How to HSP 言語

2021.06.28

1. 準備

1.1. 「hsp351」をダウンロードしよう

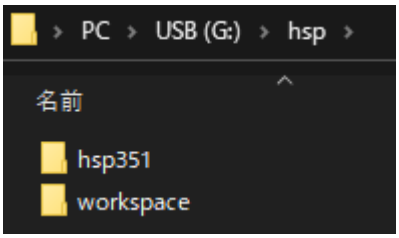
「hsp351」とは hsp 言語の開発環境です。まずは USB にダウンロードしてください。hsp 言語での開発は、すべて「hsp351」を用いて行います。

i. 先輩から USB で直接もらう。
ii. 自分で公式サイトからダウンロードする。
iii. 学校の share フォルダからダウンロードする。

まず始めに、**ダウンロードした「hsp351」フォルダの中身を、他の場所に移動させない**ように注意しましょう。フォルダ内にはたくさんのファイルがありますが、そのうちのいくつかを使って「hsp351」を立ち上げます。それに直接使われないファイルでも、例えばコードの書き方を調べるための「ヘルプ」といった必要なファイルであり、もし欠けていると開発ができなくなる可能性があります。

1.2. フォルダ構成の確認

ファイルエクスプローラーで今ダウンロードした「hsp」の場所を開いてみてください。**「hsp」フォルダ直下に、「hsp351」フォルダと「workspace」フォルダがある状態**のはずです。もしその状態になっていなければ修正しましょう。

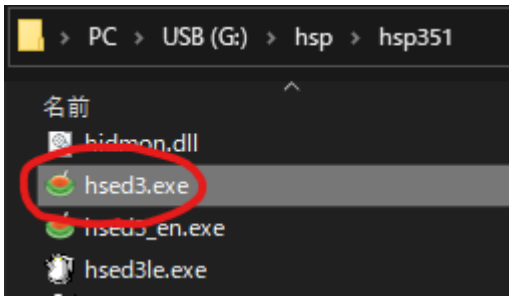


「workspace」フォルダに、これから作っていくプロジェクトを保存します。プロジェクトごとにフォルダを作る方法をおすすめします。詳しくはこの後、実際に作ってみましょう。

1.3. 「hsp351」を開いてみよう

ここまでずっと「hsp351」とフォルダ名で読んでいたものですが、要するにこれはエディタです。メモ帳の上位互換と思って問題ありません。ではエディタを開いてみましょう。

エディタを開くには、「hsp351」内の**「hsed3.exe」**ファイルを開いてください。



「HSP スクリプトエディタ」が立ち上がるはずです。

1.4. 「HSP ヘルプ」を開いてみよう

Hsp 開発環境には、親切にもヘルプがついています。開発のときはヘルプを横に開きながらコードを書きます。ヘルプを開くには、エディタ上で**「F1」キー**を押してください。「HSP Docs Library」が開かれます。それを使って、命令の使い方・解説をみます。調べたい単語の上にカーソルを置いた状態で「F1」キーを押すと、ヘルプの検索欄にその単語が入力された状態で開きます。便利です。

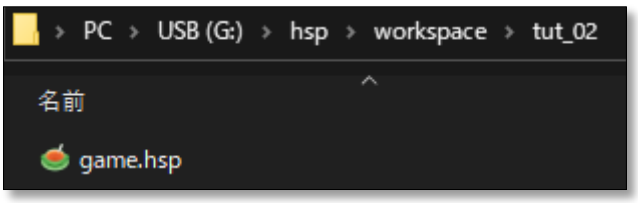
1.5. プロジェクトの作成

プロジェクトというのは、ある一つの開発のことです。例えば、ブロック崩しを作ろうと思ったら、「ブロック崩しプロジェクト」ということです。

プロジェクトの作成というのは日本語がおかしい気がしますが、つまりはファイルの作成です。Word 等で文章を作成したことがある人は、名無しのファイルを編集して、最後に名前を付けて保存する人が多いかもしれません。しかしプログラミングでは先に編集するファイルを作成し、こまめに保存する習慣をつけましょう。

