Pくthon

(ピクソン)

（20220817 版）

１章　日常生活におけるピクトグラム

ピクトグラムとは？

ピクトグラムとは日本語で絵記号，図記号と呼ばれるグラフィックシンボルで，意味するものの形状を使ってその意味概念を理解させる記号です. ピクトグラムは案内, 安全, 施設，機器等々，様々な用途で標準化されています.

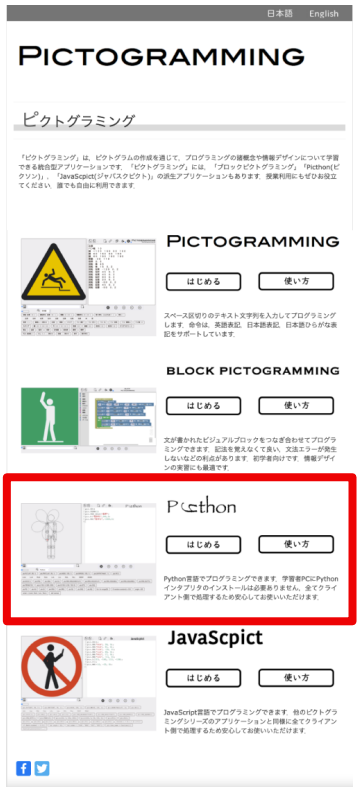
ピクトグラムは，世界共通の記号表現として世界中で用いられていますが，特に近年のグローバル化やその流れに伴う外国人観光客の急激な増加などの理由もあり，ピクトグラムを題材とする研究が盛んになっています.

ピクトグラム作成入門

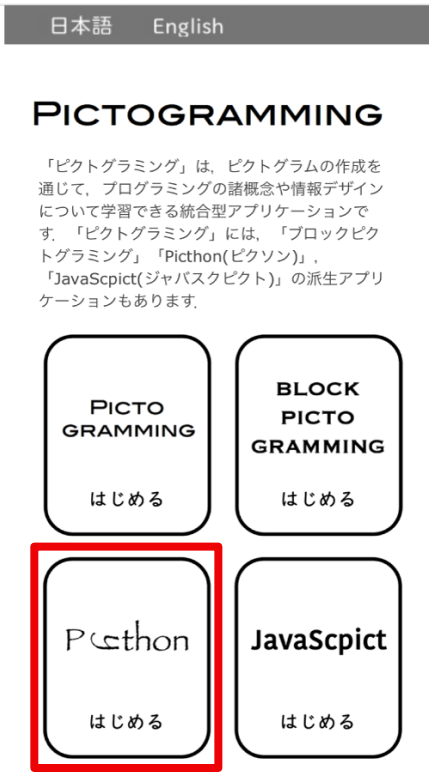
　では実際に，これからピクトグラムを作ってみましょう．
ブラウザでピクトグラミングのホームページへアクセスしてください．アドレスはこちらです．「ピクトグラミング」と検索エンジンで検索してアクセスしても結構です．

`https://pictogramming.org`

ピクトグラミングのトップページが表示されました．PC やノート PC でアクセスする場合は，Picthon の「はじめる」ボタンを押してください．スマートフォンで使えるバージョンも用意されています．　スマートフォン版では左上のボタンをタップしてください。



PC 版



スマートフォン版

ボタンを押すと「図　アプリケーションの画面」に示すアプリケーションが現れます．直接アドレスを入力してもアクセスできます．PC 版のアドレスは，

https://pictogramming.org/editor/picthon.html

スマホ版のアドレスは，

https://pictogramming.org/editor/picthonsp.html

です．

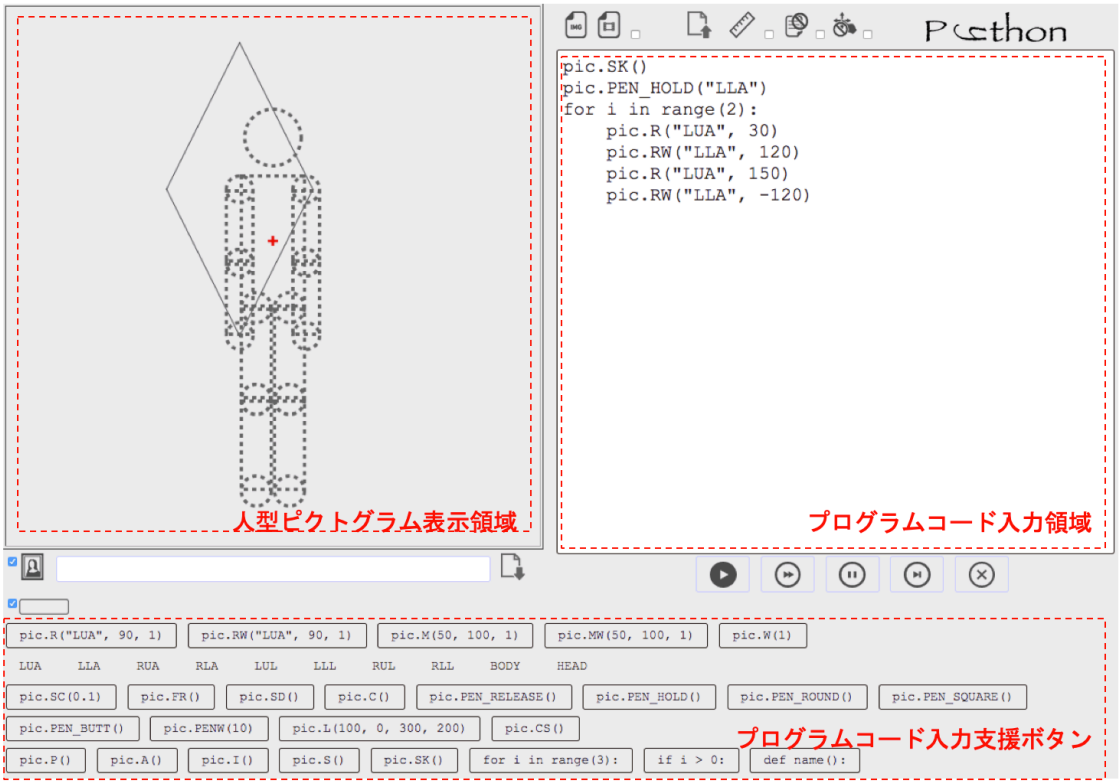


図　Picthon の画面(PC 版)

画面は主に 3 つの部分から構成されています．スクリーンショットの図において上部左側は，プログラムの実行結果を表示する人型ピクトグラム表示領域，上部右側はプログラムを入力するプログラムコード入力領域，下部にはプログラムの入力を支援するプログラムコード入力支援ボタン群が配置されています．左上部分のパネルに表示されているのが，人型ピクトグラムです．

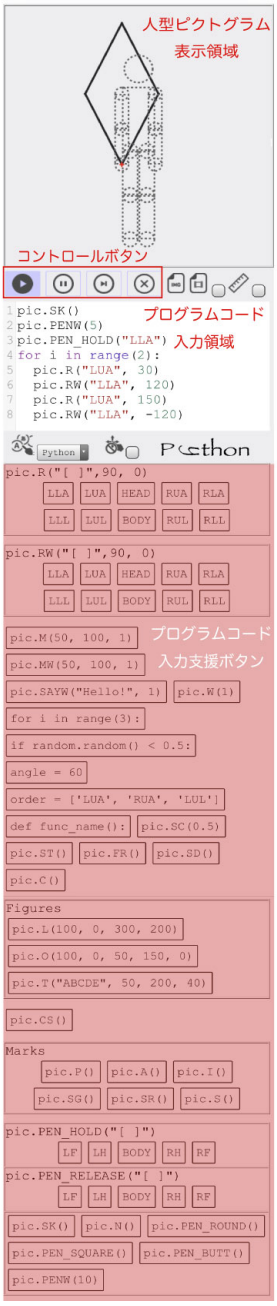


図　Picthon の画面(スマートフォン版)

スマートフォン版の画面は,PC 版と同様に主に 3 つの部分から構成されています. スクリーンショットの図において上部には, プログラムの実行結果を表示する人型ピクトグラム表示領域，中央にはプログラムを入力するプログラムコード入力領域，下部にはプログラムの入力を支援するプログラムコード入力支援ボタン群が配置されています. 上部のパネルに表示されているのが，人型ピクトグラムです.

