CONTENTS

ビジュアルヘルプ – 変数

変数

このセクションでは、グローバルな数値変数と文字列変数の特性と用途について解説します。
グローバル変数を使ったプログラミングの詳細については、ヘルプ Accessing Global Variables And Waves を参照してください。

数値変数は倍精度浮動小数点型であり、実数または複素数をとることができます。
文字列変数は、任意のバイト数を格納できます。
エクスペリメントを保存する時にすべてのグローバル変数を保存し、エクスペリメントを再開する時にそれらを復元します。

数値変数、または数値変数を含む数値式は、リテラル数値が適切な場所であればどこでも使用できます。
これには、代入文のオペランドや、演算、関数、マクロのパラメーターとしての使用も含まれます。

コマンドのフラグのパラメーターで数値変数を使う場合は、括弧が必要です。
ヘルプ Operations and Functions Syntax を参照してください。

文字列変数や文字列式は、文字列の使用が適切な場所であればどこでも使用できます。
また、Igor Pro がウェーブ、変数、グラフ、テーブル、ページレイアウトなどのオブジェクト名を期待する場所では、文字列変数をパラメーターとして使うこともできます。
詳細は、ヘルプ Converting a String into a Reference Using $ を参照してください。

Igor Pro 7 以降では、文字列変数の内容が UTF-8 でエンコードされているものとみなします。
Igor Pro 6 以前に作成された文字列変数に非 ASCII テキストを格納している場合は、Igor Pro 7 以降で使う時に変換を行う必要があります。
詳細は、ヘルプ String Variable Text Encodings を参照してください。

グローバル変数の作成

常に存在する、システム変数と呼ばれる 20 個の組み込み数値変数（K0 … K19）があります。
Igor Pro では、主に CurveFit コマンドの結果を返すためにこれらを使います。
システム変数を他の目的で使うことは控えることを推奨します。

その他の変数はすべてユーザー変数です。
ユーザー変数は、次の 2 つの方法のいずれかで作成できます。

- 特定のコマンドの過程で、Igor Pro によって自動的に行われる
- ユーザーによる明示的な操作（Variable/G と String/G コマンドを介して）

コマンドラインから Variable または String コマンドを使って変数を直接作成する場合、その変数は常にグローバル変数となるため、/G フラグを省略できます。
プロシージャ内で変数をグローバル変数にするには、/G フラグが必要です。
/G フラグには副次的な効果があり、既存グローバル変数の上書きが可能になります。

グローバル変数の用途

グローバル変数には、有用である理由となる 2 つの特性があります。
それは、「グローバル性」と「永続性」です。

グローバルであるため、どのプロシージャからでもアクセスできます。
また、永続性があるため、設定を長期にわたって保存するために利用できます。

グローバル変数を「グローバル性」そのもののために使うと、プロシージャ間に明示的でない依存関係が生じます。
これにより、プロシージャの理解やデバッグが困難になります。
パラメーターを使用できるにもかかわらず、あるプロシージャから別のプロシージャへ情報を渡すためにグローバル変数を使うことは、悪いプログラミング慣行であり、ごく稀な場合を除いて避けるべきです。

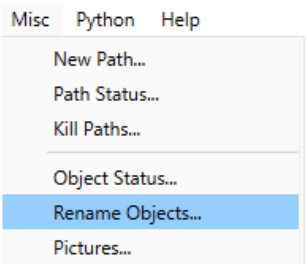
グローバル変数をその「グローバル性」を活かして適切に活用できるのは、値がほとんど変化せず、かつ多くのプロシージャからアクセスする必要がある場合です。

変数名

変数名は 1～255 バイトで構成されます。
使用できるのは ASCII 文字のみです。
最初の文字はアルファベットでなければなりません。
残りの文字は、アルファベット、数字、またはアンダースコアを使用できます。
Igor Pro における変数名は大文字と小文字を区別しません。

