

マイコン部 2024

講習会資料

第1回

目次

第 0 章 HSP の準備

- 0-0 はじめに
- 0-1 HSP とは
- 0-2 HSP の使い方
 - 0-2.1 スクリプトエディタ
 - 0-2.2 HSP アシスタント
 - 0-2.3 HSP ドキュメントライブラリ

第 1 章 基本の命令

- 1-0 HSP に命令を出す
- 1-1 screen 命令
- 1-2 メッセージ関連
 - 1-2.1 mes 命令
 - 1-2.2 font color
 - 1-2.3 pos
- 1-3 変数
 - 1-3.1 変数
 - 1-3.2 型変換
 - 1-3.3 配列
- 1-4 メディアの利用
 - 1-4.1 画像
 - 1-4.2 音楽
- 1-5 繰り返し
 - 1-5.1 repeat loop
 - 1-5.2 cnt

目次

1-5.3 wait await

1-5.4 break

第 2 章 基本の命令 ver2

2-1 条件分岐

2-1.1 if else

2-1.2 end

2-2 サブルーチン

2-2.1 ラベル

2-2.2 goto stop

2-2.3 gosub return

2-3 乱数

2-4 オブジェクト

2-4.1 HSP のオブジェクト

2-4.2 オブジェクト管理命令

2-5 キー・マウスの読み取り

2-5.1 getkey

2-5.2 stick

第 3 章 プログラム作成

3-1 画面の初期化

3-1.1 boxf

3-1.2 cls

3-1.3 redraw

3-2 時計の作成

3-2.1 プログラムの作成

3-2.2 時計の装飾

目次

- 3-3 ボールを動かす
 - 3-3.1 ボールの移動
 - 3-3.2 ボールが跳ね返るようにする
 - 3-3.3 操作できるボールの追加
 - 3-3.4 ボールの数を増やす
- 3-4 じゃんけんゲーム
 - 3-4.1 素材の用意
 - 3-4.2 プログラムの作成
 - 3-4.2 勝敗の表示
- 3-5 プログラム紹介
 - 3-5.1 合成音声プログラム
 - 3-5.2 ブロック崩し
 - 3-5.3 タイピングゲーム
 - 3-5.4 RPG 風ゲーム

第4章 発展内容

- 4-1 オブジェクト・ウィンドウ ID
- 4-2 関数
- 4-3 ファイル操作
- 4-4 割り込みラベルジャンプ
 - 4-4.1 割り込み関数
 - 4-4.2 キー・マウスの読み取り発展版
- 4-5 プリプロセッサ
 - 4-5.1 プリプロセッサ命令
 - 4-5.2 HSP のプラグイン
 - 4-5.3 WindowsAPI

第0章 HSP の準備

0-0 はじめに

- ☆ この資料は、HSP3.x 系のバージョンを想定して作成しています。
- ☆ この資料では HSP を使用するうえで一通り必要になる命令や躰きやすい箇所へのアドバイス、コラム、ゲームを作成するうえでのポイント、ある程度の経験者向けの発展内容を取り扱っています。
- ☆ 説明量の関係上端折っている関数や、深く解説していない命令などがあります。より深く知りたい場合などはドキュメントライブラリ、ネット検索等を活用して調べてみてください。

0-1 HSP とは

HSP (Hot Soup Processor) は手続き型/インタプリタ式のプログラミング言語です。したがって、プログラムを実行すると **1 行ずつ順々に実行**されていきます。また、プラグイン (外部ツール) を使うことで、3D グラフィックスなど高度な機能を利用することができます。

プログラミング言語の特徴として、HSP では、半角英数の大文字・小文字は区別しません。また、変数に 2 バイト文字 (日本語) を使うことができます。

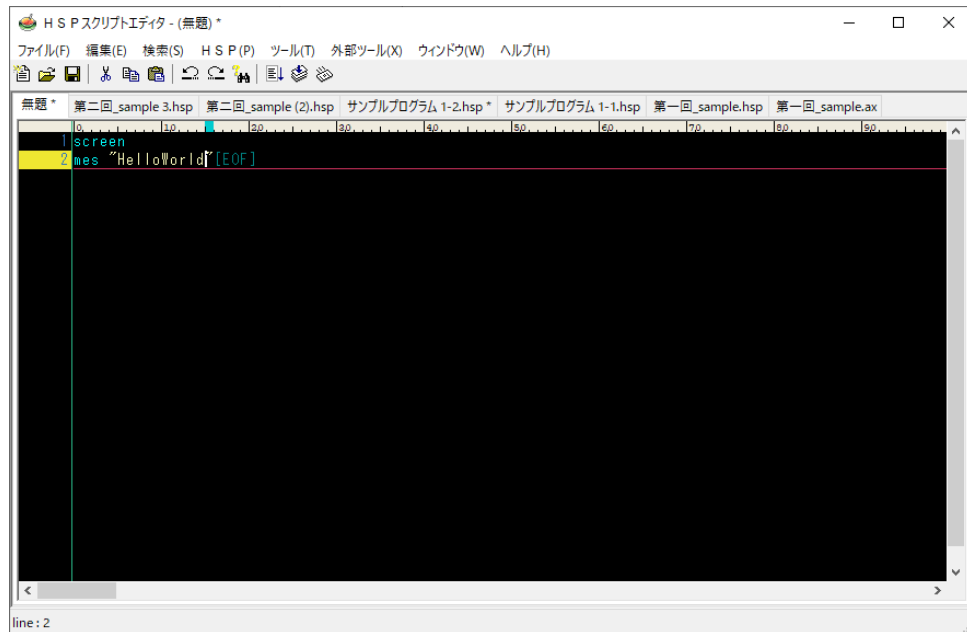
0-2 HSP の使い方

本節では、実際の HSP の機能やその使い方を説明します。

0-1.2 スクリプトエディタ

スクリプトエディタは、HSP を起動したときに出てくる黒い画面のことです。(図 0-1)ここにプログラムを記述して実行していきます。また、この画面からプログラムの実行、ドキュメントライブラリの実行、EXE ファイルの作成なども行えます。いわば HSP で最もよく使う、メインのウィンドウになります。

第0章 HSP の準備



(図 0-1) ↑スクリプトエディタのウィンドウ

スクリプトエディタの基本的な機能は以下の通りです。

- ✧ ファイル(F)より、HSP ファイルの新規作成、読み込み、保存
- ✧ 検索(S)より、文字列の検索
- ✧ HSP(P)より、プログラムの実行、実行ファイル作成
- ✧ ツール (T) より、様々な補助ツールの起動
- ✧ ウィンドウ (W) より、複数プログラムを開いている場合の管理
- ✧ ヘルプ(H)より、アシスタントやライブラリの起動

