# Python音楽プログラミング
# 第１０回

# 集合
# set型

## 集合

Pythonで集合を扱うにはset関数を使用します。
そしてPythonでは集合をset型と呼びます。
リスト型のように、set型も同じく複数の値を格納できる型です。
リスト型との決定的な違いは

- **重複した要素がない**
- **要素に順番がない**

という点です。

```
set1 = set([1,2,3])
set2 = set([1,2,3,3,2,1]) #出力では重複がない事に注目
print(set1)
print(set2)
```

```
{1, 2, 3}
{1, 2, 3}
```

## 集合

### 要素の追加にはaddメソッドを使用します

```
# addメソッドで要素を追加
myset = set([1,2,3])
myset.add(4)
print(myset)
```

```
{1, 2, 3, 4}
```

### 要素の削除にはremoveメソッドを使用します

```
# removeメソッドで要素を削除
myset.remove(4)
print(myset)
```

```
{1, 2, 3}
```

### 要素を全て削除するにはclearメソッドを使用します

```
# clearメソッドで全て削除
myset.clear()
print(myset)
```

```
set()
```

# 集合の計算

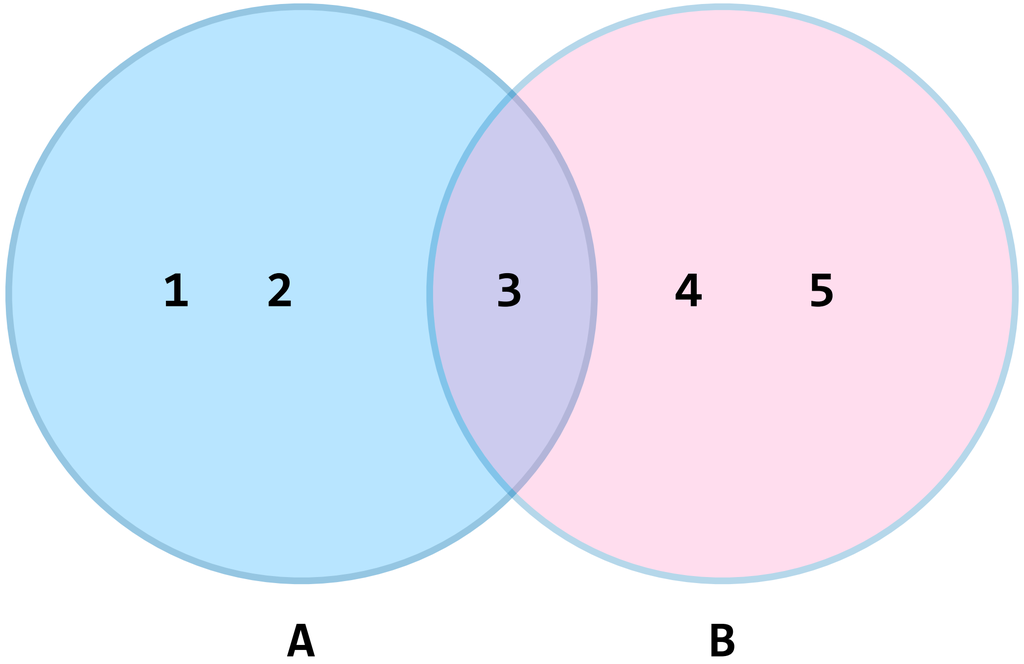
## 集合の計算

set型オブジェクトで**集合の計算**が簡単に出来ます。

集合の計算とは

- **和集合**
- **積集合**
- **差集合**
- **排他的論理和集合**

などのことです。



## 集合の計算

### 和集合

和集合は右図のAとBどちらかに含まれる全ての集合です。

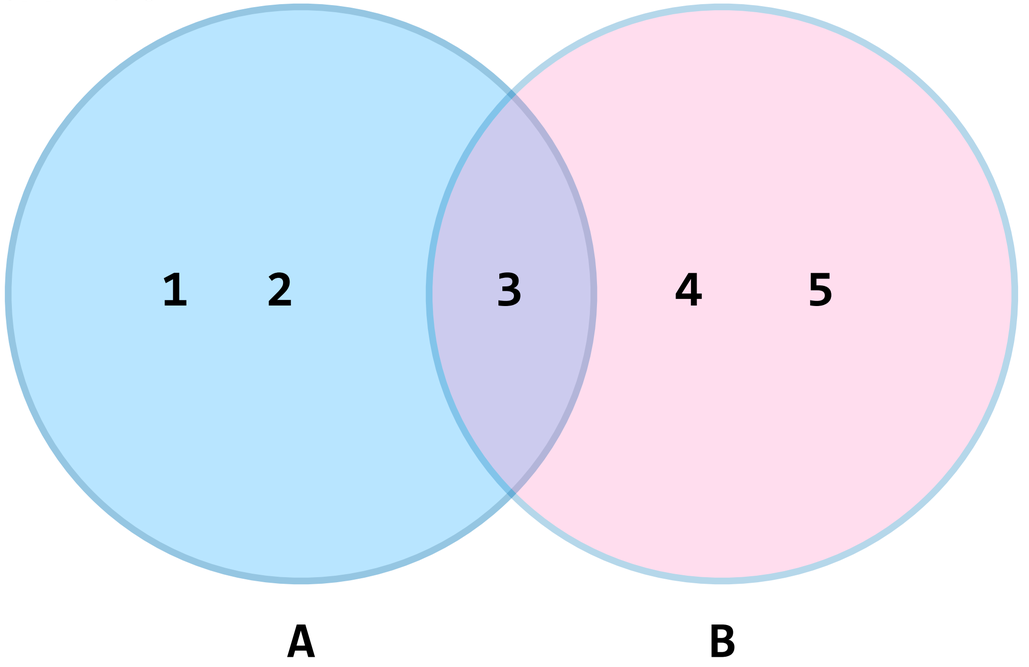
| または unionメソッドを使用します

```
# | を使用し和集合
A = {1,2,3}
B = {3,4,5}

C = A | B
print(C)
```

```
{1, 2, 3, 4, 5}
```

```
# unionメソッドで和集合
C = A.union(B)
print(C)
```

```
{1, 2, 3, 4, 5}
```