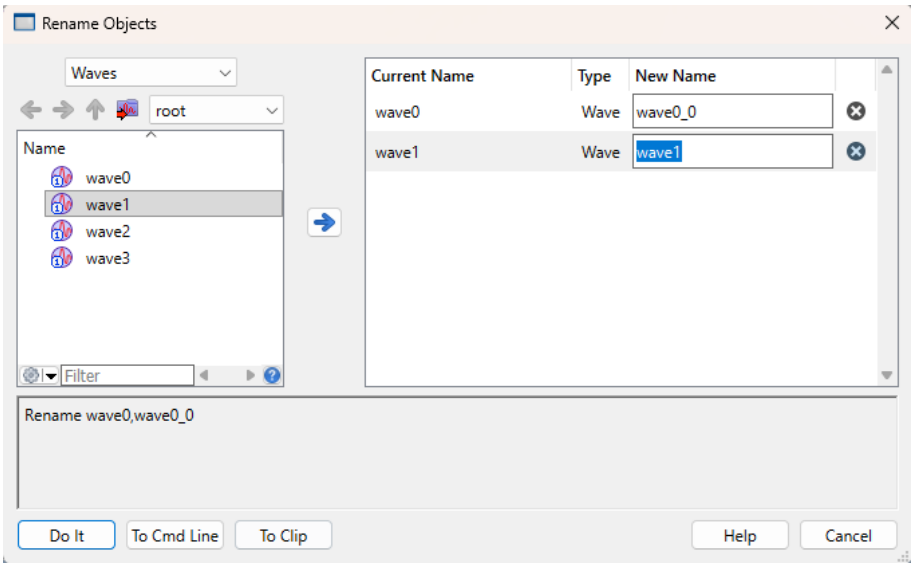
Igor Pro 8.0 以前のバージョンでは、変数名は 31 バイトまでに制限されていました。
長い変数名を使う場合は、エクスペリメントに Igor Pro 8.0 以降が必要となります。

変数名は、リベラル名ではなく、標準的な名前でなければなりません。
詳細は、ヘルプ Object Names を参照してください。



変数名は、他のオブジェクト、関数、または演算子の名前と重複してはなりません。

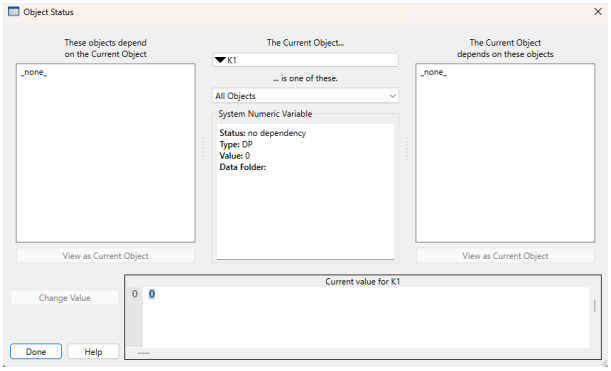
変数の名前を変更するには、Rename コマンドを使うか、Misc メニューから Rename Objects ダイアログを開いて行うことができます。



システム変数

Igor Pro にはシステム変数が組み込まれています。
これらは主に、Igor Pro の旧バージョンとの互換性を確保するために用意されているものであり、一般的な用途で

の使用は推奨されません。
Misc メニューの Object Status を選択すると、システム変数とその値の一覧を確認できます。



K0、K1…K19 という名前のシステム変数が 20 個と、
veclen という名前のシステム変数が 1 個あります。
K 系列の変数は、カーブフィッティング処理で使われます。

システム変数は、プリエンプティブスレッドからは使用できず、通常の「メインスレッド」からのみ使用できます。
プリエンプティブスレッドの詳細は、ヘルプ ThreadSafe Functions and Multitasking と Thread Data Environment を参照してください。

veclen 変数は、互換性を確保するために用意されています。
以前のバージョンの Igor Pro では、この変数には Make コマンドによって生成されるウェーブのデフォルトのポイント数が格納されていました。
しかし、現在はそうではありません。
/N=(<ポイント数>) フラグを使って明示的に指定しない限り、Make コマンドでは常に 128 ポイントのウェーブが生成されます。

CurveFit コマンドは、結果を K 変数に格納しますが、これは互換性の理由によるものであり、同じ結果を格納するためにユーザー変数やウェーブも作成します。

ただし、CurveFit コマンドでは、手動推測モードを指定した場合、初期パラメーターの推定値を設定するためにシステム変数が使われます。
また、kwCWave キーワードを使えれば、この目的でウェーブを利用することも可能です。
CurveFit コマンドのヘルプを参照してください。