動きを記述する

さて, まずは右上の「プログラムコード入力領域」に以下のように入力してみましょう.

```
pic.R("LUA", -120)
```

続けて, 実行ボタン  を押してみましょう. はい. 左腕が上がりました.

この記述の意味を説明します.
はじめに, R を説明します. R は「回転（Rotate)」の意味です. R に続くカッコ（）の中身は, あとで説明します.
最初の pic は人型ピクトグラムを示す名称（識別子）です. みなさんも自分の名前がありますよね. 次の". "はドット演算子と呼ばれる記号で, ". "のあとの操作（ここでは R）をする主体となります. pic.R で「pic が R の操作（動作）をする」という意味になります. 言い換えると, 人型ピクトグラムが「回転」という動作をするという意味になります.
次の LUA というのは, 体の部位を示しており, この場合, 左上腕（LUA: Left Upper Arm）の意味です.

さらに 2 行目, 3 行目を追加しました, 下の例では, 左上腕を反時計回りに-120 度, つまり時計回りに 120 度回転させています. さらに右下腿（RLL: Right Lower Leg）を反時計回りに 37 度回転します. それに続けて体全体(BODY)を反時計回りに-52 度, つまり時計回りに 52 度回転させています.

```
pic.R("LUA", -120)
pic.R("RLL", 37)
pic.R("BODY", -52)
```

それぞれの行を**文（statement）**と呼びます. つまり文を複数記述することで, あなたが画面上の人型ピクトグラムに対して命令する訳です. 命令には様式があります. 体の部位を回転する場合の命令の様式を表に示します.

命令の様式	処理
pic.R(arg1, arg2)	arg1 で指定される体の部位を反時計回りに arg2 度だけ支点を中心に回転する.

初めの R が回転を意味する命令の種類です．今後 R 命令だけでなく色々な命令が出てきます．次の arg1，arg2 はいずれも引数（argument）といい，外部から与えられる数や文字のことを言います．Picthon（ピクソン）では，あなたがあなたの分身である人型ピクトグラムに命令をしますので，外部とはあなたのことです．普通，回転（R）しろ！と言われただけでは，どこを（左腕なの？右足なの？）とか何度回転するのとか聞き返すと思います．つまり曖昧なく命令を実行するために与える数字や文字（ここでは RLL とか 37 とか）のことを引数というわけです．

ここで，1 つ目の引数（argument）arg1 は体の部位を示す文字列を指定します．体と頭を組み合わせた部位が 1 つと，上腕，前腕，大腿，下腿が左右それぞれ 1 つの計 9 種の部位から構成されます．「図　人型ピクトグラムを構成する部品」に示します．BODY 以外の体の部位は英字 3 文字で表しています．1 文字目は左側（Left）か右側（Right），2 文字目は上側（Upper）か下側（Lower），3 文字目は腕（Arm）か足（Leg）かです．

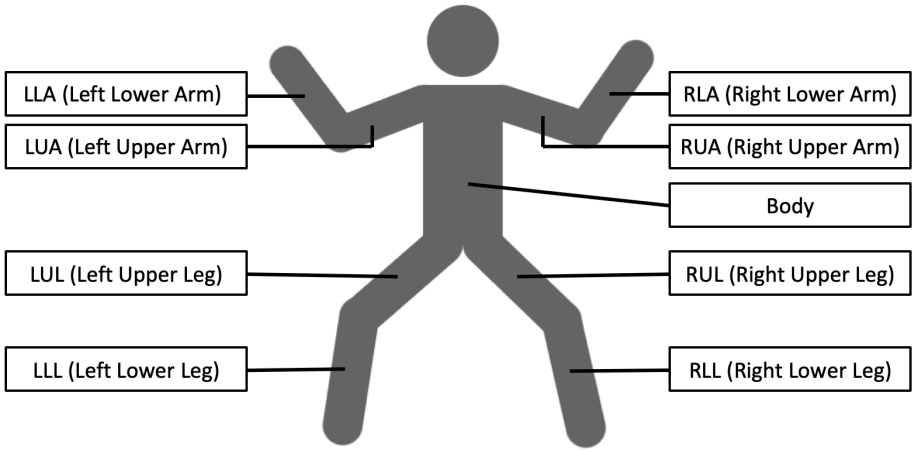


図　人型ピクトグラムを構成する部品

次の 2 つ目の引数 arg2 は回転する角度を示しています．反時計回りが正の値になります．負の値を使って時計回りの角度も表現できます．

ちなみに，命令の名称や体の部位は大文字小文字どちらでも構いません．よって

```
pic.r("rll", 37)
pic.R("Body", -52)
```

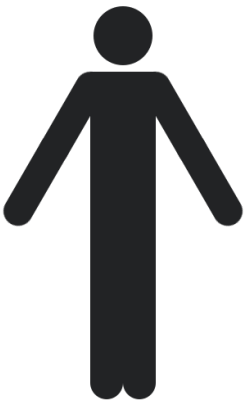
も同じように動きます．

他には次のような命令があります．SD という命令を使うことで，正面からでなく横から見た側面方向のピクトグラムも作ることができます．

命令	処理
pic.FR()	人型ピクトグラムを正面向きにする．（初期状態）
pic.SD()	人型ピクトグラムを側面向きにする．
pic.C()	人型ピクトグラムの状態を直立状態（初期状態）にする．

[演習課題]

(1)から（3）のそれぞれについて，画像と同じ姿勢を作成してください．



(1)

(2)

(3)

2章　逐次（ちくじ）実行，並列（へいれつ）実行

```
pic.R("LUA", -140)
```

と入力して実行ボタンを押しましょう．はい．左腕が上がりました．

ではこの命令の-140 のあとに","（カンマ）を挟み，さらに 1 と記述しましょう．以下のようになります．そして，実行ボタンを押しましょう．

```
pic.R("LUA", -140, 1)
```

どうなりましたか？徐々に腕が上がりました．これは 1 秒間かけて左上腕を関節点中心に反時計周りに-140 度回転するという意味です．言い換えると時計回りで 140 度という意味です．では，さらにこれから，","（カンマ）を挟んで 3 と記述して実行ボタンを押しましょう．

```
pic.R("LUA", -140, 1, 3)
```

プログラムの実行は実行ボタンが押された瞬間から始まりますが，この場合しばらくは回転せず 3 秒後から 1 秒かけて反時計周りに-140 度回転しました．
前回は，命令 R は 2 つの引数を持つと紹介しましたが，実は，命令 R は最大 4 つまで引数を持つことができます．実際の命令 R の仕様は以下の通りです．

命令の表記	処理
pic.R(arg1, arg2, arg3, arg4)	arg4 秒後に arg1 で指定される体の部位を反時計回りに arg2 度だけ arg3 秒かけて支点を中心に等速回転する．arg4 が省略された時は，arg4 に 0 が，arg3，arg4 の両方が省略された時はいずれも 0 が入力されているものとして取り扱う．

では次に以下の 2 行からなるプログラムを記述して実行してみましょう.

```
pic.R("LUA", -140, 1)
pic.R("RUA", 140, 1)
```

1 秒間かけて右図のように両腕を同時に上げるアニメーションが再生されます.

では，左腕が上げ終わってから右腕を上げるようにするにはどうすればよいでしょう.

そのための命令として，RW（Rotate Wait: 回転待ち）命令があります.