## 集合の計算

### 積集合

積集合は右図のAにもBにもどちらにも含まれる集合です。

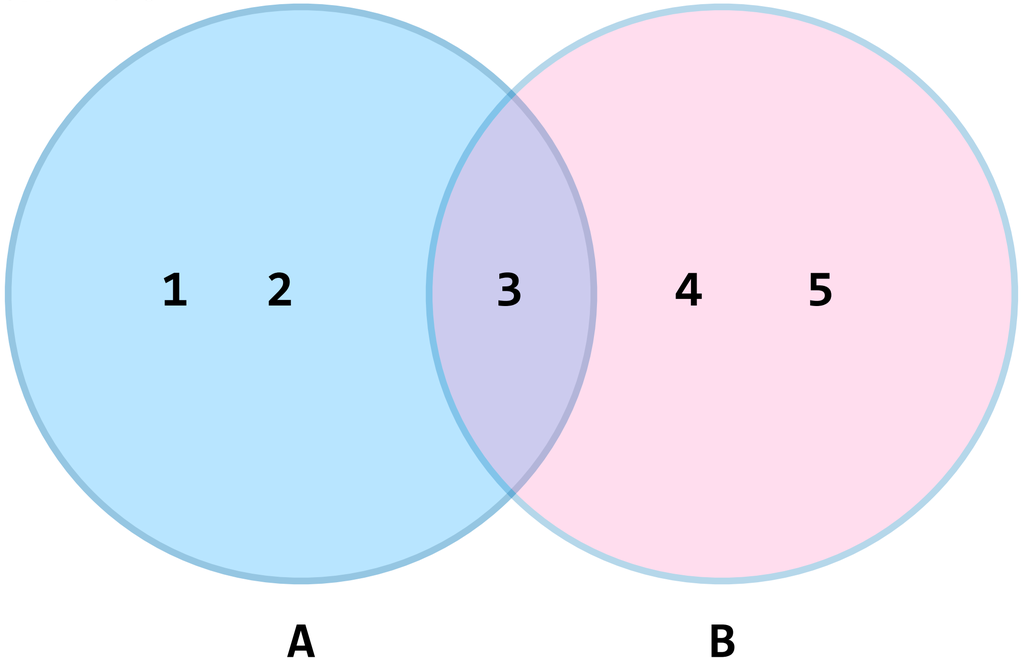
& または intersectionメソッドを使用します

```
# &で積集合
A = {1,2,3}
B = {3,4,5}

C = A & B
print(C)
```

```
{3}
```

```
# intersectionで積集合
C = A.intersection(B)
print(C)
```

```
{3}
```



## 集合の計算

### 差集合

差集合は右図のAからBに含まれる要素を抜いた集合です。

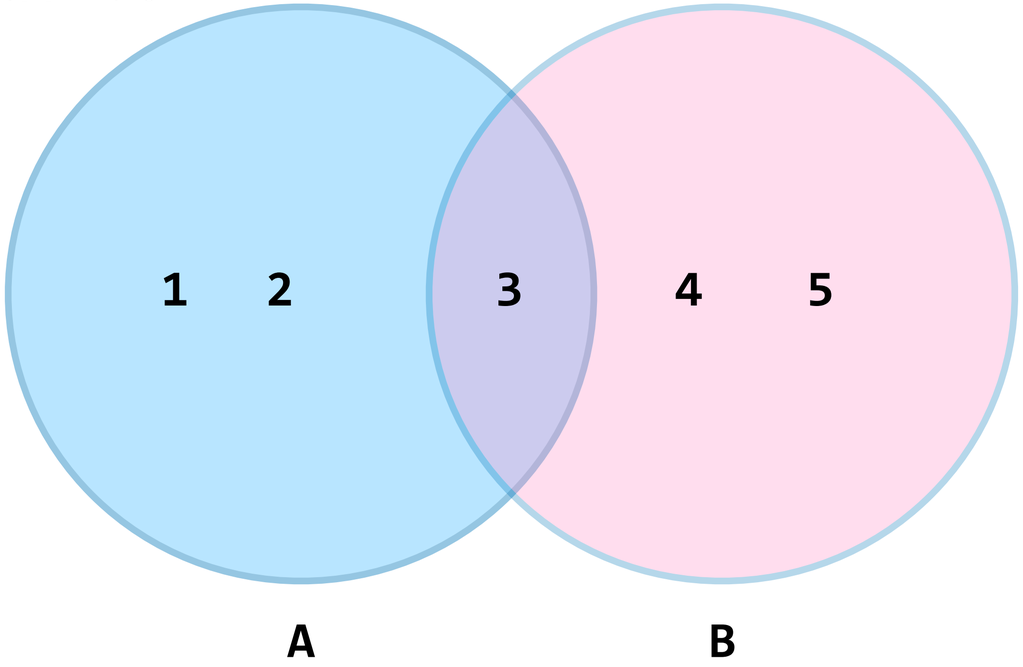
- または `difference`メソッドを使用します

```
# -で差集合
A = {1,2,3}
B = {3,4,5}

C = A - B
print(C)
```

```
{1, 2}
```

```
# differenceで差集合
C = A.difference(B)
print(C)
```

```
{1, 2}
```



## 集合の計算

### 排他的論理和集合

排他的論理和集合は右図の集合からABに両方に含まれる要素を抜いた集合です。

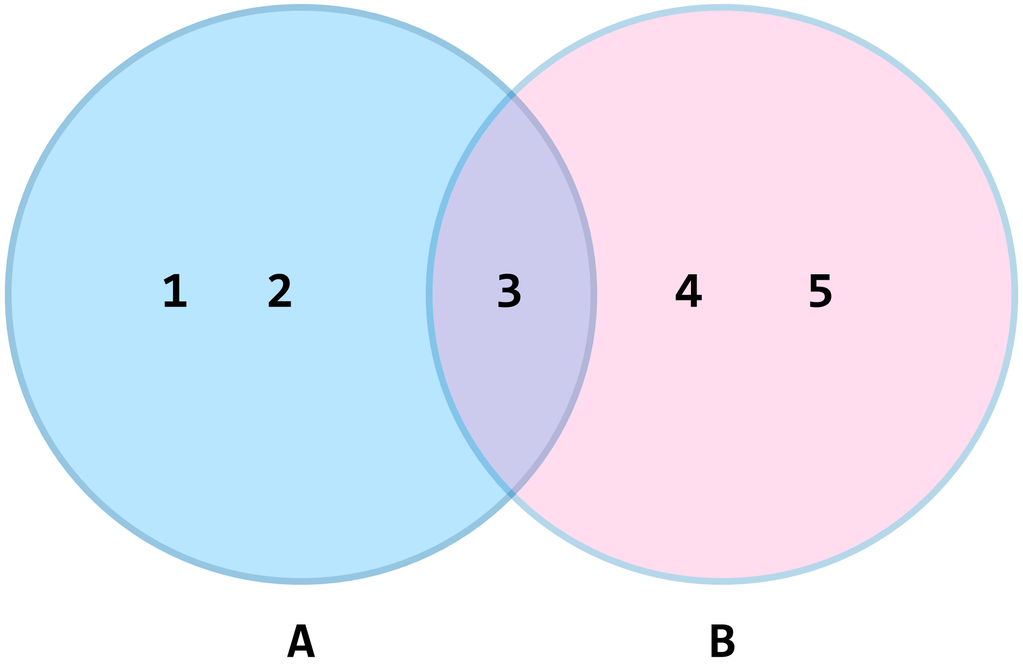
**^（キャレット）　または　symmetric_differenceメソッドを使用します**

```
# ^　で排他的論理和集合
A = {1,2,3}
B = {3,4,5}

C = A ^ B
print(C)
```

```
{1, 2, 4, 5}
```

```
# symmetric_differenceで排他的論理和集合
C = A.symmetric_difference(B)
print(C)
```

```
{1, 2, 4, 5}
```