必要がない限り、システム変数に頼らない方がよいでしょう。
Igor Pro がさまざまなタイミングでシステム変数に書き込みを行うため、予期しないタイミングで値が変更される可能性があります。

Data Browser には、システム変数は表示されません。

システム変数は、旧バージョンの Igor Pro でも読み込めるように、単精度の値としてディスクに保存されます。
したがって、無期限に保持したい値は、独自のグローバル変数に保存してください。

ユーザー変数

Variable/G（「数値変数」のセクションを参照）と String/G（「文字列変数」のセクションを参照）コマンドを使うことで、独自のグローバル変数を作成できます。
作成した変数は、数値変数であれ文字列変数であれ、「ユーザー変数」と呼ばれます。
Misc メニューの Object Status を選択すると、グローバルユーザー変数を閲覧できます。
Data メニューの Data Browser を使って、変数を表示することもできます。

グローバルユーザー変数は、主にプロシージャで使われる永続的な設定を格納するために使われます。

特殊なユーザー定数

一部のコマンドを実行すると、Igor Pro は自動的に特別なユーザー変数を作成します。
例えば、CurveFit コマンドでは、カーブフィッティングによって生成されたさまざまな結果を格納するために、ユ

ーザー変数 V_chisq などが作成されます。
これらの変数の名前は、数値変数の場合は常に “V_” で、文字列変数の場合は常に “S_” で始まります。
これらの変数の意味については、ヘルプ Igor Pro Reference に、それらを生成するコマンドとともに記載されています。

さらに、Igor Pro では、特定のルーチンのデフォルトの動作を変更するために作成できる V_ 変数を、場合によってはチェックすることがあります。
例えば、V_FitOptions という名前の変数を作成すると、その変数を使って CurveFit、FuncFit、FuncFitMD コマンドをコントロールします。
これらの変数の使用方法については、影響を受けるコマンドとともにドキュメントに記載されています。

コマンドによって V_ と S_ 変数が作成される場合、そのコマンドがコマンドラインから実行された場合はグローバル変数となり、プロシージャ内で実行された場合はローカル変数となります。
詳細は、ヘルプ Accessing Variables Used by Igor Operations を参照してください。

数値変数

数値のユーザー変数は、コマンドラインまたはプロシージャ内で Variable コマンドを使って作成します。
Variable コマンドの構文は次のとおりです。

```
Variable [flags] varName [=numExpr] [,varName [=numExpr]]...
```

オプションのフラグは 3 つあります。

/C　複素変数を指定します

/D　非推奨。以前のバージョンでは、倍精度を指定するために使われていました。現在は、すべての変数が倍精度となっています

/G　変数をグローバル変数として指定し、既存の変数があれば上書きします

=numExpr を使って数値式で初期値を指定すると、変数は作成時に初期化されます。
数値変数を作成する時に初期化子を指定しない場合、その変数は 0 に初期化されます。

複数の変数を一度に作成するには、変数名と（任意の）初期化子をコンマで区切ります。

プロシージャ内で使った場合、/G フラグが指定されていない限り、新しい変数はそのプロシージャのローカル変数となります。
コマンドラインで使った場合、新しい変数は常にグローバル変数となります。

以下は、初期化を伴う変数の宣言の例です。

```
Variable v1=1.1, v2=2.2, v3=3.3*sin(v2)/exp(v1)
```

/C フラグが指定されないため、v1、v2、v3 のデータ型は倍精度実数型となります。

/G フラグが指定されていないため、コマンドラインから直接 Variable コマンドを実行した場合はこれらの変数はグローバル変数となり、プロシージャ内で実行した場合はローカル変数となります。

Variable/G varname は、指定された名前の変数がすでに存在するかどうかに関係なく呼び出すことができます。
その変数がすでに存在する場合、その操作に変数の初期値が含まれていない限り、その内容はコマンドによって変更されません。

複素変数に値を代入するには、cmplx() 関数を使います。

```
Variable/C cv1 = cmplx(1,2)
```

グローバルユーザー変数は、Data Browser または KillVariables コマンドを使って削除できます。
構文は次のとおりです。

```
KillVariables [flags] [variableName [,variableName]...]
```

