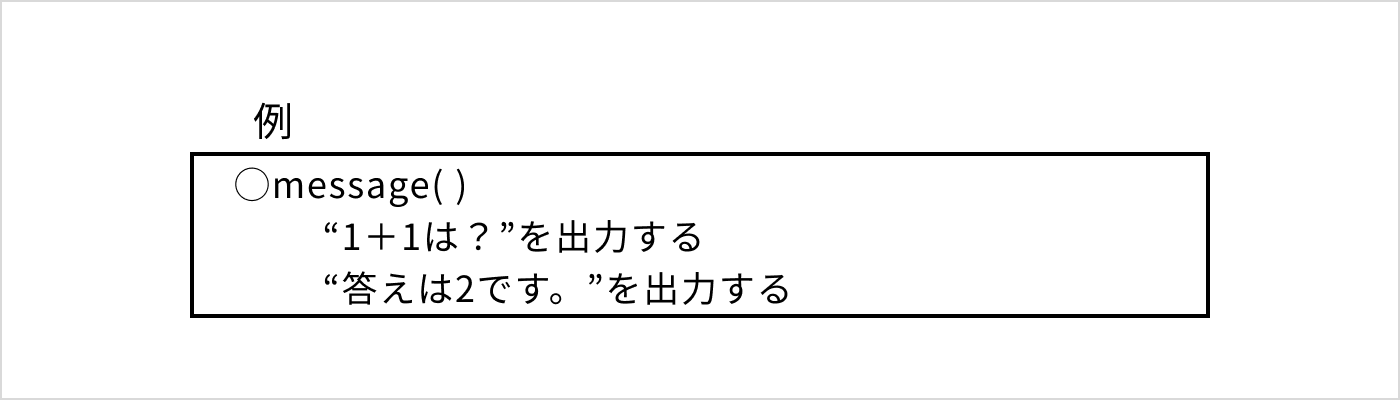
関数の定義の書式は、以下のとおりです。



* 最初に「〇」を書いて、その後に「関数名」と「( )」を書く
* 次に「処理」を書く

関数名は、プログラムを書く人が自由に決められます。関数名を書いた後は、実行したい処理をまとめて記述します。

以下の例は、「1＋1は？」「答えは2です。」という2つのメッセージを1つにまとめて出力する関数の定義です。関数名は「message」としました。

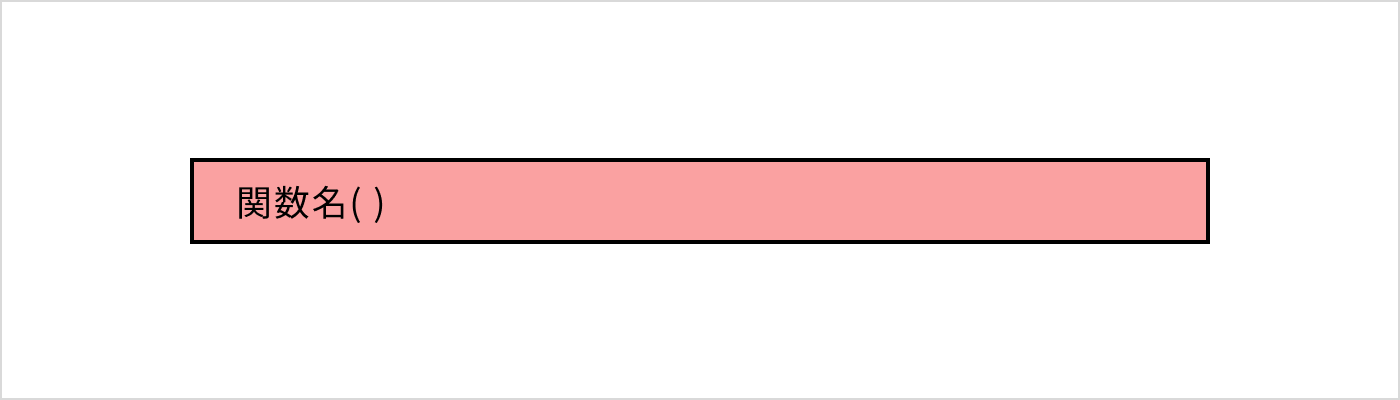


これで、関数messageの定義が完了しましたが、**関数を定義しただけでは実行されません**。

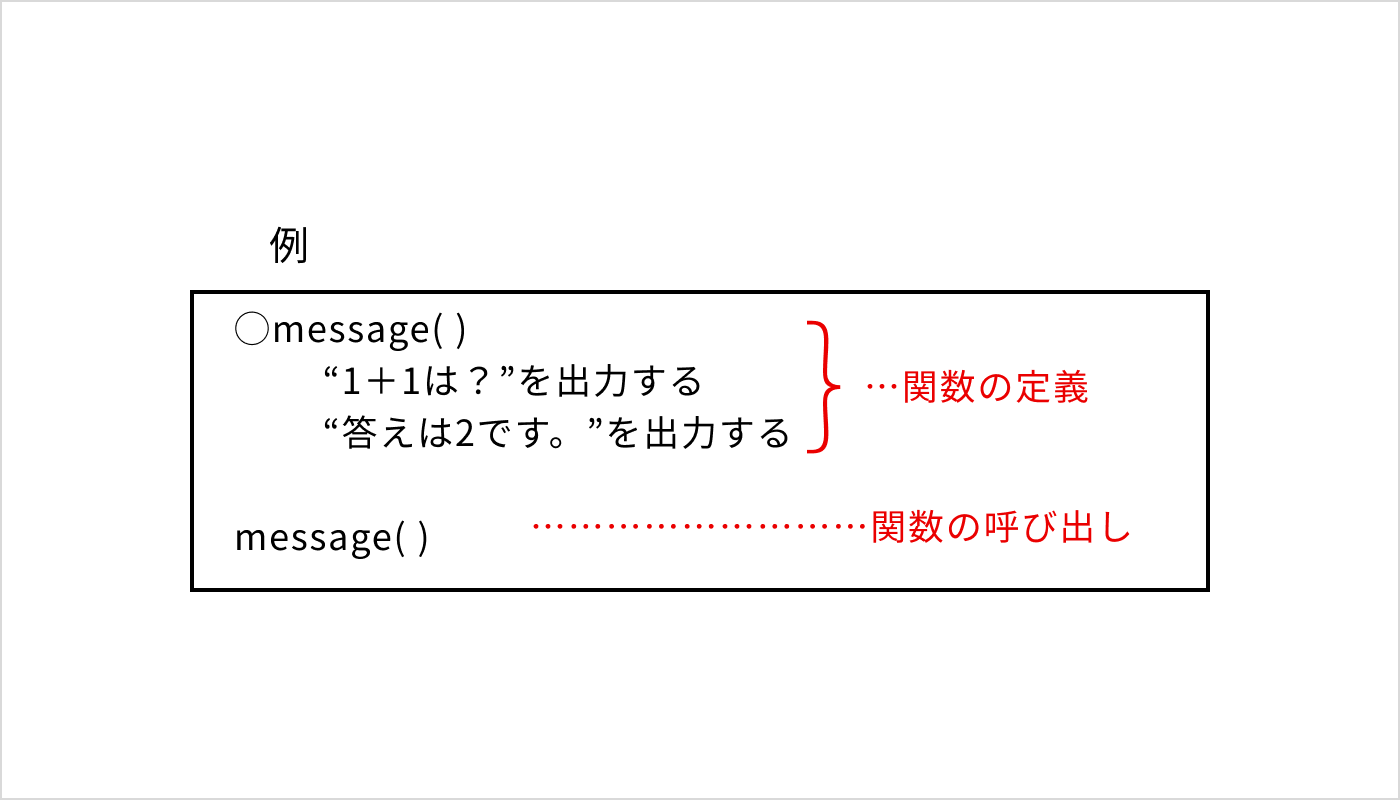
関数を実行するためには、2つ目のステップである「**関数の呼び出し**」が必要です。

