# CONTENTS

# Volume IV User's Guide (IV-1 Working with Commands #1)

## 概要

**Igor Pro マニュアル：IV-2 ページ以降をもとに編集**

コマンドラインにコマンドを入力し、Enter キーを押すことでコマンドを実行できます。

コマンドを入力するのにノートブックを使うこともできます。
マニュアル III-4 Notebooks as Worksheets を参照してください。

コマンドを一から入力することもできますが、多くの場合、Igor のダイアログでコマンドを生成させ、実行させることになります。
コマンドウィンドウの履歴領域では、これまでに実行してきたコマンドの記録を確認でき、保存されているコマンドを簡単に再入力、編集、再実行することができます。
詳細は、マニュアル II-13 Command Window Shortcuts を参照してください。

## 複数のコマンド

複数のコマンドをセミコロンで区切れば、1 行に複数記述できます。
例えば、

```
wave1= x; wave1= wave2/(wave1+1); Display wave1
```

最後のコマンドの後にはセミコロンは不要ですが、あっても問題はありません。

## コメント

コメントは // で始まり、コマンドラインの実行可能な部分を終了します。
コメントは行の終わりまで使います。

## コマンドの最大長

コマンドラインの最大の長さは、2,500 バイトを越えてはいけません。

コマンドラインには次の行へ繋げる文字はありません。
しかし、中間変数を使うことで、ほとんどの場合、1 つのコマンドを複数の行に分割することができます。
例えば、

```
Variable a = sin(x-x0)/b + cos(y-y0)/c
```

は、次にようにも書くことができます：

```
Variable t1 = sin(x-x0)/b
Variable t2 = cos(y-y0)/c
Variable a = t1 + t2
```

## パラメーター

Igor が数値パラメーターを必要とする、コマンド内のすべての場所で、数値式を使うことができます。
演算フラグ（例：/N=<数値>）では、式を括弧で囲む必要があります。
詳細は、マニュアル IV-12 Expression as Parameters を参照してください。

## リベラルなオブジェクト名

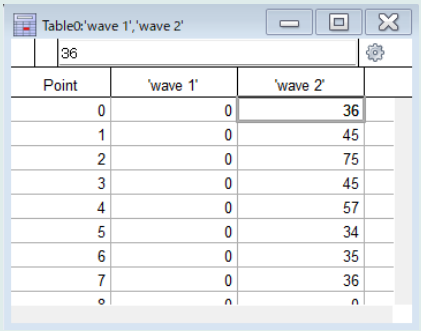
一般的に、Igor のオブジェクト名は、制限された文字セットに限定されています。
使用できるのは、英数字とアンダースコアのみです。
このような名前を「標準名」と呼びます。
コマンドで名前を使うときに、名前がどこで終わるかを特定するために、この制限は必要です。

ウェーブおよびデータフォルダーのみ、「リベラルな」名前を使うことができます。
リベラルな名前には、スペースやドットなど、ほとんどすべての文字を使うことができます（詳細は、マニュアル III-501 Liberal Object Names を参照）。
ただし、リベラルな名前の終わりを定義するには、シングルクオートを使って囲む必要があります。

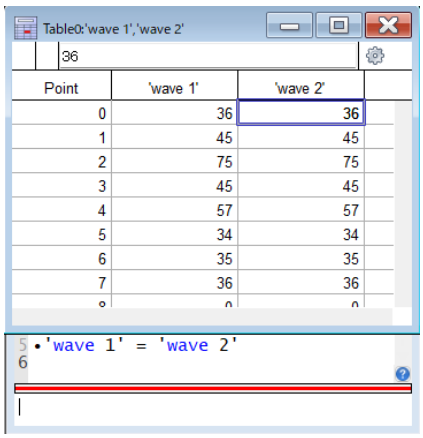
次の例では、スペースを含むため、ウェーブ名はリベラルでなければならず、クオートで囲まれています。

```
'wave 1' = 'wave 2'   // 正しい
wave 1 = wave 2       // 不正 - リベラルな名前はクオートで囲む必要がある
```

