

スクリプト文法

日本語の翻訳状況 translated 100%

Outline



1. スクリプト文法
 1. メッセージウィンドウ
 1. メッセージウィンドウへの表示
 2. 改行/
 3. 改ページ
 4. クリック待ち
 5. 改行のエスケープ
 6. 表示色
 7. 表示ディレイ
 8. 表示ウェイト
 9. 変数の表示
 2. 制御構造
 1. if文
 2. else文
 3. elif文
 4. while文
 3. 四則演算
 1. 演算の優先順位
 2. 括弧による優先順位
 4. 変数
 1. 有効な値
 2. 変数への代入
 5. 論理式
 1. 論理演算子一覧
 6. ラベル
 1. ラベルの定義
 2. ラベルジャンプ
 7. ファンクション
 1. ファンクションの定義
 2. ファンクションコール
 3. return命令
 8. スクリプトジャンプ
 9. 選択肢
 1. 問題文の複数行表示
 2. 選択肢に対する条件式の設定
 10. コメント
 11. 乱数
 12. 描画命令
 1. 描画命令一覧
 2. エフェクトID一覧
 13. 画面演出
 14. サウンド命令
 1. サウンド命令一覧
 15. セーブ命令

メッセージウィンドウ

メッセージウィンドウへの表示

「」で囲まれた文字列を記述すると、メッセージウィンドウへ表示する文字列となります

1. // 「メッセージを表示します」とメッセージウィンドウに表示します
2. "メッセージウィンドウを表示します"

改行/

メッセージを連続して記述すると、改行してメッセージを表示します

1. "春はあけぼの。"
2. "やうやうしろくなりゆく山ぎは、すこしあかりて、"
3. "紫だちたる雲のほそくたなびきたる。"
- 4.
5. // ■上記記述は、以下のように表示します
6. // "春はあけぼの。"
7. // "やうやうしろくなりゆく山ぎは、すこしあかりて、"
8. // "紫だちたる雲のほそくたなびきたる。"

なお、メッセージウィンドウ内に表示できるメッセージのサイズには制限があるため、次の項で説明する「改ページ」を適切に入れる必要があります。

改ページ

メッセージ末尾に「/」を指定することで、改ページすることができます。そして改ページ記号の位置で、キーの入力待ちをします。

1. "春はあけぼの。"
2. "やうやうしろくなりゆく山ぎは、すこしあかりて、"
3. "紫だちたる雲のほそくたなびきたる。/"
4. "夏は夜。月のころはさらなり、"
5. "闇もなほ、蛍のおほく飛びちがひたる。/"
6. "また、ただ一つ二つなど、"
7. "ほのかにうち光りて行くもをかし。/"
8. "雨など降るもをかし。/"
- 9.
10. // ■上記記述は、以下のように表示します
11. // "春はあけぼの。"
12. // "やうやうしろくなりゆく山ぎは、すこしあかりて、"
13. // "紫だちたる雲のほそくたなびきたる。" // ←ここで改ページ待ち
- 14.
15. // "夏は夜。月のころはさらなり、"
16. // "闇もなほ、蛍のおほく飛びちがひたる。" // ←ここで改ページ待ち
- 17.
18. // "また、ただ一つ二つなど、"
19. // "ほのかにうち光りて行くもをかし。" // ←ここで改ページ待ち
- 20.
21. // "雨など降るもをかし。" // ←ここで改ページ待ち

クリック待ち

改ページを行わずにプレイヤーからの入力待ちをする場合には、メッセージ末尾に「@」を指定します。

1. "春はあけぼの。@" // ←ここでクリック待ちをします
2. "やうやうしろくなりゆく山ぎは、すこしあかりて、"
3. "紫だちたる雲のほそくたなびきたる。"

改行のエスケープ

「\n」によるメッセージ改行を回避するには、メッセージ末尾に「_」を指定します

1. "春はあけぼの。_"
2. "やうやうしろくなりゆく山ぎは、すこしあかりて、"
3. "紫だちたる雲のほそくたなびきたる。"
- 4.
5. // ■上記記述は、以下のように表示します
6. // "春はあけぼの。やうやうしろくなりゆく山ぎは、すこしあかりて、" // ←1行目を改行せずに2行目をつなげている
7. // "紫だちたる雲のほそくたなびきたる。"

表示色

メッセージ開始の「"」の直後に「#」を指定することで、文字列の色を指定することができます。書式は#rrggbbで指定します。値は先頭からR成分、G成分、B成分を16進数で指定します。