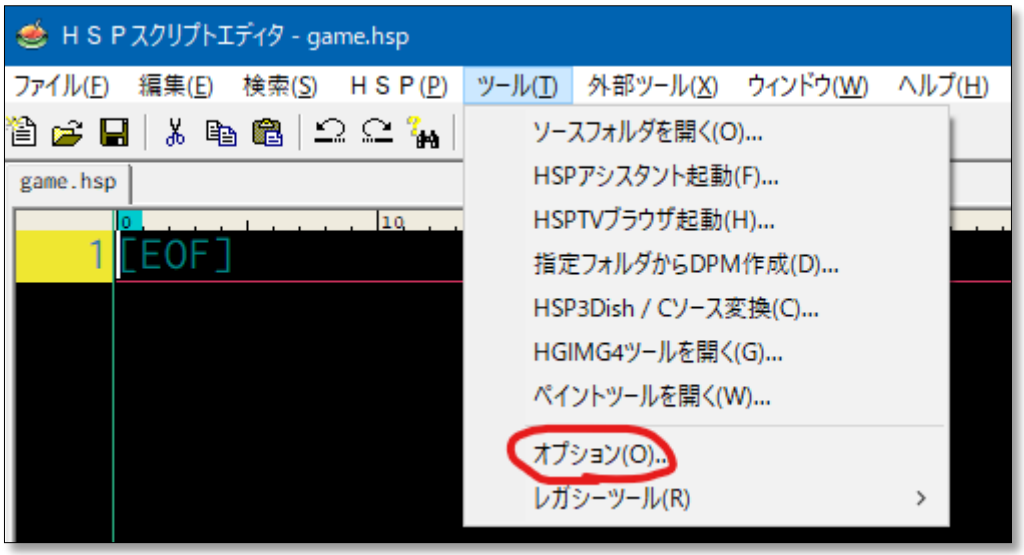
では実際に 2 章で使うプロジェクトを作成してみましょう。1 つのプロジェクトにつき、1 つのフォルダを「workspace」フォルダ直下に作ります。エディタ上で**「Ctrl+S」か、保存ボタンを押して保存する**と、ファイルエクスプローラーが開かれます。

1. 「workspace」フォルダに移動します。
2. 適当に右クリックか上のタブから、フォルダを新規作成し、名前を「tut_02」にします。
3. 「tut_02」フォルダに移動します。
4. 名前を**「game」**にして保存します。
   コンピュータにより勝手に「game.hsp」と拡張子を付けて保存されます。



エディタに戻ると開いているファイル名が「game.hsp」になっています。ちなみに、名前は各々がわかるなら何でもいいです。ただ最初なので、真似することをおすすめします。

初期設定ではかなり文字が小さいので、見にくい人は「ツール→オプション→フォント」から文字の大きさとフォントを変更できます。



1.6. 実行してみよう

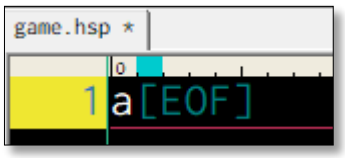
「F5」キーで実行できます。上のタブからも実行できますが、これに関してはショートカットを覚えた方がいいと思います。

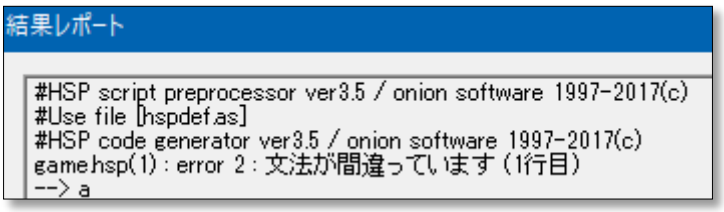
早速実行してみましょう。実行すると、小さな白いウィンドウが表示されます。hsp 言語ではこのウィンドウに図形・文字・画像等を表示させます。ウィンドウの大きさも変えることができます。