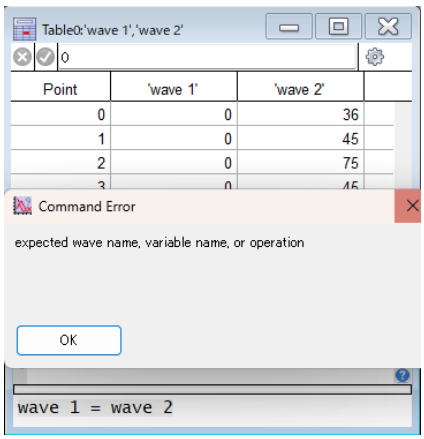
（この構文は、コマンドラインとマクロのみに適用され、ユーザー定義関数には適用されません。ユーザー定義関数では、ウェーブの読み込み／書き込みには　Wave References を使う必要があります）




wave 2 にデータがある状態で、




'wave 1' = 'wave 2'
を実行すると wave 2 のデータが
wave 1 にコピーされる




wave 1 = wave 2
を実行すると、エラーになる

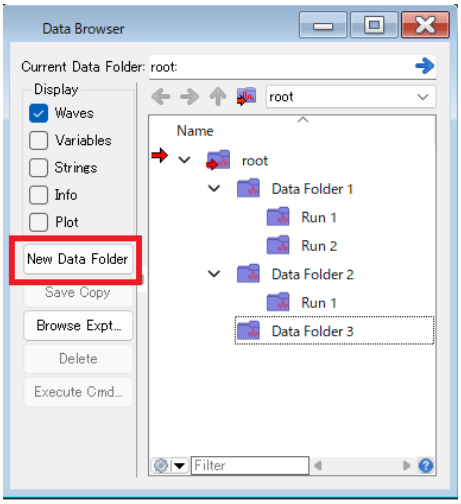
**注記：**　リベラルな名前をサポートするには、追加の作業とテストが必要になるため、ユーザー定義プロシージャでリベラル名を使うと、ときどき問題が発生することがあります（マニュアル IV-168 Programming with Liberal Names を参照）。

## コマンドとデータフォルダー

データフォルダーは、複数のデータセットが互いに干渉しないようにする方法を提供します。
データフォルダーは、Data Browser（メニュー Data → Data Browser）を使って、確認および作成（New Data Folder ボタン）することができます。

Root データフォルダーは常に存在し、多くのユーザーにとって必要なデータフォルダーはこれだけです。

上級ユーザーは、データを整理するために追加のデータフォルダーを作成したいと思うかもしれません。




現在のデータフォルダー、特定のデータフォルダー、または現在のデータフォルダーと相対的な位置にあるデータフォルダーにあるウェーブと変数を参照できます。

```
// wave1 は、現在のデータフォルダー内
wave1 = <expression>
// wave1 は、別のデータフォルダー内
root:'Background Curves':wave1 = <expression>
// wave1 は現在のデーターフォルダー内のあるデータフォルダー内
:'Background Curves':wave1 = <expression>
```

（この構文は、コマンドラインとマクロのみに適用され、ユーザー定義関数には適用されません。
ユーザー定義関数では、ウェーブの読み書きには、Wave References を使う必要があります。）

最初の例では、オブジェクト名を単独で使っています（`wave1`）。
Igor は現在のデータフォルダーでオブジェクトを探します。

2 番目の例では、完全なデータフォルダーパス（`root:'Background Curves':`）＋オブジェクト名を使っています。
Igor は指定されたデータフォルダーでオブジェクトを探します。

３番目の例では、相対的なデータフォルダーパス（`:'Background Curves':`）＋オブジェクト名を使っています。
Igor は現在のデータフォルダーで「Background Curves」というサブデータフォルダーを探し、そのデータフォルダー内のオブジェクトを探します。

重要：代入文の右辺（`<expression>`）（マニュアル IV-4 Assigned Statements を参照）は、代入先のオブジェクトが含まれるデータフォルダーのコンテキストで評価されます。
例えば、

```
root:'Background Curves':wave1 = wave2 + var1
```

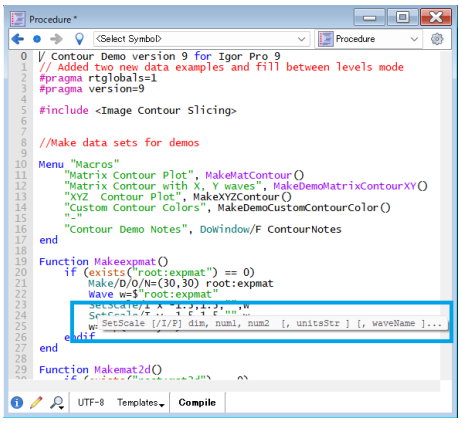
これを機能させるには、wave2 と var1 が Background Curve データフォルダー内に存在していなければなりません。