先ほどのプログラムの命令名 R のところを RW に変えて実行してみてください. つまり以下の通りです.

```
pic.RW("LUA", -140, 1)
pic.RW("RUA", 140, 1)
```

RW 命令は R 命令と同じですが,「回転が終了するまで次の命令は実行されない.」という意味の命令です.

命令の様式	処理
pic.RW(arg1, arg2, arg3)	arg1 で指定される体の部位を反時計回りに arg2 度だけ arg3 秒かけて支点を中心に等速回転する. 回転が終了するまで次の命令は実行されない.

一方，R 命令は,「次の命令も同時に実行される.」という違いがあります. なので，R 命令では両腕が同時に上がりました. R 命令は,「次の命令も同時に実行される.」ということを注意しなければいけません. 例えば,

```
pic.R("LUA", -140, 1)
pic.RW("LUA", 140, 1)
```

という命令を実行しても人型ピクトグラムは何も反応しません．これは，左上腕を反時計回りに 140 度 1 秒間で回転する命令と，左上腕を反時計回りに-140 度 1 秒間で回転する命令が同時に実行されるため，実際には両方が相殺されるからです．では，次に

```
pic.R("LUA", 40, 1)
pic.R("LUA", 40, 1)
pic.R("LUA", 40, 1)
```

という命令を実行するとどうなるでしょう？これは pic.R("LUA", 120, 1) と同一になります．

一方，

```
pic.RW("LUA", 40, 1)
pic.RW("LUA", 40, 1)
pic.RW("LUA", 40, 1)
```

という命令を実行するとどうなるでしょう？これは pic.RW("LUA", 120, 3)と同一になります．
このように R 命令と RW 命令をうまく組み合わせることで，人型ピクトグラムに対する様々なアニメーションが可能となります．

次に平行移動してみましょう．座標は XY 座標系（X 方向と Y 方向の値，両方を組み合わせて位置を表現する）です．横方向が X 座標で左端の-320 から右端の 320 まで，縦方向が Y 座標で上端の-320 から下端の 320 までをとります．つまり，中心が（0,0），左上が（-320,-320），右上が（320,-320），左下が（-320,320），右下が（320,320），つまり X 軸正方向が右，Y 軸正方向が下となります．数学の座標軸と Y 軸だけ逆方向なので注意して下さい．画面上部の定規アイコンの右のチェックボックスをオンにすると座標系が表示されますので，最初のうちはこの座標系を表示しながら命令を実行すると良いでしょう．

（←画面上部の定規アイコンの右のチェックボックスをオンにする）

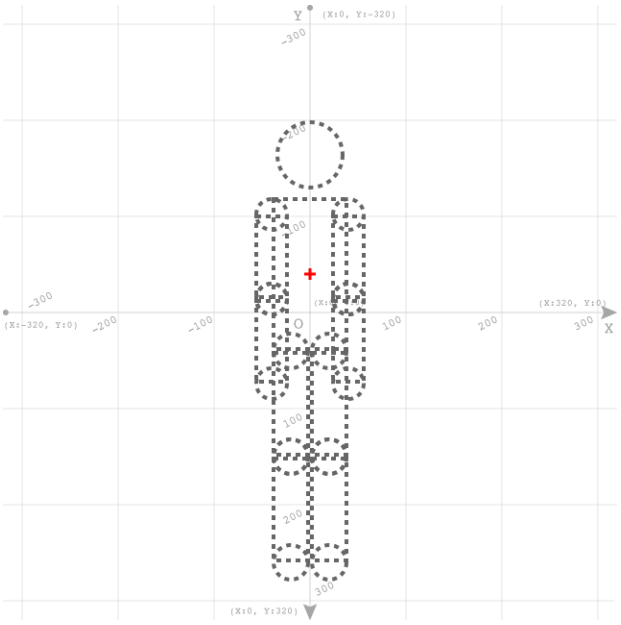


図　座標系を表示した場合の人型ピクトグラム表示領域

ピクトグラムは M 命令または MW 命令を用いて移動できます．平行移動についても W を記述するか否かで，次の命令も同時に実行するか，並行移動が終了するまで次の命令が実行されないかが決まります．

命令の様式	処理
pic.M(arg1, arg2, arg3, arg4)	arg4 秒後に arg3 秒かけて x 軸正方向に arg1 ピクセル， y 軸正方向に arg2 ピクセルだけ全体を等速直線移動する．arg4 が省略された時は，arg4 に 0 が，arg3, arg4 の両方が省略された時はいずれも 0 が入力されているものとして取り扱う．
pic.MW(arg1, arg2, arg3)	arg3 秒かけて x 軸正方向に arg1 ピクセル，y 軸正方向に arg2 ピクセルだけ全体を等速直線移動する．直線移動が終了するまで次の命令は実行されない．

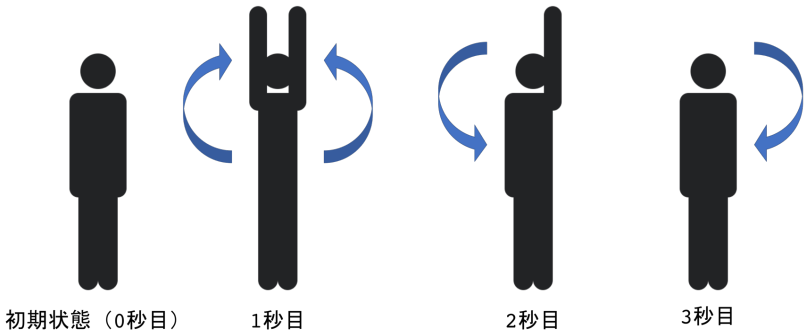
[演習課題]

（1）次のようなアニメーションを作成してください.

0 秒目から 1 秒かけて両腕を上げる.

1 秒目から 1 秒かけて左腕を下ろす.

2 秒目から 1 秒かけて右腕を下ろす.



(2) アニメーションを使った自由作品を作成してみましょう.

3章　似たような処理をまとめる　繰返し処理

```
pic.RW("LUA", -140, 1)
```

と入力して実行ボタンを押しましょう．はい．左腕が上がりました．
では次に，手を振ってみましょう．友達と別れる時を想像してみてください．

```
pic.RW("LUA", -140, 1)
pic.RW("LLA", -60, 0.3)
pic.RW("LLA", 60, 0.3)
pic.RW("LLA", -60, 0.3)
pic.RW("LLA", 60, 0.3)
pic.RW("LLA", -60, 0.3)
pic.RW("LLA", 60, 0.3)
```

3 回左前腕（LLA）を左右に振ることができました. 10 回 20 回振るようにもできます. ただ，プログラムがとても長くなってしまいますし，プログラムを見て何回振っているのかをすぐ確認することも難しくなります. そこで繰り返しの処理を書く方法を学んでみましょう．使う命令は以下の通りです．

命令の表記	処理
`for i in range(arg1):`	対応する命令群を arg1 回繰り返す.

繰返しを使うと以下のように書くことができます．

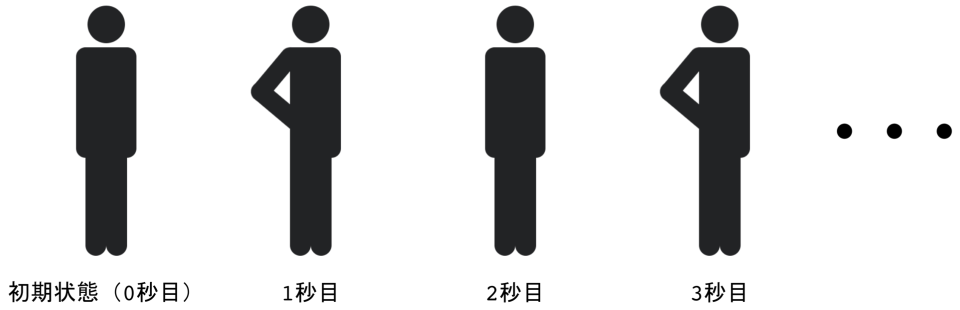
```
pic.RW("LUA", -140, 1)
for i in range(3):
    pic.RW("LLA", -60, 0.3)
    pic.RW("LLA", 60, 0.3)
```

繰り返す対象の命令は 1 段（半角スペース 4 個分）字下げします．ここでは，

`pic.RW("LLA", -60, 0.3)` と `pic.RW("LLA", 60, 0.3)` が繰り返す命令になります.