# Scipy

## Scipy

scipyはpythonで高度な科学計算を行うためのライブラリです。
numpyの機能拡張版のモジュールとも言え、numpyの機能に加え、より高度な信号処理や統計といった計算ができるようになっています。

Scipyのインストール

```
pip install scipy
```

でインストール可能です。


※numpyのインストールが必要です。

Scipyは非常にたくさんのモジュールがあるので、それぞれの確認は公式ドキュメントを参照してください。
https://docs.scipy.org/doc/scipy/reference/

# Scipy

## Scipyで積分

scipyを使えば関数の積分（定積分）が簡単に行えます。
積分をするにはscipy.integrateモジュールのquad()を使います。
次の様なコードで実行できます。

```
from scipy import integrate  #インポート
変数1, 変数2 = integrate.quad(関数名, 積分区間の始まり, 積分区間の終わり)
```

quad()は与えられた関数を[区間の始まり, 区間の終わり]の区間で定積分し、2つの値を返します。
変数1には積分した結果が、変数2には積分計算をした際の誤差が返されます。

## Scipyで積分

```
#Scipyで積分
from scipy import integrate

#関数 2x+5
def func(x):
    return 2*x + 5

result, err = integrate.quad(func, 0, 5)

print('積分結果は{0}です\n誤差は{1}です'.format(result, err))
```

積分結果は50.0です
誤差は5.551115123125783e-13です

# Scipy

## Scipyで行列の計算

行列の計算はnumpyでもできますが、scipyを使えばより高度な計算が可能です。
行列の計算には線形代数用の機能をまとめたscipy.linalgを使います。

```
from scipy import linalg #linalgモジュールのインポート
import numpy as np  #配列作成のためnumpyもインポートします
```

Scipyには他にも信号処理や画像解析、統計処理、フーリエ変換など高度な科学計算を行うためのモジュールが非常に多く用意されています。

## Scipyで行列の計算

```
#Scipyで行列の計算

from scipy import linalg
import numpy as np  #配列作成のためnumpyもインポートします

narray = np.array([[1, 2], [3, 4]])  #numpyのarrayで2 x2の行列を作成


inv_narray = linalg.inv(narray) #逆行列
print('narrayの逆行列は\n{}\n'.format(inv_narray))


result_det = linalg.det(inv_narray) #行列式
print('inv_narrayの行列式は {}\n'.format(result_det))


inv_narray_norm = linalg.norm(inv_narray) #ノルム
print('inv_narrayのノルムは{}'.format(inv_narray_norm))
```

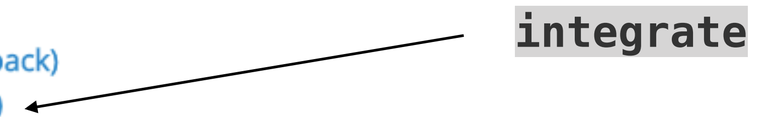
```
narrayの逆行列は
[[-2.   1. ]
 [ 1.5 -0.5]]

inv_narrayの行列式は -0.49999999999999967

inv_narrayのノルムは2.73861278752583
```

# Scipy

## API Reference

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

- Clustering package (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms (scipy.fftpack)
- Integration and ODEs (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Miscellaneous routines (scipy.misc)
- Multi-dimensional image processing (scipy.ndimage)
- Orthogonal distance regression (scipy.odr)
- Optimization and Root Finding (scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial algorithms and data structures (scipy.spatial)
- Special functions (scipy.special)
- Statistical functions (scipy.stats)
- Statistical functions for masked arrays (scipy.stats.mstats)
- Low-level callback functions




# format関数

## format関数

### format関数とは？

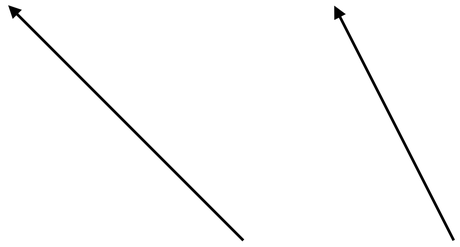
format関数は、Pyhonで変数の文字列への埋め込みに使われます。
print関数で変数を任意の文字列とともに出力したいときに使われる事が多いです。
format関数の基本的な構文は以下のようになります。

```
'任意の文字列{}任意の文字列'.format(変数)
```

format関数

```
#format関数
CD1  = 15
CD2 = 18
total = CD1 + CD2

print('２枚合わせて：{}曲'.format(total))
```

２枚合わせて：33曲



ここで指定された変数totalが{}内に渡される

# format関数

## 複数の引数を与える

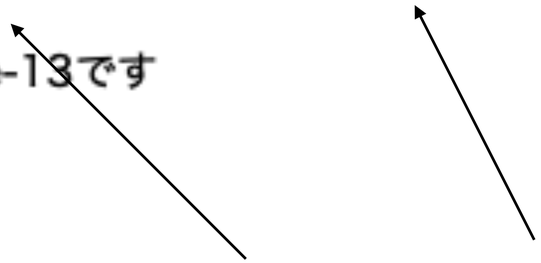
format関数には複数の変数を引数として渡し、文字列に埋め込むことができます。
複数の変数を埋め込む場合は、どの引数をどの{}に埋め込むかを指定する必要がありますが、指定の方法はいくつかあります。
代表的なものに、以下の様に
インデックス（順番）、キーワード引数、辞書引数、などです。

```
{0}, {1}, {2}...'.format(変数1, 変数2, 変数3….)  #インデックス（添え字）で指定
'{h1}, {h2},{h3}..'.format(h1=変数1, h2=変数2, h3=変数…)  #キーワード引数で指定
'{h1},{h2},{h3}…'.format(**{'h1':変数1, 'h2':変数2, 'h3':変数3….} )  #辞書引数で指定
```