以降の例では、オブジェクト名のみを使っているため、現在のデータフォルダー内のデータを参照します。

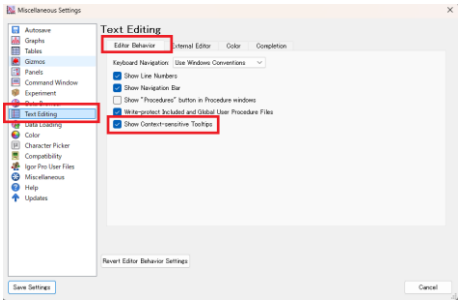
# コマンドのツールチップ

**Igor Pro マニュアル：IV-3 ページ以降をもとに編集**

プロシージャウィンドウやコマンドウィンドウで、マウスカーソルをコマンドや関数などの上に置くと、そのコマンドに関する情報を含むツールチップを表示します。
WMButtonAction などのビルトイン構造体の名前にカーソルを合わせると、構造体の定義を表示します。



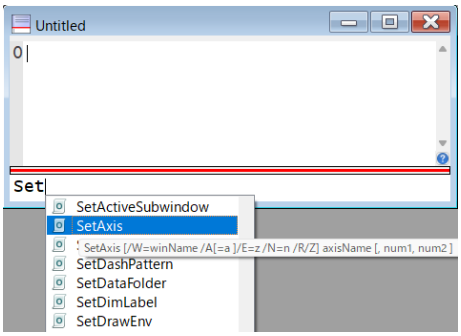

この機能をコントローするには、メニュー Misc → Miscellaneous Settings を選択し、Miscellaneous Settings ダイアログを表示します。
Text Editing セクションを選択し、Editor Behavior タブを選択し、Show Context-sensitive Tooltips チェックボックスを外すことで無効にできます。



# コマンドのオートコンプリート

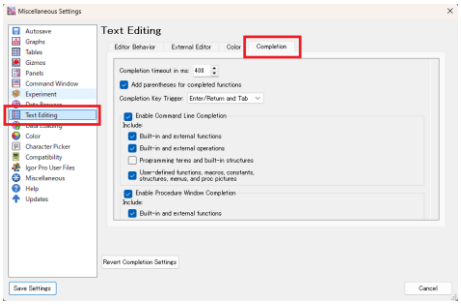
**Igor Pro マニュアル：IV-3 ページ以降をもとに編集**

プロシージャウィンドウとコマンドラインでは、コマンドのオートコンプリートが可能です。




コマンドの最初の数文字を入力すると、ポップアップが表示され、コマンドを素早く選択することができます。
矢印キーを使って、候補のリストを上下に移動します。
Enter キーまたは Tab キーを押して、ハイライトされたテキストを文書に挿入します。

マウスカーソルを候補オプションの上に移動させたり、矢印キーを使ってハイライトされたオプションを変更すると、選択したオプションのテンプレートを示すツールチップが表示されます。




コマンドのオートコンプリートの設定は、Miscellaneous Settings ダイアログの Text Editing カテゴリにある Completion タブで調整できます。
プロシージャウィンドウとコマンドラインで、それぞれ個別にオートコンプリートを有効/無効にすることができます（Enable Procedure Window Completion と Enable Command Line Completion チェックボックス）。

最後のキー入力とオートコンプリートオプションのポップアップ表示の間の遅延をコントロールし、括弧を必要とする項目を挿入するときに、左括弧を追加するかどうかをコントロールすることもできます。

現状、ウェーブや変数などのオブジェクト名のオートコンプリートはサポートされていません。
これらのサポートは、将来のバージョンで追加される可能性があります。
コマンドのオートコンプリートはコンテキスト依存ではありません。
これは、入力中のテキストのコンテキストでは意味をなさないオプションが提示されることがあることを意味します。
例えば、「Display/HID」と入力すると、次のようなオートコンプリート候補が表示される場合があります：「HideIgorMenus」「HideInfo」「HideProcedures」「HideTools」。
これらのオプションは、Display コマンドのフラグとしては有効ではありませんが、コマンドのオートコンプリートのアルゴリズムではこれらを除外することはできません。

# コマンドの形式

**Igor Pro マニュアル：IV-4 ページ以降をもとに編集**

コマンドラインから実行できるコマンドには、基本的に 3 つの異なる形式があります。

- 代入文

```
wave1 = sin(2*pi*freq*x)
```

- 操作コマンド

```
Display wave1,wave2 vs xwave
```

- ユーザー定義プロシージャコマンド

```
MyFunction(1.2,"hello")
```

Igor が入力されたコマンドを実行するときには、 3 つの基本的なコマンド形式のうち、どの形式のコマンドが入力されたかを判断する必要があります。
コマンドがウェーブ名または変数名で始まる場合、それは代入文と判断されます。
コマンドが組み込みコマンドまたは外部コマンドの名前で始まる場合、それは操作コマンドと判断されます。
コマンドがユーザー定義マクロ、ユーザー定義関数、外部関数の名前で始まる場合、それはそれぞれの定義に応じて処理されます。