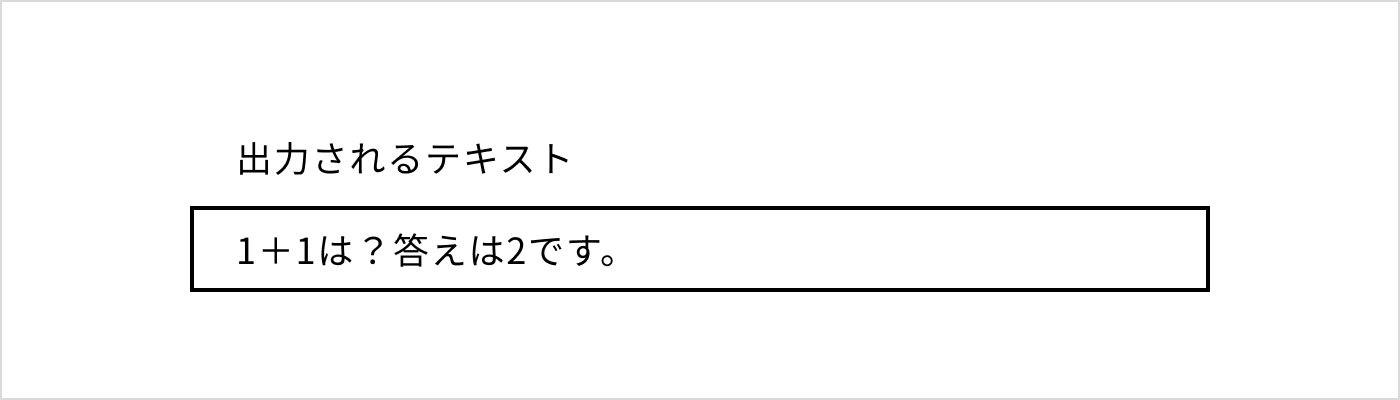
関数の呼び出しとは、定義した**関数を実行するステップのことです。**

関数を呼び出す書式は、以下のとおりです。



以下の例は、先ほど定義した「関数message」を呼び出したものです。



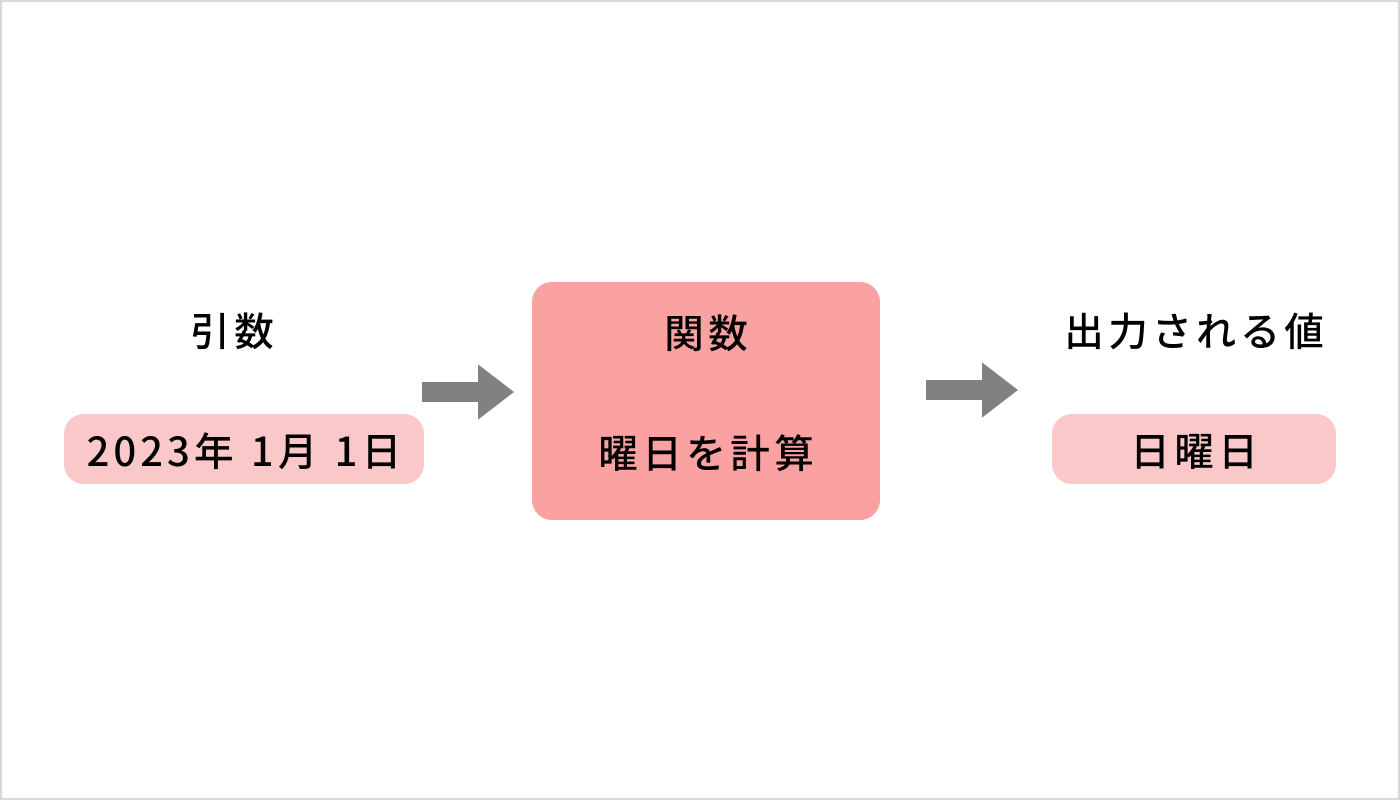


引数（ひきすう）とは、簡単に説明すると**関数に渡す値**です。関数内の命令で使われる数値や文字列など、具体的な値を入れます。

日付（2023年1月1日）を入力すると、曜日（日曜日）を計算して出力する関数を例に説明します。

この関数の**引数は「2023年1月1日」** となります。

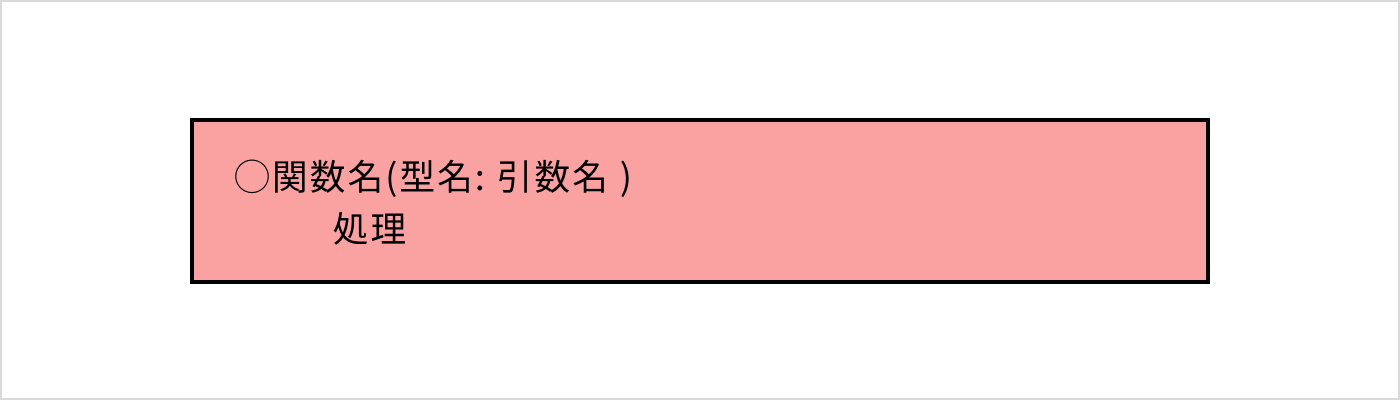
引数を使用することで、関数をより便利に使えます。



#### **引数のある関数の定義**

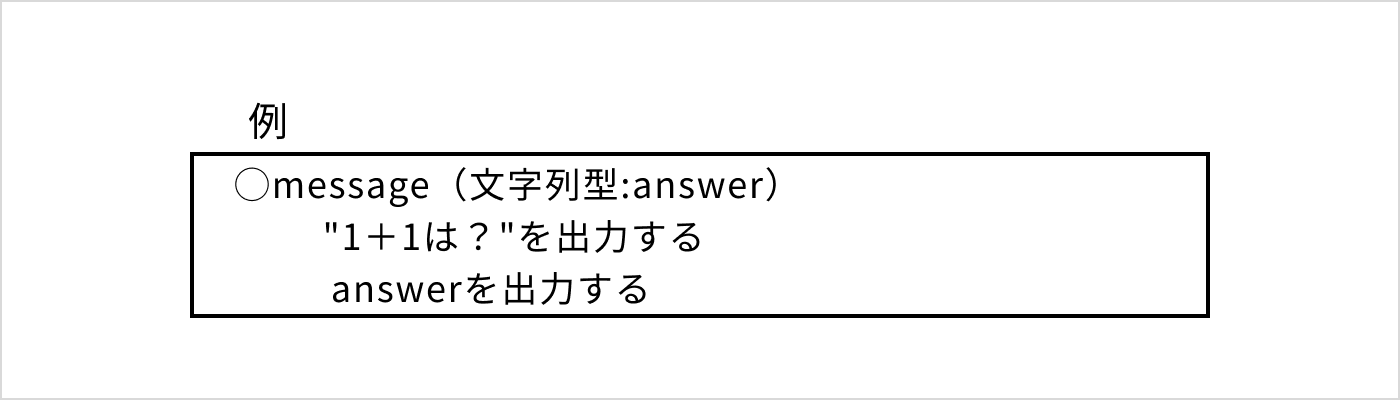
引数のある関数を定義する場合、「()」の中に「**型名：引数名**」を書きます。