そのウィンドウは実行するたびに生成されるので、**動作確認を終えたら閉じましょう**。無駄にたくさん残しておくと、パソコンが重くなってしまいます。

1.7. エラーメッセージ

「a」とだけ入力して、実行してみましょう。先ほどよりも小さなウィンドウが表示され、そこに文字がたくさん書いてあります。





これがエラーメッセージです。書いたコードが間違っているときに実行すると表示されます。**エラーメッセージにはどの行が間違っているのか書いてある**ので、これを見ながらコードを修正しましょう。今回のエラーメッセージを見ると、1 行目が間違っていると書いてありますね。

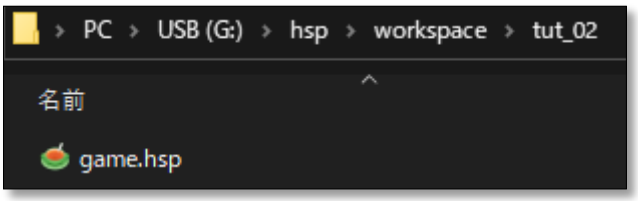
文字を入力したとき、ファイル名の横に「*」が表示されたのに気づきましたか？これはそのファイルが保存されていないことを表しています。こまめに保存することを心掛けましょう。「Ctrl+S」で保存すると、このマークは消えます。

2. ゲーム作りを始めよう

2. 1. 準備

まずは**プロジェクトを作成**しましょう。

今回のプロジェクトは「1.5」項で作成しました。これから「game.hsp」ファイルを編集していきます。

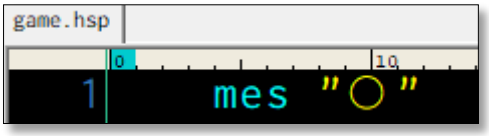


2. 2. 文字の表示

2.2.1. コード

文字の表示には「mes」命令を使います。エディタに「mes」と入力し、そこにカーソルを置いて「F1」キーを押してみましょう。ヘルプが開かれ、「mes」命令について書かれています。もしうまく表示されなければ、ヘルプの左上にある検索ボックスを使ってください。今ヘルプを見てもわからないと思いますが、いずれは必ずわかるようになります。

では次のように入力してください。「mes」の後ろにはスペースが 1 つ入っています。



- ✓ カレントポジションに「""」の中身をそのまま表示します。(カレントポジションについては、後に説明します。)
- ✓ 見た目を変えたければ、「mes」命令より前の行に、「font」「color」「pos」命令を書くことで、文字のフォント・大きさ、色、表示される座標を変えることができます。

2.2.2. インデント

なお文頭の空白は「TAB」キーで作りました。この状態で改行すると、次の行も文頭が空いた状態で改行されます。このように字下げすることを**インデント**といいます。見やすいようにすべての行でインデントを 1 つしましょう。またコードを書く上で、かっこ等で「囲む」ときに、インデントで 1 つ字下げすることがあります。これも後々出てきますが、例えば「repeat / loop」や「if」などです。

インデントやスペースは見やすさのためだけに行っていますが、かなり大事だと思います。どこをインデントするべきか、どこにスペースを入れるべきかわからないうちは、とにかく**真似**をして書いてください。「解読」が困難になってしまうとストレスがたまり、モチベーションが下がってしまうので、面倒くさがらずに入れるようにしましょう。しかもプロジェクトを進めていくうえでコード量が増えていくと、見やすく心がけて書いていないコードは、何が起きているかわからなくなってしまいます。

2.2.3. コード解説

さて、入力したコードに話を戻しましょう。今書いたコードを日本語に訳すと、

```
「○」という文字を表示する
```

という意味です。あくまで使っているのはプログラミング**言語**なので、当然訳せます。というより、何をしたいか考えてからそれをコードに直すわけなので、順序が逆と言えます。このようにコードの内容を日本語等の一般言語で表したものを**疑似コード**といいます。大雑把にでも、コードを書き始める前に、どんなコードを書くか把握しておくと楽にコードを書けます。