組み込み関数は、代入文の右辺、または演算子や関数のパラメーターとしてのみ、使うことができることに注意してください。
したがって、コマンド

```
sin(x)
```

だけを入力することはできず、エラー「Expected wave name, variable name, or operation.」（ウェーブ名、変数名、コマンドが必要です）となります。
一方、次のコマンドは使うことができます。

```
Print sin(1.567)            // sin は print コマンドのパラメーター
wave1 = 5*sin(x)            // sin は代入の右辺にある
```

スペルミスなどにより、入力されたものを解釈できない場合、エラーダイアログが表示され、コマンドラインでエラーがハイライト表示されます。

# 代入文

**Igor Pro マニュアル：IV-4 ページ以降をもとに編集**

代入文コマンドは、ウェーブまたは変数名で始まります。
コマンドは、指定されたオブジェクトのすべてまたは一部に値を割り当てます。
代入文は、代入先、代入演算子、式の３つの部分で構成されます。
例えば、

```
wave1 = 1 + 2 * 3^2
```

（代入先[wave1] 代入演算子[=] 式[1 + 2 * 3^2]）

これにより、wave1 のすべてのポイントに 19 が代入されます。
上記の例のスペースは必要ありません。
次のように書くこともできます。

```
wave1=1+2*3^2
```

ウェーブの代入文に関する詳細は、マニュアル II-74 Waveform Arithmetic and Assignments を参照してください。

以下、いくつかの例を示します。
str1： String コマンドで作成された文字列変数
var1： Variable コマンドで作成された数値変数
wave1：Make コマンドで作成されたウェーブ

```
str1 = "Today is " + date()                 // 文字列の代入

str1 += ", and the time is " + time()       // 文字列の連結

var1 = strlen(str1)                         // 変数の代入

var1 = pnt2x(wave1,numpnts(wave1)/2)        // 変数の代入

wave1 = 1.2*exp(-0.2*(x-var1)^2)            // ウェーブの代入（ウェーブ全体）

wave1[3] = 5                                // ウェーブの代入（ウェーブの特定のポイント）

wave1[0,;3]= wave2[p/3] *exp(-0.2*x)        // ウェーブの代入（ウェーブの特定の範囲）
```

これらはすべて、現在のデータフォルダー内のオブジェクトに対して操作を行います。
別のデータフォルダー内のオブジェクトに対して操作を行うには、次のようにデータフォルダーのパスを使う必要があります：

```
root:'run 1':wave1[3] = 5
```

（この構文は、コマンドラインとマクロのみに適用され、ユーザー定義関数には適用されません。
ユーザー定義関数では、ウェーブの読み書きには Wave References を使う必要があります。）

