# デジタルメディア処理1

担当: 井尻 敬

## ディジタルメディア処理: python入門

ディジタルメディア処理のプログラミング演習部分の補足資料です．既にpythonに関するプログラミング経験のある方は読む必要が有りません．

※ 本講義で取り扱うのはあくまでほんの触りの部分だけです．もし興味が湧いた方は，デジタルメディア処理2や3年後期の高度情報処理演習を履修するか，独学で学修を進めてください．

※本講義では，コードを書きながらPythonの表面的な使い方を体験します．網羅的な機能・文法の紹介は行ないません．

# 自分のPCでpythonを動かしたい人向けメモ

## Anacondaをインストールする

- 本家ページより，インストーラーをダウンロード
  - https://www.continuum.io/downloads
  - 今回はPython3, 64bit installerを選択
  - ファイルサイズが大きいので多少時間がかかる

  ※2017/3/9現在，Anaconda4.3.0が最新だが，このバージョンはOpenCVがうまくインストールできない

  ※ https://repo.continuum.io/archive/index.html ←こちらから「 Anaconda3-4.2.0-Windows-x86_64.exe 」を利用するとうまくいく．

- 『Anaconda3-4.2.0-Windows-x86_64.exe』を起動しインストール
  - コマンドプロンプトで > python –versionとして以下の感じなら成功

```
C:¥Users¥takashi>python --version
Python 3.5.2 :: Anaconda 4.2.0 (64-bit)

C:¥Users¥takashi>
```

# OpenCVをインストール

1. コマンドプロンプトを右クリックして管理者権限で起動
2. > conda install –c menpo opencv3
3. 途中でyキーを押す
4. 5分くらい待つ
5. > python sample.py

```
1 sample.py
2 img = cv2.imread('sample.jpg')
3 print( img.shape )
4 cv2.imshow('sample', img)
5 cv2.waitKey(0)
```

引くほど簡単にインストールできた
本当はもっと大変な思いをすると思ったからポイントをメモしておくためにこの文章を書き始めたのに。。。

# エディタ

- エディタはなんでも良いと思います（Atom/Vim/Emacs/limetext/xzy）
- 私は手元にあったのでVisual Studio 2015 communityを利用

- Python environmentをインストール（20分位かかった）
- 右側にあるPython Environmentにおいて

『+ Custom』をクリックし新しいenvironmentを生成

Confgureタブより以下を指定

| | |
|---|---|
| Prefix path | : **Program Files(x86)/Anaconda2** |
| Interpreter Path | : **Program Files(x86)/Anaconda2/python.exe** |
| Windowed Interpreter | : **Program Files(x86)/Anaconda2/pythonw.exe** |
| Library path | : **Program Files(x86)/Anaconda2/python.exe** |

Intelli senseタブよりrefreshする (時間かかる)

- これでインテリセンスが効くようになる

## その他

python環境について知っておいてほしいこと

- 『Anaconda 仮想環境構築』で検索
- 『Anaconda condaコマンド』で検索
- 
- 『mutableとimutable』については話すとむしろ混乱させてしまいそうなので難しいところ

# 初めてのPython

※デジタルメディア処理1にて解説したのでスキップします
※デジタルメディア処理1を受けていない方は独習してください
※変数の取り扱い・スライスのみ，大まかに話します

# Python

- 最近流行りのスクリプト言語
- 機械学習関連のライブラリが充実
- 画像処理関連のライブラリも充実（OpenCV）
- 開発コストが低い（井尻が普段利用しているC++に比べて）

- コードの可読性が高い
- インデントでブロックを強制
  → 変なコードが生成されにくく，学生のコードを読む側としてはとてもありがたい.

# Ex1.py “Hello world”

**実習: 右のコードを動かしてください**

1. “ex1.py”というファイルを作成
2. 右のコードを記入
3. コマンドラインで以下を入力
   $ python ex1.py

```
# -*- coding: utf-8 -*-
# ex1.py

print("hello, world")
```

※ 『#』でコメントアウト
※ 『print(“文字列”)』で文字列を出力
※ 『# -*- coding: utf-8 -*-』は文字コード指定．コード内で日本語を利用可能に.