また、便利なショートカットキーとして、

- ✧ F5 キーで現在作成しているプログラムをすぐに実行
- ✧ Ctrl+F キーで文字列の検索

などを行うことができます。（ほかにも様々なショートカットキーがあります）

コラム：実行ファイル（EXE ファイル）の作り方

HSP で作成したプログラムを実行ファイル（exe ファイル）にしたい場合、以下の方法で作成することができます。

方法その1

1. F5 キーでプログラムをコンパイルして実行します。
2. エラーがないことを確認します。
3. F9 キーor メニューの HSP→オブジェクトファイル作成より、オブジェクトファイルを作成します。
4. メニューの HSP→START.AX ファイル作成より START.AX を作成します。
5. メニューの「ツール」→「PACKFILE 編集」を選択します。

PACKFILE 編集では作成する EXE ファイルに組み込むファイルを編集します。前ステップで作った START.AX はプログラムファイルとなるので、必ず入れましょう。そのほかにもプログラムに必要な画像、音声ファイル、一部外部ファイル（入れられないものあり）も一緒に EXE ファイルに組み込むことができます。すべて選択し終わったら閉じるを押してください。

6. メニューの「ツール」→「EXE ファイル作成」を選択します。

方法その2

メニューの HSP より、実行ファイル自動作成を選択します。大体的場合はこの方法で実行ファイルを作成することができます。

方法その2の方が簡単ですが、細かい設定ができないため、一度方法2を試してみてもエラーが出たりするようなら方法その1を試してみるとよいかもしれません。

第0章 HSP の準備

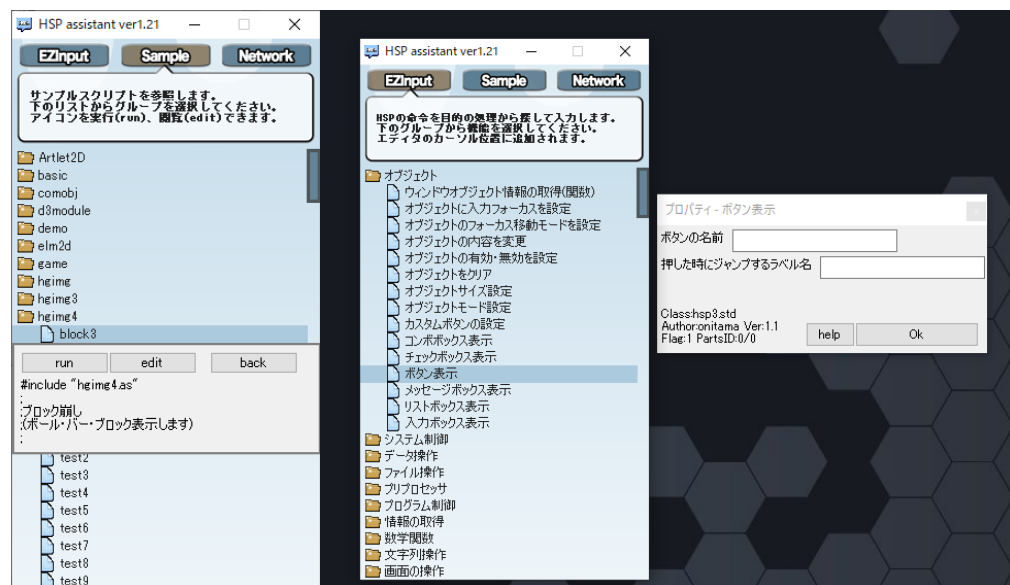
コラム：デバックウィンドウ

メニューの HSP より、「Debug ウィンドウ表示」にチェックを入れてから F5 キー時で実行を行うことで、デバックウィンドウありでプログラムが実行されます。

デバックウィンドウには、どの変数にどのような値が代入されているかや、エラーのログ、プログラムに関する全般的な情報などが表示されます。これを活用することで、プログラムのエラーが出た際の原因の特定が行いやすくなります。

0-2.2 HSP アシスタント

HSP を起動した際に、スクリプトエディタと同時に起動されるウィンドウが HSP アシスタントです。HSP アシスタントのサンプルタブではさまざまなサンプルプログラムを見ることができます。また EZInput ではプログラムの関数を簡単に入力することができる機能も備わっています。



(図 0-2,0-3) ↑ サンプルプログラムの実行と EZInput の利用画面

Sample では ran を押すことによってプログラムの実行を、edit を押すことによって HSP スクリプトエディタにサンプルプログラムを表示することができます。

EZInput では必要な引数を入力することによって、テキストエディターに該当する関数が入力された状態の命令を挿入することができます。

第0章 HSP の準備

0-2.3 HSP ドキュメントライブラリ

HSP ドキュメントライブラリは HSP スクリプトエディタのヘルプタブより起動、もしくは命令の部分にカーソルを合わせて F1 キーを押すと起動できるツールで、命令の検索を行うことができます。左上の検索ボックスに検索した命令を入力すると、その命令の詳細な仕様や利用例を見ることができます。

HSP アシスタントや他の人のプログラムを見ていて知らない命令が出てきたときや、使っている命令の細かい使い方を知りたい時などは活用してみましょう。



(図 0-4) ドキュメントライブラリ

第 1 章 基本の命令

1-0 HSP に命令を入力する

以下のスクリプトを見てみましょう。

```
screen 0,640,480  
  
mes "HelloWorld"  
  
stop
```

1 行目がウィンドウの表示、2 行目が“HelloWorld”という文字列の表示、3 行目がプログラムの停止という内容がなんとなく想像できるでしょうか。HSP では、このような命令を記述していき、それが 1 番目から順番に処理されるようになっています。（インタプリタ型言語）

screen や mes のような作業を表す単語のことを**命令**または**ステートメント**と呼びます。また、命令と同時に使う 0,640,480 や HelloWorld の部分を**パラメータ**と呼びます。1 行目の screen 命令のように、命令によってはパラメータを複数設定でき、カンマ（,）で区切ります。これらのパラメータは 1 番目から順に第 1 パラメータ、第 2 パラメータ…と呼ばれます。3 行目の stop も命令の 1 つで、スクリプトの実行を停止する指示を出すことができます。この命令のように、**パラメータがない命令も存在**します。