詳細は、マニュアル II-8 Data Folders を参照してください。

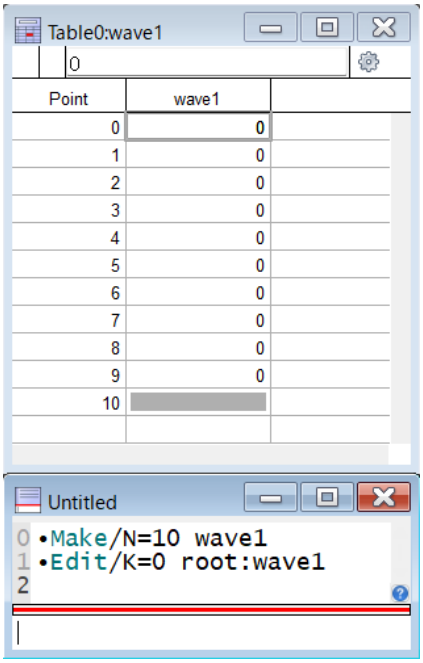
## 代入演算子

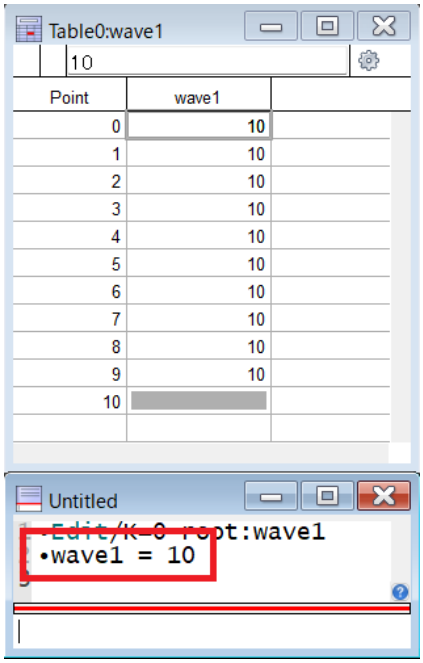
代入演算子は、式と代入先が結合される方法を決定します。
Igor では、以下の代入演算子をサポートしています。

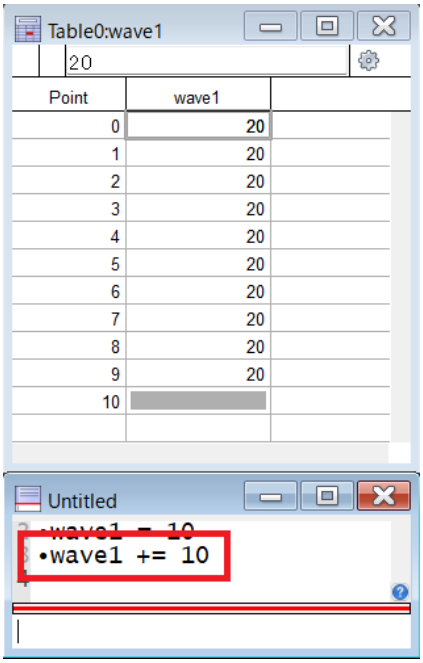
| 演算子 | 代入処理 |
|---|---|
| = | 代入先の内容は、右辺の式の値に設定されます。 |
| += | 右辺の式が、代入先の内容に加算されます。 |
| -= | 右辺の式が、代入先の内容から減算されます。 |
| *= | 代入先の内容が、右辺の式によって乗算されます。 |
| /= | 代入先の内容が、右辺の式によって除算されます。 |
| := | 右辺の式の値が変更されるたびに、代入先の内容が式の値に動的に更新されます。<br>:=演算子は、式に対する代入先の「依存関係」を確立するといわれます。 |

```
wave1 = 10
```
は、wave1 の各 Y 値を 10 に設定します。

```
wave1 += 10
```
は、wave1 の各 Y 値に 10 を加算します。
これは次の式と等価です。
```
wave1 = wave1 + 10
```




代入演算子 =、:=、+= は文字列の代入文でも動作しますが、-=、*=、/= は動作しません。
例えば、
```
String str1; str1 = "Today is "; str1 += date(); Print str1
```
は、“Today is Mon, Jan 6, 2025” のような文字列を出力します。

:= 演算子に関する詳細は、マニュアル IV-9 Dependencies を参照してください。

## 演算子

代入文の式（右辺）部分でサポートしている演算子の一覧を優先順に説明します。

| 演算子 | | | 処理 |
|---|---|---|---|
| ++ | -- | | プリフィックスとポストフィックスをインクリメント／デクリメント（Igor Pro 7 以降が必要）。<br>ユーザー定義関数内のローカル変数でのみ使うことが可能。 |
| ^ | << | >> | 指数、ビット単位の左シフト／右シフト。<br>シフトは Igor Pro 7 以降が必要。 |
| - | ! | ~ | 否定、論理補数、ビット単位の補数。 |
| * | / | | 乗算、除算。 |
| + | - | | 加算や文字列の連結、減算。 |
| == | != | > | 比較オペレーター。 |
| < | >= | <= | |
| & | \| | %^ | ビット単位の AND、OR、XOR。 |
| && | \|\| | ? : | 論理的 AND、OR、条件オペレーター。 |
| $ | | | 次の文字列式を名前として代入。 |

比較演算子は NaN パラメーターでは動作しません。
定義上、NaN は他の NaN と比較しても、それ以外のものと比較しても、常に False となるためです。
numtype を使って、値が NaN であるかどうかを判定できます。

比較演算子、ビット単位の AND、OR、XOR は右から左に結合します。
したがって、

```
a==b>=c
```

は

```
(a==(b>=c))
```

を意味します。
例えば、

```
2==1>=0
```

は、1 ではなく、0 と評価されます（ `(2==(1>=0))` と評価されるため ）。

その他すべての二項演算子は左から右に結合されます。

誰もが直感的に理解している一般的な演算子の優先順位を除いて、混乱を避けるために括弧を使うと便利です。
あなたは優先順位と結合性を理解しているかもしれませんが、あなたのコードを読む他の人は理解していないかもしれません。