## format関数

```
#Scipyで積分
from scipy import integrate

#関数 2x+5
def func(x):
    return 2*x + 5

result, err = integrate.quad(func, 0, 5)

print('積分結果は{0}です\n誤差は{1}です'.format(result, err))
```

```
積分結果は50.0です
誤差は5.551115123125783e-13です
```



**指定された変数resultとerrがインデックスで番号指定された{}内に渡される**

# scikit-learn

# scikit-learn

**scikit-learn**は、
Pythonの機械学習ライブラリです。
多くの機械学習アルゴリズムが実装されており、どのアルゴリズムでも同じような書き方で利用することができます。
サンプルデータセットが付属しているため、インストールしてすぐ機械学習を試すことができます。

AnacondaやGoogle Colabで標準でインストールされているためすぐに使用できますが、未インストールの方は

```
pip install scikit-learn
```

でインストールできます

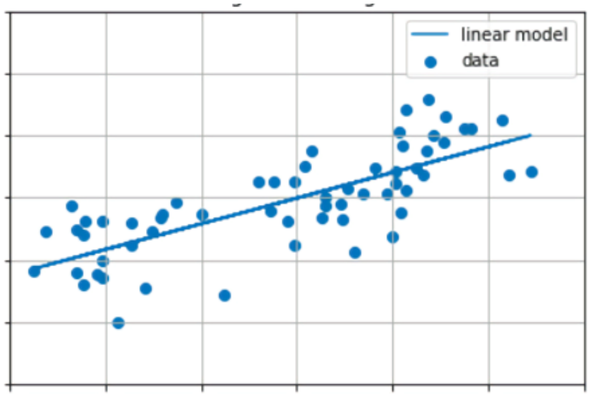
https://scikit-learn.org/stable/

# 機械学習

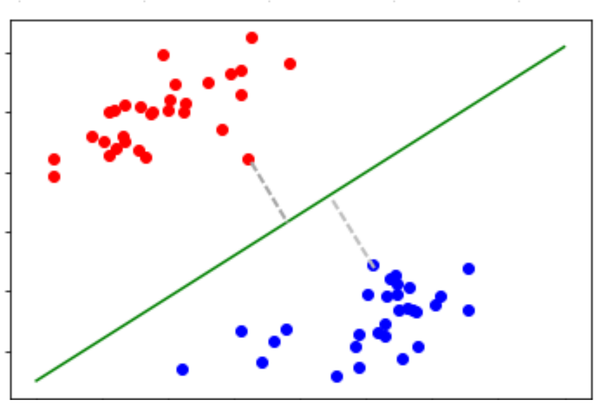
**機械学習**

知能を模した機械、人工知能が自ら学習する仕組みでコンピュータのプログラムによって実現される。

機械学習は、学習のためのデータが多ければ多いほど望ましい学習結果、つまり出力結果が得られる。




**回帰（家賃の相場予測等）**

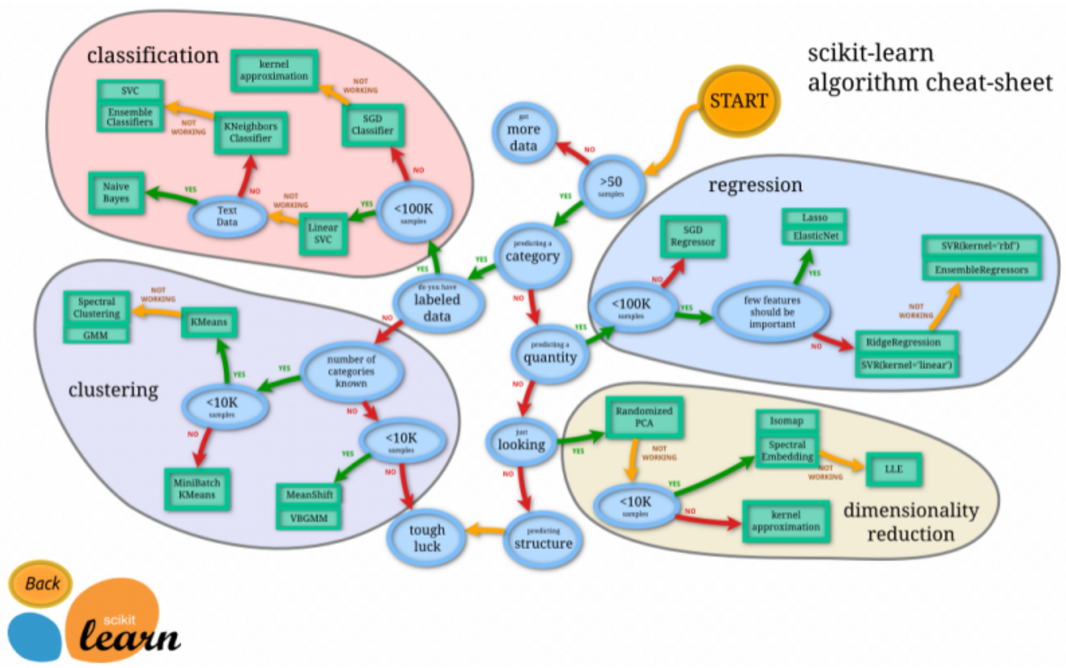


**分類（犬と猫の画像分類等）**

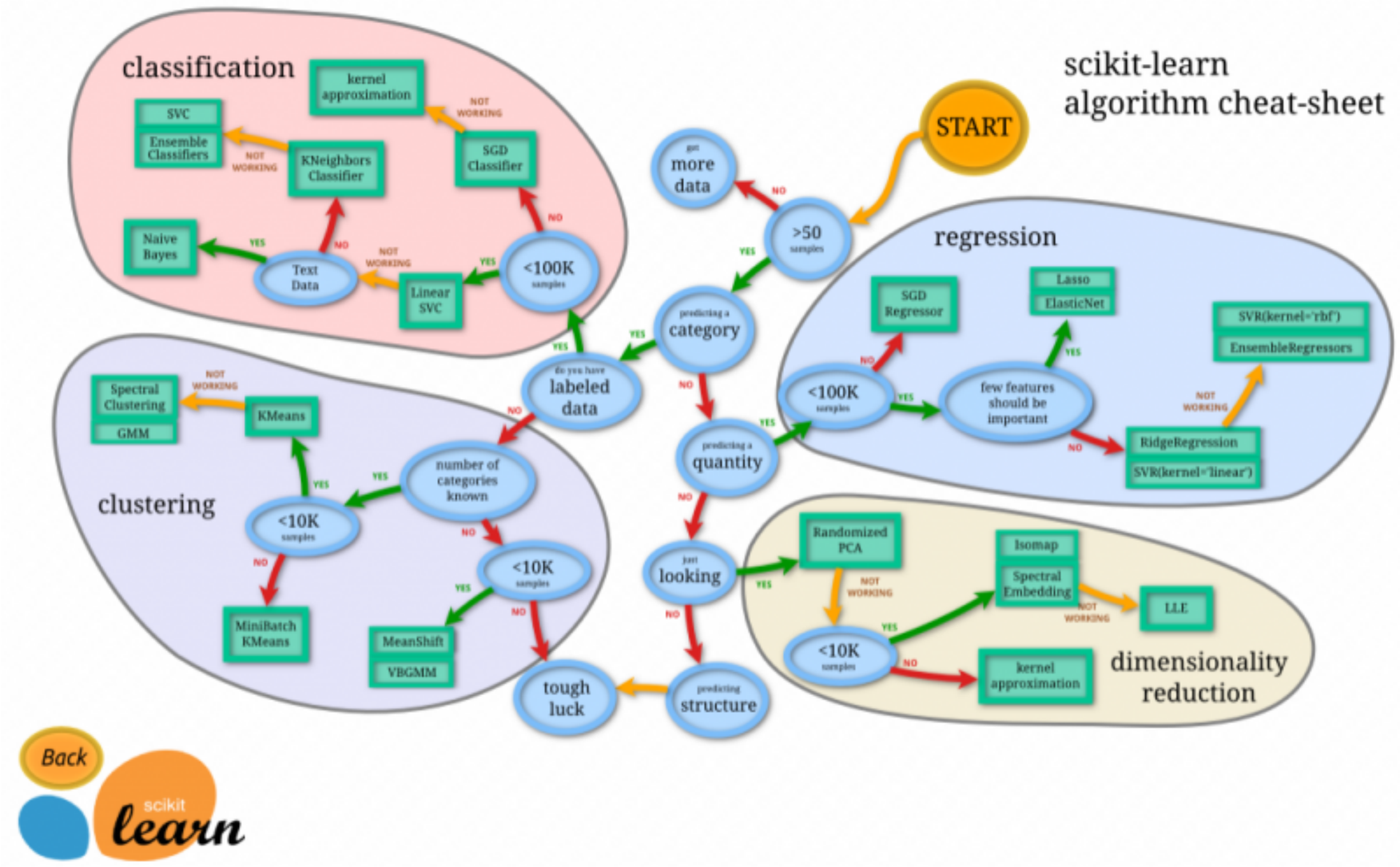
そのため機械学習の実用性を高めるには大量のデータの確保が必要でした。
１９９０年代以降のパーソナルコンピューターとインターネットの普及は現在の機械学習の活性化に大きな役割を果たしています。

## scikit-learn

scikit-learnでは行いたい機械学習ごとに、たくさんのモデルが提供されています。
以下の図はscikit-learnのアルゴリズムチートシートと呼ばれるもので、scikit-learnを用いて機械学習を行う際、自分が行いたい分析（分類／回帰／クラスタリングなど）について、適切なモデルを視覚的に探せる様に作られているものです。



# scikit-learn



scikit-learn

# 分類（classification）

与えられたデータを任意に分類するアルゴリズムです。

