

# デジタルメディア処理

担当: 井尻 敬

## Python による画像処理 Quick Start (自習用)

### 達成目標

- Python+OpenCV環境における簡単なプログラムを作成できる
- 本講義にて解説した画像処理手法をプログラムとして記述できる

注：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。網羅的な機能・文法の紹介は行ないません。もし興味が湧いた方は、独自に学修を進めてください。

# Python

- 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（と言われている）
  
- 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。
  - 一方、インデントの変化が挙動の変化を引き起こすので思わぬバグがおきることも

## 準備

- Python 3のインストールされたマシンを用意する
  - 自分のPCにpythonをインストール → 環境構築資料参照
  - 学情のPCにpythonをインストール → 環境構築資料参照
  
- 作業ディレクトリを用意してください
  - どこでもよいです
  
- サンプルコードを講義web pageよりダウンロードしてください
  - この資料を写経してもよいのですが面倒だと思うので用意しておきました

# Ex1.py “Hello world”

**実習: pythonで書かれた右のコードを動かしてください**

0. 作業用ディレクトリを作成
  1. “ex1.py”というファイルを作成し作業ディレクトリに配置
  2. “ex1.py”に右のコードを記入
  3. コマンドプロンプトを起動し, 作業ディレクトリへ移動
- ※次ページ参照
4. コマンドプロンプトにおいて, 以下のコマンドを入力

```
> python ex1.py
```

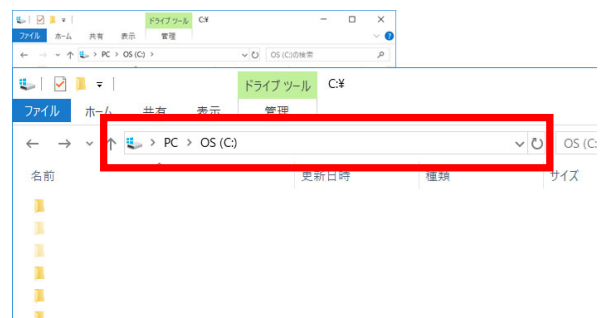
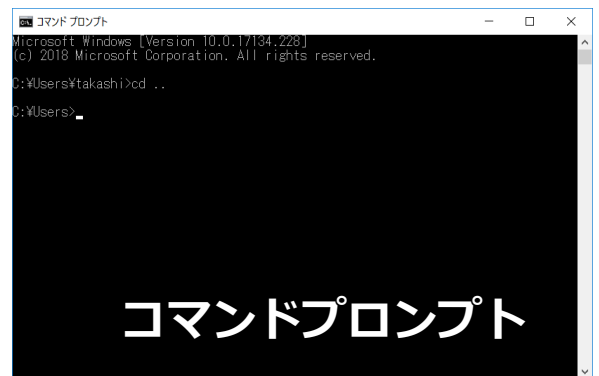
5. hello, worldと出力されたら成功

```
# ex1.py  
  
print("hello, world")
```

- ※ 『#』でその行をコメントアウト
- ※ 『print(“文字列”)』で文字列を出力
- ※ 今後日本語も利用したいのでファイルは UTF-8で保存する

## コマンドプロンプトについて

- 使ったことがない人もいると思うので解説します
- コマンドプロンプトとは, コマンドで実行ファイルを起動できるCUIアプリです (windowsにおけるUnixターミナルみたいな認識でOKです (右図) )
- Windowsのタスクバーの検索ウインドウに『cmd』と打ち込むと起動できます
- Unixのターミナル同様に『cd』コマンドでディレクトリを移動できます
- ディレクトリ内のフォルダ・ファイルを参照するには『dir』コマンドを打ち込みます (Unixのlsのようなもの)
- エクスプローラーのアドレスバー (右図) に『cmd』と書いてエンターを押すと, 開いたディレクトリをカレントとするコマンドプロンプトを起動されます (とても便利!)



## 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 ) : 変数型を取得する関数

※ id (変数名) : オブジェクトidを取得する関数  
(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。意味が分からない人は、とりあえず無視してOK。)

※ print (変数1, 変数2) で複数変数を出力可能

※ 型変換も可能

```
#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)
```

始めにC言語を学んだ皆さんからすると…

- 型指定が暗黙的に行なわれる
- int型の変数にfloat型を突っ込むとfloat型になる

あたりが気持ち悪いと感じるかもしれません

Pythonは型付けが動的（実行時）に行なわれます。  
上からプログラムを実行していった、変数が必要になった瞬間にその型が決まるようなイメージです

```
#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 コマンドライン引数

コマンドライン引数とは、コマンドラインからpythonを起動する際に、下のように与える引数の事です

```
> python ex3.py arg1 arg2 5
```

この例では、3個の文字列『arg1』『arg2』『5』を引数として与えています

**実習:** 3個の引数を受け取る右のコードの動作を確認してください。また、引数を変化させてみてください。実行コマンドは以下の通り;