型名とは引数の種類を表したもので、文字列型や整数型などがあります。引数名は関数名と同じく、自由に決められます。



**なお指定した引数名は、関数内では「変数」（処理の途中で中身が変わってもよい数）として扱われます。**

以下の例は、関数messageに引数を追加したプログラムです。



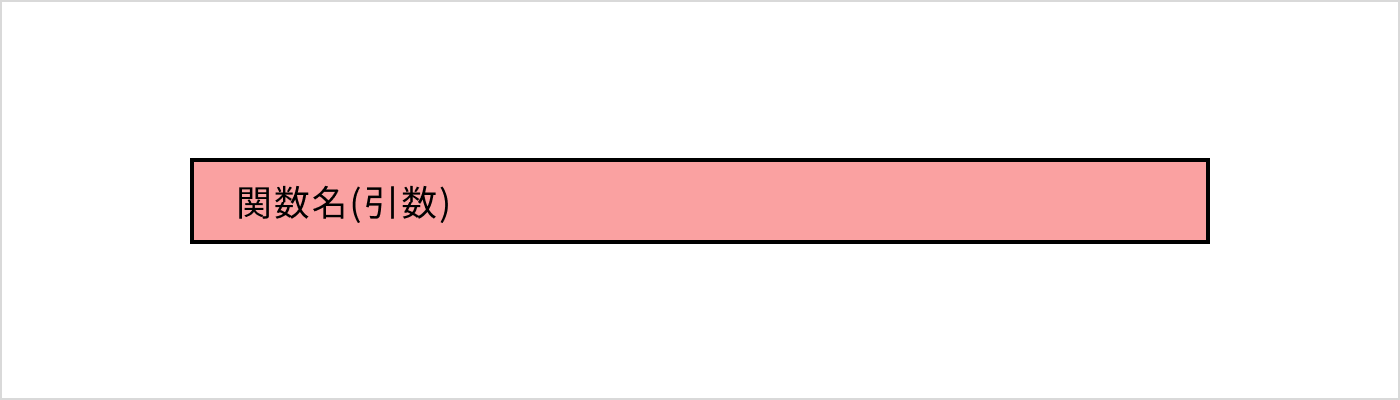
* 型名は「文字列型」、変数は「answer」と書く

上記のように書くことで、関数messageにて「文字列型：answer」が使用できます。

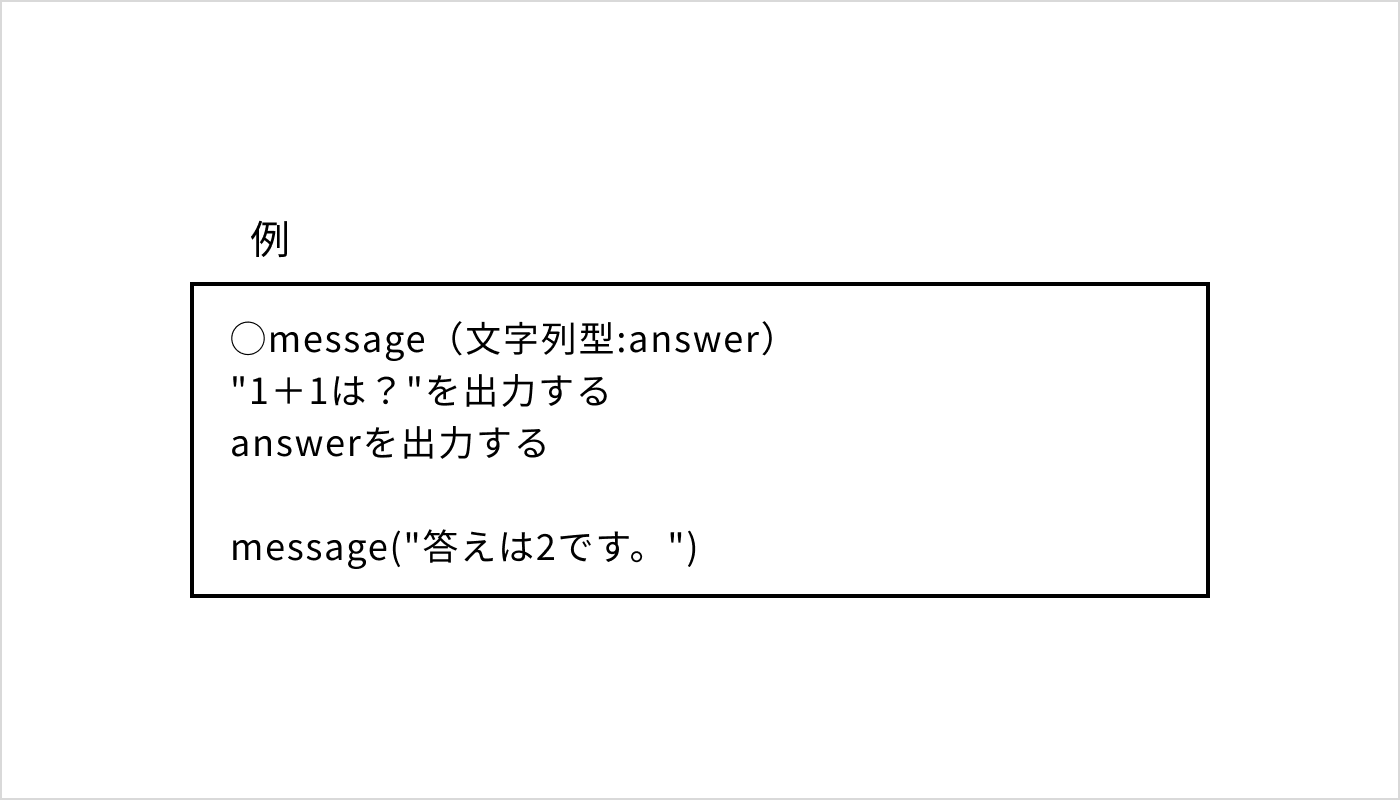
「answer」の値は、次に解説する「引数のある関数の呼び出し」で指定します。

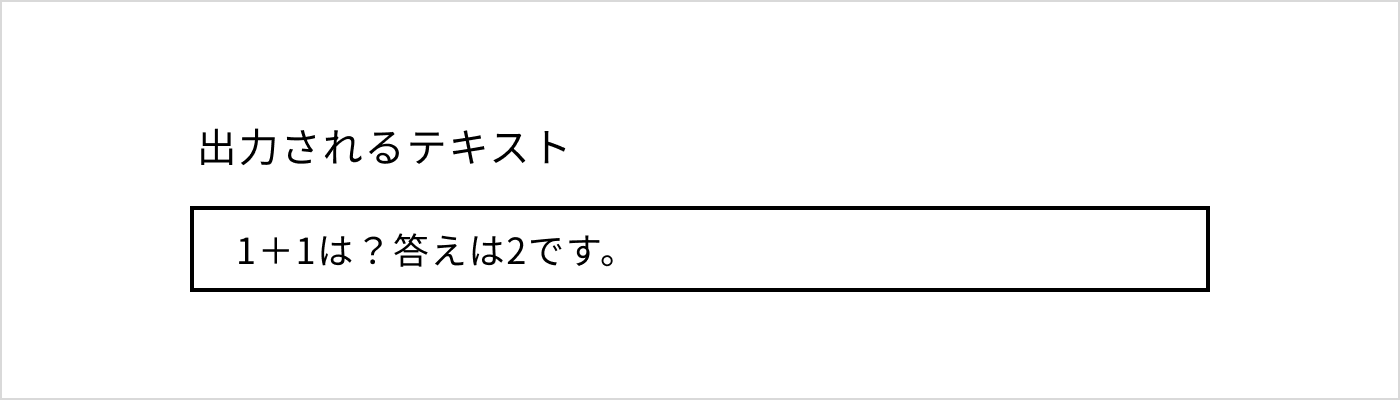
#### **引数のある関数の呼び出し**

引数のある関数を呼び出すときの書式は、以下のとおりです。



以下の例は、関数messageの引数に「答えは2です。」を代入し、2つの命令が実行されるようにしたプログラムです。

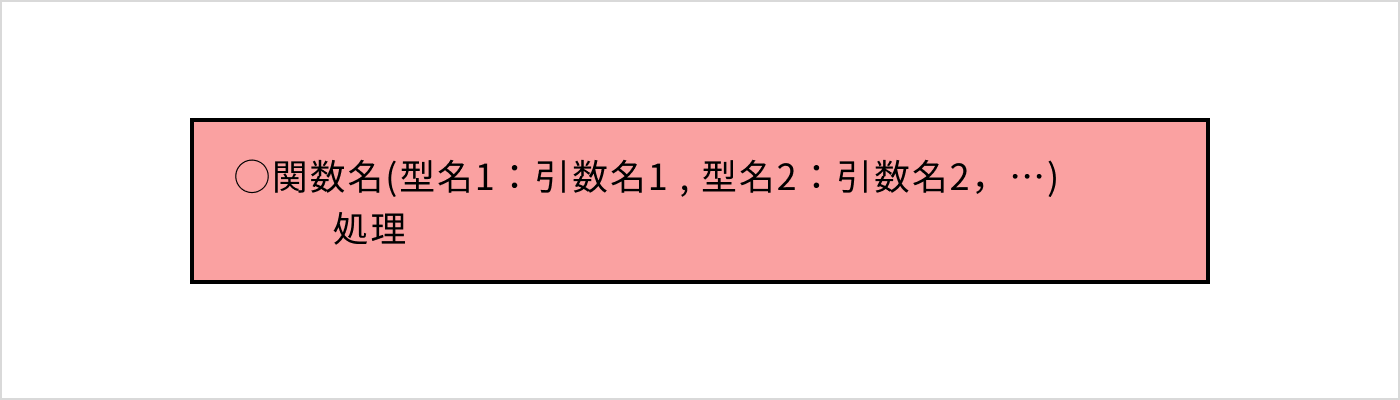




#### **複数の引数を受け付ける関数**

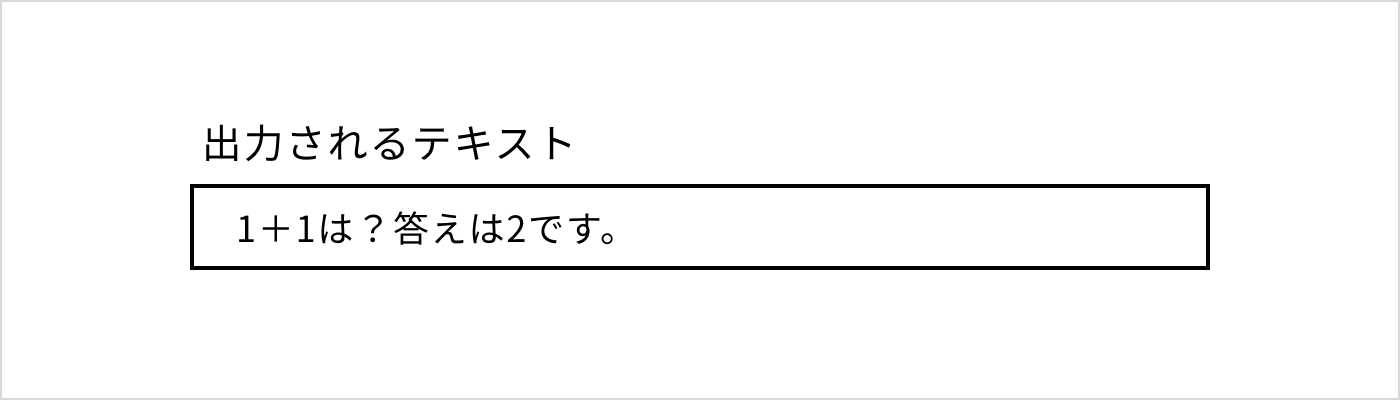
関数には、複数の引数を渡すことが可能です。

複数の引数を指定する場合は、引数を「,」（カンマ）で区切ります。



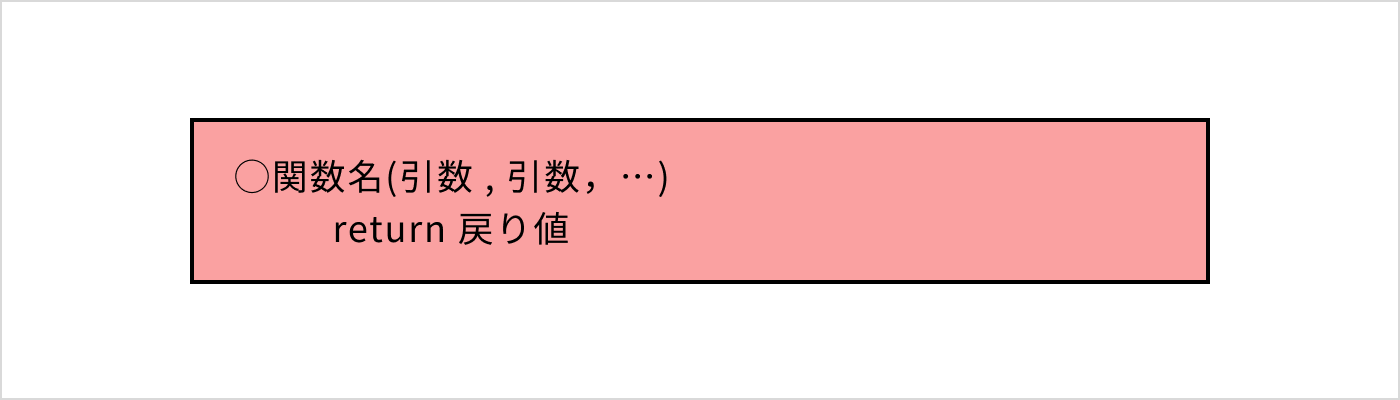
以下の例では、関数messageに「引数x」と「引数y」を用いています。



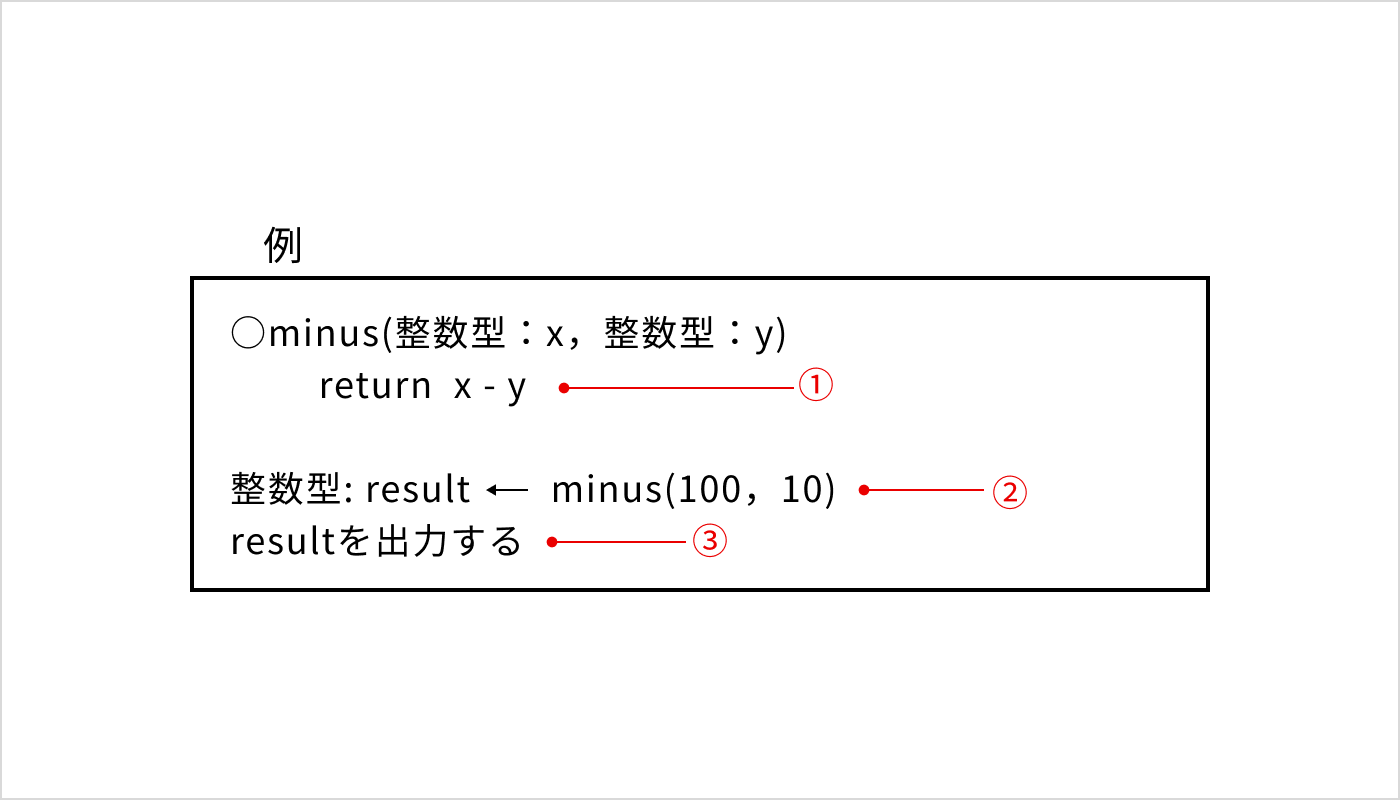


り値（返り値）とは、**関数実行後に出力される値**のことです。

「戻り値」を得るには、「return」を使用します。戻り値の書式は、以下のとおりです。



以下の例は、戻り値を使用したプログラムです。





1. 関数minusの定義：「x-y」の計算結果を戻り値として指定
2. 関数minusの実行：引数に100と10を渡して、関数minusを呼び出し、returnで戻ってきた計算結果を「result」に代入
3. 変数「result」（計算結果の90）を出力

②のステップではminus関数の実行と、reslut変数への代入を同時に行っています。

| **用語** | **説明** |
| --- | --- |
| 関数の定義 | 複数の命令を1つにまとめること |
| 関数の呼び出し | 関数を実行すること |
| 引数 | 簡単に説明すると**関数に渡す値** プログラム内では関数で使われる数値や文字列など、具体的な値を入れる |
| 戻り値 | 関数の呼び出しによって得られる結果 |