1. "#ff0000このメッセージは赤色で表示されます。" // 赤色でメッセージを表示します

表示ディレイ

メッセージ開始の「"」直後に「!d数字(ミリ秒)」を指定することで、1文字ごとのメッセージ表示速度にディレイをかけることができます。

1. "!d1000このメッセージは、ゆっくり、ゆっくり、表示されます" // 1秒ごとに1文字ずつ表示する

表示ウェイト

特定の位置だけウェイト(待ち)を入れたい場合には、「!w数値(ミリ秒)」を使用します

1. "3秒待ちます。!w3000表示を再開します" // "3秒待ちます"、の表示3秒後に次にメッセージを表示します

変数の表示

「\$数字」により変数を表示することができます

1. \$100 = 250
2. "変数の中身を表示します。\$100は250です"
- 3.
4. // 上記記述は、以下のように表示されます
5. // "変数の中身を表示します。250は250です"

※変数の詳細については、変数の項目を参照

制御構造

if文

```
1.  if(式)
2.  {
3.      文
4.  }
```

「式」が真である場合「文」を実行します（式については論理式の項目を参照）

else文

```
1.  if(式)
2.  {
3.      文1
4.  }
5.  else
6.  {
7.      文2
8.  }
```

「式」が真である場合「文1」を実行します。そして「式」が偽である場合「文2」を実行します。

elif文

```
1.  if(式1)
2.  {
3.      文1
4.  }
5.  elif(式2)
6.  {
7.      文2
8.  }
```

「式1」が真の場合、「文1」を実行します。この場合、「式2」は評価せず、「文2」が実行されることはありません。また「式1」が偽の場合、「式2」を評価します。そして「式2」が真であれば「文2」を実行します。「式2」が偽であれば、何も実行しません

while文

ループを実現するために、while文を使うことができます。

```
1.  $1 = 0
2.  while($1 < 5)
3.  {
4.      $1 += 1
5.  }
```

上記コードは変数\$1が5になるまでループを続ける例です。また「break」「continue」による制御も可能です。

```
1. $1 = 0
2. while(true)
3. {
4.     $1 += 1
5.     if($1 < 5) {
6.         continue
7.     }
8.     else {
9.         break
10.    }
11. }
```

while-else文を使うこともできます。

```
1. $1 = 0
2. while($1 < 5) {
3.     $1 += 1
4. }
5. else {
6. }
```

このように記述すると、while文のブロックを抜けた際、else文のブロックを実行します。

```
1. $1 = 0
2. while(true) {
3.     $1 += 1
4.     if($1 < 5) {
5.         break
6.     }
7. }
8. else {
9. }
```

breakで抜けると、else文のブロックは実行されません。

四則演算

四則演算として以下のものが使用できます。

演算子	演算方法
+	加算
-	減算
*	乗算
/	除算

```
1. $100 = 50 + 200 // 変数$100に、「50+200」の計算結果を代入します
```

※変数・代入については、変数の項目を参照してください

演算の優先順位

'*'と'/'は、'+'と'-'よりも優先して演算を行います

```
1. $100 = 2 + 3 * 4 // この計算結果は、20ではなく、14となります
```

括弧による優先順位

'('と')'を使用することで、演算を優先して行うことができます

```
1. $100 = (2 + 3) * 4 // この計算結果は、20となります
```

変数

「\$」 + 「数値」による値を、変数（変更可能な値）として使用することができます。

有効な値

有効な変数の範囲と値は以下のようになります。（※浮動小数や文字列を使用することはできません）

有効な変数の範囲	\$0～\$999
有効値	-2147483648 ～ +2147483647(4byte整数値)

変数への代入

「=」により変数へ値を代入することができます

```
1. $100 = 20 // 変数$100に20を代入する
```

論理式

if文などの制御を行う場合、制御パラメータとして論理式を与える必要があります。（※if文については、「制御構造」の項目を参照）

論理演算子一覧

論理式のための演算子としては、以下のものが使用できます

演算子	真となる条件	例
==	左辺と右辺が同一である	100 == 100 → 真
!=	左辺と右辺が同一でない	100 != 100 → 偽
<	左辺が右辺より小さい	50 < 100 → 真
<=	左辺が右辺以下	100 <= 100 → 真
>	左辺が右辺よりも大きい	100 > 50 → 真
>=	左辺が右辺以上	100 >= 100 → 真
!	論理演算結果が偽（演算結果を反転する）	!(100 == 100) → 偽
&&	左辺が真かつ右辺が真	(100 == 100) && (200 == 200) → 真
	左辺が真または右辺が真	(100 == 100) (200 == 100) → 真