# Ex2.py 変数の型

- **int, float, string, boolなどの型を利用可能**
- **変数の型は代入する値に応じて自動で決まる**
  （型を明示した変数宣言は行わない）
- **後から異なる型に変更することも可能**
  （その都度新しい変数が生成される）

**実習：右のコードを動かしてみてください**

**実習：右のコードを色々と編集し型の挙動を確認してください**

※ type ( 変数名 )：型を取得

※ id (変数名 )　　：オブジェクトidを取得 (idを見ると，数値代入のたびに新たなオブジェクトが生成されているのが分かる)

※ print (変数1, 変数2) で複数変数を出力可能

※ 型変換も可能

```
# -*- coding: utf-8 -*-
# ex2.py

#int
a = 1234
print(a, type(a),id(a))

#float
a = 1.234
print(a, type(a),id(a))
a = 1.2345
print(a, type(a),id(a))
a = 1
print(a, type(a),id(a))

#bool
a = True
print(a, type(a),id(a))

#string
a = "hello, world"
print(a, type(a),id(a))

#型変換例: float->int, string->int, int->float
a = int(16.2)
a = int('16')
a = float(16)
```

# Ex3.py 配列 (1)

Pythonでは, tuple・list・np.array という配列表現が利用可能

| | | |
|---|---|---|
| (1,2,3) | tuple | ：**長さ&値 変更不可**の配列 |
| [1,2,3] | list | ：可変長配列 |
| np.array(1,2,3) | np.array: | *n*次元配列（画像などはこれで表現される） |

- np.arrayは高速処理のための制約がある配列
  - np.array では要素がメモリ内の連続領域に配置される
  - np.array では各次元の要素数は等しい（行列の形になる）
  - np.array では原則的に要素は同じ型
  - 参考：http://www.kamishima.net/mlmpyja/nbayes1/ndarray.html

# Ex3.py 配列 (2)

**実習：右のコードの出力を予想してください**

output1
output2
output3
outout4

**実習 :コードを実行し，予想と比べてください**

**実習 :コードの中身を色々変化させ，配列とタプルの挙動を確認してください**

※ 配列は[], tuppleは()で表現される
※ 配列要素の変更・追加・削除ができる
※ len(配列名)で長さを取得できる
※ 2次元配列（行列）も表現可能

```
# -*- coding: utf-8 -*-
# ex3.py

#1D array
A  = [1,3,5,7]
print("output1:", A )

N  = len(A)  #要素数
a2 = A[2]    #2番目の要素を参照
A.append(4)  #後ろに"4"を挿入
a = A.pop(2) #2番目の要素をpop
A.remove(4)  #値が4の最初の要素を削除
print( "output2", N, a, a2, A )

#2D array
A = [[1,2],[3,4],[5,6]]
print( "output3", A[0][1], A, len(A))

#tupple
T = (1,2,3)
# T[1] = 2  #error tappleは変更不可
print("output4",T, T[1])
```

# Ex4.py np.array (1)

- np.arrayを含むコードを右に示す

**実習：右のコードにprint文を挿入し，計算結果を確認してください**

**実習：コードの中身を色々変化させ，np.array の挙動を確認してください**

※『import numpy as np』はnumpy関連のモジュールのインポート文

※ python & openCV環境では，np.arrayで画像を表現

※ 要素ごとの演算が一行で書ける（画像と画像の和など）

※ **np配列名.shape**で配列サイズを取得

```
# -*- coding: utf-8 -*-
# ex4.py
import numpy as np

A=np.array([5,6,7,8])
B=np.array([1,2,3,4])
print(A.shape, A)
print(B.shape, B)

#要素ごとの演算(和差積商余べき)
C = A + B
D = A - B
E = A * B
F = A / B
G = A % B
H = A ** B
I = A - 1

#2D
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[1,2,3],[4,5,6]])
C=A+B
print(C,C.shape)
```