単項否定は、そのオペランドの符号を変更します。
論理補数は、ゼロ以外のオペランドをゼロに、ゼロのオペランドを 1 に変更します。
ビット単位の補数は、そのオペランドを切り捨てにより符号なし整数に変換し、その後、２進数の補数に変更します。

指数関数は、左辺のオペランドを右辺のオペランドで指定された累乗まで累乗します。
つまり、$3^2$ は、`3^2` と表記されます。
a^b という式において、結果が実変数またはウェーブに代入される場合、b が整数でない場合は、a は負であってはいけません。
結果が複素式で使われる場合、負の a、小数 b、または複素数 a または b の任意の組み合わせが許可されます。

指数が整数である場合、乗算のみを使って式を評価します。
効率的な評価を得るために a^2 を a*a と記述する必要はありません。
自動的に同等の処理が行われます。
一方、指数が整数でない場合は、対数を使って評価が行われるため、実数式における負の a に対する制限が存在します。

論理和 OR（||）と論理積 AND（&&）は、２つの式の真偽を決定します。
AND 演算は、両方の式が真の場合のみ真を返します。
OR 演算は、どちらかが真であれば真を返します。
この演算は NaN に対しては定義されていません。
また、複素式では使うことができません。

論理演算子は左から右の順に評価され、必要のないオペランドは評価されません。
例えば、

```
if(MyFunc1() && MyFunc2())
```

において、MyFunc1() が偽（ゼロ）を返すとき、MyFunc2() は評価されません。
これは式全体が既に偽であるためです。
右辺の式に別の作用（ウェーブの作成やグルーバル値の設定など）がある場合、予期せぬ結果が生じる可能性があります。

ビット演算の AND（&）、OR（|）、XOR（%^）は、オペランドを切り捨てにより符号なし整数に変換し、２進数の AND、OR、排他的 OR を返します。

ビット単位のシフト << と >> は、左辺のオペランドを指定されたビット数だけシフトして返します。
この機能は Igor Pro 7 以降が必要です。
左辺の演算はシフト前に整数に切り捨てられます。

条件演算子（? :）は、if-else-endif 式の省略形です。

```
<expression> ? <TRUE> : <FALSE>
```

最初のオペランド <expression> はテスト条件です。

ゼロ以外の場合、<TRUE> オペランドを評価します。
そうでない場合は、<FALSE> が評価されます。
テスト条件に従って、オペランドは 1 つだけ評価されます。
これは次のように書いたことと同じです。

```
if( <expression> )
        <TRUE>
else
        <FALSE>
endif
```

条件演算子内の「:」文字は、常に 2 つの隣接するオペランドとスペースで区切らなければなりません。
どちらかのスペースを省略すると、エラー（"No such data folder"）が発生します。
これは、この式がデータフォルダーのパスとして解釈される可能性もあるためです。
安全のため、オペランドと演算子記号の間には常にスペースを挿入してください。

オペランドは数値でなければなりません。
文字列の場合は、SelectString コマンドを使ってください。
条件演算子と複素式を併用する場合、演算子が式を評価するときには実部のみが使われます。

条件演算子は、混乱を招きやすいので、使うときには注意が必要です。
例えば、次の式が何を返すかは、見ただけではわかりにくいです：

```
1 ? 2 : 3 ? 4 : 5
```

この場合は「4」です。一方、

```
1 ? 2 : (3 ? 4 : 5)
```

は「2」を返します。
常に括弧を使って、あいまいさを排除してください。

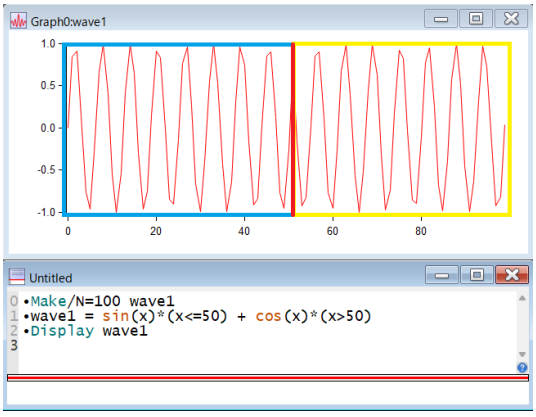
比較演算子は、比較結果が真であれば 1 を返し、偽であれば 0 を返します。
例えば、== 演算子は、そのオペランドが等しい場合は 1 を返し、等しくない場合は 0 を返します。
!= 演算子はその逆です。