[演習問題]

(1) 次のようなアニメーションを作成して下さい. ただし 繰返し文を使って下さい. 1 秒かけて左腕を曲げて, 続けて 1 秒かけて再び左腕を伸ばすということを 10 回繰り返す.



(2) 繰返しを使った作品を自由に作成してみましょう.

４章　色々なバイバイを考えてみよう

前回，手を振るプログラムを作りました.

```
pic.RW("LUA", -140, 1)
for i in range(3):
    pic.RW("LLA", -60, 0.3)
    pic.RW("LLA", 60, 0.3)
```

手の振り方を変えたいと思えば，引数の値を変えればいいですね.

```
pic.RW("LUA", -120, 1)
for i in range(5):
    pic.RW("LLA", -90, 0.8)
    pic.RW("LLA", 90, 0.8)
```

ここで，手を振る角度を変えようと思うと 2 箇所変える必要が出てきます．この例では，2 箇所ですが，場合によっては数カ所，数十カ所使われることもあるでしょう．
そこで手を振る角度を表す**変数（variable）**を定義してみましょう．変数とはその名の通り状態や状況によって「変わる数」です．みなさん自身も多くの変数を所持しています．年齢や身長，体重などが代表例ですね.

変数への値の代入は記号 = を用います．命令の様式は以下の通りです． 数学ですと記号 = は左辺と右辺が等しいことを示しますが，多くのプログラミング言語では代入を意味します.

命令の様式	処理
`arg1 = arg2`	変数 `arg1` に `arg2` を代入する.

例えば,

```
waveAngle = 60
```

は,「変数 `waveAngle` に `60` を代入する.」と表現します．すると変数 `waveAngle` を

他の命令の引数としても使えるようになります．このプログラムの場合，以後手を振る角度を変更したい場合，60 の値の部分だけ変更すれば良いわけです．

```
waveAngle = 60
pic.RW("LUA", -140, 1)
for i in range(3):
    pic.RW("LLA", -waveAngle, 0.3)
    pic.RW("LLA", waveAngle, 0.3)
```

さらに腕をあげる角度や，手を振る角度や時間や回数を変数で設定できます．より可読性の高いプログラムが書けます．

```
raiseAngle = 140
raiseTime = 1
waveAngle = 60
waveTime = 0.3
numOfWave = 3
pic.RW("LUA", -raiseAngle, raiseTime)
for i in range(numOfWave):
    pic.RW("LLA", -waveAngle, waveTime)
    pic.RW("LLA", waveAngle, waveTime)
```

[演習課題]

(1)

上のプログラム，raiseAngle, raiseTime, waveAngle, waveTime, numOfWave の値を自由に変更して，あなたが考える最高の「バイバイ」を作ってください．

(2)

上のプログラム，raiseAngle, raiseTime, waveAngle, waveTime, numOfWave の値を自由に変更して，「バイバイ」以外の動作に見えるプログラムにしてください．またその動作の名称を名付けてください．

5章　体の部位を動かして絵を描こう

Picthon(ピクソン)では，人型ピクトグラムの体の部位を動かすことで絵を描くことができます．まずは，下のプログラムを実行してみましょう．

```
pic.pen_hold("LH") # 左手にペンを持つ
pic.R("LUA", 360, 1) # 円を描く
```

線画は，ペンで描きます．そのペンを持ったり離したりできます．１行目の`pic.pen_hold("LH")`は，左手にペンを持つという命令です．ペン関係の命令の様式は以下の通りです．ペンを持つ，放す体の部位は下の図に示すように，いくつか指定できます．上のプログラムで # は行のそれ以降がコメント（人間がわかりやすく理解するためのメモ書き）であることを示しています．ですので実習の時は"#"および"#"以降の文字を入力する必要はありません．

命令の様式	処理
`pic.PEN_HOLD(arg1)`	ペンを持つ．ペンを持つ体の部位の名称を `R,RW` 命令と同様の表記で `arg1` に指定できる．`arg1` が省略された場合は `BODY` が記述されているものと見なされる．
`pic.PEN_RELEASE(arg1)`	ペンを離す．ペンを持つ体の部位の名称を `R,RW` 命令と同様の表記で `arg1` に指定できる．`arg1` が省略された場合は `BODY` が記述されているものとみなされる．
`pic.PENW(arg1)`	ペンの太さ（幅）を `arg1` にする．初期値は 15.

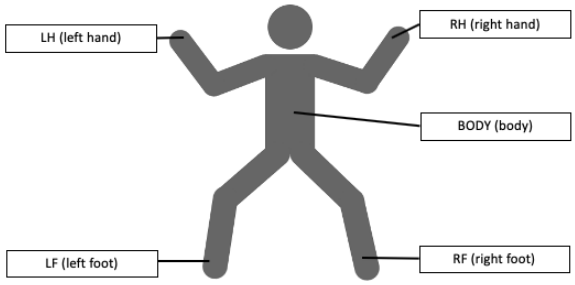
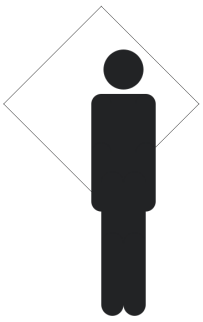


図　ペンを持つことができる位置の名称

では四角形を書いてみましょう.

```
pic.PENW(1)
pic.PEN_HOLD("LH")
for i in range(4):        # 4 回繰り返す
    pic.RW("LUA",90,0)
```

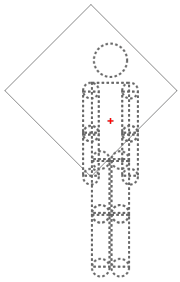


人型ピクトグラムで線画を描くときに, 人型ピクトグラム自身のせいで描いている線画の概形がわかりにくくなることがあるので，人型ピクトグラムを透明にする SK 命令もあります．透明からまた戻すときは，N 命令を実行してください．また，これまでに描いた図形を消す場合は，CS（Clear Screen）命令を使います.

命令の様式	処理
pic.SK()	スケルトンモード（Skeleton mode）に変更する。
pic.N()	ノーマルモード(Normal mode)に変更する。
pic.CS()	ペンによって描画された図形を消去する.

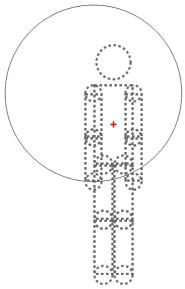
先ほどのプログラムの 1 行目に pic.SK() を挿入しました．人型ピクトグラムが透明になるので，描いた線画が見やすくなります.