教師あり学習として以下の様なものが用意されています。

### SGD（stochastic gradient descent）

大規模データ（10万件以上）に向く、線形のクラス分類手法

### カーネル近似

SGDではうまく分類できない場合に利用する、非線形なクラス分類手法。大規模データ向き

### Linear SVC

中小規模（10万件未満）に向く、線形のクラス分類手法

### k近傍法

Linear SVCではうまく分類できない場合に利用する、非線形なクラス分類手法。中小規模データ向き。

scikit-learn

# 回帰（regression）

与えられたデータの相関性を元に、出力結果を予測するものです。
教師あり学習の回帰問題として以下の様なものが用意されています。

**SGD（stochastic gradient descent）**
大規模データ（10万件以上）向きの、線形の回帰分析手法

**LASSO、ElasticNet**
中小規模（10万件未満）向き。説明変数の一部が重要な場合に使用される回帰分析手法

**Ridge、Liner SVR**
中小規模（10万件未満）向き。説明変数の全てが重要な場合に使用される回帰分析手法

**SVR（ガウスカーネル）、Ensemble**
Ridge、またはLinerSVRではうまく分析できない場合に利用する、非線形な回帰分析手法

# クラスタリング（clustering）

与えられたデータを、それぞれの特徴によって分類する。
教師なし学習の分類問題として以下の様なものが提供されています。

**KMeans**
いくつのクラスタに分かれるのか、事前に決めることができる場合におすすめな、クラスタリング分析手法
大規模データの場合、MiniBatchといって、データを分けながら学習させる手法を取ります

**スペクトラルクラスタリング、GMM**
KMeansではうまく分析できない場合に利用する、非線形なクラスタリング分析手法

**MeanShift、VBGMM**
いくつのクラスタに分かれるのか、事前に決めることができない場合におすすめな、クラスタリング分析手法

scikit-learn

# その他機械学習で使用される機能

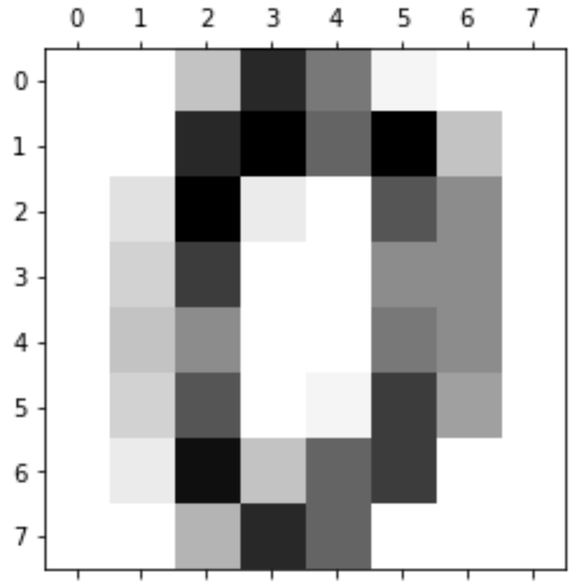
**次元削減**
与えられたデータの次元数が多い場合、学習効率を上げるため、次元削減という前処理を行う。
PCA、カーネルPCA、Isomap、SpectralEmbeddingなどの手法がある

**ハイパーパラメータ自動最適化**
機械学習を行う際、学習の方法などを調整する数値のことを「ハイパーパラメータ」と呼ぶ。
本来手動で調整される事が多いが、それを機械学習として自動調整する機能。
グリッドサーチ、クロスバリデーションなどの手法がある

