max script 覚書① 基本編

maxオブジェクトノードへのアクセス

名前で指定

$オブジェクト名

例) $sphere01

カレントオブジェクト

$

現在選択されてる
オブジェクトを指定

参照オブジェクト

例)
```
obj_a = $shpere01
obj_a.pos.x=100;
```
obj_aへ参照代入
以降obj_a で shpere01を参照できる

maxオブジェクトのプロパティへのアクセス

$（オブジェクト名）. プロパティ

$sphere01.pos.x　位置のX座標へアクセス　<オブジェクト>.name　オブジェクト名

$sphere01.pos.x　位置のX座標へアクセス

$sphere01.modifiers[#Bend].BendAngle　ベンドの角度へのアクセス

キャスト

as string 文字
as integer 数値
as float 浮動小数点

例)
```
atai = 100;
messageBox("値は" + atai as string)
```

演算子

算術演算子	+ 加算	* 乗算	除算の余りは、modを利用　mod 10 3
	- 減算	/ 除算	10÷3の余り 1を返す

代入演算子	=
加算代入演算子	+=
減算代入演算子	-=

比較演算子	A == B	等しい	A >= B	AはB以上
	A != B	等しくない	A <= B	BはA以上
	A > B	AがBより大きい		
	A < B	AがBより小さい		

変数

変数は、タイプ自由変数と呼び、宣言なしに、好きな型のデータを保存できる

変数のスコープ

変数を宣言以下の方法で宣言すると、変数の有効範囲を決めることができる

global　グローバル変数として宣言
local　ローカル変数として宣言

※宣言なしの場合はローカルとして扱う

ユーザー定義関数の作成

```
Function 関数名 引数 , 引数 , ・・・・ =
(
    処理
    return 戻り値
)
```

Function は、fnと略せる

関数の呼び出し

関数名 引数 引数 ・・・・

a = 関数名 引数 引数 ・・・・

a = 関数名()

※引数をとらない関数は()をつける

例)
```
a = tasu 4 3

function tasu ia , ib =
(
  return (ia + ab)
)
```

条件分岐

if文

if 条件式 then (条件があってるとき処理) else (条件にあわないときの処理)

case文

```
case 変数 of
(
    値A:    値Aの処理
    値B:    値Bの処理
    default:  あてはまらない場合の処理
)
```

配列の作成

変数名 = #(インデックス数)

※インデックス数省略化
a = #()

配列の参照

変数名[インデックス]

※配列のインデックスは1から始る

繰り返し処理

while文

```
while <条件式> do
(
    処理
)
```

条件に一致してる間繰り返し

for文

for <変数名> = <初期値> to <終了値> do (処理)

for i = 1 to 10 do print i

exit

ループ処理を抜ける

コメント

-- 1行コメント（ハイフン2個）

オブジェクトのプロパティ、機能名 など、を調べるには?

メニューバーの、[max script] → [max scriptリスナー] を開き
リスナーのメニューから , マクロレコーダーをONにする。
以降、maxの操作が、リスナー上にでるので、それを参照する

メニューバーの [ヘルプ] → [max scriptヘルプ] を参考にする。

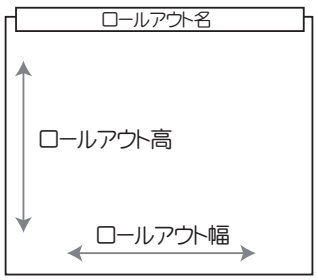
max scriptは、大文字、小文字は判別しない

max script 覚書② UIの作成

ユーティリティーパネルで作成

```
Utility ロールアウト名 "ロールアウトのタイトル" width:幅 height:高
(
    UIの作成
    イベント処理
)
```

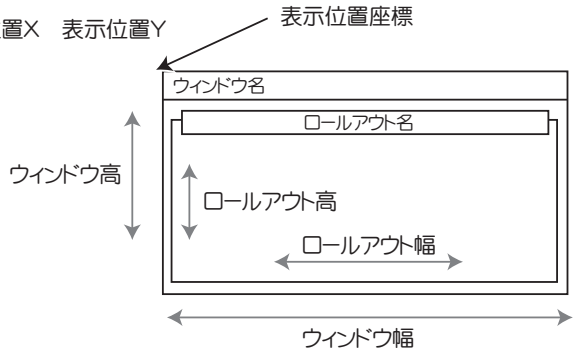
（コマンドパネル内に作成）



フローティグウィンドウで作成

```
myRollFloater = newrolloutfloater "ウィンドウ名"  ウィンドウ幅  ウィンドウ高  表示位置X  表示位置Y

rollout ロールアウト名 "ロールアウトのタイトル" width:幅 height:高
(
    UIの作成
    イベント処理
)
addRollout ロールアウト名 myRollFloater rolledUp:false
```



UIの作成

グループ

```
group "グループ名"
(
    UI
)
```

```
group "サンプル"
(

)
```

ボタン

button <名前>

例）

```
button btn1 " 押して " pos:[50 , 100] width:200 height:50 enabled:true
```

ラベル

label <名前>

テキストボックス

edittext <名前>

スライダー

slider <名前>

スピナー

spinner <名前>

プログレスバー

progressBar <名前>

チェックボックス

checkbox <名前>

ピックオブジェクト

pickButton <名前>

共通プロパティ

```
"ラベル" pos:[x位置,y位置] width:幅 height:高 enabled:true
```

UIの持つプロパティへのアクセス

UI名 . プロパティ

.caption　UIのラベル文字
.text　ラベルなどの文字を表示するUIにある設定値
.value　スピナーなど数値のあるUIの設定値
.enabled　UIの利用可能、不可能を設定　false or true

例）

```
lablel1.caption = "ラベルだよー"
lablel1.text = "ラベルだよー"
Btn1.enabled = false
Spn1.value = 30
```

イベント

on <UI名> changed <変数> (　処理　) UIが変更されたら　変数には、変更された値が入る

on <UI名> pressed do (　処理　)　UIが押されたら

on <ロールアウト名> open do (　処理　)　ロールアウトが実行されたとき

UI ボタンやチェックボックスのプロパティの種類を知るには?