比較演算子は値 1 または 0 を返すため、興味深い方法で使うことができます。
次の代入文：

```
wave1 = sin(x)*(x<=50) + cos(x)*(x>50)
```

は、x=50 より下では Sin 波、x=50 より上では Cos 波となるように wave1 を設定します。

マニュアル II-82 Example: Comparison Operators and Wave Synthesis も参照してください。




二重等号（==）は等価を意味し、単一等号（=）は代入を意味することを忘れないでください。

丸め誤差により、2 つの数値の等値性をテストするために == を使うと、不正確な結果が返される場合があります。
ある数値が狭い範囲に収まっているかどうかを確認するには、<= および >= を使うのが安全です。

例えば、ある変数が 3 分の 1 と等しいかどうかを比較したいとします。
式、
```
(v1 == 1/3)
```
は、丸め処理のため、失敗する可能性があります。
次のようにするのがより安全です：
```
((v1 > .33332) && (v1 < .33334))
```

数値が整数である場合、$2^{53}$（約 $10^{16}$）より小さい整数は倍精度浮動小数点で正確に表現されるため、== を使っても安全です。

演算子に関するここまでの説明は、数値のオペランドが前提とされています。
+ 演算子は、数値と文字列の両方のオペランドで動作する唯一の演算子です。
例えば、str1 が文字列変数である場合、代入文
```
str1 = "Today is " + "a nice day"
```
は、str1 に "Today is a nice day" という値を代入します。
もう 1 つの文字列演算子、$ については、マニュアル IV-18 String Substitution Using $ を参照してください。

括弧で指定がない限り、単項の否定または補数演算が最初に実行され、その後、累乗、乗算または除算、加算または減算、比較演算子の順に実行されます。

単項の否定・補数演算　＞　累乗　＞　乗算・除算　＞　加算・減算　＞　比較演算子

次のウェーブの代入、
```
wave1 = ((1 + 2) * 3) ^ 2
```
は、wave1 のすべてのポイントに値 81 を代入しますが、
```
wave1 = 1 + 2 * 3 ^ 2
```
は、値 19 を代入します。

-a^b は、このルールに関する例外であり、-(a^b) として評価されます。

文字列の置換、部分文字列、ウェーブインデックスの優先順位はやや複雑です。
不明な場合は、必要な優先順位を強制するために括弧を使ってください。

## オペランド

3.141 や 27 のような文字通りの数値に加えて、演算子は変数や関数値にも作用することができます。
次の代入式：
```
var1 = log(3.7) + var2
```
では、演算子 + は、log が返した関数値と変数 var2 を対象として処理を行います。

## 数値型

Igor では、各数値型の代入先オブジェクト（ウェーブまたは変数）は、それぞれ独自の数値型を持っています。
数値型は、数値の精度（倍精度浮動小数点など）と数値の型（実数または複素数）で構成されます。

ウェーブは単精度または倍精度浮動小数点、またはさまざまなサイズの整数とすることができますが、変数は常に倍精度浮動小数点です。

代入先の数値の精度は計算に影響を与えません。
FFT のような一部の演算を除いて、すべての計算は倍精度で行われます。

ウェーブには整数数値型を使用できますが、ウェーブの式は常に倍精度浮動小数点で評価されます。
浮動小数点は、ウェーブに値を保存する前の最終ステップとして、四捨五入により整数に変換されます。
格納される値が、指定された整数型で表現できる値の範囲を超える場合、結果は undefined となります。

Igor Pro 7 以降では、整数型ローカル変数と 64bit 整数ウェーブをサポートしています。
これらのいずれかが代入文の代入先である場合、整数演算を使って計算を実行します。
詳細は、マニュアル IV-38 Expression Evaluation を参照してください。

代入式の初期数値の型（実数または複素数）は、代入先の数値の型によって決定されます。
Igor は数値型の「予期しない」または「実行時」の変更に対応できないため、これは重要です。
例えば、負の数の平方根を求める場合、その後のすべての演算は複素数を用いて行う必要があります。

以下に例を示します。

```
Variable a, b, c, var1
Variable/C cvar1
Make wave1

var1= a*b
cvar1= c*cmplx(a+1,b-1)
wave1= var1 + real(cvar1)
```