ラベル

スクリプトの流れに分岐を作るために、ラベルの定義とラベルジャンプ(goto)を使用することができます。

ラベルの定義

行頭に「名前」 + 「:」（コロン）を指定するとラベルとなります。

```
1. scene001: // 「scene001」というラベルを定義
2. draw_bg(0)
3. draw_ch(100, c)
```

このラベル定義により、後述する「goto」命令を使用して実行位置のジャンプを行うことができます

ラベルジャンプ

goto命令により、ラベルジャンプを行うことができます。

```
1. $1 = 0
2. Label_Loop:
3. $1 += 1
4. if($1 < 5)
5. {
6.     goto Label_Loop
7. }
```

上記コードは、変数"\$1"が5を超えるまで、Label_Loopラベルにジャンプし続けます。なお、存在しないラベル名を指定した場合、コンバート時に未定義エラーとなります。

ファンクション

ファンクションの定義

「def」で開始するブロックはファンクションとなります。

```
1. def scene002
2. {
3.     // ファンクションブロック
4. }
```

ファンクションブロックは、call命令以外では実行されることはありません。

ファンクションコール

call命令により、ファンクションコールをすることが可能です。

```
1. call scene002
```

上記コードは、scene002ラベルをコールします。存在しないファンクション名を指定した場合、コンバート時にエラーとなります。なお、呼び出したラベルのブロックを超えた場合には、call命令により呼び出した位置に戻ります。

return命令

call命令により呼びだされたラベル内で「return」命令を行うと、その時点で呼び出し元に戻ります

```
1. def scene002
2. {
3.   if($1 == 5)
4.   {
5.     // $1 が 5 だったら処理を行わない
6.     return;
7.   }
8.   "$1は5ではありません\"
9. }
```

スクリプトジャンプ

「load」命令を使用すると、実行中のスクリプトを変更することができます。

```
1. load("scene2") // 現在のスクリプトを終了して、"scene2"を開始します
```

goto / call 命令との違いは以下のようになります

1. 対象スクリプトを#includeする必要はない
2. 呼び出し元に戻ることはできない

選択肢

選択肢により分岐を行う場合は、「select」命令を使用します。

```
1. select("問題文")
2. {"選択肢1"
3.   // 選択肢1を選んだ場合、このブロックの処理を行う。
4. }
5. {"選択肢2"
6.   // 選択肢2を選んだ場合、このブロックの処理を行う。
7. }
8. {"選択肢3"
9.   // 選択肢3を選んだ場合、このブロックの処理を行う。
10. }
```

例えば、選択肢1を選んだ場合、選択肢1のブロックを実行を実行します。そして選択肢1のブロックの末尾に達すると、この全ての選択肢ブロックの終端にジャンプします。（選択肢2／選択肢3のブロックは実行されません）

問題文の複数行表示

```
1. select("問題文1行目"
2.   "問題文2行目"
3.   "問題文3行目"
4. )
5. {"選択肢1"
6. }
7. {"選択肢2"
8. }
```

という記述をすることで問題文を複数行表示することができます。
もしくは、


```

1. select("問題文1行目" "問題文2行目" "問題文3行目")
2. {"選択肢1"
3. }
4. {"選択肢2"
5. {
6. }
```

というようにスペース区切りで1行にまとめて定義することが可能です。

選択肢に対する条件式の設定

```

1. select("問題文" "問題文2行目" "問題文3行目")
2. {($1==0);"A" // $1 が 0ならばこの選択肢を表示する
3.   "Aを選択しました\"
4. }
5. {($2);"B" // $2 が 0でなければこの選択肢を表示する
6.   "Bを選択しました\"
7. }
8. {($3>=3);"C" // $3 が 3以上であればこの選択肢を表示する
9.   "Cを選択しました\"
10. }
```

選択肢の前に (式) + 「;(セミコロン)」 を定義することで、その選択肢を表示するかどうかの判定が可能となります。

コメント

スクリプトの補足説明をするために、変数やラベルの開始部分などにコメントを入れることができます。コメントを開始する部分に「//」を入れると、以降の文字列は行末までコメントとみなし、実行しません

```

1. // 学校の背景を描画します。 ←コメントとして扱い、実行しない
2. draw_bg(30)
```