**コンピュータシステム**

システムとは、コンピュータを使って手作業を自動化するための仕組み

日常的に使っているスマホもシステムの一つになる、システムはハードウェア（スマホ本体）とソフトウェア（アプリ）の２つで構成されている

**処理形態によるシステムの分類**

３つの分類法がある

・**処理形態による分類**

・**利用形態による分類**

・**システム構成による分類**

**処理形態による分類**

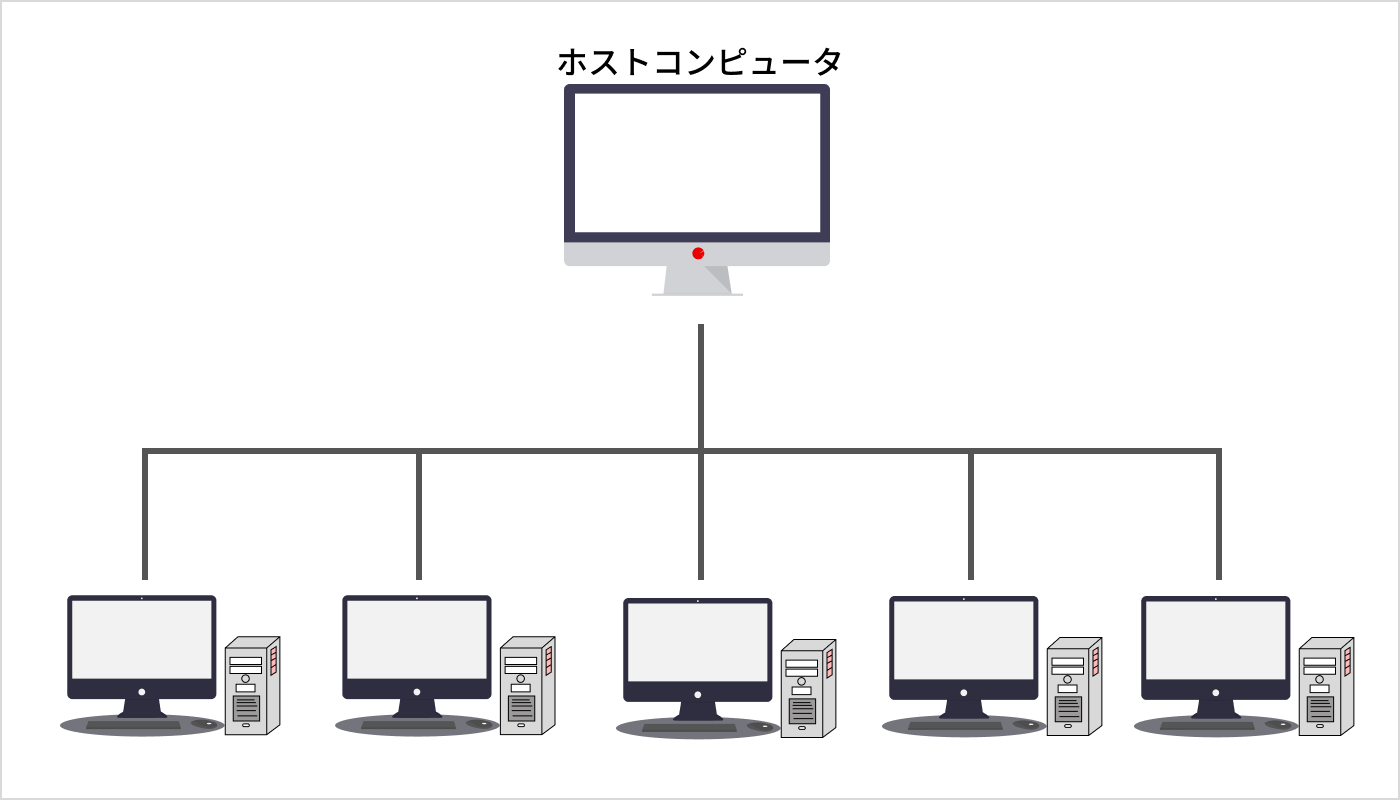
システムには「集中処理」と「分散処理」の２つの形態がある

**集中処理**

１台の大型コンピュータが複数の処理を引き受ける

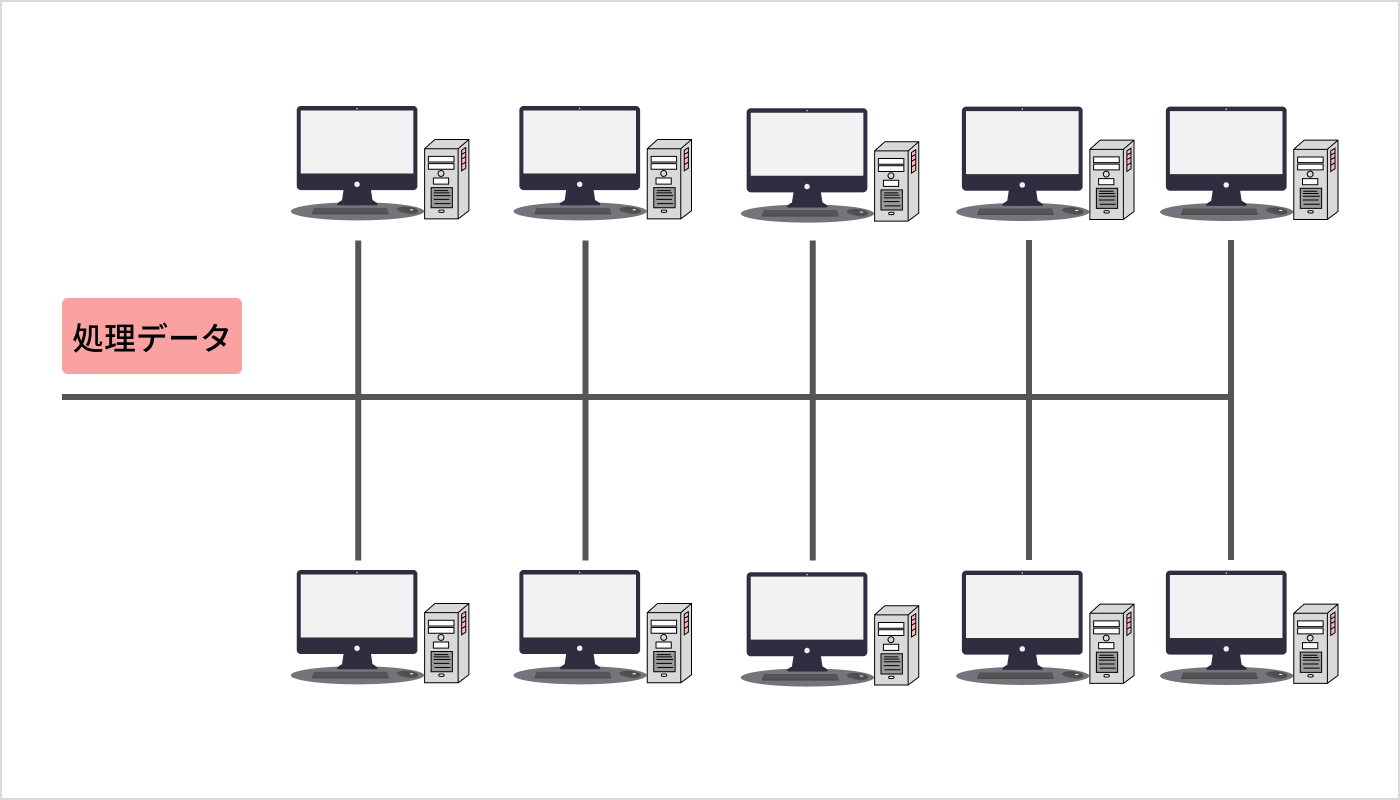
端末から送られてくるすべての要求を、その１台のコンピュータが処理を行う

大型コンピュータを「ホストコンピュータ」と呼ぶ



**分散処理**

ネットワークで繋がれている複数のコンピュータが複数を分担して行う



分散処理が主流であり、ITパスポートでもよく出題される

分散処理システムの種類

| **クライアントサーバシステム** | **クライアント側**が出した処理要求を**サーバ**が処理する分散処理システム。基本的には、専用のソフトウェアをクライアント側にインストールする。 クライアント：サービスを利用する側 サーバ：サービスを提供する側 |
| --- | --- |
| **Webシステム** | **インターネットとWebブラウザを使用するシステム** **Webブラウザ**を使用してサーバに処理要求を出す。そのため、専用のソフトウェアをクライアント側にインストールしない。 |
| **シンクライアント** | クライアント側には、**必要最低限の機能**しか持たせず、サーバ側でアプリやデータなどを集中管理するシステム。 クライアント側から情報漏洩するリスクが軽減される |

各分散処理システムについて、クライアント側の説明をまとめておきます。

| **システム** | **クライアント側でデータ表示** | **クライアント側の記憶装置** | **クライアント側でデータ処理** |
| --- | --- | --- | --- |
| クライアントサーバシステム | する | あり | する |
| Webシステム | **する** | **あり** | **しない** |
| シンクライアント | **する** | **なし** | **しない** |

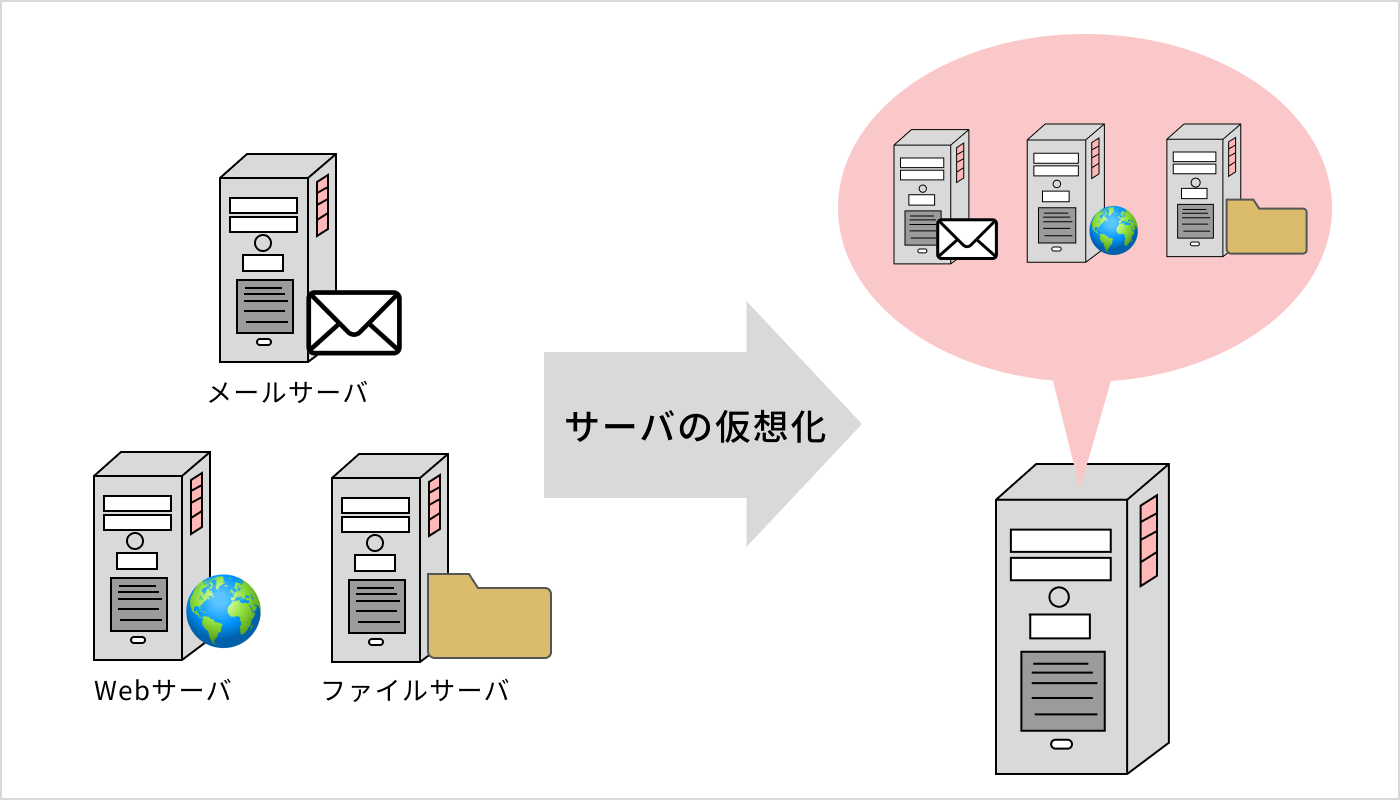
**利用形態やシステム構成による分類**

システムの利用形態はさまざまだが、その一つにサーバーの仮想化がある

サーバの仮想化とは１つのサーバの中に複数の仮想的なサーバを作り出す事

基本は「１つのサーバにOSが一つ」、しかしサーバを仮想化すると「１つのサーバに複数のOS」をインストールできる

実際は1つしかないサーバを、サーバの仮想化により「サーバが3つあるように見せる」ことができます。



OS（オペレーティング・システム）について補足します。

OSとは、「文書作成ソフトや画像編集ソフト」などのプログラムを実行するとき、コンピューターの中でプログラムが動くための環境のことです

**システム構成による分類**

障害が起きたときの対策として重要な要素になる、システム分類すると次の３種類に分けられる

**シンプレックスシステム**

コンピュータが１つだけのシンプルな構成、システムのどこか１箇所でも障害が起きると、システムは停止してしまう

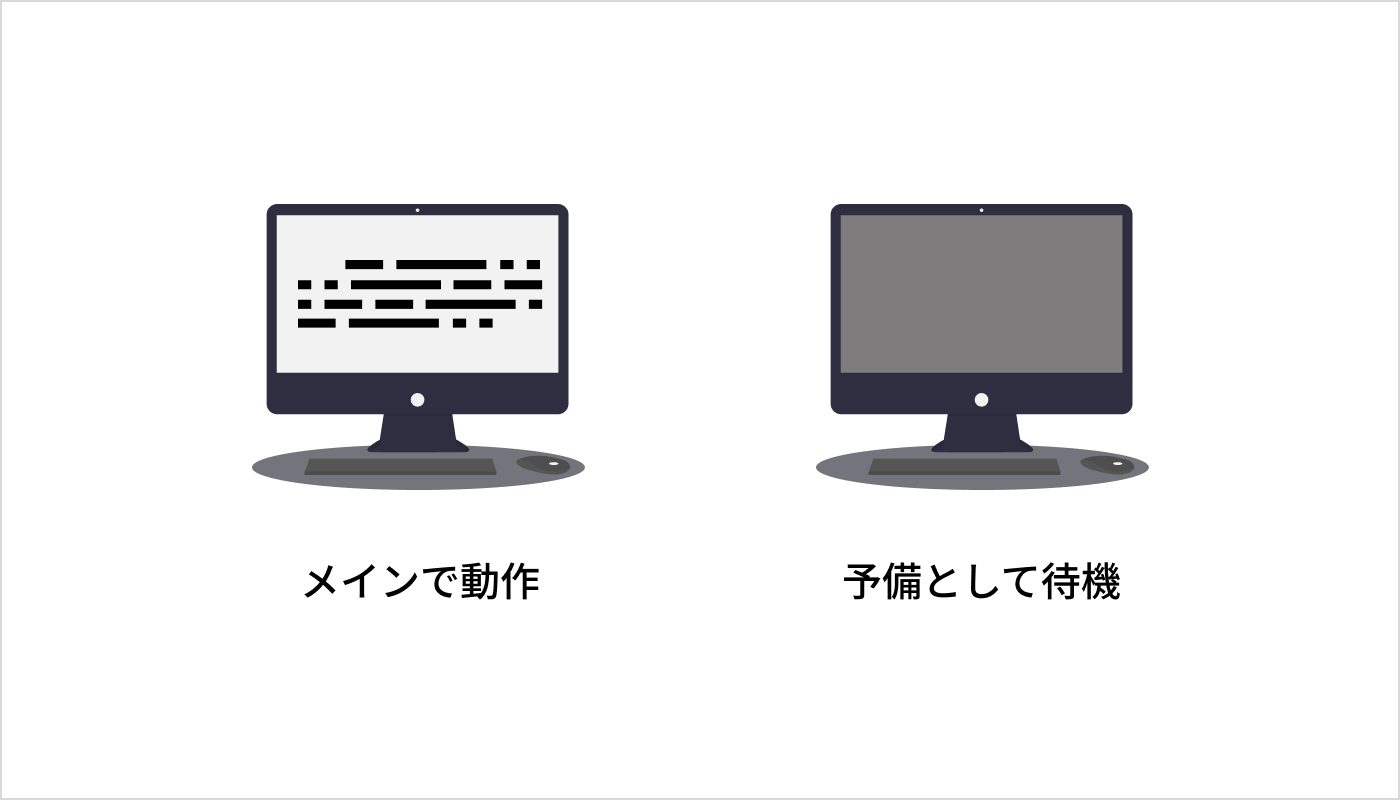


**デュプレックスシステム**

１つの処理に対して、メインで動作するコンピュータと

障害が発生したときの予備として待機している２ペアのシステム構成

Duplexは２重という意味



**デュアルシステム**

２つのコンピュータが同時に処理を行い

実行結果を照合して正しいことをチェックするシステム構成

２つのコンピュータが同じ処理を行うため

処理能力としてはシンプレックスシステムとほとんど同じ

特徴は「障害を検知できる」こと、同じ処理を行い、結果を照合しているため、結界に違いがあれば異常であると判断できる



ITパスポート試験では、「**デュアルシステム**」と「**デュプレックスシステム**」を区別する問題が出題されやすいです。名前が似ているので混同しないよう注意しましょう。

| デュプレックスシステム | 1つの処理に対して、メインで動作するコンピュータと、障害が発生したときの予備として待機している2組のコンピュータからなるシステム構成。 |
| --- | --- |
| デュアルシステム | 2つのコンピュータが同時に同じ処理を行い、実行結果を照合して正しいことをチェックするシステム構成。 |