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
for i in range(4):
    pic.RW("LUA",90,0)
```



5 行目の最後の 0 を 1 にしてみましょう.

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
for i in range(4):
    pic.RW("LUA",90,1)
```



今度は，円になりました．線画は人型ピクトグラムの移動の履歴を描きます．よって普通に回転すれば，その履歴は円になります．
ただし，時間が 0 の時は，瞬間移動です．Picthon（ピクソン）では瞬間移動した場合は，移動元と移動先を両端とする線分（直線）を描きます．よって上の例では四角形が書けるのです．

次に 5 行目の RW を R に変更してみましょう．

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
for i in range(4):
    pic.R("LUA",90,0)
```

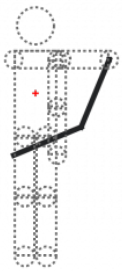
何も線画が描かれなくなりました．これは，次の 2 つのルールがあるからです．

（1）時間 0 の R 命令，M 命令は，元の位置と，R 命令，M 命令からなる複数の命令の動きを全て実行したあとの位置を両端とする線分を描画します．

（2）時間 0 の RW 命令，MW 命令は，元の位置とその命令を実行したあとの位置を両端とする線分を描画します．

この違いを例に示します．

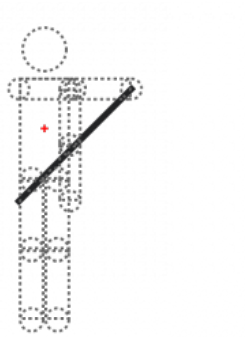
```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
pic.RW("LUA", 45)
pic.RW("LUA", 45)
```



は右のようになります．左腕の初期位置から，左上腕を反時計回りに 45 度回転した位置まで線分を描きます．さらに左上腕を反時計回りに 45 度回転した位置まで別の線分を描きます．

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
```

```
pic.R("LUA", 45)
pic.RW("LUA", 45)
```



は右のようになります．左腕の初期位置から，左上腕を反時計回りに 90（=45+45）度回転した位置まで線分を描きます．

よって先ほどの例は何も描きません．なぜなら，繰り返しを展開すると

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
pic.R("LUA", 90, 0)
pic.R("LUA", 90, 0)
pic.R("LUA", 90, 0)
pic.RW("LUA", 90, 0)
```

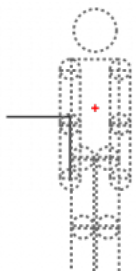
となりますが，これは，

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
pic.RW("LUA", 360, 0)
```

と等しいからです．

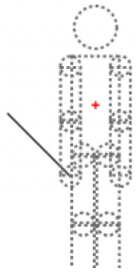
同様に，

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
pic.MW(100, 0)
pic.MW(0, 100)
```



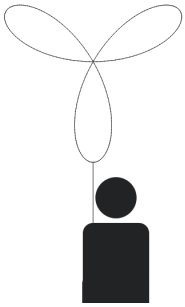
は右のようになりますが，

```
pic.SK()
pic.PENW(1)
pic.PEN_HOLD("LH")
pic.M(100, 0)
pic.MW(0, 100)
```
は右のようになります.



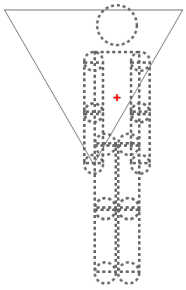
ピクトグラムの体の部位を動かす際と同様，R, RW, M, MW 命令をうまく使い分けることで，様々な線画を描くことができます．例えば，下は三つ葉のクローバーです．

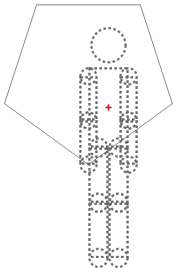
```
pic.PEN_HOLD("LH")
pic.PENW(1)
pic.R("LUA", 360, 5)
pic.RW("LLA", -1080, 5)
pic.MW(0,300)
```



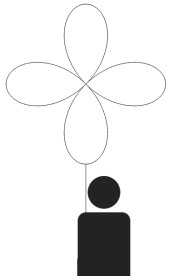
[演習課題]

(1) 正三角形，正五角形をそれぞれ描いてみましょう．

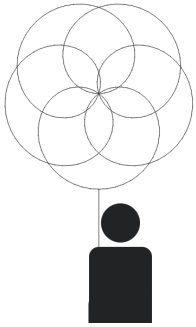




(2) 三つ葉のクローバーのプログラムを変更して，幸せを呼ぶ四つ葉のクローバーを描いてみましょう．

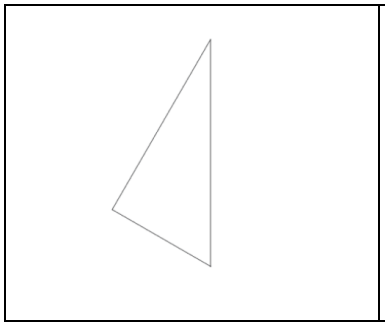


(3) 三つ葉のクローバーのプログラムのいずれかの命令のある引数を一箇所変えるだけで，右のようなお花を描くことができます．やってみましょう．

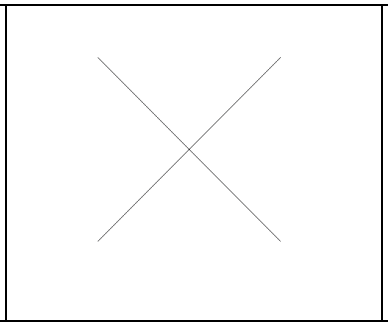


(4) 次の図形を描いてみよう

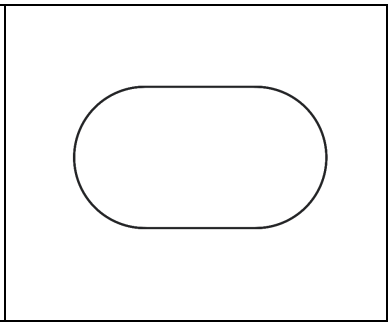
(A) 内角がそれぞれ 90 度，
60 度， 30 度の直角三角形



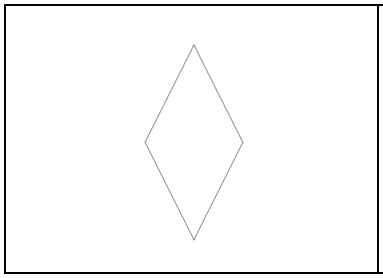
(B) バツマーク



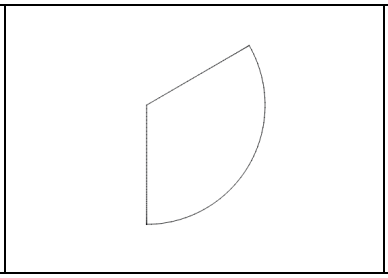
(C) グラウンド



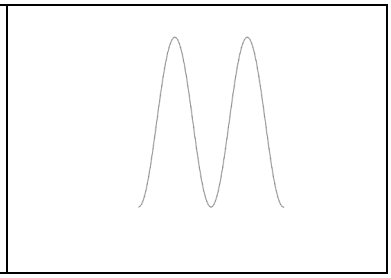
(D) ひし形



(E) 中心角が 120 度の扇型



(F) サインカーブ



6章　レッツ，ダンシング

下のプログラムを実行してみましょう．その後その動作を真似てみましょう．

```
order = ['LUA', 'LUL', 'RUA', 'RUL']
for parts in order:
    pic.RW(parts, -45, 0.5)
    pic.R(parts, 45, 0.5)
```

実行すると順番に「左腕」,「左足」,「右腕」,「右足」を上げ下げします．
ここで，複数の値を順番付けしてまとめたものをリスト（list）といいます．
上のプログラムでは，これがリストになります．

```
['LUA', 'LUL', 'RUA', 'RUL']
```

前回の変数への代入と同様，代入文を使って，
`order = ['LUA', 'LUL', 'RUA', 'RUL']`
`order` に代入しています．

２行目の

```
  for parts in order:
```

はこれまで

```
  for i in range(5):
```

のように書いてきたと思います．これは実は，

```
  for i in [0,1,2,3,4]:
```

の意味でした．

リストのそれぞれの値のことを要素といいます．要素はリスト名のあとに中かっこ番号で指定します．番号は先頭の要素が `0` で，順番に `1,2,3` と続いていきます．

order = ['LUA', 'LUL', 'RUA', 'RUL'] の

order[0] は 'LUA' ，order[1] は 'LUL' ，order[2] は 'RUA'，order[3] は 'RUL' となります．リストの要素は新しい値で更新できます．

例えば，先ほどのプログラムに１行追加して，