HSP 言語で命令を記述する際、以下の点に気を付けてください。今回に関してはエラーは起きませんが、命令によってはエラーが起こる可能性があります。

- ★ 命令とパラメータの間には 1 つ以上のスペースを空ける
- ★ 命令の途中にスペースを入れない



(図 1-1) ↑ 命令の書き方

第1章 基本の命令

1-1 screen 命令

まずは、最も基本の命令となる screen 命令を紹介します。screen 命令は文字や画像を表示させるためのウィンドウを作る命令です。

screen 命令の仕様は以下のようになっています。

```
screen p1,p2,p3,p4,p5,p6,p7,p8
```

p1=0～(0) : ウィンドウ ID

p2=0～(640): 初期化する画面サイズ X (1 ドット単位)

p3=0～(480): 初期化する画面サイズ Y (1 ドット単位)

p4=0～1(0) : 初期化する画面モード

p5=0～(-1) : ウィンドウの配置 X (1 ドット単位)

p6=0～(-1) : ウィンドウの配置 Y (1 ドット単位)

p7=0～ : ウィンドウのサイズ X (1 ドット単位)

p8=0～ : ウィンドウのサイズ Y (1 ドット単位)

p はパラメータの略で、ここに各種設定の値が入ります。

☆ p1 は、複数のウィンドウを表示する際に、その番号を割り振るためのパラメータです。今回の講習会では基本 0 で大丈夫です。

☆ p2, p3 では、それぞれウィンドウの X (横方向) サイズと Y (縦方向) のサイズを指定します。

☆ p4 では、ウィンドウのサイズを固定したり、見えないウィンドウを作成したりできます。

実は、screen 命令は HSP の実行に必須ではなく、省略することができます。実際に、screen 命令を記述せず、mes 命令のみを記述すると、新しいウィンドウに mes 命令で指定した内容が表示されます。省略された場合は以下の初期値が自動で設定されます。

```
screen 0, 640, 480, 0, -1, -1, 640, 480
```

第1章 基本の命令

コラム：パラメータの省略

HSP の命令の中にはパラメータを省略することができるものがあります。例えば、先ほどの screen 命令では、

```
screen , 640, 480
```

と書くこともできます。こうすると、何も書かれていない第 1 パラメータには自動で初期値の 0 が入り、記載されていない第 4 パラメータ以降も初期値が補完されます。しかし、中には mes 命令や mmload 命令などパラメータを省略できないものも存在します。

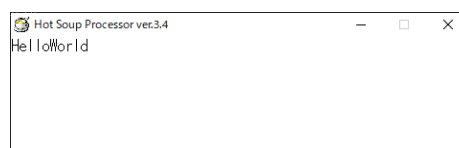
1-2 メッセージ関連

次に、メッセージ関連の命令を見ていきましょう。これらの命令を使用することにより、先ほど設定したウィンドウに文字を表示したり、文字のフォントや色、表示位置を変更したりすることができます。

1-2.1 mes 命令

mes 命令を使用することで、ウィンドウにメッセージを表示することができます。パラメータは、入力したい文字列(日本語でも OK)をダブルクォーテーション (") で囲うことで設定します。

```
mes "入力したい文字列"
```



(図 1-2) ↑実際に mes 命令を用いたプログラムの実行結果

1-2.2 font/color 命令

Font 命令は mes 命令で表示させる文字のフォントを変更する際に使用します。また、color 命令も同じように色を変更する際に使用します。

第1章 基本の命令

Font や color, pos などの命令は、指定した次の行からその設定が適用され、もう一度 font, color, pos などの命令で指定しなおさない限りずっと適用され続けます。したがって元のフォントや大きさ、位置に戻したいときは、再び指定する必要があります。

font "fontname", p1, p2

"fontname" : フォント名

p1=1~99(12) : フォントの大きさ

p2=0~(0) : フォントのスタイル

color p1, p2, p3

p1,p2,p3=0~255(0) : 色コード (R,G,B)

font 命令では最初にフォント名 (MS 明朝、MS ゴシックなど)、次にフォントのサイズ (デフォルトは 12)、最後にフォントのスタイルを設定します。スタイルの値によって以下のように変化します。

- ◇ スタイル 1 : 太文字
- ◇ スタイル 2 : イタリック体
- ◇ スタイル 4 : 下線
- ◇ スタイル 8 : 打ち消し線
- ◇ スタイル 16 : アンチエイリアス

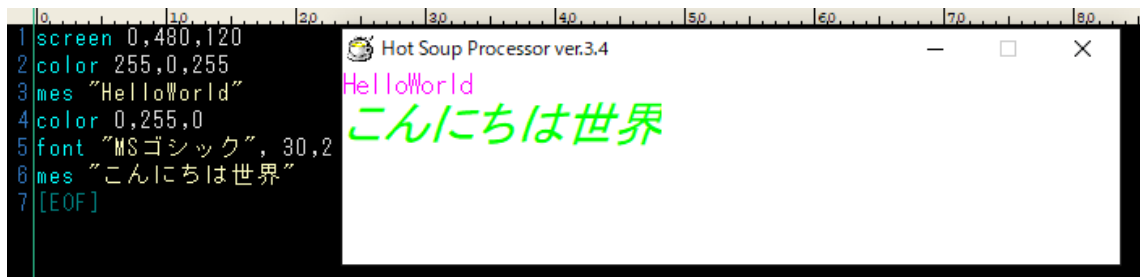
color 命令ではそれぞれ R(Red),G(Green),B(Blue)の値を 256 段階(0~255)で設定することによって、色を表現します。

代表的な色

黒■ : 0, 0, 0 白 : 255, 255, 255 赤■ : 255, 0, 0 緑■ : 0, 255, 0 青■ : 0, 0, 255 水色■ : 0, 255, 255 黄色■ : 255, 255, 0 ピンク■ : 255, 0, 255

ちなみに、color 命令は文字の色だけではなく、後ほど解説するウィンドウ全体の色などの変更にも使用されます。

第1章 基本の命令



(図 1-3)font, color 命令を利用したプログラムと実行結果

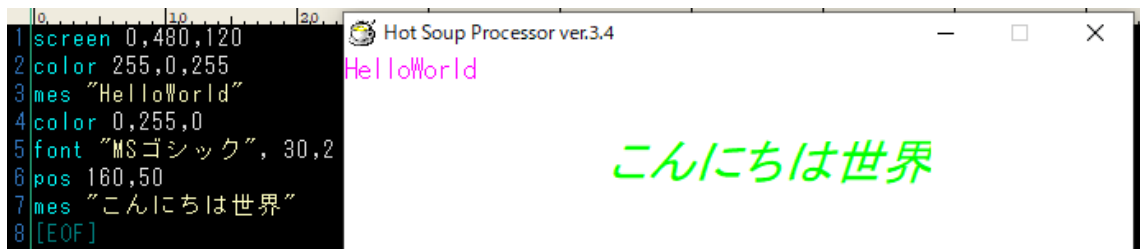
1-2.3 pos 命令

Pos 命令では、文字やオブジェクトなどをウィンドウに表示する位置を変更することができます。デフォルトでは左上から順番に表示されますが、pos 命令を利用することで、ウィンドウの好きな位置にコンテンツを表示させることができます。

pos p1,p2

p1=0～ : カレントポジション（表示させる位置）の X（横）座標

p2=0～ : カレントポジション（表示させる位置）の Y（縦）座標



(図 1-4)pos 命令を利用したプログラムと実行結果

コラム：コメントアウト

HSP を含む多くのプログラム言語では、プログラムの途中にコメントを残すことができます。コメントを残る利点は、あとからプログラムを見返したときになぜこのようなプログラムにしたのかがわかりやすくなるからです。大規模なプログラムを作る際はできるだけコメントを付けて、あとから見返してもわかりやすいプログラムを作ることを心がけましょう。