最初の式（`var1= a*b`）は実数型を使って評価されます。
２番目の式（`cvar1= c*cmplx(a+1,b-1)`）には、２つの型が混在しています。
c と cmplx 関数の結果の乗算は複素数として評価されますが、cmplx 関数の引数は実数として評価されます。
３番目の式（`wave1= var1 + real(cvar1)`）は、実関数の引数が複素数として評価されることを除いて、実数として評価されます。

## 定数型

Igor のプロシージャファイルでは、名前付きの数値定数および文字列定数を定義し、ユーザー定義関数の本体でそれらを使うことができます。
定数は、以下の構文を使ってプロシージャファイルで定義できます。

```
Constant <name1> = <literal number> [, <name2> = <literal number>]

StrConstant <name1> = <literal string> [, <name2> = <literal string>]
```

`static` プリフィックスを使うと、スコープを指定のソースファイルに限定することができます。
デバッグ用に、関数と同様に Override キーワードを使うことができます。

これらの宣言は、以下のような形で使うことができます。

```
Constant kFoo=1,kBar=2
StrConstant ksFoo="hello",ksBar="there"
```

```
static Constant kFoo=1,kBar=2
static StrConstant ksFoo="hello",ksBar="there"

Override Constant kFoo=1,kBar=2
Override StrConstant ksFoo="hello",ksBar="there"
```

プログラミングをする方は、「k」と「ks」プリフィックスを使うことでコードが読みやすくなっていることに気づくかもしれません。

数値定数、文字列定数の名前は、他のすべての名前と競合する可能性があります。
異なるファイルの静的定数、Override と併用する場合を除き、与えられた型の定数の重複は許可されません。
唯一の真の名前の競合は、パラメーターを受け取らない特定の組み込み関数（pi など）との間でのみ発生します。
変数名は定数を上書きしますが、定数は pi などの関数を上書きします。

## 依存代入文

グローバル変数やウェーブを設定すると、他のグローバルオブジェクトが変更されたときに、その内容が自動的に再計算されるようにすることができます。
詳細はマニュアル IV-9 Dependencies を参照してください。

# 操作コマンド

**Igor Pro マニュアル：IV-10 ページ以降をもとに編集**

操作コマンドとは、組み込みまたは外部のルーチンで、アクションを実行しますが、関数とは異なり、直接値を返すことはありません。
以下のように使います。

```
Make/N=512 wave1
Display wave1
Smooth 5, wave1
```

操作コマンドは、Igor の大部分の作業を実行し、ダイアログを使って作業を行うときに自動的に生成、実行されます。

これらのダイアログを使って、興味のある操作を試してみることができます。
各種のダイアログで何かをクリックすると、Igor がコマンドを生成します。
これにより、操作のコマンドを確認したり、ユーザー定義プロシージャで使うコマンドを生成したりする便利な方法を提供します。
すべてのビルトイン操作コマンドの完全なリストは、マニュアル V-1 Igor Reference を参照してください。
ヘルプブラウザの Command Help タブを使って構文を学習することもできます。

操作コマンドの構文は多様ですが、一般的には「コマンド名+フラグのリスト（例：/N=512）+パラメーターリスト」の順で構成されます。
「コマンド名」は、コマンドの主な動作を指定し、残りのコマンドの構文を決定します。
「フラグのリスト」は、コマンドのデフォルトの動作のバリエーションを指定します。
デフォルトの動作で問題ない場合は、フラグは不要です。
「パラメーターリスト」は、コマンドの対象となるオブジェクトを特定します。
いくつかのコマンドはパラメーターを持ちません。

例えば、コマンド
`Make/D/N=512 wave1, wave2, wave3`
の操作コマンド名は「`Make`」です。
フラグのリストは「`/D/N=512`」です。
パラメーターのリストは「`wave1, wave2, wave3`」です

数値パラメーターを期待するコマンドのパラメーターリストでは数値式を使用できますが、フラグでは式を括弧で囲む必要があります。
例えば、

```
Variable val = 1.0
Make/N=(val) wave0, wave1
Make/N=(numpnts(wave0)) wave2
```

もっとも一般的なパラメーターの種類は、リテラル数値または数値式、リテラル文字列または文字列式、名前、ウェーブです。
上記の例では、Make コマンドに渡される　wave1 は名前パラメーターです。
これは、例えば、Display と　Smooth コマンドに渡されるときのウェーブパラメーターです。

名前パラメーターは、既に存在している場合も、そうでない場合もあるウェーブを参照することができます。
一方、ウェーブパラメーターは、必ず既存のウェーブを参照しなければなりません。