# scikit-learn
# で機械学習の実践

## 手書きデータセットの読み込み

```
# scikit-learn
from sklearn import datasets #インポート

# 手書きデータセットの読み込み
digits = datasets.load_digits()

# データ確認
import matplotlib.pyplot as plt
plt.matshow(digits.images[0], cmap="Greys")
plt.show()
```



### 訓練データとテストデータの準備

データセットには、「手書き数字の画像データ」と、それに対する「数字」が含まれます。
今回は教師あり学習の教師＝正解データが数字になります。正解データはラベルとも呼びます。
データは訓練データとテストデータに分け、訓練データで学習した結果を、テストデータで検証します。

```
# 画像データを配列にする
X = digits.data

# 数字（ラベル＝正解データ）
y = digits.target

# 訓練データとテストデータに分ける（今回は５：５）
X_train, y_train = X[0::2], y[0::2] # 訓練データ ：偶数行
X_test, y_test = X[1::2], y[1::2] # テストデータ：奇数行
```

**学習**

モデルに学習を行います。

今回はSVMというアルゴリズムを選択しています。

```
# 学習器の作成。SVM使用
from sklearn import svm
clf = svm.SVC(gamma=0.001)

# 訓練データとラベルで学習
clf.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

## モデルの評価

モデルの学習結果を、テストデータを用いて評価しレポート表示します

```python
#モデルの評価
from sklearn.metrics import classification_report #評価レポート用のモジュール

# テストデータで検証した正解率
accuracy = clf.score(X_test, y_test)
print(f"正解率: {accuracy}")

# テストデータを分類した結果を返す
predicted = clf.predict(X_test)


# 詳しいレポート
# precision(適合率): 選択した正解/選択した集合
# recall(再現率) : 選択した正解/全体の正解
# F-score(F値) : 適合率と再現率はトレードオフの関係にあるため
print(classification_report(y_test, predicted))
```

```
正解率: 0.9866369710467706
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.98      1.00      0.99        89
           2       1.00      1.00      1.00        91
           3       1.00      0.98      0.99        93
           4       0.99      1.00      0.99        88
           5       0.98      0.97      0.97        91
           6       0.99      1.00      0.99        90
           7       0.99      1.00      0.99        91
           8       0.97      0.97      0.97        86
           9       0.98      0.97      0.97        91

    micro avg       0.99      0.99      0.99       898
    macro avg       0.99      0.99      0.99       898
 weighted avg       0.99      0.99      0.99       898
```

**学習器の変更ができます**

ロジスティック回帰というアルゴリズムを選択してみましょう

```
# 学習器の作成。ロジスティック回帰使用
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()

# 訓練データとラベルで学習
clf.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

**モデルの評価**

ロジスティック回帰でもテストデータを用いて評価しレポート表示します

```python
#モデルの評価
from sklearn.metrics import classification_report #評価レポート用のモジュール

# テストデータで検証した正解率
accuracy = clf.score(X_test, y_test)
print(f"正解率: {accuracy}")

# テストデータを分類した結果を返す
predicted = clf.predict(X_test)


# 詳しいレポート
# precision(適合率): 選択した正解/選択した集合
# recall(再現率) : 選択した正解/全体の正解
# F-score(F値) : 適合率と再現率はトレードオフの関係にあるため
print(classification_report(y_test, predicted))
```

```
正解率: 0.9398663697104677
              precision    recall  f1-score   support

           0       0.97      0.98      0.97        88
           1       0.87      0.98      0.92        89
           2       0.97      0.99      0.98        91
           3       0.93      0.92      0.93        93
           4       0.90      0.98      0.93        88
           5       0.96      0.96      0.96        91
           6       0.99      0.98      0.98        90
           7       0.96      0.97      0.96        91
           8       0.91      0.86      0.89        86
           9       0.96      0.79      0.87        91

    micro avg       0.94      0.94      0.94       898
    macro avg       0.94      0.94      0.94       898
 weighted avg       0.94      0.94      0.94       898
```

# open cv

# open cv

**OpenCV（Open Source Computer Vision Library）**は、コンピューターで画像や動画を処理するための機能がまとめて実装されているライブラリです。
Python以外にもC++やJavaなど色々な言語から使用する事ができます。
非常にたくさんの高度な機能が提供されていますので是非公式サイトで確認をしてみてください。

https://opencv-python-tutroals.readthedocs.io/en/latest/index.html


**anaconda**やgoogle colabでは標準でインストールされていますが未インストールの方はpipでインストールしてください。
なお必須の依存ライブラリーであるnumpyも一緒にインストールされます（未インストールの方）

```
pip install opencv-python
```

# open cv
# で顔検出

## open cvで顔検出

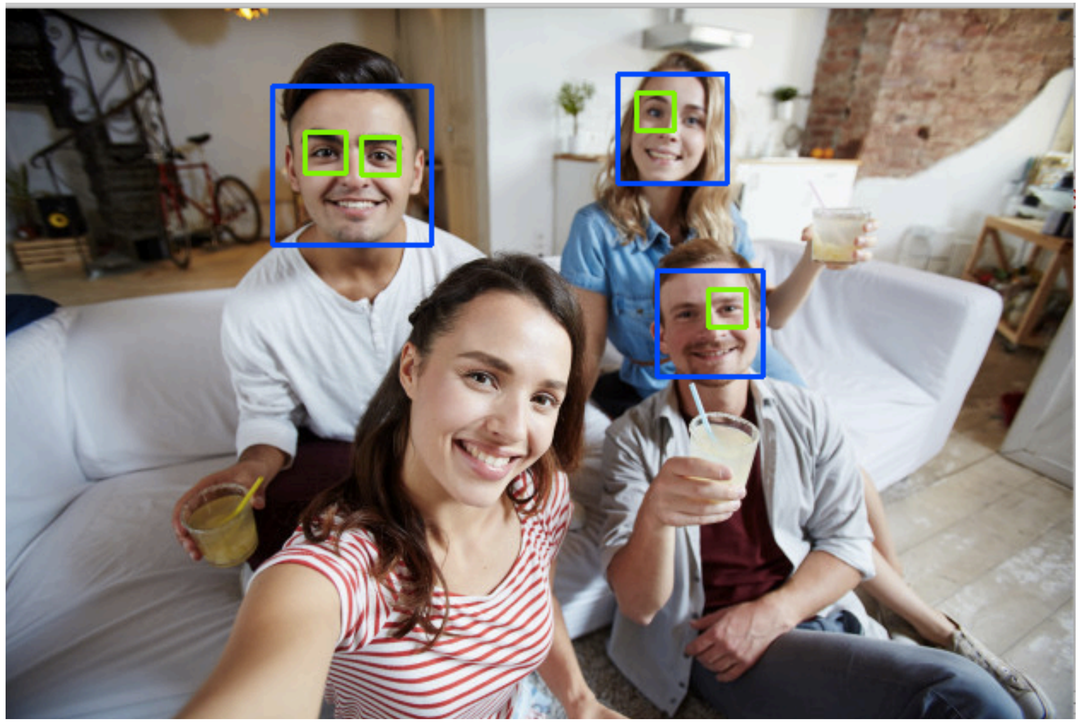
## cv2.CascadeClassifier()を使用し、顔と目を検出できます

```
import numpy as np
import cv2

#ご自身の仮想環境に合わせパスを指定
face_cascade = cv2.CascadeClassifier('/anaconda3/lib/python3.6/site-packages/cv2/data/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('/anaconda3/lib/python3.6/site-packages/cv2/data/haarcascade_eye.xml')


#任意の画像または添付画像のパスを指定
img = cv2.imread('_1098-15079.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## open cvで顔検出

`cv2.CascadeClassifier()`を使用し、顔と目を検出できます

元画像



検出

# open cv
# Canny法でエッジ検出

## open cv Canny法でエッジ検出

cv2.Canny()を使いCanny法でエッジを検出する事ができます。
cv2.Canny()の2つ目と3つ目の引数を調整すると、検出できる輪郭が変わります。