```
order = ['LUA', 'LUL', 'RUA', 'RUL']
order[2] = 'LUA'
for parts in order:
    pic.RW(parts, -45, 0.5)
    pic.R(parts, 45, 0.5)
```

として実行してみましょう．

要素群を囲う記号を中かっこ（[]）の代わりに，かっこ（()）で囲ってみましょう．

```
order = ('LUA', 'LUL', 'RUA', 'RUL')
for parts in order:
    pic.RW(parts, -45, 0.5)
    pic.R(parts, 45, 0.5)
```

このように，複数の値を順番付けしてかっこ（()）で囲ってものをタプル（tuple）がと言います．タプルとリストの大きな違いは，要素の値を更新できないことです．なので，

```
order = ('LUA', 'LUL', 'RUA', 'RUL')
order[2] = 'LUA' # エラーとなる
for parts in order:
    pic.RW(parts, -45, 0.5)
    pic.R(parts, 45, 0.5)
```

はエラーとなります．実行時に不用意に値が変更されないようにできます．

[演習課題]

あなたが踊ることができるオリジナルダンスをリストまたはタプルを使って表現してみましょう．

７章　さらに激しくダンシング

値，変数，演算子の組み合わせのことを**式**と言います．値と変数はこれまでに出てきました．例えば，

```
angle = 60
```

は変数 angle に値 60 を代入するという意味でした．演算子とは演算を表す記号のことをいいます．例えば下にあるような +，-，*，/，//，% などです．特に計算のために用いる演算子なので算術演算子と呼ばれています．

算術演算子の様式	評価
A + B	Ａと Ｂを足す．
A - B	Ａから Ｂを引く．
A * B	Ａと Ｂを掛ける．
A / B	Ａを Ｂで割る．
A // B	Ａを Ｂで割る．（小数点以下切り捨て）
A % B	Ａを Ｂで割ったあまり．

繰り返しますが，式とは，値，変数，演算子の組み合わせです．ここでは，

```
angle + 10
```

は式になります．代入式の右辺には，式を記述できます．
angle に angle + 10 を代入するというのは，

```
angle = angle + 10
```

と書けます．つまり，angle の値を 10 増やす処理になります．次のプログラムを見てみましょう．

```
angle = 30;
for i in range(3):
    pic.RW("LUA", -angle, 0.5)
```

```
pic.RW("LUA", angle, 0.5)
angle = angle + 30
```

実行すると順番に「左腕」30 度 60 度 90 度と上げ下げします.

[演習課題]

徐々に人型ピクトグラムの動きが激しくなるダンスを作ってみましょう.

８章　分岐を使って異なった処理をする

日常生活において，人は条件によって異なった意思決定をします.
例えば,「もし，明日晴れならばピクニックに行こう」「もし，財布の中に 5000 円以上あるなら買おう」などです．つまり条件によって実行する命令を変えたいときです．その場合は，if 命令を使います.

命令の様式	処理
if exp1:	もし 条件式 exp1 が真ならば対応する命令群を実行する.

if 命令の引数には比較演算子を使った条件式を記述します．比較演算子には以下のものがあります.

比較演算子の様式	評価
A > B	A が B より 大きい.
A >= B	A が B より 大きい か 等しい.
A < B	A が B より 小さい.
A <= B	A が B より 小さい か 等しい.
A == B	A と B が 等しい.
A != B	A と B が 等しくない.
A in B	A は B の要素になっている.

例えば，以下の通りです．for 文と同様に，if の条件式が真のときに実行される命令群は４文字分インデント（字下げ）します.

```
pic.RW("LUA", -140, 1)
if random.random() < 0.5:
    pic.RW("LLA", -60, 0.3)
    pic.RW("LLA", 60, 0.3)
```

ここで,random.random()は 0 以上 1 未満の中からランダムに実数の値を返します.
つまり，random.random() < 0.5 は字下げされた命令群が 50%の確率で実行されます.

.

for 文を入れ子にした場合と同様，if 文と for 文が組み合わされた場合も，インデントの文字数は 4 文字ずつ増えていきます.

```
pic.RW("LUA", -140, 1)
if random.random() < 0.3:
    for i in range(3):
        pic.RW("LLA", -60, 0.3)
        pic.RW("LLA", 60, 0.3)
```

これは，30%の確率で手を 3 回振ります．もう少し複雑な条件を記述してみましょう.

命令の様式	処理
if exp1:	もし 式 exp1 が真ならば対応する命令群を実行する.
elif exp2:	もし対応する先述の if または elif の条件が全て満たされなくて，かつ条件式 exp2 が真ならば対応する命令群を実行する.
else:	もし対応する先述の if または elif の条件が全て満たされない場合，対応する命令群を実行する.

例えば下のプログラムは，左股，右腕，左腕のいずれかをそれぞれ，30%, 35% (=(1−0.3)×0.5)) , 35% (=(1−0.3)×(1−0.5)) の確率で動かすことを 20 回繰り返します.

```
for i in range(20):      # 20 回繰り返す
    if random.random() < 0.3: # 30%の確率で
        pic.RW("LUL", 90, 0.2)
        pic.RW("LUL", -90, 0.2)
    elif random.random() < 0.5:  # それ以外で 50%の確率で
        pic.RW("RUA", 90, 0.2)
        pic.RW("RUA", -90, 0.2)
    else:  # それ以外で
        pic.RW("LUA", 90, 0.2)
```

```
pic.RW("LUA", -90, 0.2)
```

[演習課題]

第 4 章で習った体の部位で図形を描くというのがありました．ランダムに動く人型ピクトグラムの特定の部位にペンを持たせて，ランダムな図形を描いてみましょう．

９章　色々なバイバイをまとめてしてみよう．　-関数定義-

一連の命令をひとまとめで考えたくなる時があります. 例えばこれまでに出てきたバイバイのプログラムです.

```
num = 3
angle = 90
pic.RW("LUA", -140, 1)
for i in range(num):
    pic.RW("LLA", -angle, 0.3)
    pic.RW("LLA", angle, 0.3)
pic.RW("LUA", 140, 1)
```

一連の命令をひとまとめにして定義することで, その一連の命令を繰り返し利用しやすくなります. またその一連の命令に対して名前を定義できるので, わかりやすくなります. そこで関数というものを使います.

命令の様式	処理
def name(arg1, .. ,argN):	arg1 から argN までの N 個の引数を伴う関数 name を登録する.

では, 関数 waveHand を定義しましょう.

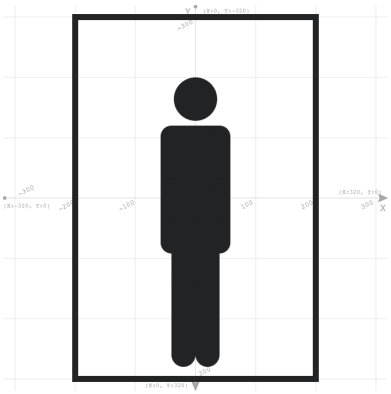
```
def waveHand(angle, num):
    pic.RW("LUA", -140, 1)
    for i in range(num):
        pic.RW("LLA", -angle, 0.3)
        pic.RW("LLA", angle, 0.3)
    pic.RW("LUA", 140, 1)