Visual Max Script内で、プロパティの一覧が見れる

max scriptの作成、編集、実行

maxのユーティリティーパネルの
max scriptを実行し、パネルから操作。
また、スクリプトはテキストなので、メモ帳でもOK
※拡張子は、ms

VisualMaxScriptの起動方法

①maxのユーティリティーパネルから、その他
　visual maxscriptを選択
②スクリプトの編集ウィンドウで[編集]→[ロールアウトの編集]

max script 覚書③ その他

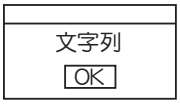
ランダム

random <最小値> <最大値>

例）
```
rn = random 10 100          変数rn には 10から100のランダム値が入る
```

メッセージボックス

messageBox ("文字列")



タイムスライダ

sliderTime　現在のスライダー位置

例）
```
now_slider = sliderTime          変数へ現在のスライダー位置を代入
sliderTime += 1                  タイムスライダーを1フレーム移動
```

at time <フレーム> (プロパティ)
指定したフレームの時のプロパティを返す

例）
```
x = at time 5( $sphere01.pos.x )
        5フレームの時のsphere01のx座標を変数xへ代入
x = at time (sliderTime - 5)( $sphere01.pos.x )
        5フレーム前のsphere01のx座標を代入
```

マテリアルエディターへのアクセス

meditMaterials[スロット番号]
スロット番号のマテリアルへアクセス

getMeditMaterial <スロット番号>(1-24)
スロット番号のマテリアルを取得

例）
```
meditMaterial[1].diffuse = red          スロット1の拡散反射光を赤
mt = getMeditMaterial 2                 スロット2の参照をmtへ代入
mt.diffuse = blue                       mtの拡散を青
mt.specular = color 255 0 255           mtの鏡面をR255 G0 B255に
$sphere01.material = mt                 sphere01 に mtを割り当て
```

マテリアルのプロパティ

.diffuse　拡散反射　　.specular　鏡面反射

マテリアルの適用

<オブジェクト>.material = マテリアル

カラー指定

col = color <R値> <G値> <B値>　　カラークラスを利用して作成

col = [R値,G値,B値] as color　point3をカラーへ変換

col = red　　カラーリテラルを利用

red,green ,blue ,white ,black ,orange
yellow ,brown ,gray

レンダリング

render　<引数> <引数>

パラメーター
- camera: <カメラ>
- frame: <フレーム番号>
- fromframe: <開始フレーム>
- toframe: <終了フレーム>
- outputwidth: <レンダリング幅>
- outputheight: <レンダリング高>
- outputFile: <ファイル名>

例）
```
render camera:$camera01 fromframe: 0 toframe:10 outputFile: test.tif
```

保存の画像形式は拡張子で決まる

静止画　.tif　.tga　.jpg など
動画　　.avi　.mov

※max script各フォーマットの内部設定にアクセスする手段を持たないので
AVIなどの圧縮設定は前回に設定した値が利用される

※引数なしの場合はrender()
指定されていないパラメーターは前回のレンダリング時の設定を引き継ぐ

Bitmapクラス

画像イメージを利用する場合は,Bitmapクラスを使う

コンストラクタ

Bitmap <幅> <高さ> filename:<ファイル> color:<色>

ファイルがあるときは、読み込む
ない場合でも、saveした時の保存ファイル名になる

例）
```
bm1 = Bitmap 100 50 color:blue
        bm1ビットマップobjの作成、初期カラー青,サイズ 100 x 50

bm2 = Bitmap 200 150 filename:"d:¥¥test.jpg"
        bm2ビットマップobjの作成 サイズ200x150でtest.jpgを読み込み


btm = Bitmap 100 80  filename:test_s.jpg
rn = render outputwidth:640 outputheight:480 outputFile:test.jpg
copy rn btm
save btm                test.jpgで640x480の画像を出力し画像をrnに代入
close btm               btmへコピーする、コピー時に自動でサイズを合わせる
                        btmを出力、縮小された画像が test_s.jpgとして保存される
```

ファイル処理

maxファイルのロード

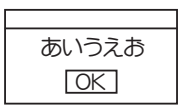
loadMaxFile <ファイル名>

テキストの読み書き　FileStreamクラスの利用

createFile <ファイル名>	新規ファイルの作成
openFile <ファイル名>	ファイルの読み込み
close <FileStream>	ファイルのクローズ
print 文字列 to:<FileStream>	一行書き出し
format フォーマット文字列 to:<FileStream>	フォーマット書き出し
readline <FileStream>	一行読み込み、&行移動
readChar <FileStream>	一文字読み込み、&行移動

例）
```
my_file = createFile "d:¥¥test.txt"
print "あいうえお" to: my_file      ←― 書き込み
close my_file

my_file = openFile "d:¥test.txt"
ch = readline my_file
messageBox ch                ←― 読み込み
close my_file
```



※フォルダ階層の区切りは、¥¥　　例）d:¥¥image¥¥test.jpg

ファイルを使わない文字情報は、StringStreamクラスの利用する。