今の段階で心がけることは、**コードの意味を日本語で説明できるようにする**ことです。これができるようになれば、ヘルプ片手に開発が一人で進められるようになります。

2.3. 座標

2.3.1. コード

「2.2」で表示させた「○」の、ウィンドウ上に表示させる位置を変えましょう。「pos」命令を使います。

```
1    pos 100, 50
2    mes "○"
```

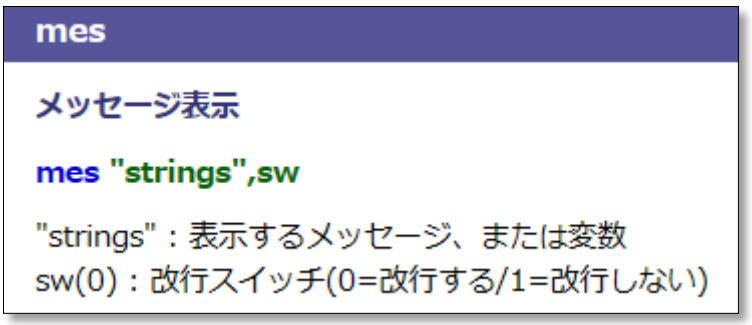
✓ 「pos」命令ではカレントポジションを変更します。

✓ 第一引数(ひきすう)は x 座標, 第二引数は y 座標をとります。

2.3.2. 引数

座標とは関係ありませんが、**引数(ひきすう)**についてです。引数とは今入力したコードの、「100」、「50」、「"○"」の部分のことです。ここでは「pos」命令は引数を 2 つ、「mes」命令は引数を 1 つとっています。「pos」命令のように引数を 2 つ以上取る場合、引数同士を「,」でつなぎます。

命令ごとに何の引数をとるかは、ヘルプに書いてあります。「mes」命令のヘルプを見てみましょう。



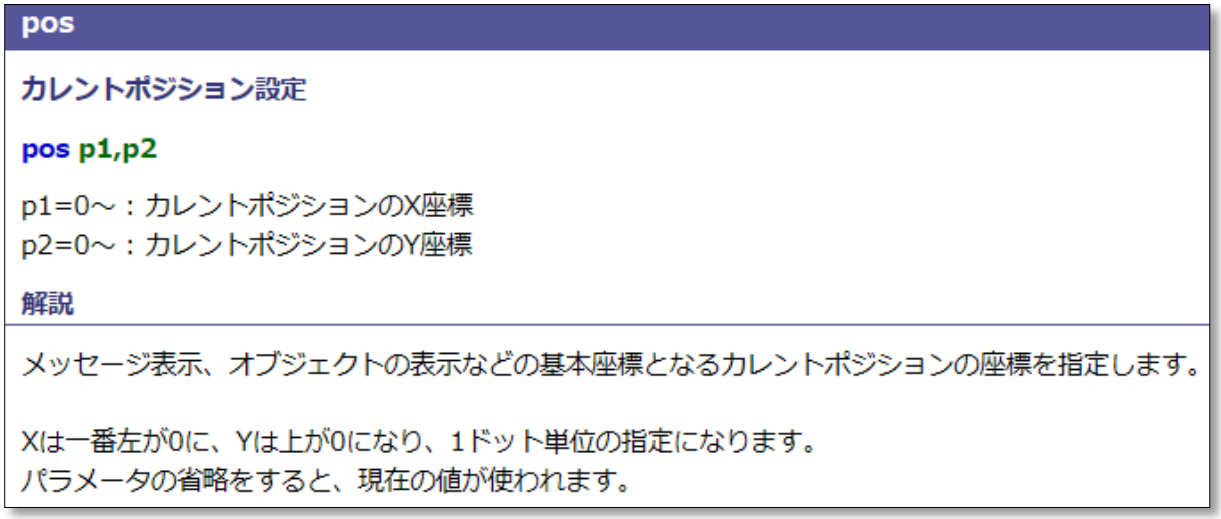
「mes」命令は 2 つ引数をとることがわかります。たださっきのコードでは、第二引数を指定していません。上のヘルプの第二引数「sw」の解説で、「sw(0)」とかっこがついています。かっこは「指定しなければ、かっこの中身を引数として指定する」という意味です。つまり今回、第二引数を指定していないことは、第二引数に「0」を指定するこ