コメントアウトの方法

その 1


プログラムの途中に `//` を付ける。

そうすると、その行のそれ以降の文字はコメントとなりプログラムの実行には反映されなくなります。

その 2

コメントを `/* */` で囲む。

こうすると、囲まれたコメントはプログラムの実行には反映されなくなります。その 1 と違うのは、複数行のコメントを付けることができるという点です



```
1 screen 0,480,120
2 color 255,0,255
3 mes "HelloWorld" //このように
4 color 0,255,0 /*コメントアウトをすることができます*/
5 font "MSゴシック", 30,2
6 pos 160,50/* この方法では
7              複数行のコメントを
8              することができます */
9 mes "こんにちは世界"
10 //これらのコメントが
11 //プログラムに反映されることは
12 //ありません
13 [EOF]
```

Hot Soup Processor ver.3.4

HelloWorld

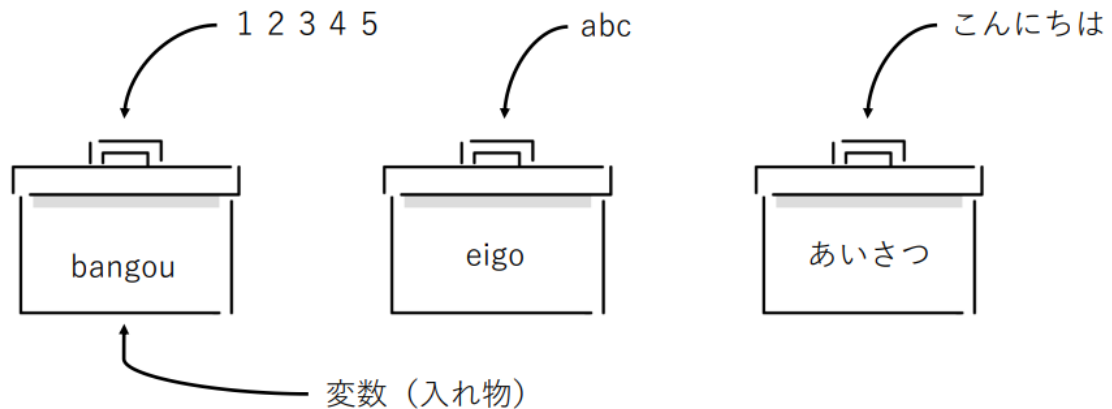
こんにちは世界

(図 1-5) ↑ コメントアウトを利用したプログラムの作成

第1章 基本の命令

1-3 変数

プログラミング言語には**変数**という**文字や数値を入れておくための箱**があり、当然 HSP にも変数の概念があります。変数には名前を決めておくことができ、あとからその名前を使って変数の中身を呼び出したり、利用したりできます。



(図 2-1)変数の説明 それぞれの箱には名前がついており、その中に数字や文字が入っている

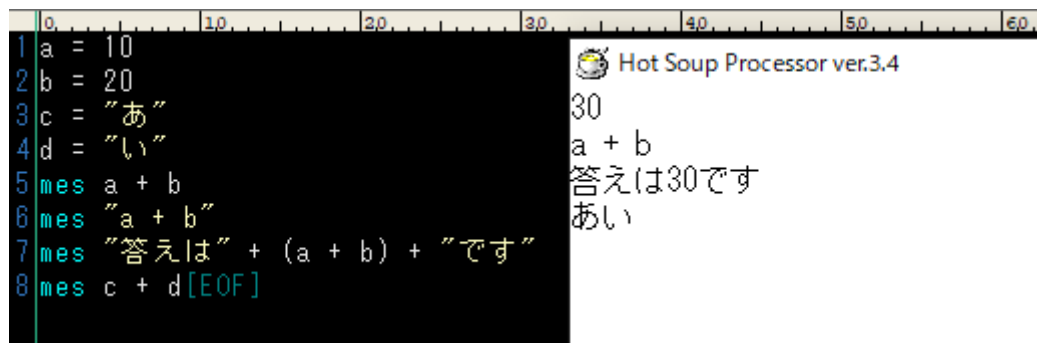
1-3.1 変数

HSP では、基本的には**変数の宣言や変数の初期化をする必要はありません**。パラメータに使いたい名前の変数を入力すれば、HSP が勝手に変数の宣言を行ってくれます。ただし、変数をプログラムの初期値として扱いたい場合など、変数の初期化を行う場合もあります。

```
0. 10. 20. 30. 40. 50. 60. 70.
1 screen 0,640,480
2 Input Naiyo //入力ボックスに入った内容を"Naiyo"という名前の変数に入れる[EOF]
```

(図 1-6)↑プログラム内で、いきなり変数を使ってもよい

また、変数の内容同士で加減乗除算を行ったりすることもできます。



```
1 a = 10
2 b = 20
3 c = "あ"
4 d = "い"
5 mes a + b
6 mes "a + b"
7 mes "答えは" + (a + b) + "です"
8 mes c + d[EOF]
```

Hot Soup Processor ver.3.4

30
a + b
答えは30です
あい

(図 1-7)変数の計算

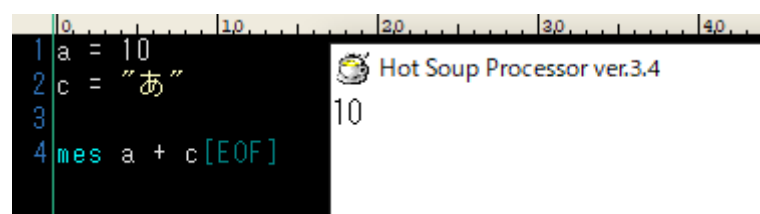
例えば、図 1-7 では、1～4 行目で数字の 10 を a という名前の変数に代入し、20 を変数 b に代入、ひらがなの"あ"を c に代入、"い"を d に代入しています。

