**－プログラミング（４）－**

１.目的

ProcessingのPythonモードによるプログラミングについて学ぶ。

２.Processingの統合開発環境のモード機能

Processingの統合開発環境はJavaシステムを基盤にして構成されている。ProcessingのプログラムはJava言語を簡素化した形式で記述され、実行コードはJava仮想マシン上で実行され、Javaのクラスライブラリを利用することができる。

Processingの統合開発環境のモード機能は、java以外のプログラム開発環境を提供できる機能で、Pythonモードやp5.js（JavaScript）モードがある。

２.１　統合開発環境のモード機能と設定

モード機能は、モードの追加変更ボタンをクリックすることで設定できる。モードの追加変更の操作の例を図１に示す。モード機能でサポートされているプログラム開発環境を次に示す。

Java Mode：　Processingにおける標準（Java）プログラム開発モード

Android Mode：　スマートフォンなどのAndroid（Java）用プログラム開発モード

p5.js Mode：　p5.js（JavaScript）用プログラム開発モード

Python Mode：　Python用プログラム開発モード

R for Processing：　R言語（統計解析）用プログラム開発モード

REPL Mode：　対話インタプリタ（コンソール・コマンド）用プログラム開発モード

Shader Mode: GLSL（OpenGL Shading Language）用プログラム開発モード

モード機能を変更してPythonやp5.js（JavaScript）でプログラム開発を行う場合は、モードの追加変更ボタンをクリックし、Pythonモードやp5.js（JavaScript）モードに変更する。これにより、Processingの統合開発環境を使用して、Pythonやp5.js（JavaScript）によるプログラム開発ができる。

モード機能の「モードの追加」を選択すると、「Contribution Manager」によるモードインストール画面が表れ、モード一覧から追加するモードを選択できる。Installボタンをクリックすることで新しいモードをインストールできる。

　「ファイル」メニューの「サンプル」から読み込むことができる拡張ライブラリのサンプルプログラムは、各モードに対応したサンプルプログラムとなっている。

２.２　Pythonモードでのプログラミング

　Pythonモードでは、Python 2.7がサポートされている。PythonはWebサービスやAI開発などで広く使用されているスクリプト言語である。Pythonモードによるプログラム例をリスト１に示す。

**def setup( ): # 初期設定のsetup( )関数**

**size(400,300) # 実行画面のサイズ**

**background(255,255,255) # 背景を白色に**

**fill(0,32) # 図形を半透明に塗りつぶし**

**def draw( ): # 繰り返し描画のdraw( )関数**

**if mousePressed: # マウスボタンが押された場合**

**ellipse(mouseX,mouseY,10,10) # 10×10で〇を描く**

　　　　　　　　　　　　リスト１　sample401.pyde

　リスト1はマウスの左ボタンが押された時にマウスカーソルの位置に〇を描くプログラムである。

プログラムにおいて、#以下はコメント文を表す。def setup( )は初期設定の関数、def draw( )は繰返し描画の関数を表す。関数名の後にはコロン「：」の記号が」付けられ、その後は字下げ（半角スペースまたはTab）して内側のブロック部分の文であることを示す。JavaやCでは{ }を使用するが、pythonでは字下げで対応する。なお、字下げの位置が揃っていないとエラーになるので、きちんと揃えること。

if mousePressed: は、マウスボタンが押されたかどうかを判定する条件文である。ifの後にはスペースを挟んで条件式を記述し（条件式に( )は必要ない）、その後にコロン「：」の記号を付ける。それ以降は内側のブロックであることを明示するために字下げを行って記述する必要がある。else if文、else文も同様の記述で内側のブロックを明示する。リスト２にif～else文の書式例を示す。

　繰り返し処理を記述するfor文およびwhile文も条件式の書き方、内側のブロックの字下げのルールは同じである。リスト３にfor文およびwhile文の書式を示す。

**if a < 10 :　　　　　　　　　　　　# if ～ else　文**

**elif a < 20: # else if 文**

**else: # else 文**

リスト２　if～else文

**for i in range(10): # for 文　range(10 )は10までの範囲を示す**

**# （ i = 0,1,2,…,9の範囲）**

**while i < 100:　　　　　　 　 　# while文**

リスト３　for文およびwhile文

　変数の型は宣言しない。文の中に変数名を記述すれば、Pythonのインタープリタが自動的に判定して変数を処理する。

　Pythonモードの命令一覧はProcessingのメニューの「Help」から「References」で参照できる。基本命令および書式はPython言語に対応し、さらにProcessingで使用できる描画命令がサポートされている。

２.３　アニメーション表示

　　ウィンドウの中を〇の図形直線的に移動し、ウィンドウの枠に当たると跳ね返るプログラムをリスト４に示す。変数x,y,dx,dyはグローバル変数であるため、関数のブロック内の最初でglobalの宣言を行う。

**x=0**

**y=0**

**dx=2.0**

**dy=2.0**

**x1=0**

**y1=0**

**def setup( ):**

**global x, y**

**size(640, 480);**

**background(64, 64, 128)**

**stroke(128, 128, 255)**

**fill(255, 200) 　　　　　　　#/図形の色を半透明で白く塗りつぶし**

**x = random(20, width-20) #xを20～ width-20の範囲でランダムに設定**

**y = random(20, height-20) #yを20～ height-20の範囲でランダムに設定**

**def draw( ):**

**global x, y, dx, dy**

**background(64,64,128)**

**x+=dx #xをdxだけ加算**

**y+=dy　　　　　　　　　　 #yをdyだけ加算**

**if x<10 or x>width–10: #ウィンドウの枠（横方向）を判定**

**dx = -dx**

**if y<10 or y>height–10: #ウィンドウの枠（縦方向）を判定**

**dy = -dy**

**ellipse(x, y, 20, 20)**

リスト４　sample402.pyde

２.４　Pyhtonモードでの画像表示

　　　画像を読み込むには、loadImage( )関数を使用する。jpg、pngなどの画像ファイルを読み込むことができる。画像に表示はimage( )関数を使用し、javaモードとほぼ同じである。なお、画像の大きさに合わせて、size( )関数の大きさを設定しておく必要がある。