とと同じということです。そしてかっこがついている引数に限り、引数を省略しても構わないというルールがあります。

ヘルプを見て、**かっこがついていたら省略可**、**省略したらかっこ内の内容が指定される**、と理解しておきましょう。

2.3.3. カレントポジション

「pos」命令のヘルプを見てみましょう。



「mes」命令は、カレントポジションに文字を表示させるという命令です。つまり今入力したコードは、

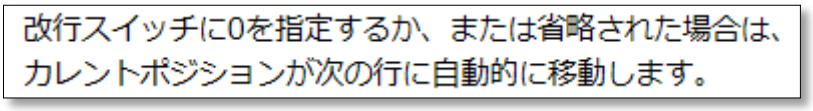
1．カレントポジションを(100, 50)にする。
2．カレントポジションに「○」と表示する。

というものだとわかります。

ここで勘違いしてほしくないのは、「mes」命令で表示させる座標を直前の「pos」命令で指定しているわけではないということです。

```
1    pos 100, 50
2    mes "○"
3    mes "これは2つめのmes"
```

上のコードを入力し、実行してみましょう。「○」の下に「これは 2 つめの mes」と表示されました。「mes」命令のヘルプには次のようにあります。



これでカレントポジションについて、なんとなくわかったと思います。実際文字を表示する場合は、そこまで深く考えなくても、感覚的にコードを書いていけば大抵うまくいきます。それで思い通りにいかなかったときに、ヘルプを見て考えれば良いと思います。

2.3.4. 座標

ウィンドウを x-y 座標系と考えた時、**左上が原点**になり、**右方向に x 軸**、**下方向に y 軸**がとられます。つまり右下に行くにしたがって、座標の数字が大きくなります。

座標が使われるのは、カレントポジションだけはありません。例えば長方形を表示する「boxf」という命令では「左上座標から右下座標まで塗りつぶす」という風に、座標を直接指定します。

2. 4. 繰り返し処理

2.4.1. コード

次のコードを入力してください。

```
repeat 10
mes cnt
loop
```

実行すると 0～9 までの数字が、1 つずつ縦に表示されました。
「repeat」と「loop」は必ずセットで使われます。

- ✓ 「repeat」は繰り返しはじめ、「loop」は繰り返し終わりの位置を表す。
- ✓ 「repeat / loop」で囲まれた内容を、「repeat」の第一引数回だけ繰り返す。
- ✓ 「cnt」は繰り返しで囲まれた部分で使うことができ、一番近くのループの**繰り返し回数**を返す。

今入力したコードは、次のような意味です。

1 巡目のループで「0」を表示。

2 巡目のループで「1」を表示。

・・・

10 巡目のループで「9」を表示。

1 巡目のループで「1」ではなく「0」が表示されることに注意しましょう。

2.4.2. ループの解体

すべての繰り返し処理は解体することができます。今入力したコードは次のように解体できます。

```
mes 0
mes 1
mes 2
mes 3
mes 4
mes 5
mes 6
mes 7
mes 8
mes 9
mes 10
```

このように規則性のあるものをまとめたものが、繰り返し処理です。

繰り返し処理を使うと、コードを書く量が少なくなるのはもちろん、**拡張性**が確保されます。つまり今は「0～9」まで表示するコードですが、簡単に「0～100」まで表示するコードにできるということです。

2.5. 変数

2.5.1. 説明

変数とは、数字・文字・配列を文字で表したものです。まずは数字と文字の変数について説明します。次のコードを入力してください。

```
1  kazu = 10
2  moji = "これは文字"
3
4  mes kazu
5  mes moji
6  mes "kazu"
7  mes "moji"
```

```
10
これは文字
kazu
moji
```