# Ex5.py np.array (2)

- np.arrayには便利な初期化方法が用意されている
- 要素の総和・平均・分散の計算など，多様な便利機能が用意されている

**実習：右のコードを実行し結果を確認してください**

**実習：最後のprint文では，配列の分散が計算されていることを確認してください**

```
#Np.array 初期化
A = np.array([1,2,3,4]) #listで初期化
B = np.array((1,2,3,4)) #tupleで初期化
C = np.zeros(3)         #要素数指定，要素は0
D = np.ones(3)          #要素数指定，要素は1
E = np.ones((2,3))      #[[1,1,1][1,1,1]]
F = np.zeros_like(E)    #Eと同じサイズの0配列
F = np.identity(3)      #3x3 単位行列
```

```
# -*- coding: utf-8 -*-
# ex5.py
import numpy as np

A    = np.array([1,2,3,4,5,6,7,8])
mean = np.mean( A )
sum  = np.sum ( A )
vari = np.var ( A )
print( mean, sum, vari)

A = A-mean # 全要素からmeanを引く
A = A**2   # 全要素を二乗
print( np.sum(A)/A.shape[0]);
```

# Ex6.py　for文

**実習：コードを実行し結果を確認してください**

**実習：右のコードを少し変更し，for分を使ってAの分散を計算してください**

※インデントによりブロックを定義する　（Cでは{}でブロックを定義した）

**※インデントは半角スペース4個を推奨**

※ブロック開始部分に 『:』が必要

※ 『for p in A :』でAのすべての要素に順にアクセスできる

※ 『for i in range(a, b) :』で i = a~b-1をループできる

※ for文はスコープを作らないのでfor文内で生成された変数をfor文の外でも参照できる

```
# -*- coding: utf-8 -*-
# ex6.py
import numpy as np

A = np.array([1,2,3,4,5,6,7,8])

#Aの全要素をまわる
sum = 0
for p in A :
    print(p)
    sum += p

print( sum / A.shape[0])

#添え字を利用し要素を参照する
sum = 0
N   = A.shape[0]
for i in range(N):
    print( A[i] )
    sum += A[i]

print(sum / A.shape[0])
```

# Ex7.py　if文

- if 文は右のコードの通り定義できる
- else if () は elif() : と書く
- for文と同様にインデントでブロックを定義する

**実習：右のコードを実行し挙動を確認してください**

※『AかつB』　→ if( 条件A and 条件B) :

※『AまたはB』→ if( 条件A or 条件B) :

※ if文はスコープを作らないのでif文内で生成された変数を外から参照できる

```
# -*- coding: utf-8 -*-
# ex7.py
import numpy as np

A = [1, 2, 4, 2, 1, 1, 3, 4]

for p in A :
    if( p == 1 ) :
        print( "a" )
    elif( p == 2 ) :
        print( "b" )
    else :
        print ("c")
```

# Ex8.py　関数

**実習：コードを実行してください.**

**実習：aとbの積も返すよう関数を修正してください**

**※複数の引数を受け取れる**

**※引数は参照渡し**

**※ただし, 組み込み型**int, float, bool, str, tuple, unicode**は、値渡しのように振舞う**

ある引数aがあるとき，aに代入をしない場合はaへの参照が保持される．関数内でaへの代入が起こると，その瞬間に新たに変数aが生成される．そのため，関数外部からみると引数変数の変化は起きないため，値渡しのように見える．

**※複数の値を返せる**

※関数はスコープを作る：関数内部で定義した変数は外に漏れない

※関数はスコープの外の変数を参照できる，
（ただし関数内部で代入をすると，その変数は関数のローカル変数に）

```
# -*- coding: utf-8 -*-
# ex8.py
import numpy as np

def func(a,b) :
    print("a is ", a)
    print("b is ", b)
    print("c is ", c) #外のcを参照できる
    wa = a+b
    sa = a-b
    return wa, sa

a = 1
b = 2
c = 5
sum, sub = func(a,b)

print( a,b, sum, sub)
```