　　　リスト５に画像ファイルを読み込み表示するsample403.pydeを示す。

def setup():

global img1　　　　　　　　　　# グローバル変数の宣言

img1=loadImage("image001.png")

size(480,360) # 画像ファイルの大きさ

def draw():

global img1

image(img1, 0, 0, img1.width, img1.height)

　　　　　　　　　　　　　　　　 リスト５　sample403.pyde

３．フラクタル図形

　３.１　木のフラクタル図形

　再帰処理によりフラクタル図形を描くことができる。ここでは木のフラクタル図形を描く方法を確認する。

　下から伸びた枝が分岐点において一定角度で左右に分かれる。次の枝は最初の枝より一定の割合で短くし、これを再帰処理で繰り返すことで木の形状の図形を描くことができる。

再帰処理はある関数の中で自分自身の関数を呼び出す処理であり、これにより同様の処理を繰り返すことが簡潔に記述できる。

リスト６のsample404.pydeは再帰処理により木のフラクタル図形を描くプログラムである。このプログラムの中で、draw( ) 関数のa=60　は左右の枝分かれの角度を決めている。さらにbranch( ) 関数のh \*=0.66は再帰呼び出し時の枝の長さの割合を決めている。

木のフラクタル図形の描画手順を下記の(1)～(3)に示す。

1. 直線を描く
2. 直線の先端で、元の直線より一定の割合で短くした次の直線を、左右に一定角度で開いて描く。
3. 上記(2)の処理を再帰的に繰り返し、直線の長さが一定の値より短くなったら終了する。

左右の枝分かれの角度a および再帰呼び出し時の枝の長さの割合h を変えることにより、いろいろな木のフラクタル図形を描くことができる。

**def setup():**

**size(640, 360)**

**def draw():**

**background(255)**

**stroke(0)**

**a=60 #** 枝分かれの角度

**translate(width / 2, height)**

**line(0, 0, 0, -120)**

**translate(0, -120)**

**branch(120, radians(a))**

**def branch(h, theta):**

**h \*= 0.66 #** 再帰呼び出し時の枝の長さの割合

**if h > 2:**

**pushMatrix()**

**rotate(theta)**

**line(0, 0, 0, -h)**

**translate(0, -h)**

**branch(h, theta)**

**popMatrix()**

**with pushMatrix():**

**rotate(-theta)**

**line(0, 0, 0, -h)**

**translate(0, -h)**

**branch(h, theta)**

リスト６　sample404.pyde

４．演習課題・検討課題

４.１　演習課題１

**［課題１-１］**

リスト４のsample402.pydeを改良して、長方形のラケットでボールを打ち返すプログラムkadai4101.pydeをpythonモードで作成せよ。長方形のラケットは次のように設定する。

**rect(x1,y1,100,10);**

カーソルキーでラケットを移動するには、keyPressed( )の関数を使用する。Pythonモードでは下記のような記述となる。

**def keyPressed( ):**

**if key==CODED:**

**if keycode==RIGHT:**

**#ここでラケットの座標を右へ移動**

**elif keyCode==LIGHT:**

**#ここでラケットの座標を左へ移動**

ソースプログラムを印刷して報告書に添付すること。さらに実行結果の画面をキャプチャして報告書に貼り付けること。

４.２　演習課題２

**［課題２-１］**

リスト５のsample403.pydeを参考にして、元画像の横に1/2スケールの画像を並べて表示するkadai4201.pydeのプログラムを作成せよ。画像ファイルにはimage001.pngを使用する。

ソースプログラムを印刷して報告書に添付すること。さらに実行結果の画面をキャプチャして報告書に貼り付けること。

**［課題２-２］**

マウスの右ボタンが押されたとき、画像データimage001.pngをフルカラー表示し、「 b 」のkeyを押すと画像が２値画像に変化し、「 g 」のkeyを押すと画像がグレイスケール画像に変化し、「 i 」のkeyを押すと反転画像（ネガ画像）に変化し、「 c 」のkeyを押すと全体が消去されるプログラムkadai2202.pydeを作成せよ。画像の返還にはfilter( )を使用すること。実行ウィンドウの下側に操作キーの説明を表示すること。

ソースプログラムを報告書に添付すること。さらに実行結果のそれぞれの画面をキャプチャーして報告書に貼り付けること。

４.３　演習課題３

**［課題３-１］**

リスト６のsample404.pydeを参考にして、枝分かれの角度aを0～90までキー操作で変更できるようにし、木のフラクタル図形の変化を確認できるプログラムを作成せよ。角度aは初期値を60とし、上向きカーソルキーまたは下向きカーソルキーを押すことで、１ずつ増減できるようにすること。

ソースプログラムを印刷して報告書に添付すること。さらに実行結果の画面をキャプチャして報告書に貼り付けること。

４.４　検討課題

**［検討１］**

リスト６における制御文の「with」（下から6行目）について、どのような制御を行っているのか調査し説明せよ。

**［検討２］**

　　　フラクタル図形としては他にどのようなものがあるか調査し、事例を上げて説明せよ。