```
> python ex3.py aaa bbb ccc
```

```
import sys

a1 = sys.argv[1]
a2 = sys.argv[2]
a3 = sys.argv[3]
print(a1, a2, a3)
```

※ 『import sys』は引数読み込みのためのライブラリを利用する準備

※ 『sys.argv[i]』にi番目の引数が入る

※ 引数は文字列型

## Ex4.py 配列

Pythonにはいくつかの配列表現がある

|                   |            |                                 |
|-------------------|------------|---------------------------------|
| (1,2,3)           | tuple      | : <b>長さ&amp;値 変更不可</b> の配列      |
| [1,2,3]           | list       | : 可変長配列 (要素を後から追加削除可)           |
| np.array([1,2,3]) | np.ndarray | : <i>n</i> 次元配列 (画像などはこれで表現される) |

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

## Ex4.py 配列

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

- ※ 配列は角括弧 [ ] で表現される
- ※ tupleは丸括弧 ( ) 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

```
#1D array
A = [1, 3, 5, 7]
N = len(A) #要素数
print("output1:", A, N)

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(3) #値が3の最初の要素を削除
print("output2", a, a2, A)

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

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

## Ex5.py np.ndarray

np.ndarray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、C,D,...,Iの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.ndarray の挙動を確認してください

- ※ 『import numpy as np』はnumpy関連モジュールを利用する準備
- ※ python & openCV環境では、np.ndarray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）のでとても便利
- ※ 配列変数名.shapeで配列サイズを取得
- ※ 要素がひとつのタプルにはカンマがつく (4, )
- ※ タプルはそもそもカンマで区切られた値なので...
- ※ a = 1,2,3 #← 丸かっこを省略することも可能
- ※ a = 3, #← これタプルになるので注意

```
import numpy as np

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

#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 + 3 #スカラーとの和

#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)
```

## Ex6.py np.ndarray

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

np.ndarray には便利な初期化方法が用意されている

**実習：右のコードを実行し結果を確認してください**

**実習：配列の分散が計算されていることを確認してください**

```
import numpy as np

A = np.array([1, 2, 3, 4, 5, 6, 7, 8])
mean = np.mean(A) #全要素の平均
sum = np.sum(A) #全要素の総和
vari = np.var(A) #全要素の分散
print(mean, sum, vari)

#分散は以下の方法でも計算可能
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])

#np.arrayの初期化方法
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
G = np.identity(3) #3x3 単位行列
```

## Ex6\_ref.py：配列の変数は参照を保持する

```
a = [1, 2, 3]
b = a
b[1] = 10
print(a)
print(b)
print(id(a))
print(id(b))
```

左のように書くと、aとbは同じものを指す。

- id(a) と id(b) も同じに
- bにした変更がaにも影響する
- メモリの同じ領域を参照しているイメージ

## Ex6\_ref.py : 配列の変数は参照を保持する

```
a = np.array([1, 2, 3])
b = a
# b = np.copy(a)
b[1] = 10
print(a)
print(b)
print(id(a))
print(id(b))
```

左のように書くと, aとbは同じものを指す.

※numpy配列でも同じ

- id(a) と id(b) も同じに
- bにした変更がaにも影響する
- メモリの同じ領域を参照しているイメージ

b = np.copy(a)

と書くとコピーが生成され, aとbは別物になる

## Ex6\_ref.py : pythonの変数はオブジェクトへの参照を保持する

```
#aは 値1を持つ int値 オブジェクトを指す
#bも aが参照する int値オブジェクトを指す
a = 1
b = a
print(a, id(a)) # aとbは同じものを参照
print(b, id(b)) # aとbは同じものを参照

# ここが問題
b = 3
# 数値オブジェクトはimmutable (不変)なので変更は起
# きない
# 新しいint値オブジェクトが生成されbはそれを指す
print(a, id(a)) # aとbは違うものを指す
print(b, id(b))
```

aとbが同じオブジェクトを参照しているというならば...  
b = 3 のタイミングで aも3になりそう

**mutableオブジェクトは変更が可能**

→ 配列などはmutableなので中身が変化する

**Immutableオブジェクトは不変**

→ 数値はimmutableなのでその値は変化しない

→ b=3とすると新しい数値オブジェクトが生成されbはそれを参照する

ちょっとややこしいので、上記の説明は一旦無視してもOKです  
現時点では『数値はc言語と同様に代入・変更でき、配列は参照が保持されるので気を付ける』のような理解でもよいかと思います



## Ex7.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

※pythonでは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  $i = a \sim b-1$  をループできる

```
import numpy as np

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

#Aの全要素の和をfor文で計算する
sum = 0
for p in A:
    print(p)
    sum += p

print(sum)

#range(N) を利用し N回繰り返しが可能
sum = 0
N = A.shape[0]
for i in range(N):
    print(A[i])
    sum += A[i]

print(sum)
```