また「/*」から「*/」までの間はコメントブロックとなります。

```

1. /*
2.   この間はコメントとなり
3.   表示されません
4. */
```

乱数

乱数を使うと、ランダムな値を取得できるため、それを使ってランダムにイベントを発生させたりすることができます。書式は以下のようになります。

```

rand(開始値, 終了値)
開始値～終了値の範囲で、乱数を発生します
```

```
1. $1 = rand(0, 99) // 0~99の値をランダムで返します
2. if($1 < 30)
3. {
4.     // 30%の確率でこちらのブロックが実行されます
5. }
6. else
7. {
8.     // 70%の確率でこちらのブロックが実行されます
9. }
```

描画命令

描画命令一覧

命令	説明	構文	引数
draw_bg	背景の描画	draw_bg(id, effectId)	<ul style="list-style-type: none">・ id : 画像ID or 色(#rrggbb 単色による塗りつぶし)・ effectId : エフェクトID (省略時は EF_NORMAL)
erase_bg	背景の消去	erase_bg(effectId)	<ul style="list-style-type: none">・ effectId : エフェクトID (省略時は EF_NORMAL)
draw_ch	キャラの描画	draw_ch(id, pos, effectId)	<ul style="list-style-type: none">・ id : 画像ID or 色(#rrggbb 単色による塗りつぶし)・ pos : 立ち位置 ("c:"中心 "l":左 "r":右)・ effectId : エフェクトID (省略時は EF_NORMAL)
draw_ch_pos	キャラの描画 (座標指定・左上)	draw_ch_pos(id, posX, posY, effectId)	<ul style="list-style-type: none">・ id : 画像ID or 色(#rrggbb 単色による塗りつぶし)・ posX : スクリーン座標(X)・ posY : スクリーン座標(Y)・ effectId : エフェクトID (省略時は EF_NORMAL)
erase_ch	キャラの消去	erase_ch(pos, effectId)	<ul style="list-style-type: none">・ pos : 立ち位置 ("c:"中心 "l":左 "r":右)・ effectId : エフェクトID (省略時は EF_NORMAL)

エフェクトID一覧

描画命令 (draw_bg / draw_chなど) に指定可能なエフェクトIDの一覧は以下のものとなります。

ID	説明
EF_NORMAL	瞬間表示
EF_SCROLL_L	左スクロール
EF_SCROLL_R	右スクロール
EF_SCROLL_U	上スクロール
EF_SCROLL_D	下スクロール

EF_SHUTTER_L	左シャッター
EF_SHUTTER_R	右シャッター
EF_SHUTTER_U	上シャッター
EF_SHUTTER_D	下シャッター
EF_ALPHA	半透明
EF_GRAY	グレースケール
EF_SEPIA	セピア
EF_NEGA	反転

画面演出

命令	説明	構文	引数
shake	画面を揺らす	shake(ms, amp)	<ul style="list-style-type: none">ms : 揺れ時間 (ミリ秒)amp (開始揺れ幅)

サウンド命令



サウンド命令一覧

命令	説明	構文	引数
play_bgm	BGMの再生	play_bgm(id, loop, fade)	<ul style="list-style-type: none">id : サウンドIDloop : ループ回数。省略時は0で無限ループ再生fade : フェードイン時間 (ミリ秒)。省略時は0でフェードなしで再生
stop_bgm	BGMの停止	stop_bgm(fade)	<ul style="list-style-type: none">fade : フェードアウト時間 (ミリ秒)。省略時は0でフェードなしで停止
play_se	SEの再生	play_se(id, loop)	<ul style="list-style-type: none">id : サウンドIDloop : ループ回数。省略時は1で1回のみ再生する。0で無限ループ再生
stop_se	SEの停止	stop_se(id)	<ul style="list-style-type: none">id : サウンドID

セーブ命令

save 命令を使用すると以下の情報が保存されます。

- 実行中のスクリプト名
- プログラム・カウンタ
- 描画している背景画像番号
- 描画しているキャラ画像番号
- 再生しているBGM番号
- 再生しているSE (ループのみ)
- 変数の値
- 実行スタック
- コールスタック
- 表示中のメッセージ
- バックログ

[ページ情報] 更新日時: 2014-02-23 08:23:31, 更新者:  syun77
[ライセンス]  クリエイティブ・コモンズ 2.1 表示
[権限] 表示:無制限, 編集:管理者, 削除/設定:管理者
[IRI] <https://ja.osdn.net/projects/adv/wiki/スクリプト文法>