実行すると、右側のように表示されます。このコードにおいて、1・2・4・5行目の「kazu」と「moji」が変数です。

4・6行目を見ると、どちらも「kazu」を出力していますが、6行目には「""」がついています。命令以外で、エディタ上で「""」がついている文字は、実行すると「""」の中身が出力されます。エディタ上で何もついていない文字は変数です。そこより前に決められた値を出力します。つまり黄色になっているのが文字です。

1・2 行目の「=」は数学的なイコールではなく、**代入**という意味です。

今入力したコードでは、4 行目の「kazu」には 1 行目で定めた「10」が代入されているので、「10」と表示されます。6 行目の「"kazu"」はそのまま「kazu」という文字を表すので、「kazu」と出力されます。

2・5・7 行目の「moji」も同様に考えることができます。5 行目の「moji」は変数で、2 行目で「"これは文字"」が代入されているので、「これは文字」と表示されます。7 行目の「"moji"」はそのまま「moji」という文字なので、「moji」と表示されます。

2.5.2. 種類

「kazu」のように、**整数が入る変数を整数型変数または int 型変数といいます**。このような言い方が他にもあります。

int 型	double 型	str 型	char 型
整数	実数	文字列	文字

- ✓ Hsp で変数は、ほとんど数字を扱う。
- ✓ 基本は int 型。小数を扱いたい場合だけ double 型を使う。
- ✓ **int 型と double 型を同時に扱いたい場合は注意**が必要。
  例えば、「4 * 2.5」の出力は「8」。
- ✓ Char 型は 1 文字だけ。
- ✓ **str 型は「""」、char 型は「'」で囲む**。

※ここから 6.28 編集

2.5.3. 主な使い方

上記のように、変数にはいくつかの種類がありますが、hsp の開発では **int 型**と **double 型**を主に使います。文字型はほとんど使う機会はありません。

変数を使うときは、上記の説明のような抽象的な使い方をすることは少なく、明確に意味を持つ**パラメーター**として使われることが多いで

す。そのため、**あくまで「数字」として扱え**ば、難しいことを考えずに感覚的に扱うことができます。エラーが起きてしまったときに、変数の使い方が間違っていないか確認する、くらいの軽い気持ちで良いと思います。

変数を使う例を挙げると、例えばプレイヤーの座標だ。次のスクリプトを入力してみましょう。

```
1    x = 100
2    y = 100
3
4    repeat
5        pos x, y
6        mes "○"
7
8        x++         // xを1増やす
9    await 20        // 20/1000秒待機する
10   loop
```

8・9行目は初めて見るコードだと思うが、意味は右の日本語で今はなんとなく理解してほしいです。

これを実行すると、○が右に移動していき、厚みを持った直線が引かれていきます。

今書いたコードでは、「x, y」は明確に「丸の座標」という意味を持っています。このように意味を持った変数を扱うことがほとんどである。意味を持たない変数を扱う例は、計算をするときなど。

2.6. 条件式(if 文)

2.6.1. 使い方

使い方等はヘルプで「if」を調べてみてください。

「{ }」と「:」の書き方を主に使います。処理が 1 行もしくは短くて簡単なら「:」を使います。

ラベルは非推奨です。

2.6.2. 例

ヘルプの例を見てみましょう。スクリプトは以下の通りです。

```
1    a = 10
2    if a > 5 {
3        mes "TRUE"
4        mes "(MULTILINE IF)"
5    } else {
6        mes "FALSE"
7        mes "(MULTILINE IF)"
8    }
```

条件の「a > 5」は「10 > 5」で正しいのですぐ後ろの「{ true }」が実行されます。ではここにコードを追記してみましょう。

```
10   a = 3
11   if a > 5 {
12       mes "2-TRUE"
13       mes "2-(MULTILINE IF)"
14   } else {
15       mes "2-FALSE"
16       mes "2-(MULTILINE IF)"
17   }
```