## インデントについて

大事な事なので繰り返します

Pythonではインデントによりブロックを定義する

Python では右の場所がブロック

参考: Cでは{}でブロックを定義した

```
for( i=0; i<N; ++i){
    // Cではここがブロック
}
```

- ・インデントは半角スペース4個を推奨
- ・インデント内にスペースとタブが混在するとエラーが出るので注意 (毎年たくさんの学生がはまる)
- ・※ if文やfor文はスコープを作らないのでブロック内で定義した変数を外から参照できる (講義中に説明)

```
A = [1, 2, 3, 4, 5, 6, 7]
N = len(A)
sum = 0

for i in range(N):
    print(i)
    print(A[i])
    sum += A[i]
print(sum)
```

} ブロック

## Ex8.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:

```
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" )
```

## Ex9.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**

※以下の構文で関数を定義できる

def 関数名 (引数1, 引数2) :

処理1

処理2

:

return 変数

※複数の引数を受け取れ, **複数の戻り値を返せる!**

※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

※引数は参照渡し

- 配列を引数とした場合、オブジェクト自体が関数内で変化する

- 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、関数外部から見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。
- 意味が分からない人はとりあえず無視してOK
- 講義中に詳しく説明する予定

```
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

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

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

## ex9main.py main 関数 (今回は使わないので飛ばしてもOK)

- 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ファイル内に定義された関数を、他の.pyファイルから呼び出せる(importできる).
- このとき、関数だけ読み込みたいのに、importしたファイルの関数以外の部分が実行されると困る
- 関数以外の部分を『`if __name__ == '__main__':`』に入れると、外からimportされた時には実行されない

```
# ex9.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(sum, sub)
```



```
# ex9main.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(sum, sub)
```

## 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**("ファイル名")で画像読み込み  
※**cv2.imshow**("キャプション", img)で画像表示  
※**cv2.imwrite**("ファイル名", img)で画像書き出し  
※**cv2.waitKey(t)** キーボード入力待ち  
t は待ち時間[ms], 0を指定すると無期限に待つ

※画像は **np.array** 形式で表現される

**img.shape** : 画像サイズ

→ 幅128, 高さ512, カラー画像なら **img.shape**は (512, 128, 3)というタプルになる

**img.dtype** : 画像データの型

```
import numpy as np
import cv2

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

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

#display img
cv2.imshow("show image", img)
cv2.waitKey(0)
```

# 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](http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html)

```
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()
```

## Ex12.py 画素へのアクセス

**実習：**右のコードを動かし、画像のred値のみを取り出した画像が表示されることを確認して下さい

**課題：**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

$r = \text{img}[y,x,2]$

$g = \text{img}[y,x,1]$

$b = \text{img}[y,x,0]$

※途中計算時のオーバー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。

$\text{img} = \text{np.float64}(\text{img})$  #float64に変換

$\text{img} = \text{np.uint8}(\text{img})$  #uint8に変換

```
# 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) #空画像(WxH)を用意

for y in range(H) :
    for x in range(W) :
        r = img[y, x, 2] #画素(y, x)のred値
        g = img[y, x, 1] #画素(y, x)のgreen値
        b = img[y, x, 0] #画素(y, x)のblue値
        img_gry[y, x] = r #red値を代入

#display img
cv2.imshow("gray image", np.uint8( img_gry ) )
cv2.waitKey(0)
```

## 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

※グレースケール画像は2次元行列となります

**！！スライス表現！！**

※  $\text{img}[10:20, 30:40]$  とすると

$y = 10 \sim 19, x = 30 \sim 39$

の矩形画像にアクセスできます

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

```
import numpy as np
import cv2

#サイズ200x200の画像を作成
N = 200
img = np.zeros((N,N), np.uint8 )

#画素値代入(縦じま)
for y in range(N) :
    for x in range(N) :
        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(スライス表現)
img[50:60, 50:70] = 255

#display img
cv2.imshow("image", np.uint8(img) )
cv2.waitKey(0)
```

# 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

```
import numpy as np
import cv2

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])

#200x200の2D配列を作製
img1 = np.zeros( (200, 200), 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(0)
```

## 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画素の平均値』を注目画素に格納するものとする

```
# 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 ):
        #ここを編集し平滑化フィルタを完成させる
        img_out[y,x] = 128

cv2.imshow( "output", np.uint8( img_out ) )
cv2.waitKey(0)
```

# まとめ

- Pythonを利用した画像処理のために必要な知識を紹介した
- Python & OpenCV環境で、初歩的なコードを書いた.
- 『Python & OpenCV環境』では、とても手軽に画像処理が行えるので、興味がある方はぜひ独習を進めてください!