```
# Canny法でエッジ検出
import cv2
import numpy as np

img = cv2.imread('_1098-15079.jpg',0) #添付してある画像の例です

edges = cv2.Canny(img,100,200)

cv2.imshow('edges',edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## open cv Canny法でエッジ検出

cv2.Canny()を使いCanny法でエッジを検出する事ができます。
cv2.Canny()の2つ目と3つ目の引数を調整すると、検出できる輪郭が変わります。

元画像

検出

# open cv
# 画像のヒストグラムを求める

## open cv 画像のヒストグラムを求める

cv2.calcHist()を使い画像のヒストグラムを求める事ができます。
ヒストグラムとは、データの分布状況を横軸、縦軸にグラフ化したものです。

画像のヒストグラムでは、R（赤）、G（緑）、B（青）のチャンネルごとに、横軸が明るさの階調（0〜255）を示し、縦軸がピクセル数を示しています。

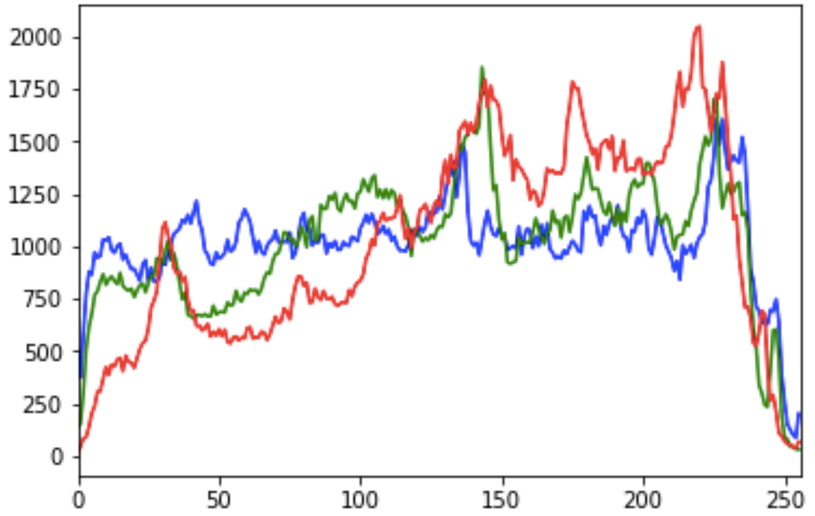
open cvで検出、グラフ表示にmatplotlibを使っています。
まだmatplotlib未インストールの場合は

```
pip install matplotlib
```

でインストールしてください

# open cv 画像のヒストグラムを求める

```
# 画像のヒストグラムを求める
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('_1098-15079.jpg') #添付してある画像の例です
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```



# open cv
# 円の検出

## open cv 円の検出

cv2.HoughCircles()を使い画像から円の部分だけを検出できます。

param1=800の数値を小さくすると、もっと多くの円が検出されるようになります。

```
#円の検出
import cv2
import numpy as np

img_org = cv2.imread('4976660182232.jpg') #添付してある画像の例です
imgray = cv2.cvtColor(img_org,cv2.COLOR_BGR2GRAY)
imgray = cv2.medianBlur(imgray,5)

circles = cv2.HoughCircles(imgray,cv2.HOUGH_GRADIENT,1,20,
                           param1=800,param2=30,minRadius=0,maxRadius=0) #param1の値を変化させると検出感度の調整ができます

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(img_org,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(img_org,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('detected circles',img_org)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## open cv 円の検出

cv2.HoughCircles()を使い画像から円の部分だけを検出できます。
param1=800の数値を小さくすると、もっと多くの円が検出されるようになります。


元画像


検出

# Pythonでシンセ作成

# Pythonでシンセ作成

配布した
Subtractive-synth.py
を開いてください。
pythonファイルなので読み込んで実行する必要があります。

例　ターミナルやjupyter notebookで
python /ご自身の保管ディレクトリ/Subtractive-synth.py

# Pythonでシンセ作成

```
import numpy as np
import pyaudio
import struct
import cv2
import threading

RATE=44100
bufsize = 64

ksx = 800
ksy = 200
keyboard = np.zeros([ksy,ksx,3])
keyboard[:,:,:] = 255
for i in range(15):
    cv2.rectangle(keyboard, (int(ksx/15*i), 0), (int(ksx/15*(i+1)),ksy), (0, 0, 0), 5)
for i in range(15):
    if i in {0,1,3,4,5,7,8,10,11,12}:
        cv2.rectangle(keyboard, (int(ksx/15*i + ksx/27 ), 0), (int(ksx/15*(i+1) + ksx/33),int(ksy/2)), (0, 0, 0), -1)

cv2.namedWindow("keyboard", cv2.WINDOW_NORMAL)

sl=np.array([0,150,150,255])
slName = np.array(['Wave_type',
                   'Attack',
                   'Release',
                   'Lowpass_freq'])

def changeBar(val):
    global sl
    for i in range(4):
        sl[i] = cv2.getTrackbarPos(slName[i], "keyboard")
cv2.createTrackbar(slName[0], "keyboard", 0, 3, changeBar)
cv2.createTrackbar(slName[1], "keyboard", 0, 255, changeBar)
cv2.createTrackbar(slName[2], "keyboard", 0, 255, changeBar)
cv2.createTrackbar(slName[3], "keyboard", 0, 255, changeBar)
for i in range(4):
    cv2.setTrackbarPos(slName[i], "keyboard", sl[i])
```