**ハードディスクの多重化**

ハードディスクに重要なデータを保存していた場合、障害の発生によりデータが取り出せなくなる可能性がある

同じデータを複数のハードディスクに書き込めば、データを障害から守れる

これがハードディスクの多重化という

障害に強いハードディスクの構成

**RAID**

Redundant Array of Inexpensive レイド　の略称で

複数のハードディスクにデータを保存する技術

複数のハードディスクを１つのハードディスクであるかのように扱う

RAIDには０～６の７つのレベルがある

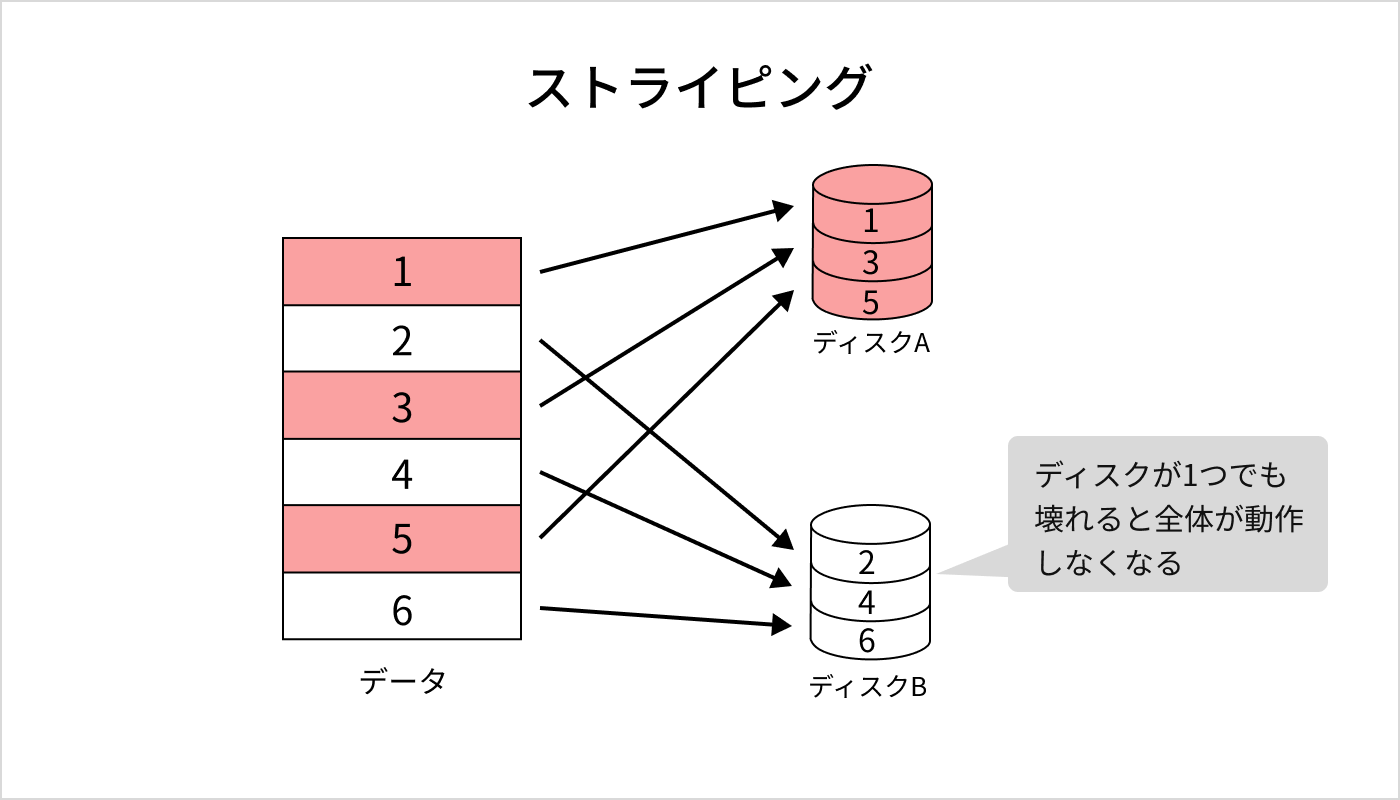
RAIDのレベルは性能の高さではなく「データの記録方式の違い」を表している、そのため「RAID1よりも2の方が高性能」という訳ではない

**RADI０（ストライピング）とRAID１（ミラーリング）**がITパスポートに出題されやすい

**RAID０**

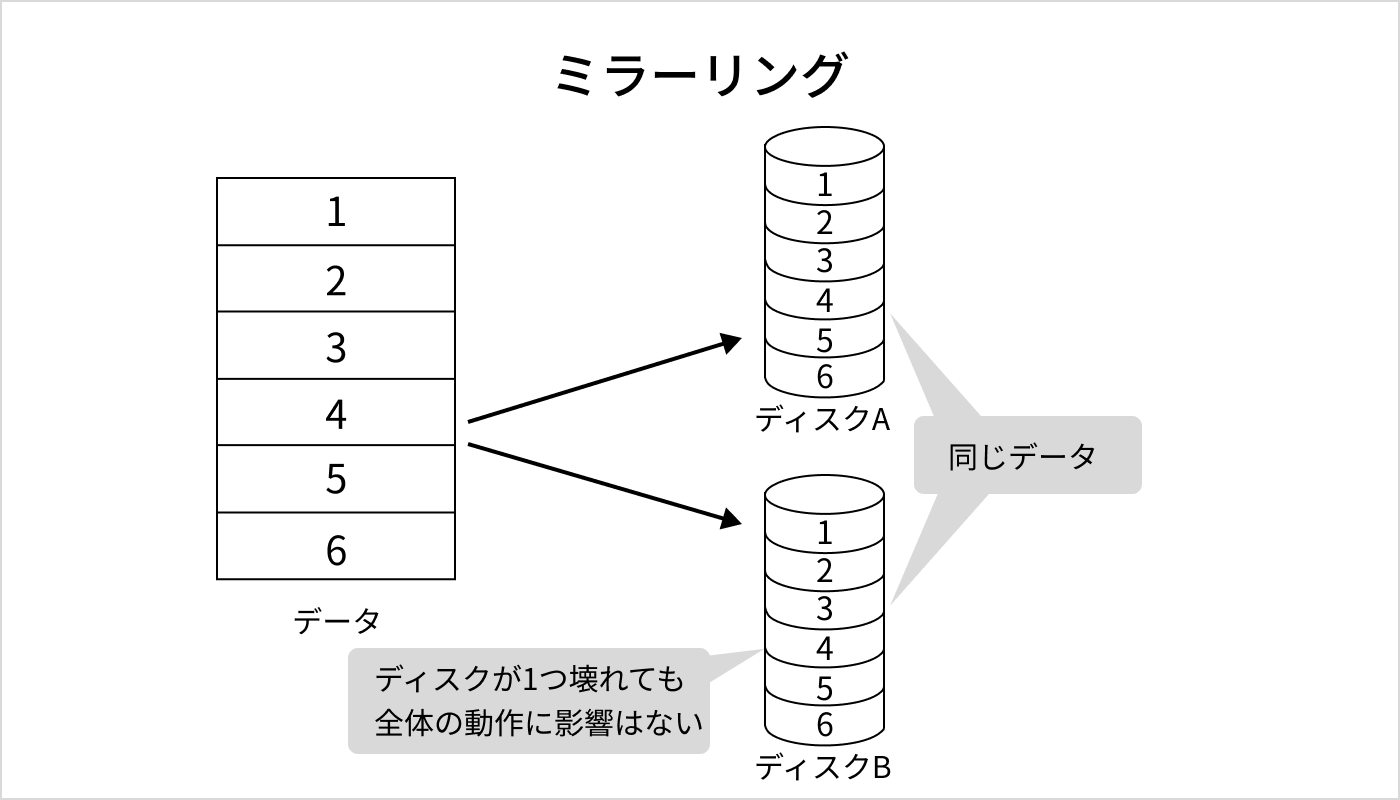
データを複数のハードディスクに分散して書き込む記憶方式

ストライピングとも呼ばれる



**RAID１**

複数のハードディスクにに同じデータを書き込む記憶方式、ミラーリングとも呼ばれる



| RAID 2 | データに誤り訂正符号を付け、複数に分散してデータを書き込む記憶方式 |
| --- | --- |
| RAID 3 | 誤り訂正符号は別の専用ディスクに書き込み、データは他に分散して書き込む記憶方式 |
| RAID 4 | RAID 3と同様の方式だが、データの分散をブロック単位で行う記憶方式 |
| RAID 5 | データに誤り訂正符号を付け、複数に分散してデータを書き込む記憶方式 |
| RAID 6 | RAID 5と同様の方式だが、誤り訂正符号を2重で生成する記憶方式 |