waveHand(60, 3)
waveHand(30, 10)
```

1 行目から 6 行目までが関数 waveHand の定義です．引数は手を振る角度 angle と手を振る回数 num です．
8 行目，9 行目でそれぞれ関数 waveHand を呼び出しています．
よって，このプログラムを実行すると，「腕を上げ，手を 3 回 60 度振って，腕を下ろす，腕を上げ，手を 10 回 30 度振って，腕を下ろす．」という一連の動作をします．

[演習課題]
人型ピクトグラムが歩く動きをする walk を関数で定義して，引数を変えて複数回関数 walk を呼び出してみましょう．

10 章　座標を指定して線画を描き動かす



5 章で，人型ピクトグラムの部位にペンを持たせて，移動の軌跡で線画を描くことを学びましたが，右の図のように，描く図形によっては，座標を直接指定して線を描くのが簡単です．そこで，Picthon（ピクソン）では，線分の両端の座標を指定して描く L 命令や楕円を描く O 命令というのが用意されています，仕様は以下の通りです．

命令の様式	処理
`pic.L(arg1,arg2,arg3,arg4)`	座標（`arg1,arg2`）から座標`(arg3,arg4)`まで線分を描く．
`pic.O(arg1,arg2,arg3,arg4,arg5)`	中心座標（`arg1,arg2`），幅 `arg3`，高さ `arg4`，中心座標を中心に反時計回りに `arg5` 度回転した楕円を描く．`arg5` が省略された時は，`arg5` に 0 が入力されているものとして取り扱う．

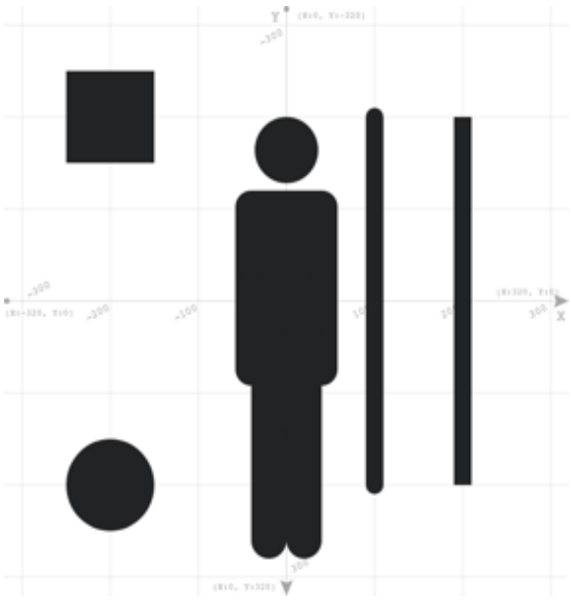
よって，例えば，`pic.L(100, -150, 200, 300)` は座標（`100, -150`）と座標（`200, 300`）を両端とする線分を（一瞬で）描きます．

描く線の太さや線種も変更することができます．

命令の様式	処理
pic.PEN_SQUARE()	以後描画する線の両端の形状を四角にする.
pic.PEN_ROUND()	以後描画する線の両端の形状を半円にする.
pic.PEN_BUTT()	以後描画する線の両端の形状を無しにする.
pic.PEN_NORMAL()	以後描画する線の線種を実線にする.
pic.PEN_ERASE()	以後描画する線の線種を消線にする.
pic.PEN_XOR()	以後描画する線の線種を反転線（すでに描かれていた部分は消し，そうでない部分は描く）にする.

線の太さや線の種類を変更して描画した例を下に示します.

```
pic.PENW(20)
pic.PEN_BUTT()
pic.L(200,-200,200,200)
pic.PEN_ROUND()
pic.L(100,-200,100,200)
pic.PENW(100)
pic.PEN_SQUARE()
pic.L(-200,-200,-200,-200)
pic.PEN_ROUND()
pic.L(-200,200,-200,200)
```



また，任意の文字の大きさの文字列を，指定した座標に描画することもできます.

命令の様式	処理
pic.T(arg1, arg2, arg3, [arg4])	座標（arg2,arg3），文字の大きさ arg4 で文字列 arg1 を描く. arg4 が省略された時は，arg4 に 80 が入力されているものとして取り扱う.

描画した線画をグループ化し名称を付けて，その物体（オブジェクト）を移動させることもできます.

リンゴを落とすプログラムを下に示します.