# main 関数

Pythonでは，スクリプトが.pyファイルの上から順に実行される

ある.pyファイルを，他の.pyファイルから呼び出す(importする)時にスクリプトが実行されてしまうと困ることがある

スクリプト部分を『if __name__ == '__main__' :』に入れると，外からimportされた時には実行されない

```
# -*- coding: utf-8 -*-
# ex8.py
import numpy as np

def func(a,b) :
    print("a is ", a)
    print("b is ", b)
    wa = a+b
    sa = a-b
    return wa, sa


sum, sub = func(1,2)
print( a,b, sum, sub)
```

```
# -*- coding: utf-8 -*-
# ex8.py
import numpy as np

def func(a,b) :
    print("a is ", a)
    print("b is ", b)
    wa = a+b
    sa = a-b
    return wa, sa


if __name__ == '__main__':
    sum, sub = func(1,2)
    print( a,b, sum, sub)
```

# スライス

**スライスとは，配列のある部分にアクセスできる表現です．便利なので使って慣れてください．**

**右はサンプルコードです**

※配列 a=[1,2,3,4,5,6,7] の2番目の要素から5番目の要素までの連続部分を取り出したい場合には

a[2:6]

とします．この a[2:6] には値の代入も可能です．

※ 2次元配列 imgに対して，

img[10:20, 30:40]

とすると y=10~19, x=30~39 の矩形画像にアクセスできます

※ img[10:20, 30:40] = 10 とすると矩形領域を値10で埋められます

```
# -*- coding: utf-8 -*-
# ex13slice.py
import numpy as np
import cv2

#1D
a1 = np.array([0,1,2,3,4,5,6,7,8,9])
a2 = np.array([2,2,2,2,2,2,2,2,2,2])

#2番目から5番目の要素を取り出す
print(a1[2:6])

#2番目から5番目に代入にする
a1[2:6] = a2[2:6]
print(a1[2:6])

#2D
#100x100の配列を作製
img1 = np.zeros( (100,100), np.uint8 )

#x=10～50, y=20～30の部分を取り出す
img2 = img1[20:31,10:51]
print(img2.shape)

#x=10～50, y=20～30の部分を白く塗る
img1[20:31,10:51] = 255

cv2.imshow("image", np.uint8( img1) )
cv2.waitKey()
```

# Python+OpenCVを利用した画像処理

# PythonとOpenCVを利用した画像処理

- OpenCVとは
  - Open sourceの画像処理ライブラリ群
  - 多様な画像処理ツールを提供する
  - BSDライセンス：『「無保証」であることの明記と著作権およびライセンス条文自身の表示を再頒布の条件とするライセンス規定』(https://ja.wikipedia.org/wiki/BSDライセンス より)
- C++, Python, java, unityなどから利用可能

以降のコードは学内環境にて動作することを確認しています
もし途中で落ちる場合は画像データの読み込みに失敗している場合があります
ファイル名や配置するフォルダを確認してください

# Ex10.py 画像の入出力

**実習:**

- **適当な画像を準備し，名前を「img.png」としてコードと同一フォルダに配置してください**
- **コードを実行し画像が表示/保存されることを確認してください**

※**cv2.imread**(“fname”)で画像読み込み
※**cv2.imshow**(“caption”, img)で画像表示
※**cv2.imwrite**(“fname”, img)で画像書き出し

※画像は np.array 形式で表現されます
img.shape：画像サイズ
img.dtype：画像データの型

```
# -*- coding: utf-8 -*-
# ex10.py
import numpy as np
import cv2

#load image
img = cv2.imread("img.png")
print( img.shape, type(img), img.dtype )

#display img
cv2.imshow("show image", img)
cv2.waitKey()

#save image
cv2.imwrite("img_save.png", img);
```

# Ex11.py 画像に図形を書き込む

**実習:**

- **コードを実行し画像に図形が書き込まれることを確認してください**
- **注意)Img.pngが小さいとうまく書かれない**

※手軽に画像への書き込みが行なえます
cv2.line　　(画像, 点1, 点2, 色, 太さ)
cv2.rectangle(画像, 点1, 点2, 色, 太さ)
cv2.circle　　(画像, 中心, 半径, 色, 太さ)