RAIDは家庭用のPCでは採用されておらず、ネットワーク上のストレージや企業内のサーバーなどで採用されている

**ストライピング**

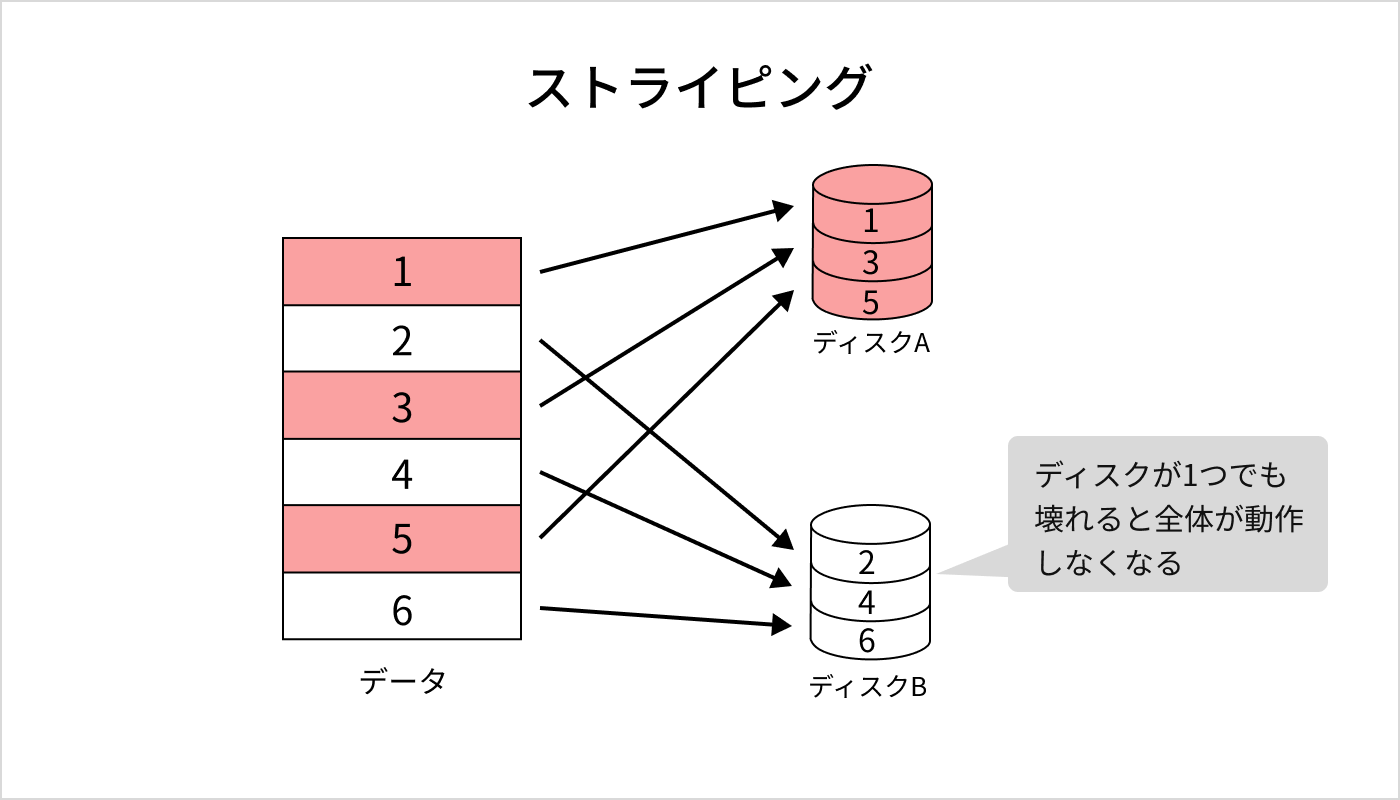
データを複数のハードディスクに分散させて書き込む記憶方式

ハードディスクに対して並列にアクセスを行うため、データの読み書きを高速化できる点がメリット

ただし、データの多重化は行っていないため、ディスクが１つでも壊れると全体が動作しなくなる

データの多重化とは同じデータを複数のディスクに書き込むこと

そのため、ストライピングを採用しても障害には強くならない、逆にディスクが増えたため故障のリスクが上がる



**ミラーリング**

複数のハードディスクに同じデータを書き込む記憶方式

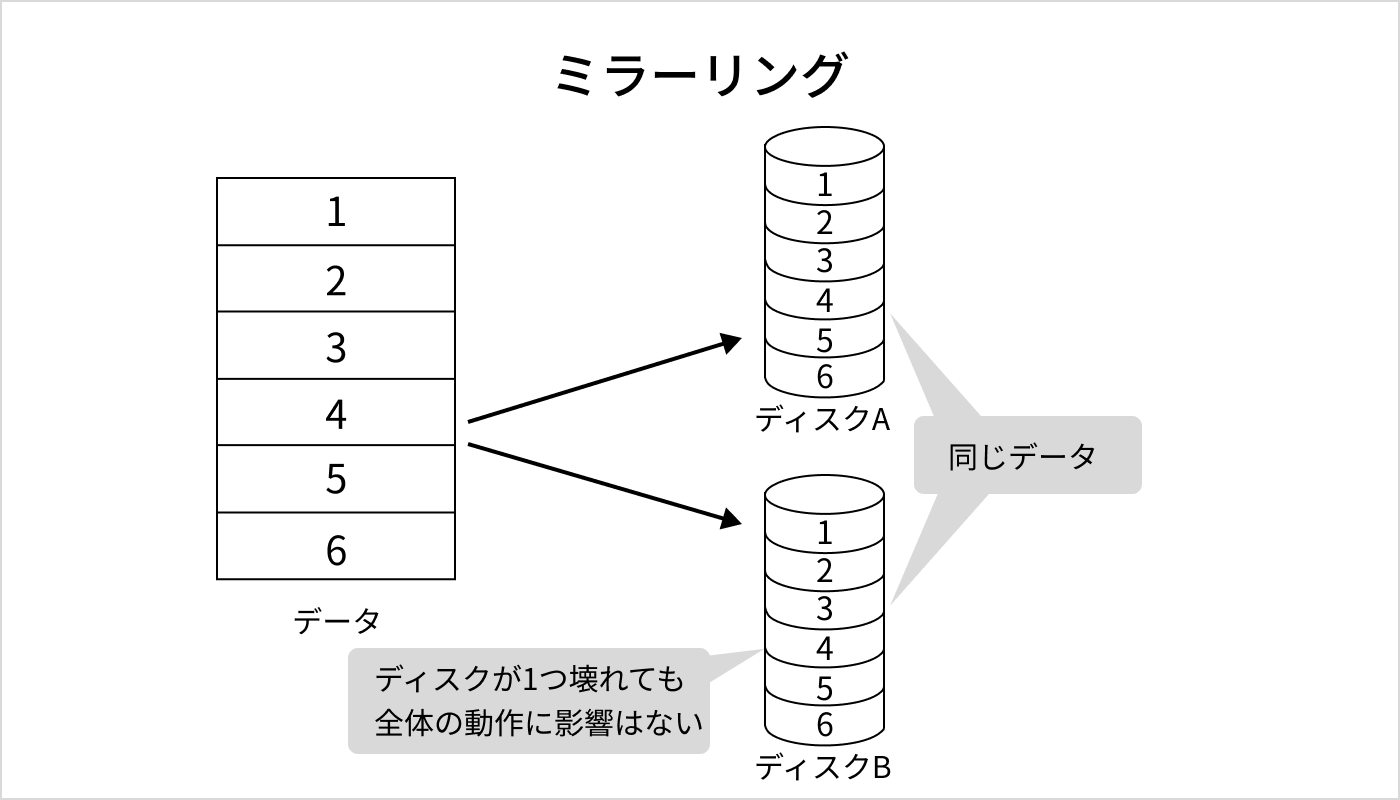
すべてのデータが多重化されるので、障害に強くなるのがメリット

デメリット

・複数のディスクが必要になるので費用が掛かる

・同時に複数のディスクにアクセスするため、処理速度が遅くなる

・ストライピングと比較して保存できる容量が半分になる

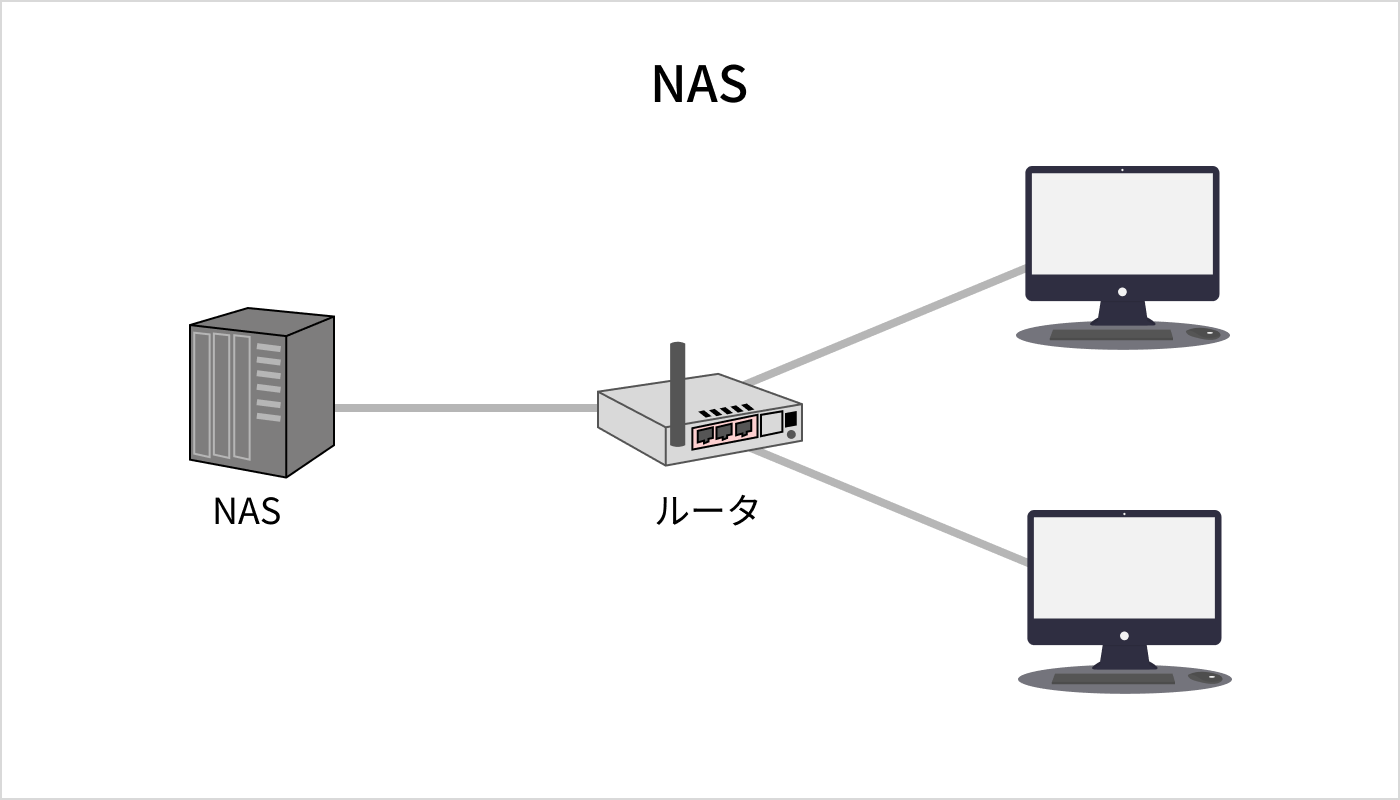


**NAS　Network Attached Storage**

ネットワークに接続して使用するハードディスク

ネットワークに接続するため、複数のコンピュータやスマートフォンでデータを共有できるのがメリット

NASはデータの共有だけでなく、バックアップ用の機器としても利用できるため、障害に強いと言える



**システムの評価指標**

構築したシステムの能力を「総合的に評価する指標」

システムを評価する指標は以下の３つがある

・性能：処理の速さ

・経済性：コスト

・信頼性：問題なく動作するか

**性能**

システムの応答時間（処理に必要な時間）、応答時間は「レスポンスタイム」ともいう

また、システムの性能を比較するためのテスト「ベンチマークテスト」という

レスポンスタイムを他のシステムと比較して評価をする

**経済性**

開発にかかるコストや運用コストなど、様々なコスト

経済性を評価する指標は以下の３つ

・初期コスト（イニシャルコスト）：システムの導入時に発生する費用

・運用コスト（ランニングコスト）：システム導入後に発生する運用、管理で発生する費用

・TCO（Total Cost of Ownership）：初期コストと運用コストを合わせた費用

まとめ

・ハードディスクの多重化とは、同じデータを複数のハードディスクに書き込む事

・システム評価指標とは、構築したシステムの能力を「総合的に評価する指標」

RAID　複数のハードディスクにデータを保存する技術

ストライピング 　データを複数のハードディスクに分散させて書き込む記憶方式

ミラーリング　複数のハードディスクに同じデータを書き込む記憶方式

NAS　ネットワークに接続して使用するハードディスク

性能　応答時間を評価すること

経済性　コストを評価すること

**システムの信頼性**

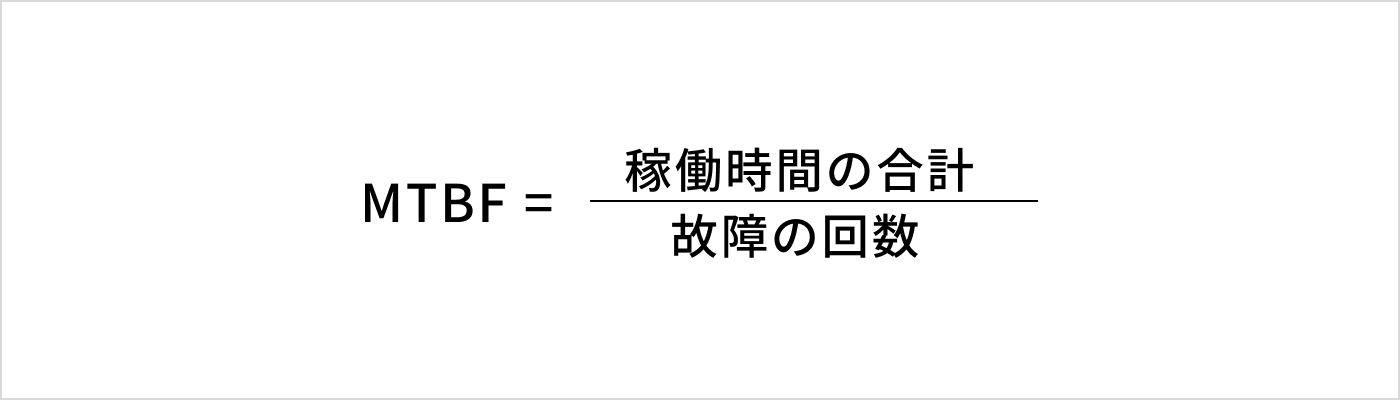
信頼性の評価指標は、システムが正常に動作するか

信頼性が高いシステムとは、故障や障害などが起こりにくいシステムのこと

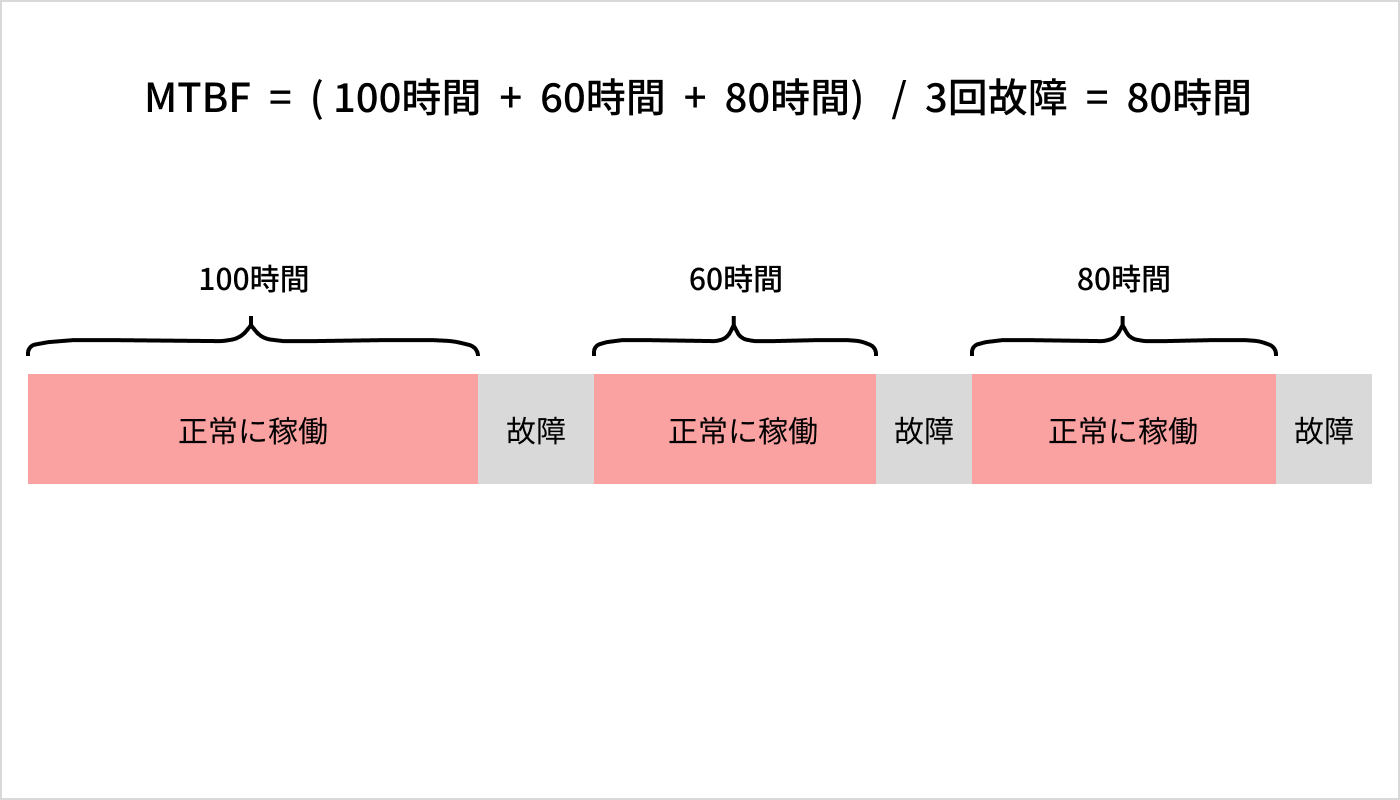