そして、5 行目で a + b を行っているので変数の中身を利用して、 $10 + 20 = 30$ となります。なお、変数を mes で表示する際はダブルクォーテーションを付けずに変数名を表記します。（※ダブルクォーテーションを付けると、変数ではなく、文字扱いになります。）したがって 6 行目はそのまま "a + b" と出力されることになります。

さらに、変数と文字を組み合わせることもできます。7 行目のようにすることで、変数の計算と文字の表示を同時に行うことができます。

1-3.2 型変換

まずは以下の画像を見てください。どこがおかしいところはありませんか。



```
1 a = 10
2 c = "あ"
3
4 mes a + c[EOF]
```

Hot Soup Processor ver.3.4

10

(図 1-8)どこがおかしい実行結果の画像

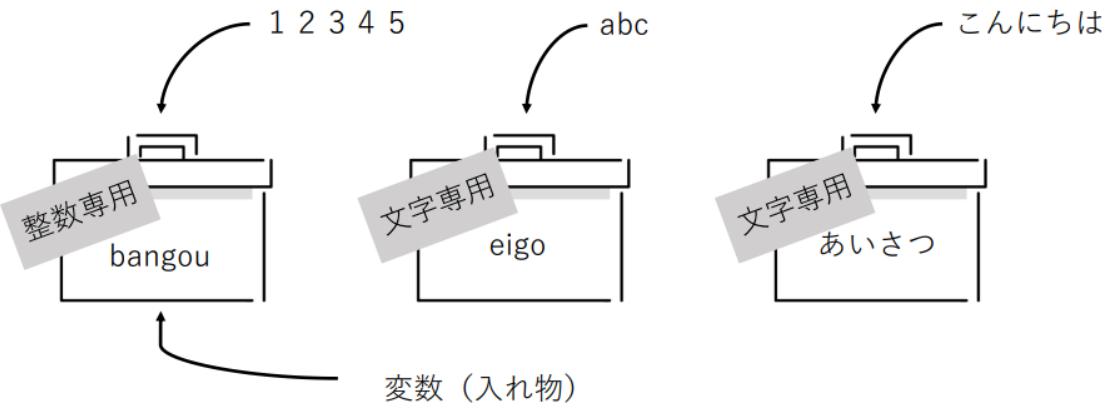
そうです、変数 a の中に数字の 10 が、変数 c の中にひらがなの"あ"が入っているのに、mes 命令で足してみるとひらがなの"あ"がなくなっています。なぜこのようなことが起きてしまうのでしょうか。

第1章 基本の命令

実はこの現象、お互いの変数の型が違うことが原因で起きてしまうのです。

H S Pには変数の概念がありますが、実は、この**変数にはいくつかの種類があります**。整数のみを入れることができる **int 型**、実数（小数点以下）も扱うことができる **double 型**、文字を扱うことができる **str 型**などがH S Pにはあります。変数の型は、私たちが変数をプログラム内に書いて実行したときに、H S P側が勝手に決めてその中に文字や数字を入れてくれます。しかし、時にはH S Pの指定してくれた変数の型では都合が悪い場合もあります。そのようなときには私たちがHSPの指定した型を別の型に変換することができます。このようにプログラムが勝手に型を決めてくれる言語のことを**動的型付け言語**といい、一方で自分たちが使う型を決める言語を**静的型付け言語**といいます。

動的型付け言語はプログラムが型を決めてくれるため便利ですが、プログラムを書き間違えているときなどもエラーが出ずにそのまま実行できてしまい、間違えている箇所がわかりづらくなったりするので注意が必要です。



(図 EX-1)変数の型のイメージ

(表 1－1)HSP の型

名称	代入できる形式	例
Int 型	整数	1,2,3...
Double 型	実数	1.000000,2.00000...
Str 型	文字列	A,b,c,1,2,あ,亜...

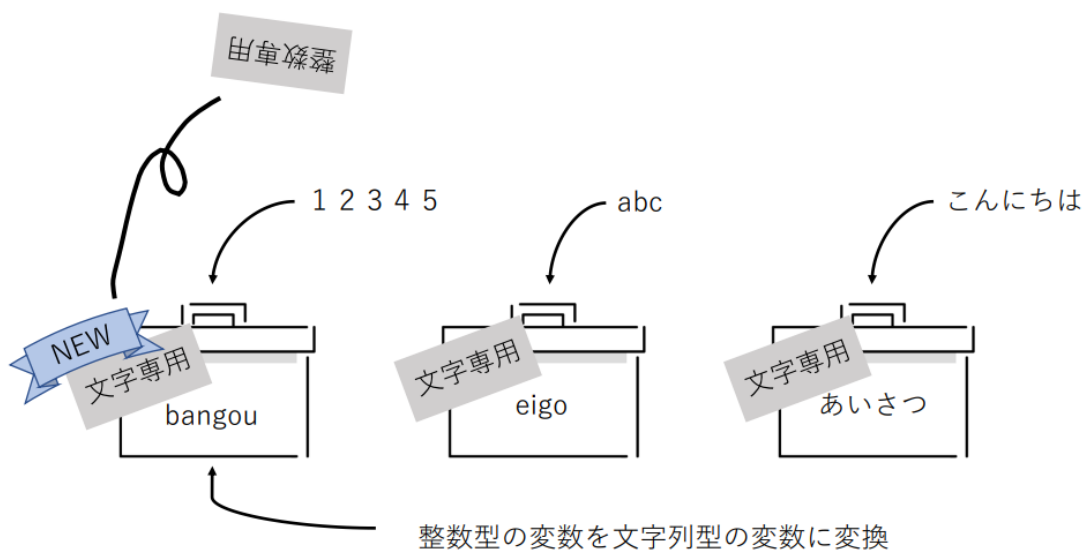
第1章 基本の命令

```
1 a = 10
2 b = 20
3 c = "あ"
4 d = "い"
5
6 e = str(a) //str命令は、パラメータ内の変数の中身を文字列型に変換します
7 f = str(b)
8 mes e + f
9
10 g = 10.5
11 h = 20.5
12 mes g + h
13 mes double(a) //double命令は、パラメータ内の変数の中身を実数型に変換します
14 i = int(g) //int命令は、パラメータ内の変数の中身を整数型に変換します
15 j = int(h)
16 mes i + j [EOF]
```