※その他の図形描画関数は以下を参照
http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

```
# -*- coding: utf-8 -*-
# ex10.py
import numpy as np
import cv2

#load image
img = cv2.imread("img.png")
print( img.shape, type(img), img.dtype )

#draw rect and dots
cv2.line      (img,(100,100),(300,200), (0,255,255),2)
cv2.circle    (img,(100,100), 50,       (255,255,0),1)
cv2.rectangle(img,(100,100),(200,200), (255,0,255),1)

#display img
cv2.imshow("show image", img)
cv2.waitKey()

#save image
cv2.imwrite("img_save.png", img);
```

# Ex12.py 画素へのアクセス

**実習：右のコードの一部を編集し画像をグレースケール化してください**

- グレースケール値は，r g bの平均とする
  I = (r+g+b)/3
- 画像の(y,x)画素の(r,g,b)値は
  r = img[y,x,2]
  g = img[y,x,1]
  b = img[y,x,0]

※途中計算時のオーバフローを避けるため，画像imgとimg_grayは，float型に変換されており，可視化時にuint8型に変換されている．

img = np.float64(img) #float64に変換

img = np.uing8 (img) #uint8に変換

```
# -*- coding: utf-8 -*-
# ex12.py
import numpy as np
import cv2

#load image
img = cv2.imread("img.png")
img = np.float64( img ) #要素をfloat型に
H   = img.shape[0]
W   = img.shape[1]
img_gry = np.zeros( (H,W), float )

print(img.dtype, img_gry.dtype)

for y in range(H) :
    for x in range(W) :
        img_gry[y,x] = img[y,x,0] #ここを編集

#display img
cv2.imshow("color image", np.uint8( img) )
cv2.imshow("gray  image", np.uint8( img_gry) )
cv2.waitKey()
```

# Ex13.py 画像の作成

- **実習：右のコードを実行し，縦じま画像が現れることを確認してください**
- **実習：縦じまが横じまになるようにコードを修正してください**

※グレースケール画像は2次元行列となります

※ img[10:20, 30:40] とすると

y = 10~19, x=30~39

の矩形画像にアクセスできます

※ img[10:20, 30:40] = 10 とすると矩形領域を値10で埋められます

```
# -*- coding: utf-8 -*-
# ex13.py
import numpy as np
import cv2

#サイズ100x100の画像を作成
H   = 200
W   = 200
img = np.zeros( (H,W), np.uint8 )

#画素値代入(縦じま)
for y in range(H) :
    for x in range(W) :
        if( x % 2==0) :
            img[y,x] = x
        else :
            img[y,x] = 255

#矩形領域に一度で代入も可能
img[50:60, 50:70] = 255

#display img
cv2.imshow("image", np.uint8( img) )
cv2.waitKey()
```

# Ex14.py 平滑化フィルタ

- **実習：右のコードの一部を編集し,平滑化フィルタを完成させよ.**
- **ただし，注目画素を中心とする9画素の平均を注目画素に格納するものとする**

```
# -*- coding: utf-8 -*-
# ex14.py
import numpy as np
import cv2

#load image, convert to grayscale float
img = cv2.imread("img.png")
img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
img = np.float64(img)

img_out = np.zeros_like( img )

for y in range( 1, img.shape[0]-1 ) :
    for x in range( 1, img.shape[1]-1 ) :
        img_out[y,x] = 128 # ここを編集
        #ヒント: img[y-1,x-1] で左上画素にアクセス

cv2.imshow( "output", np.uint8( img_out ) )
cv2.waitKey()
```

# まとめ

- 画像処理プログラミングの雰囲気を味合うために，Python & OpenCV環境で，初歩的なコードを書いてみた.
- このPython & OpenCV環境は，比較的手軽にプロトタイピングが行えるので，興味がある学生は是非学修を進めてほしい