**MTBF　Mean Between Failures：平均故障間隔**

システムが故障しないで正常に動作していた時間の平均値、システムの信頼性を表す指標になる

MTBFは次の式で求める



例えば下図のシステムのMTBFは８０時間になる



MTBFの値が大きいほどシステムの信頼性が高く、MTBFの値が小さいほどシステムの信頼性は低いと言える

MTBFの値を大きくするには、システムのメンテナンスを定期的び行うことが必要

例えば、古くなったハードウェアを交換することで、予期しない故障や障害を防げる

MTBFの値を大きくする（予期しない故障や障害を未然に防ぐ）メンテナンスのことを「予防保守」という

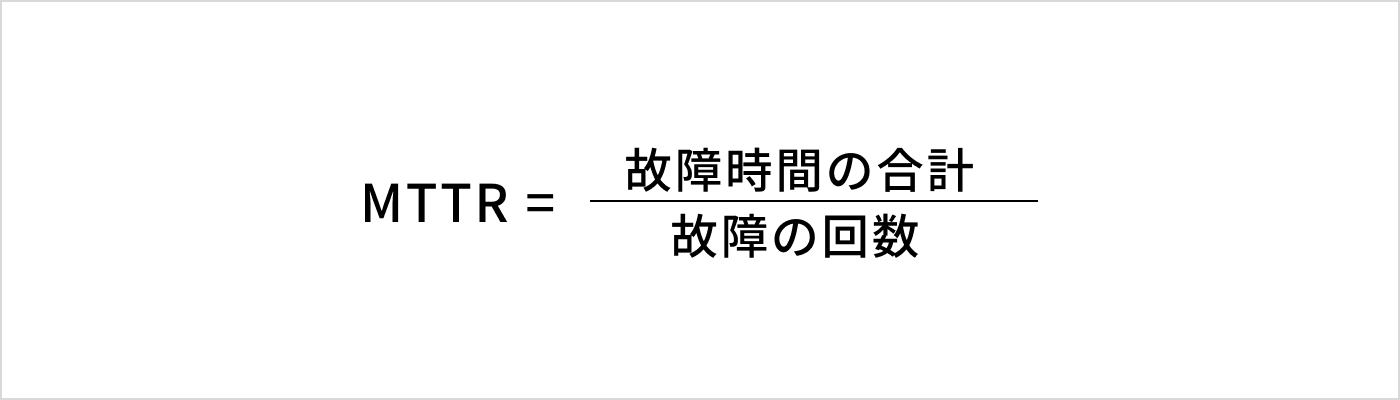
**MTTR　Mean Time To Repair：平均修理時間**

システムが稼働していない時間（修理していた時間）の平均値のこと

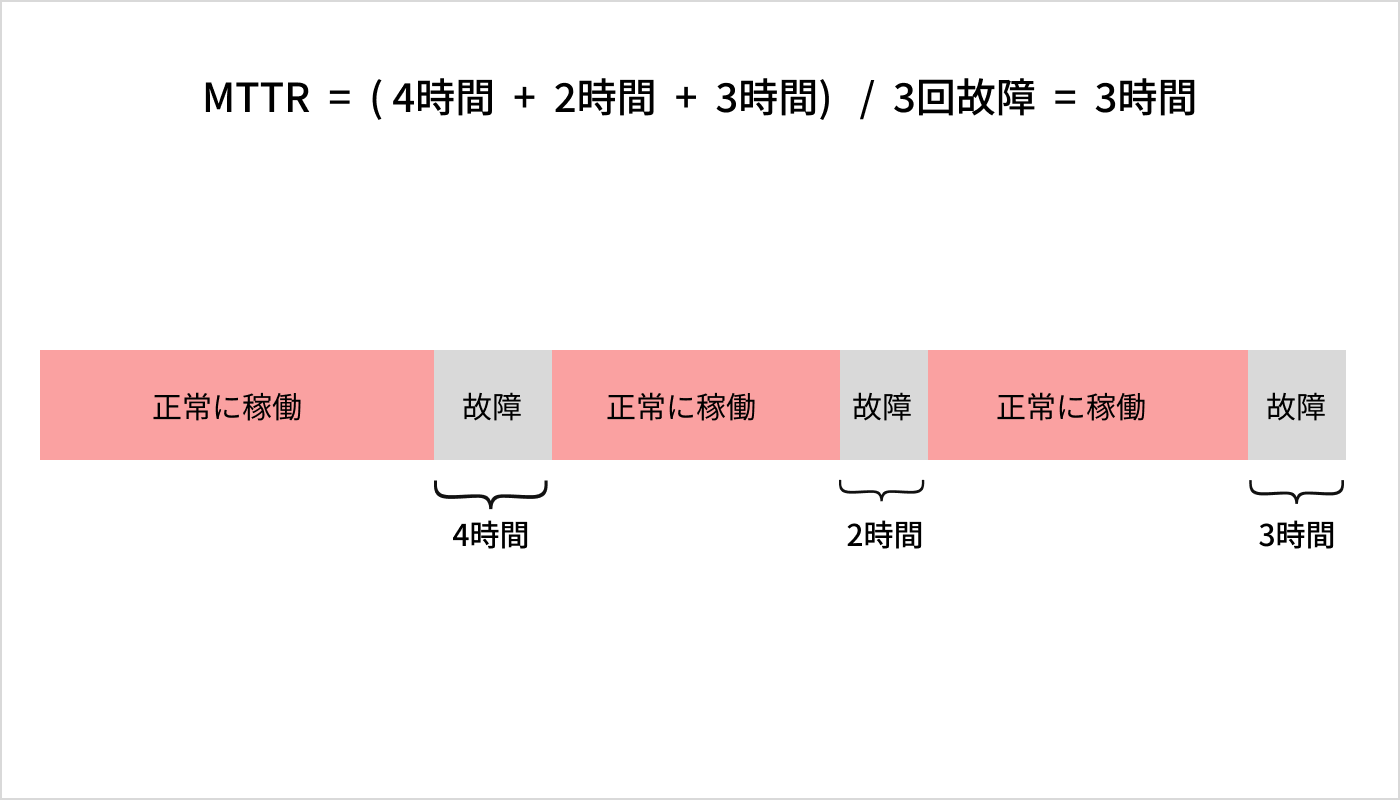
システムの保守性（修理や交換を行うことで機能や状態を維持）を表す指標。

保守性と信頼性は「正常に動作するか」という点て関連している

MTTRは次の式で求める



例えば下図のMTTRは３時間になる



MTTRの値が小さいほどシステムの保守性が高く、MTTRの値が大きいほどシステムの保守性が低いと言える

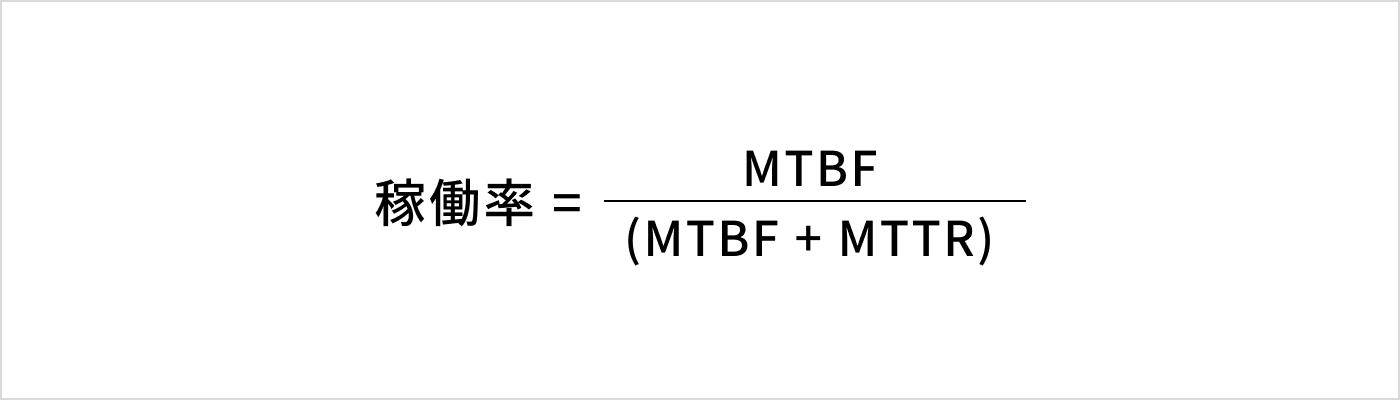
MTBFとMTTRを使い、システムの稼働率を求める

**稼働率**

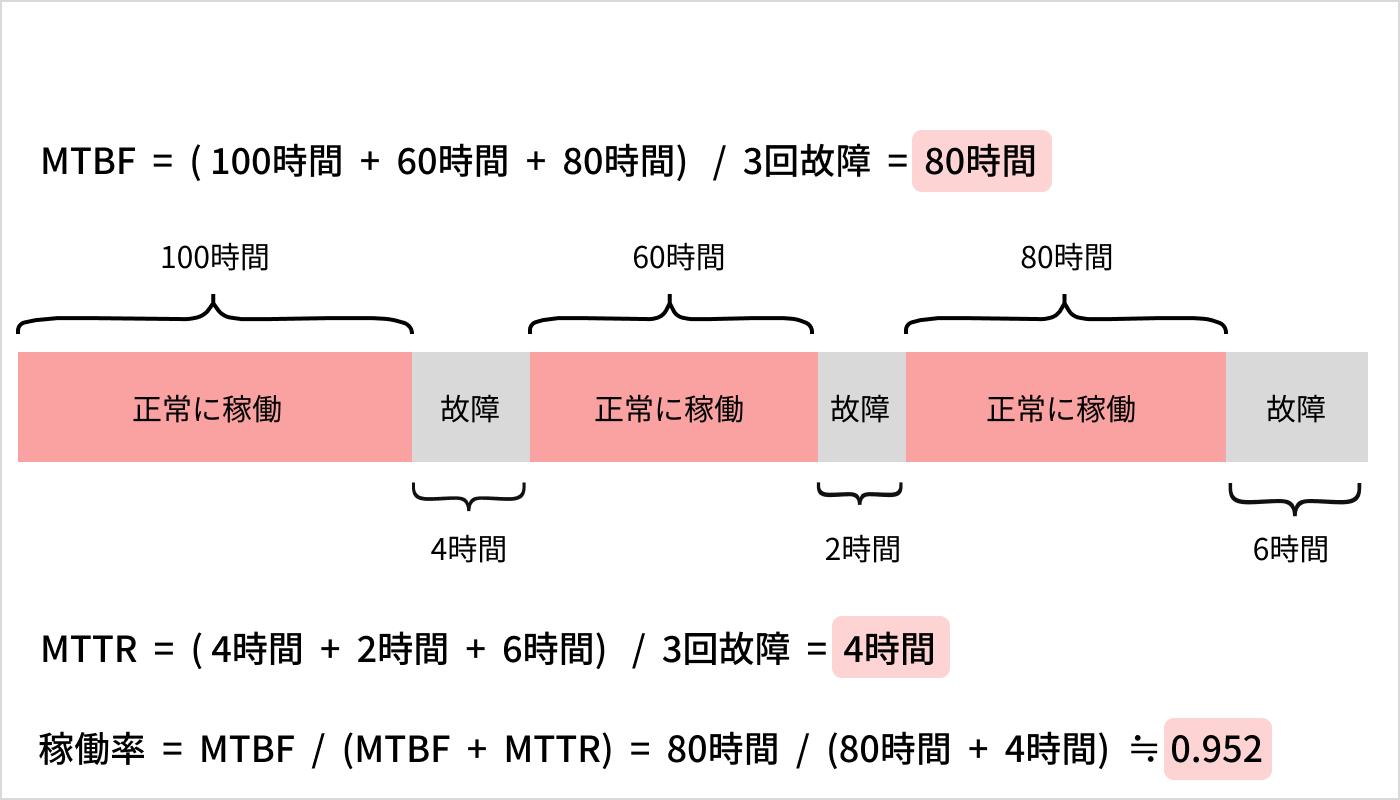
システムが故障しないで稼働していた時間の割合

システムの可用性（システムが継続して稼働できる度合いや能力）を表す指標

稼働率は次の式で求める



例えば下図のシステムの稼働率は0.952



稼働率が値が高いほどシステムは可用性が高く、稼働率の値が低いほどシステムは可用性が低いと言える

稼働率が「１」に近いほど、正常に稼働し続けている

**システムを止めないために**

ユーザが求めているものは、システムが提供するサービスで、システムそのものではない

そのため、障害が発生した場合でも、できる限りシステムが止まらないように設計することが重要になる

このように、システムが止まらないように設計することを「フォールトトレラント」と言う

**フォールトトレラント**

コンピュータが故障してもシステムが止まらないように設計すること

コンピュータは電気機器製品のため、いつか必ず故障する、故障しないコンピュータは存在しない

重要なことは「絶対に故障しないコンピュータを用意すること」ではなく「コンピュータが故障してもシステムは稼働を続けられること」になる

例えば、原子力発電や飛行機などの重要のシステムは、このフォールトトレラントが重要になる

他にも信頼性に関する用語がある

| フェールソフト | 障害が発生した場合、**機能が落ちてもシステム全体は動作させる設計** 例えば、飛行機のエンジンは1つ故障しても飛行を継続できる設計 |
| --- | --- |
| フェールセーフ | 障害が発生した場合、**システムを止めてでも安全を優先させる設計** 例えば、工場で使用する機械に人間が手を入れてしまった場合、機械が停止する設計 |
| フールプルーフ | **人が間違った使い方を**しても、誤作動が起きないようにする設計 例えば、洗濯機はフタを閉めないと稼働しない設計 |