「a = 3」が上書きされます。条件は「3 > 5」となり偽なので、「else」の後ろの処理が実行されます。

2.7. 「stick」命令

2.7.1. 定義

入力を管理する命令です。

```
x = 100                    // 「●」のx座標

repeat
    color 255, 255, 255
    boxf                   // 白で画面塗りつぶし

    stick key, 5           // 変数「key」に数字の値を入れる
    if key&1: x--          // 「←」キーが押されていたら
    if key&4: x++          // 「→」キーが押されていたら

    color 0, 0, 0          // 黒
    pos x, 100
    mes "●"
await 20
loop
```

使い方の例は上記の通りです。この命令で扱えるキーやマウスボタンはヘルプの通りです。数字キー等、そこにないキーは扱えません。次の「getkey」命令を用いてください。

「stick」命令の特徴は、**何のキーが押されているかの情報を 1 つの変数で管理する**点です。そのため、条件式で「&」という一見理解しづらい表現をしています。「&」は「それが含まれているか」という意味にとれます。表のとおり、「←」キーが押されていたら「key」に「1」が、「→」キーが押されていたら「key」に「4」が代入されます。

「←」と「→」が同時に押されていたら「key」に 1+4 で「5」が代入されます。表のどの数字の組み合わせでも、またそれらの要素に分解できるという仕組みになっています。例えば、「key」に「22」が入っていたら、「22=2+4+16」で「↑」「→」「スペース」が同時に押されていることがわかります。このように、「22」に「2」の要素が含まれていることを表すのが「key&2」という表現です。

設定によって、**押されている状態**と**押された瞬間**を検知できます。

2.7.2. 使い方

第 1 引数で、何のキーが押されているかの情報を扱う変数を定義します。ここでは「key」という変数名にします。

第 2 引数で、押し続けられたときに扱うことにするキーを設定します。ここで設定しなければ、「stick」命令は基本的にキーが押された瞬間しか検知されません。第 2 引数の数字は、例えば「→」「←」「スペース」の 3 つを押し続けて検知されるように設定したいとき、「1+4+16=21」で第 2 引数に 21 を設定します。この数字はヘルプの表を見てください。

処理の部分では「if」を用いて、変数にキーに設定されている数字が含まれているかを調べます。例えば「key」に「1: ←」が含まれるか調べるには、下の最初の行のように書きます。

2 つのキーが同時に押されているか調べるには、「if」をつなげるか、含まれているか調べる数字の和にすることで実装できます。

```
1    stick key, 22
2    if key&1: //  「←」が押されていたら
3    if key&5: //  「←」「→」が同時に押されていたら
```

※上のスクリプトをそのまま入力しても動作しません。入力系はループと組み合わせて使います。

2.8. 「getkey」命令

2.8.1. 定義

この命令では、あるキーが入力されている情報が、1つの変数に対応します。「A」キーが押されているか調べる変数に「keyA」という変数を設定すると、「A」キーが押されているときは「1」が、押されていないときは「0」が代入されます。

押されている状態を検知します。押された瞬間だけは検知できません。

2.8.2. 使い方

入力されているか調べたいキーに、一つの変数を対応させます。以下の例では「A」キーが入力されているか調べるために、「keyA」という変数を用意しています。

条件式はただ「keyA」と書くだけでも問題ありません。

```
1    getkey keyA, 65
2    if keyA = 1: //  「A」キーが押されたとき
```