Hot Soup Processor ver.3.4

1020
31.000000
10.000000
30

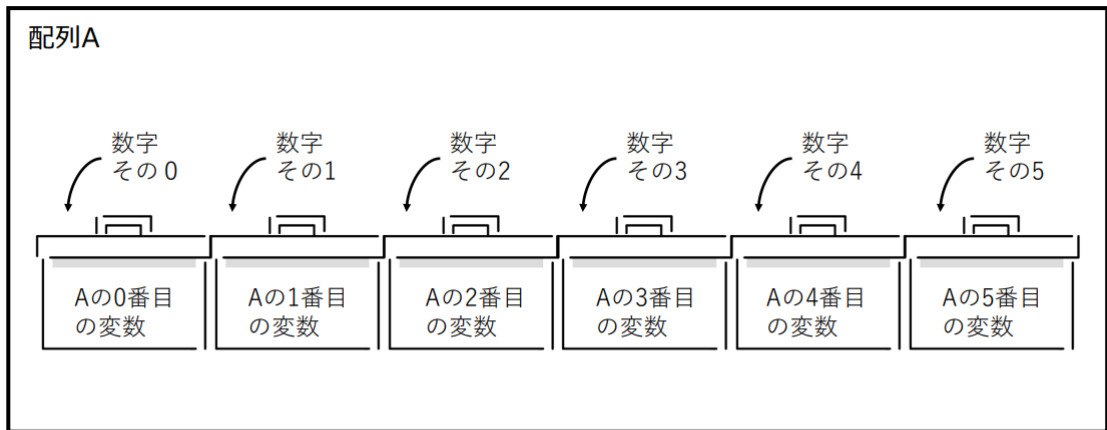
(図 1-9)型変換のプログラムと実行結果



(図 EX-2)型変換のイメージ

1-3.3 配列

プログラミング言語には、連なった変数をひとまとまりにするための**配列**という概念があります。つまり、値を入れる箱（変数）が連なったものを配列といいます。**配列**は通常、0番目から始まることに注意してください。



(図 1-10)配列の説明 変数が横 1 列に連なっているイメージ

上の図 1-10 のように、数字を代入する変数を 1 つにまとめることができます。HSP では、整数、実数、文字列ごとに別の名前の命令が用意されています。

(表 1-2)HSP の配列

命令名	入力できる形式
dim	整数
sdim	文字列
ddim	実数

```
1 a1 = 10
2 a2 = 15
3 a3 = 20
4 a4 = 25
5 a5 = 30
6 wa_a = a1 + a2 + a3 + a4 + a5
7 mes "合計は、" + wa_a + " です。"
8 //-----
9 dim b,5 //b(0)~b(4)までの5つの入れ物がある配列変数bを設定
10 b = 10, 15, 20, 25, 30 //配列bに値を代入
11 wa_b = b(0) + b(1) + b(2) + b(3) + b(4)
12 mes "合計は、" + wa_b + " です。"
13
14 [EOF]
```

(図 1-11)配列変数の利用例 上下とも同じ結果になることがわかる

図 1-11 のように、配列変数を利用することによって、変数の代入や計算を大幅に短縮することができます。

第1章 基本の命令

1-4 メディアの利用

HSP では、文字や数字だけでなく、画像や動画、音声ファイル等も扱うことができます。ここでは、ゲームを制作するときに特に重要となる画像ファイルと音声ファイルの利用方法について解説します。

1-4.1 画像

HSP で画像を扱うには2通りの方法があります。1つは `picload` 命令を利用する、もう一つは `imgload` 命令を利用することです。

```
picload "filename", p1
```

"filename": ロードするファイル名

p1=0~1(0): 画像ロードモード

対応フォーマット: bmp, gif, jpeg, ico, png, psd, tga

```
imgload "ファイル名"
```

"ファイル名": 読み込むファイル名

対応フォーマット: bmp, jpeg, gif, ico, png

Picload 命令と imgload 命令の違いは、picload 命令の方は、画像のサイズに合わせてウィンドウのサイズも変化し、ウィンドウ全体に画像が表示されるのに対し、imgload 命令ではウィンドウのサイズは変化せず、そのまま貼り付けられます。したがって、picload 命令は背景に、imgload 命令はそれ以外に使うと有効です。

また、画像ファイルを細かく動かしたい（連続した画像ファイルでアニメーションさせたいなど）場合は `cellload` 命令を使う方がプログラムを記述しやすいと思います。（詳しくは調べてみてください）

第1章 基本の命令

1-4.2 音楽

HSP で音声ファイルと取り扱う場合、基本的には mm~~と続く命令を使います。

mm 系の命令の一覧は以下の通りです。

```
mmload "filename",p1,p2 //サウンドファイル読み込み
```

p1=0~(0) : 割り当てるバッファの番号

p2=0~2(0): 割り当てるモード

```
mmplay p1 //サウンド再生
```

p1=0~(0): 再生するバッファの番号

```
mmstop //サウンド停止
```

複数のサウンドファイルを読み込む場合、バッファ（データの記憶場所）を 0 から順番に割り当ててください。（1 つ目のサウンドファイルはバッファ 0、2 つ目はバッファ 1、3 つ目はバッファ 2 …）

また、これ以外にも Windows の標準命令である mci 命令を用いる方法や、HSP 拡張プラグインを用いる方法もあります。



(図 1-12,1-13)体験入部プログラムで使われた mmload,mmplay 命令

コラム：ファイルの指定方法

画像や、音声ファイルを指定する場合、方法は2通りあります。1つは**絶対パス**、もう一つは**相対パス**です。

絶対パス：目的ファイルまでの道筋をすべて記述する

相対パス：現在位置(カレントディレクトリ)から目的ファイルまでの道筋を記述する

例えば、デスクトップにプログラムファイルを保存していて、同じデスクトップ上にあるファイル“test”を指定する場合、

絶対パス：C:\User\owner\desktop\test

相対パス：test

となります。また、フォルダの区切り記号「\」について、これは HSP 上で特別な意味の記号として用いられているため、そのままでは\として認識してくれません。かわりに、「\\」とすると、1文字の「\」として認識してくれるようになります。

× C:\User\owner\desktop\test

○ C:\\User\\owner\\desktop\\test

1-5 繰り返し

HSP は基本的に上から下に向かって順番に命令が実行されますが、中には実行する順序を変える命令もあります。それらの命令を**制御文**といい、ここでは同じ命令を繰り返し処理する繰り返し命令について解説します。