２つのオプションフラグがあります。

/A　　現在のデータフォルダー内のすべてのグローバル変数を削除します。/A オプションを使う場合は、variableName を省略してください

/Z　　削除対象のグローバル変数が存在しない場合でも、エラーは発生しません

例えば、グローバル変数 cv1 が以前に定義されていたかどうかを気にせずに、それを破棄するには、次のコマンドを使います。

```
KillVariables/Z cv1
```

グローバル変数を削除すると、コードがすっきりし、メモリも少し節約できます。
システム変数やローカル変数は削除できません。

現在のデータフォルダー内のすべてのグローバル数値変数を削除するには、次のように入力します。

```
KillVariables/A/Z
```

文字列変数

ユーザー文字列変数は、コマンドラインまたはプロシージャ内で String コマンドを呼び出すことで作成します。
構文は次のとおりです。

```
String [/G] strName [=strExpr] [,strName [=strExpr]... ]
```

オプションの /G フラグは、その文字列をグローバル変数として扱うことを指定し、既存の文字列変数があれば上書きします。

コマンドラインやマクロから String を呼び出すと、文字列変数は指定された初期値で初期化されます。
値が指定されていない場合は、空の文字列（””）で初期化されます。

ユーザー定義関数内でローカル文字列変数を宣言した場合、初期値またはその後の代入文によって値が割り当てられるまでは、その変数は null（値がない）状態になります。
ユーザー定義関数内で null のローカル文字列変数を使うと、エラーを発生させます。

プロシージャ内で String を呼び出す場合、/G フラグを指定しない限り、新しく作成される文字列変数はそのプロシージャのローカル変数となります。
コマンドラインから String を呼び出す場合、新しく作成される文字列は常にグローバル変数となります。

複数の文字列変数を一度に作成するには、それらの名前と、必要に応じて初期化子をコンマで区切ります。

以下は、初期化を伴う変数の宣言の例です。

```
String str1 = "This is string 1", str2 = "This is string 2"
```

/G オプションが指定されていないため、コマンドラインから直接 String を呼び出した場合はこれらの文字列はグローバル変数となり、プロシージャ内で呼び出した場合はローカル変数となります。

String/G strName は、指定された名前の変数がすでに存在するかどうかに関係なく呼び出すことができます。
その変数が文字列としてすでに存在する場合、そのコマンドに文字列の初期値が含まれていない限り、その内容は操作によって変更されません。

グローバル文字列は、Data Browser または KillStrings コマンドを使って削除できます。
構文は次のとおりです。

```
KillStrings [flags] [stringName [,stringName ]...]
```

２つのオプションフラグがあります。

/A　　現在のデータフォルダー内のすべてのグローバル文字列を削除します。/A オプションを使う場合は、stringName を省略してください

/Z　　削除対象のグローバル文字列が存在しない場合でも、エラーは発生しません

例えば、グローバル文字列 “myGlobalString” が以前に定義されていたかどうかを気にせずにそれを破棄するには、次のコマンドを使います。

```
KillStrings/Z myGlobalString
```

文字列を破棄すると、不要なデータが減り、メモリを少し節約できます。
ローカル文字列は破棄できません。

現在のデータフォルダー内にあるすべてのグローバル文字列変数を削除するには、次のように実行します。

```
KillStrings/A/Z
```

プロシージャにおけるローカル変数とパラメーター変数

プロシージャ内では、パラメーターまたはローカル変数として変数を作成できます。
これらの変数は、プロシージャの実行中にのみ存在します。
プロシージャの外部からはアクセスできず、プロシージャの呼び出しごとに値が保持されることはありません。
詳細は、ヘルプ Local Versus Global Variables を参照してください。

文字列変数とテキストのエンコーディング

Igor Pro は内部で Unicode を使っています。
Igor Pro 7 以前は、MacRoman、Windows-1252、Shift JIS などの Unicode 以外のテキストエンコーディングを使っていました。

過去のエクスペリメントで、非 ASCII 文字を含む文字列変数がある場合、それらは誤って解釈されてしまいます。
この問題に関する詳細は、ヘルプ String Variable Text Encodings を参照してください。