※上のスクリプトをそのまま入力しても動作しません。入力系はループと組み合わせて使います。

2.9. 図形の描画

2.9.1. 「boxf」命令

四角形を描画します。引数を 4 つ「左上 x, 左上 y, 右下 x, 右下 y」と順番にとります。

2.9.2. 「circle」命令

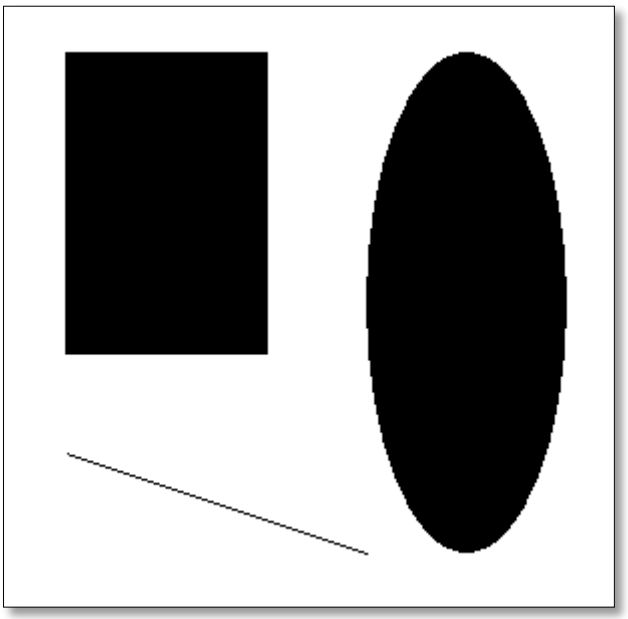
円を描画します。第 4 引数までは「boxf」命令と同じです。第 5 引数に「0」を指定すると円を塗りつぶします。第 5 引数を指定しないか、「1」を指定すると、円を線で描画します。

2.9.3. 「line」命令

線を描画します。引数を 4 つ「始点 x, 始点 y, 終点 x, 終点 y」の順にとります。

2.9.4. 例

```
1    boxf 50, 50, 150, 200
2    circle 200, 50, 300, 300
3    line 50, 250, 200, 300
```



2. 10. ゲーム作りの基本形

2.10.1. 説明

```
1    /* 変数定義 */
2
3    repeat
4    redraw 0
5
6    /* 処理 */
7
8    redraw 1
9    await 20
10   loop
```

```
1    /* 変数定義 */
2
3    repeat
4
5    /* 処理 */
6
7    redraw 0
8
9    /* 描画 */
10
11   redraw 1
12   await 20
13   loop
```

左側をタイプ 1、右側をタイプ 2 と呼ぶ。

コードの量が増えていき、表示するオブジェクトの量が増えると、描画する順番を見やすくするためにタイプ 2 をお勧めします。左側のメリットは、あるオブジェクトの処理と描画を一か所で見ることができる点です。

コメントを残しながら、タイプ 2 で書く方がやりやすいと思います。

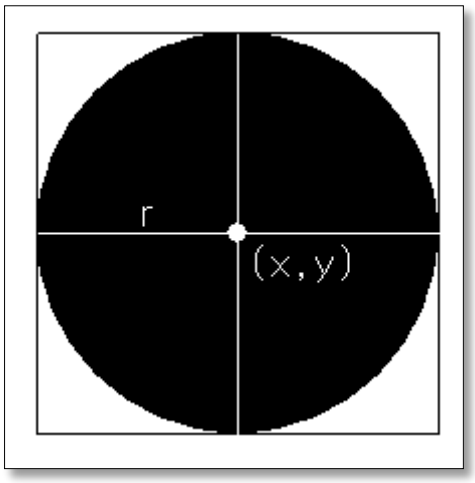
2.10.2. 練習

円が矢印キーで上下左右に動くコードを書いてみよう。

2.10.3. ヒント

定義する変数

- ✧ 「x」「y」: 円の中心座標
- ✧ 「r」: 円の半径
- ✧ 「key」: stick 命令の変数



2.10.4. 正解例

```
x = 100 // 円のx座標
y = 100 // 円のy座標
r = 10  // 円の半径

repeat
    // 円の動き
    stick key, 15
    if key&1: x--
    if key&2: y--
    if key&4: x++
    if key&8: y++

redraw 0
    // 背景
    color 255, 255, 255
    boxf

    // 円
    color 0, 0, 0
    pos x, y
    circle x-r, y-r, x+r, y+r

redraw 1
await 20
loop
```