**システムを構成する装置の配置方法**

配置方法は直列、並列、複合システムの３種類に分類できる

**直列システム**

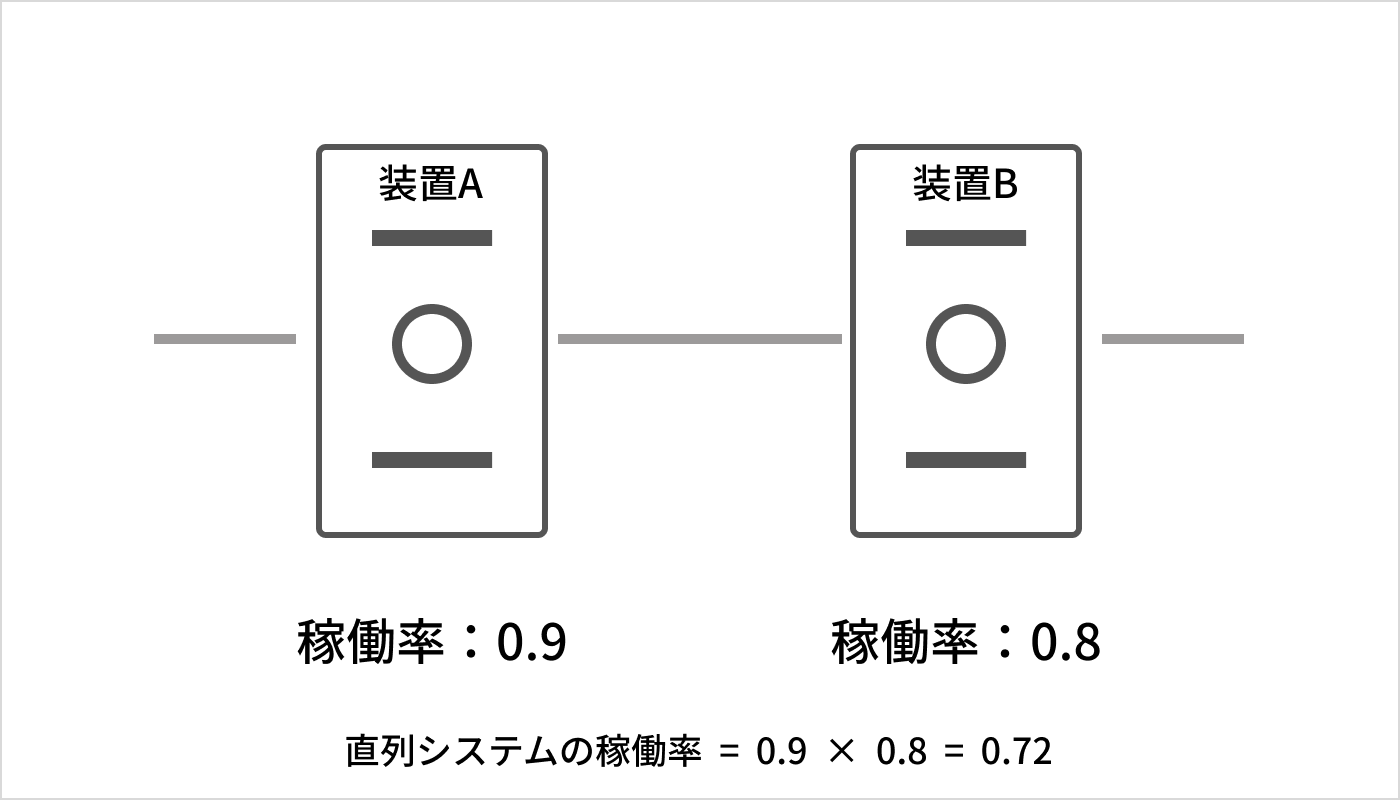
複数の装置が直列に接続されたシステム

直列システムの代表例はシンプレックスシステムになる、システムを構成する装置が１つでも故障すると全体が停止する

直列システムの稼働率は次の式で求める

**直列システムの稼働率 = 装置Ａの稼働率 × 装置Ｂの稼働率**

例えば、下図の直列システムの稼働率は0.81となる



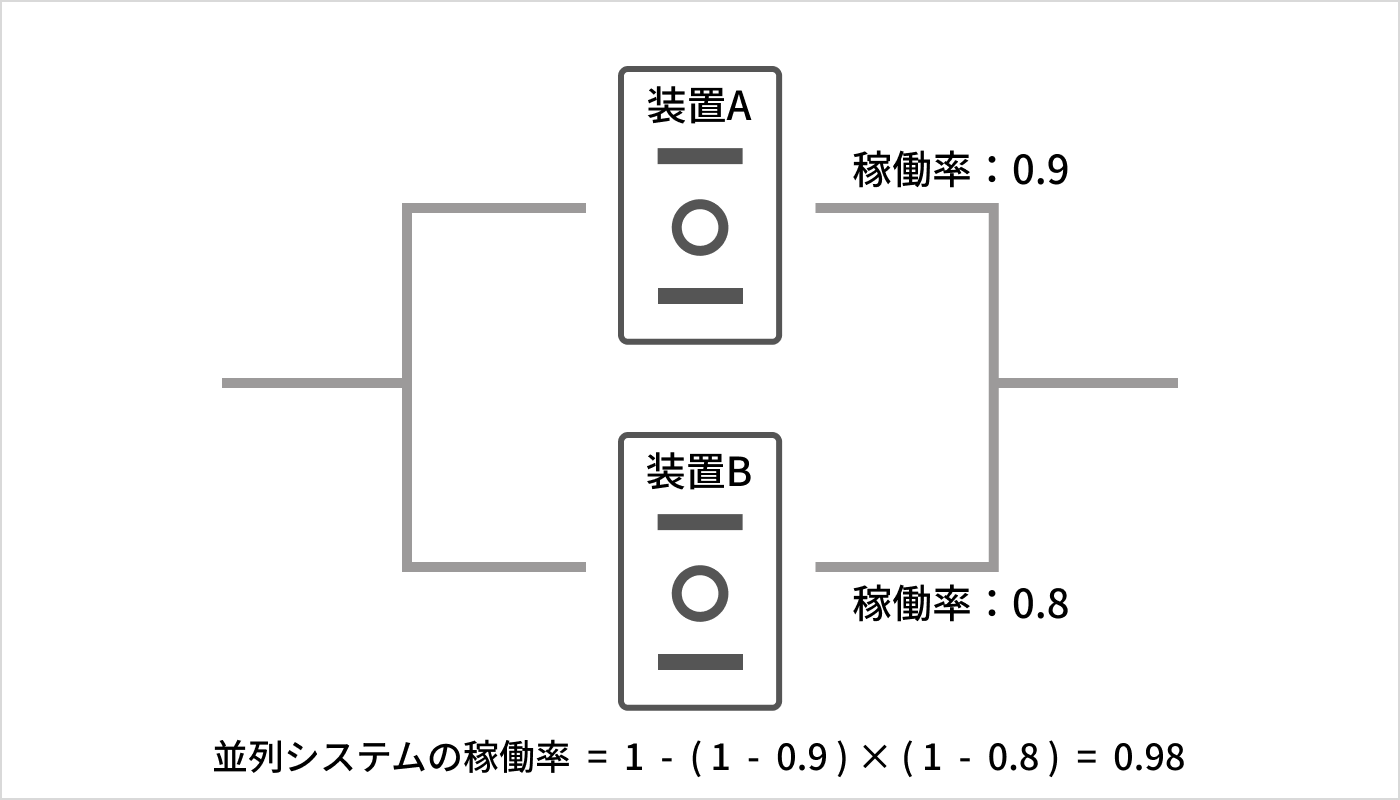
**並列システム**  
複数の装置が並列に接続されたシステム

並列システムの代表例は、デュアルシステムやデュプレックスシステムになる、システムを構成する装置が１つ故障しても、システム全体は稼働し続ける

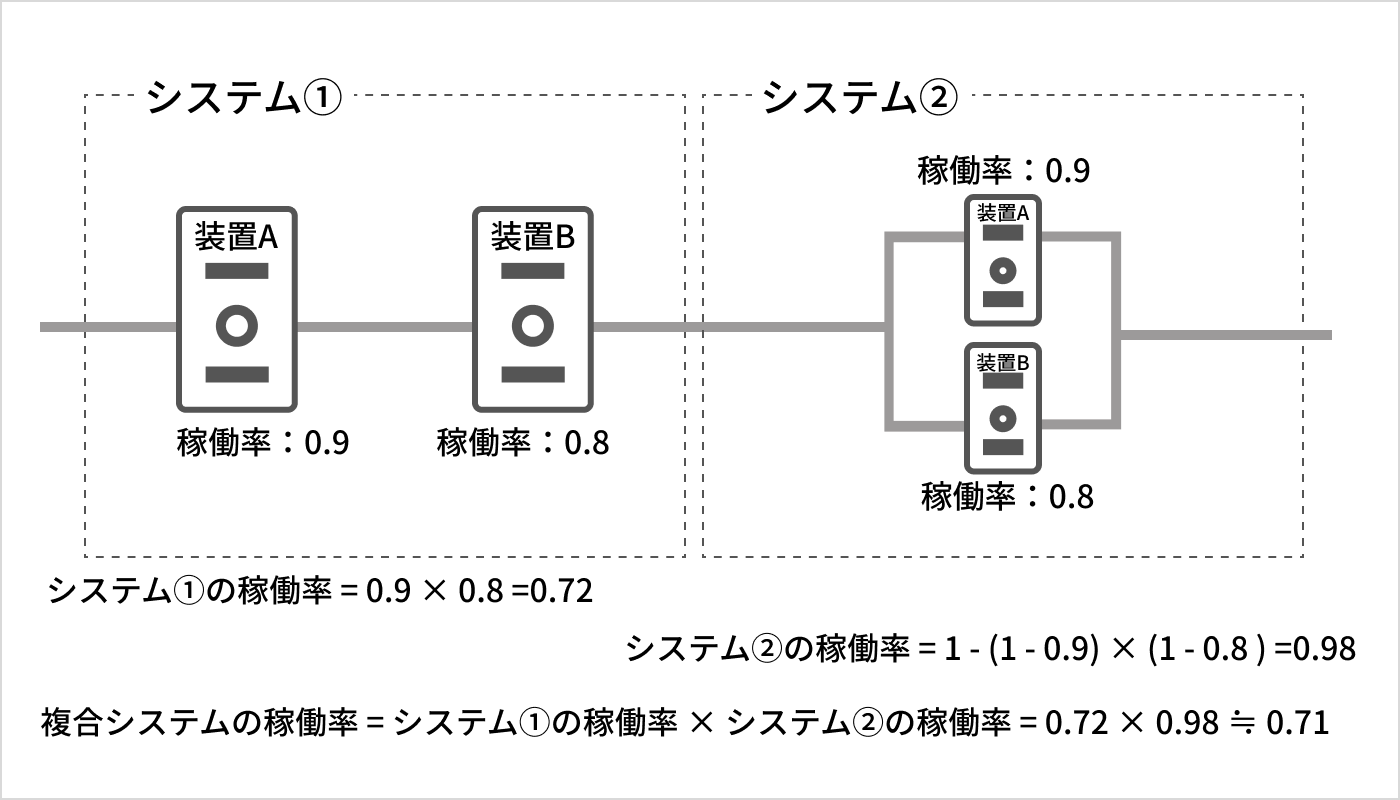
並列システムの稼働率は、次の式で求める

**並列システムの稼働率 = 1 ‐（ 1 - 装置Ａの稼働率 ）×（ 1 - 装置Bの稼働率）**

例えば、下図の並列システムの稼働率は0.98となる



**複合システム**  
複数の装置やシステムを組み合わせて作るシステム  
複合システムの稼働率は次の式で求める  
**複合システムの稼働率 = システム①の稼働率 × システム②の稼働率**例えば、下図の複合システムの稼働率は0.71となる  
システム１直列システムとシステム２並列システムを組み合わせた複合システム



**バスタブ曲線**  
装置の故障率と時間経過の関係を表した曲線

曲線の形がバスタブと似ているため、バスタブ曲線と呼ばれる  
バスタブ曲線は時間経過によって「初期故障期間」「偶発故障期間」「摩耗故障期間」に分けられる  
  
・初期故障期間：稼働を始めた直後は初期不良により故障率が高くなる  
・偶発故障期間：稼働が安定してくると、故障率が低くなる  
・摩耗故障期間：装置が寿命に近づくと、摩耗などから故障率が高くなる  