1-5.1 repeat/loop 命令

repeat-loop 命令は、HSP 言語の中でも最もポピュラーな繰り返し命令です。以下のように記述します。

第1章 基本の命令

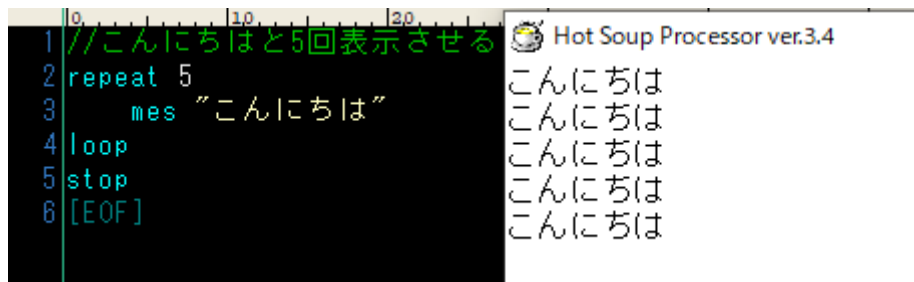
`repeat p1,p2` //ループの始まるの場所を示す

`p1=1~(-1)` : ループ回数

`p2=0~(0)` : システム変数 `cnt` のスタート値

`loop` //ループの始まるに戻る

`repeat` 命令のパラメータに入力しなければ、無限ループとなり、強制的にループを脱出しない限りプログラムが繰り返され続けます。また、`repeat-loop` 命令の中にさらに `repeat-loop` 命令を配置する（入れ子構造）こともできますが、その際後述する **break** 命令などで強制的にループを抜けた場合にエラーが起こる場合があるので、ほかの方法でループをさせたり、気を付けてループを抜けたりする必要があります。



```
1 //こんにちはと5回表示させる
2 repeat 5
3     mes "こんにちは"
4 loop
5 stop
6 [EOF]
```

Hot Soup Processor ver.3.4

こんにちは
こんにちは
こんにちは
こんにちは
こんにちは

(図 1-14)repeat-loop 命令を利用したプログラムの例

コラム：それ以外の繰り返し命令

HSP には、repeat-loop 以外にもいくつかの繰り返し命令があります。

命令	例	内容
do – until 命令	do 実行する命令 until 条件式	条件式が真になるまで 繰り返す。
while – wend 命令	while 条件式 実行する命令 wend	条件式が真の間、 繰り返す。
for – next 命令	for 変数, 初期値, 終値, 増分 実行する命令 next	for x, 1, 5, 1 mes x next 1, 2, 3, 4 と表示され る。
foreach – loop 命令	foreach 配列変数 実行する命令 loop	配列変数の要素数だけ 繰り返す

条件式については、「2-1 条件分岐」で取り扱います。

1-5.2 cnt 変数

HSP には**システム変数**という、あらかじめシステムによって作成された特別な変数があります。その一つがこの cnt 変数です。**cnt 変数は、ループの数を勝手に数えておいてくれる変数**です。この変数を利用することによって、より簡単に繰り返し処理を行うことができます。

```
1 //1から100までの和を計算する
2 saigo = 100//最後の値
3 goukei = 0//合計の数を入れる変数を初期化
4 repeat saigo,1//cntの初期値を1にして、saigoの値まで繰り返す
5   goukei = goukei + cnt//変数goukeiに現在の合計値とcntの値を足し合わせた数を代入する
6 loop
7 mes "合計は" + goukei + "です"[EOF]
```

Hot Soup Processor ver.3.4
合計は5050です

(図 1-15)cnt 変数を利用したプログラムの例

1-5.3 wait/await 命令

繰り返し命令を行う際、コンピュータはできる限り早く繰り返しを行います。そうすると、あまりにも早すぎてPCが固まってしまったり、人間の目で追いつけない速度で繰り返しを行ったりしてしまいます。その対策として、**wait/await 命令を用いて一定時間プログラムを停止させる**ことでちょうどよい速度の繰り返しになるように調節することができます。

```
wait p1 //実行を一時中断する
```

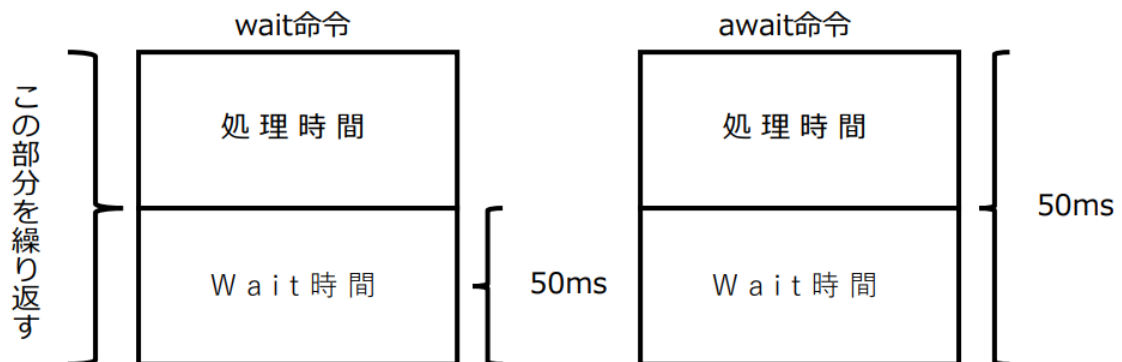
p1(100): 待ち時間(10ms 単位)

```
await p1 //一定の時間で待つ
```

p1=0~(0): 待ち時間(1ms 単位)

※wait は 10ms 単位、await は 1ms 単位であることに注意してください。

wait と await の違いについて、wait はその命令が実行された瞬間から指定された時間待ちます。一方で、await は前の await の終わった瞬間から指定時間待ちます。つまり、間の処理時間を考慮するかしないかの違いとなります。