```
def apple():
    pic.O(211, -50, 42, 42, 0)
    pic.L(211, -55, 211, -85)
pic.RV()
pic.RW("RS", 69, 0)
pic.RW("RE", 41, 0)
pic.MO(apple, 0, 320,0.5)
```

1 行目から 3 行目で関数 apple を定義しています. 関数に含まれる L 命令と 0 命令の集合が apple(リンゴ)を構成する線画の集合です. 7 行目の MO(Move Object)命令が関数で定義されたオブジェクトを移動させる命令です. 関数自体を第一引数にします. この例では, 0.5 秒かけて, x 軸正方向に 0, y 軸正方向に 320 リンゴを移動させます.
同様に, MOW(Move Object Wait)という命令（メソッド）もあり, それぞれ次のとおりです.

命令の様式	処理
pic.MO(func, arg1, arg2, arg3)	arg3 秒かけて x 軸正方向に arg1 ピクセル, y 軸正方向に arg2 ピクセルだけ func という名称で定義された関数で構成される線画を等速直線移動する.
pic.MOW(func, arg1, arg2, arg3)	arg3 秒かけて x 軸正方向に arg1 ピクセル, y 軸正方向に arg2 ピクセルだけ func という名称で定義された関数で構成される線画を等速直線移動する. 直線移動が終了するまで次の命令は実行されない.

人型ピクトグラムの動きと同様, 線画オブジェクトに関しても MO 命令と MOW をうまく組み合わせると複数のオブジェクトを表示させたり, 複雑な動きをさせることができます.

11 章　正しく伝える難しさを知る -ピクトグラムデザイン-

これまで，自由な発想で様々な作品を作成してきたと思います．今回は，社会や身の回りに役立つピクトグラムというのを作成してみましょう．下のピクトグラムは，いずれも安全図記号で指定されている図例です．さらに禁止，注意，指示，安全の 4 項目に関するピクトグラムデザインのガイドラインも策定されています．通常，世の中に広く普及しているピクトグラムは作成ガイドラインに則ってデザインされています．ピクトグラムで大切な事は誰にでも正しく伝わるという事なのです．ピクトグラミングでは通常のモードに加え，禁止，注意，指示，安全用のピクトグラムを作るための 6 つのモードを備えています．

禁止	注意	指示	安全	安全 (緑)	安全 (赤)
pic.P()	pic.A()	pic.I()	pic.S()	pic.SG()	pic.SR()

命令の様式	処理
pic.P()	禁止モード（Prohibit mode）に変更する．
pic.A()	注意モード（Attention mode）に変更する．
pic.I()	指示モード（Instruction mode）に変更する．指示モード中に描画した線画の色は人型ピクトグラムと同じ白となる．
pic.S()	安全モード（Safety mode）に変更する．安全モード中に描画した線画の色は人型ピクトグラムと同じ緑色となる．
pic.SG()	安全緑モード(Safety Green mode)に変更する．安全緑モード中に描画した線画の色は人型ピクトグラムと同じ白色となる．
pic.SR()	安全赤モード(Safety Red mode)に変更する．安全赤モード中に描画した線画の色は人型ピクトグラムと同じ白色となる．
pic.N()	ノーマルモード(Normal mode)に変更する。

さてこれらは，社会の中でよく見かけるピクトグラムの例です．

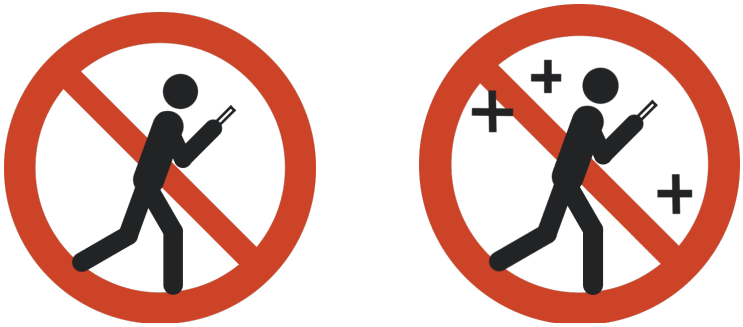
人型ピクトグラムにポーズをとらせて，伝えたい情報を正しく伝えることができるように，色付きのマークをつけて，さらに線画を付け加えてみましょう．

さて，世の中に溢れるピクトグラムはポスターなど紙媒体上で提示されているので基本は画像（静止画）です．一方最近では，デジタルサイネージ（電子掲示板）の発達により，街中でもアニメーションのピクトグラムを提示させることができるように技術的にはなっています．またスマートフォンなどは位置情報や無線によって，状況に応じて画面上に何らかの表示をすることができるようになっているので，静止画や動画のピクトグラムを表示させて，注意喚起や情報提供をすることができるでしょう．

ピクトグラミングは，静止画だけではなく動画（アニメーション）のピクトグラムを作成することができます．ぜひ，単なる静止画のピクトグラムではなく，オリジナルのアニメーションピクトグラムを作成してみましょう．

「図形は最低限に」

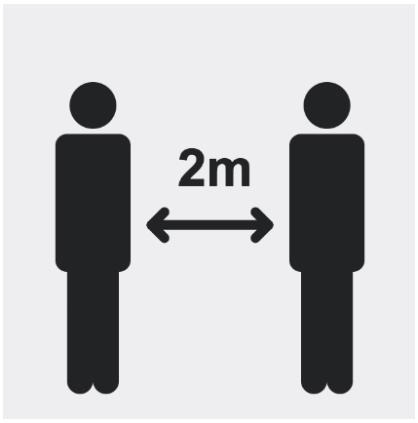
例えば，左のピクトグラムは，歩きスマホ禁止を表現していますが，素っ気ないので少しアクセントを加えようと，右のように十字のマークをいくつかつけたとしましょう．するとこのピクトグラムを見た人は，何かこの十字には意味があるのではないかと推察してしまい，結果として誤った解釈を誘発してしまう可能性があります．図形描画は必要最低限にしましょう．



「文字は使用しない」

例えば，左のピクトグラムは，歩きスマホ禁止を表現していますが，上部に「歩きスマホ禁止！」の文字列を付与しています．ピクトグラムは，図的記号でこれでは，文字を

読んで理解しようとしてしまうので NG です. 右のピクトグラムは，ソーシャルディスタンス（物理的距離）に関するものですが，伝えたいことは 2m という距離です．このように数量自体が伝えるべき具体的内容である場合に限り使用するようにしましょう.

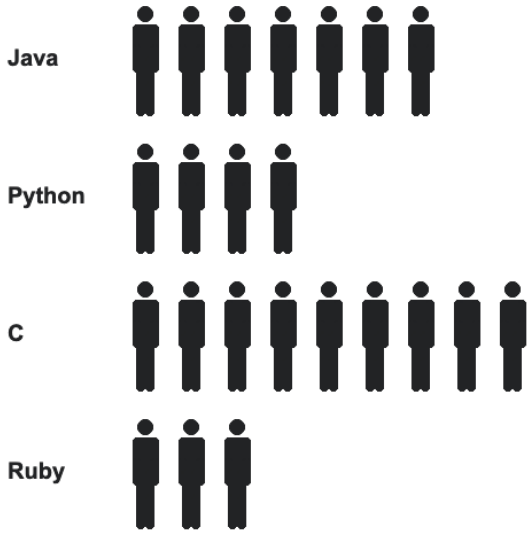


[演習課題]

身の回りにあると良いピクトグラムをデザインしましょう. 正しくメッセージが伝わるか他の人に確認してもらいましょう.

第 12 章　数量や統計的情報を瞬時に正確に伝える　-インフォグラフィック-

下の画像は, 好きなプログラミング言語のアンケートを行なった結果をまとめた例です. この画像を見るとそれぞれの言語が好きな割合がすぐに把握できます.



このように数量的なデータや統計データをグラフィック表示することでわかりやすくしたものを現代では，インフォグラフィック(infographics)といいます．時代をさかのぼれば，20 世紀初頭，哲学者でもあり社会経済学者でもあったオットー・ノイラートは社会情勢が不安定の中で, 一般人が経済データや統計データを瞬時に情報として得られる環境を提供することが重要であると考え，アイソタイプ(ISOTYPE: International System of Typographies Picture Education)を提唱しました．アイソタイプ(ISOTYPE)は，ピクトグラムの起源と言われています.

では，ピクトグラミングを使って，インフォグラフィックスを作ってみましょう.

命令が実行される時点での人型ピクトグラムを描画する **ST** 命令というのがあります. 様式と処理内容は以下のとおりです．**ST** は stamp の略で，命令実行時の人型ピクトグラムの状態をスタンプするイメージです.

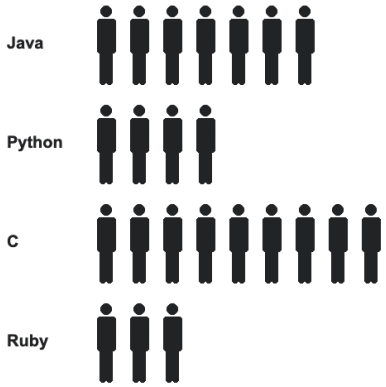
命令の様式	処理
pic.ST()	命令が実行される時点での人型ピクトグラムを描画する.

複製機能を追加しました. これにより複数の人型ピクトグラムを含むピクトグラムがより簡単に作成できるようになります. また, ピクトグラムを主要構成素とする様々なインフォグラフィックスの作成が期待できます. 例えば, 8 人の手を繋いだ人型ピクトグラムを描くプログラムの例は次のようになります.

```
pic.SC(0.3)
pic.MW(-250, 0)
pic.RW("LUA", -20)
pic.RW("RUA", 20)
for i in range(8):
    pic.MW(60, 0)
    pic.ST()
```

この例では, 全ての人型ピクトグラムは同じ姿勢ですが, 命令が実行される時点でのピクトグラムの状態がスタンプされますので, 様々な姿勢の人型ピクトグラムを混在させることもできます.

```
data = {'Java':7, 'Python':4, 'C':9, 'Ruby':3}
pic.SC(0.2)
pic.M(-260, -200)
for name, count in data.items():
    pic.T(name, 20)
    pic.MW(60,0, 0.06)
    for i in range(count):
        pic.MW(40, 0, 0.06)
        pic.ST()
    pic.M(-(40 * count + 60), 120)
pic.SC(0)
```



1 行目の表記は辞書(dict)を作成しています.

```
data = {'Java':7, 'Python':4, 'C':9, 'Ruby':3}
```

波カッコ({})の中に，キー(key)と値(value)の組合せを複数書き，カンマ(,)で繋げます． この場合は，キー Java , 値 7 となります．みなさんもデータを扱う場合は，その数値がなんの数値なのかを示した文字列とセットで扱うと思いますが，Python ではそのような場合は辞書を用いることが多いです．一般的にはこのようなデータ構造を連想配列(associative array)とか連想リスト(associative list)とかいいます．この dict をこのプログラムでは変数 data に代入しています.

2 行目の pic.SC(0.2)は人型ピクトグラムの大きさを標準の 0.2 倍にしています．SC はスケール（scale）の意味です.

あと，5 行目の記述が目新しいと思います.

```
for name, count in data.items():
```

data.items() は辞書(dict)である，data をキーと値の組合せからなるリストとタプルからなる二次元配列に変更しています.

```
(['Java', 7], ['Python', 4], ['C', 9], ['Ruby',3])
```

その上で，最初に['Java', 7]のそれぞれの要素が name, count に代入されます．つまり name には，'Java', count には 7 が代入されます．繰返すごとに順次要素が name, count にそれぞれ代入されます.

[演習課題]
適当なデータを作成して，サンプルのプログラムを改変して，人型ピクトグラムを構成素とするインフォグラフィックを作成しましょう. 正しくメッセージが伝わるか他の人に確認してもらいましょう.