※どちらも50msの待ち時間を指定した場合

(図 1-16)wait 命令と await 命令の違い

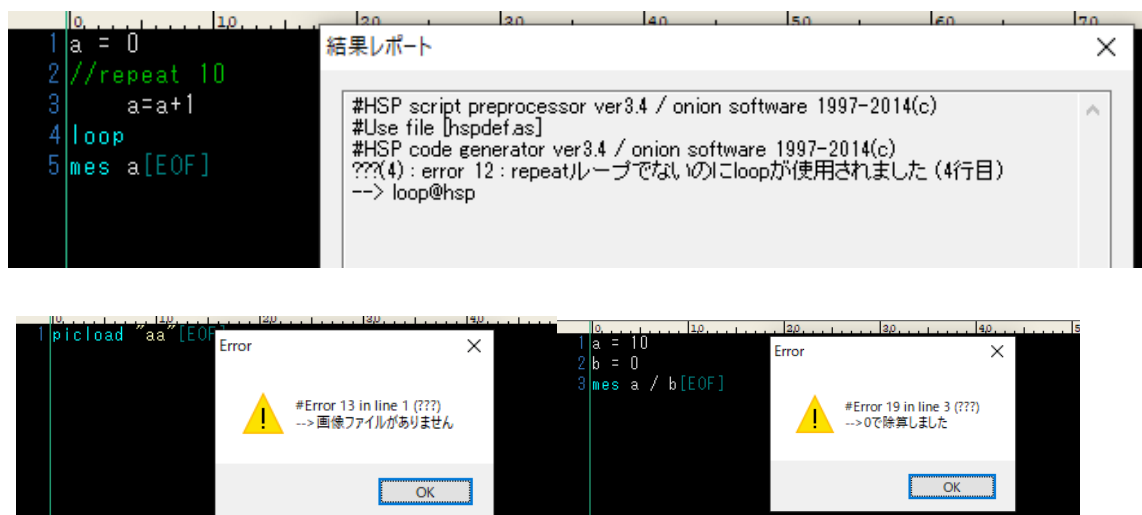
1-5.4 break 命令

1-5.1 の repeat-loop 命令の中で、無限ループを紹介しましたが、その無限ループやその他の繰り返し命令を強制的に脱出する命令があります。それがこの break 命令です。この命令が実行されると、現在行っている繰り返し命令を抜け出し、繰り返しが終わった次の行へジャンプします。ただし、先述したように入れ子構造の repeat-loop 命令の中で break 命令を実行するとエラーが起こる可能性があるため、慎重に使用する必要があります。ちなみに、break 命令とは逆に実行されると強制的に繰り返し命令の始まるの行に戻る continue 命令というものもあります。

コラム：エラーの対処方法

プログラミングをしていると、実行した際にエラーが出る場合があります。その時は以下の方法でエラーの原因を探り、解決してみてください。

まず、プログラムを実行した際に、エラーがあると HSP は途中で処理を中断してエラーダイアログを表示します。そのダイアログにはエラーが起こってしまいプログラムが停止した位置とエラーコード、エラーの原因が記載されています。したがって、記載されているエラー位置の付近を再確認し、エラーの原因と照らし合わせて怪しい箇所がないか確認してください。



(図 1-17～19)エラー表示の例

第1章 基本の命令

エラーの原因がよくわからない場合、エラーコード（上の画像だとエラー12、13、16）を確認し、インターネットで「HSP エラー 「エラーコード」」と検索してみてください。詳しいエラーの説明や解決策が記載されていることがあります。

どうしてもエラーが解決できない場合、エラーが起きる前の状態に巻き戻すという方法もあります。そのためには、プログラムを保存する際に名前を付けて保存からプログラム名の後に番号を振る方法が最も簡単です。

例：test1.hsp test2.hsp test3.hsp test4.hsp test4-1.hsp など

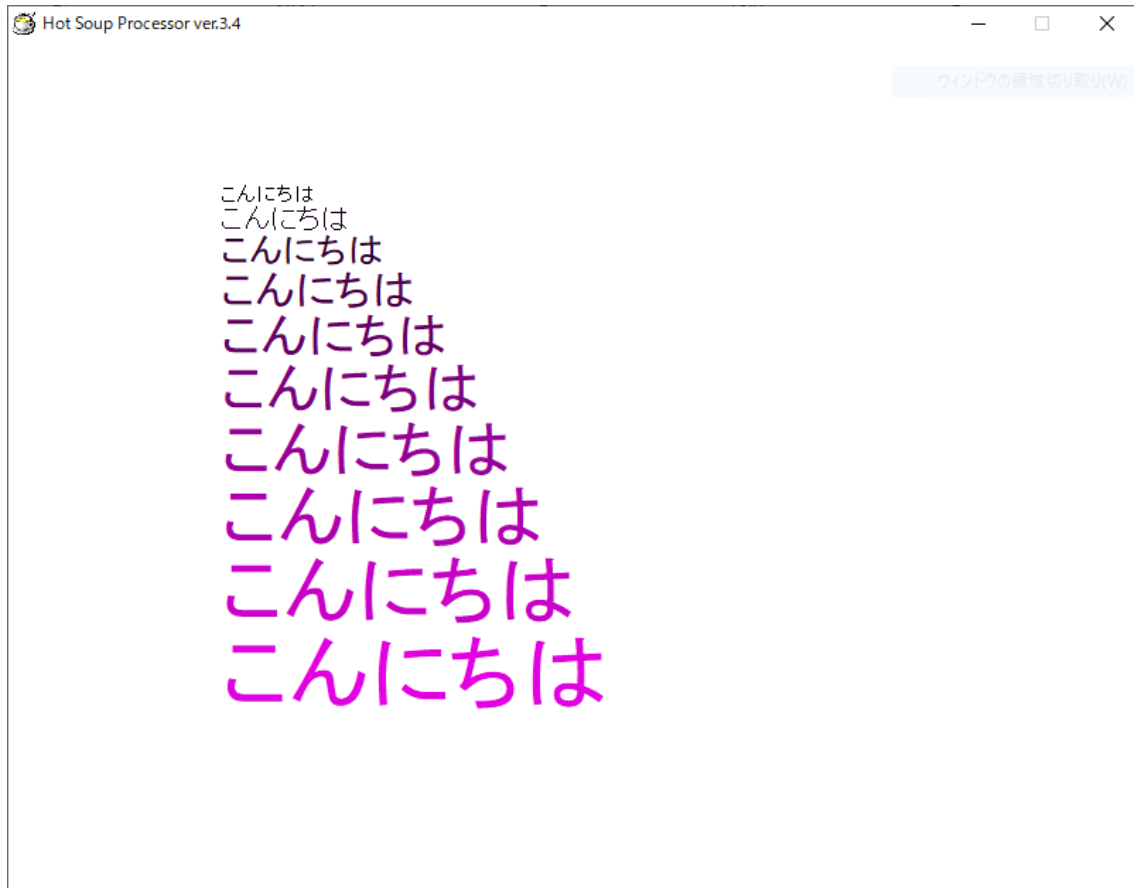
しかし、この方法だとプログラムを何度も書き直すうちにどれがどのプログラムなのかわからなくなってしまうので、詳しい人は Git などを活用してみてもよいと思います。

第1章問題

以下のプログラムを作成してください

- 「こんにちは」のピンク文字がだんだん大きくだんだん濃くなっていくプログラム
- 「こんにちは」の回数は10回
- 200msごとに新しいこんにちはが表示されていくようにする
- RGB値の初期値は0,0,0で1回あたりの増分値は25
- 文字の大きさの初期値は15で1回あたりの増分値は5、フォントは自由
- ウィンドウのサイズは横幅が800、縦幅が600
- 1回目の「こんにちは」の場所はx=150,y=100の位置

第1章 基本の命令



↑ 完成したプログラムの実行結果