# Pythonでシンセ作成

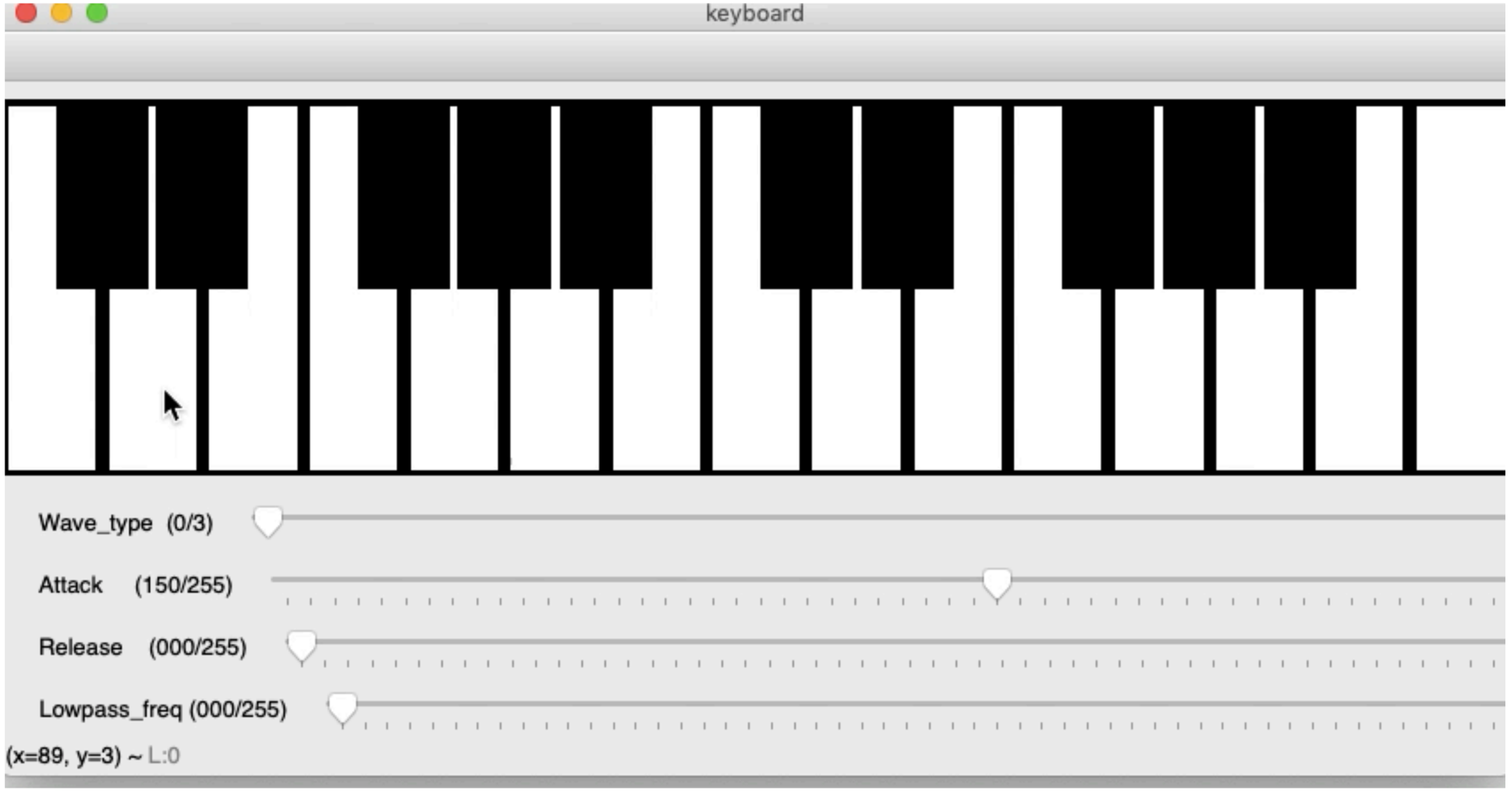
```
keyon = 0
pre_keyon = 0
pitch = 440
velosity = 0.0
highkeys = np.array([0,1,1,3,3,4,5,6,6,8,8,10,10,11, 12,13,13,15,15,16,17,18,18,20,20,22,22,23, 24,24])
lowkeys = np.array([0,2,4,5,7,9,11, 12,14,16,17,19,21,23, 24])
def mouse_event(event, x, y, flags, param):
    global keyon,pre_keyon,pitch,velosity
    if event == cv2.EVENT_LBUTTONDOWN:
        keyon = 1
        if y >= ksy/2:
            note = lowkeys[int(15.0*x/ksx)]
        elif y < ksy/2:
            note = highkeys[int(30.0*x/ksx)]
        pitch = 440*(np.power(2,(note-9)/12))
    elif event == cv2.EVENT_LBUTTONUP :
        keyon = 0
    if pre_keyon ==0 and keyon ==1:
        velosity = 0.0
    pre_keyon = keyon
cv2.setMouseCallback("keyboard", mouse_event)


lpfbuf=np.zeros(4)
outwave=np.zeros(bufsize)
def lowpass(wave):
    global lpfbuf,outwave
    w0 = 2.0*np.pi*(200+(sl[3]/255.0)**2*20000)/RATE;
    Q = 1.0
    alpha = np.sin(w0)/(2.0*Q)
    a0 =   (1 + alpha)
    a1 =  -2*np.cos(w0)/a0
    a2 =   (1 - alpha)/a0
    b0 =  (1 - np.cos(w0))/2/a0
    b1 =   (1 - np.cos(w0))/a0
    b2 =  (1 - np.cos(w0))/2/a0
    for i in range(bufsize):
        outwave[i] = b0*wave[i]+b1*lpfbuf[1]+b2*lpfbuf[0]-a1*lpfbuf[3]-a2*lpfbuf[2]
        lpfbuf[0] = lpfbuf[1]
        lpfbuf[1] = wave[i]
        lpfbuf[2] = lpfbuf[3]
        lpfbuf[3] = outwave[i]
    return outwave
```

# Pythonでシンセ作成

```
x=np.arange(bufsize)
pos = 0
def synthesize():
    global pos,velosity


    t = pitch * (x+pos) / RATE
    t = t - np.trunc(t)
    pos += bufsize


    if sl[0]==1:
        wave = t*2.0-1.0
    elif sl[0]==2:
        wave = np.zeros(bufsize);wave[t<=0.5]=-1;wave[t>0.5]=1;
    elif sl[0]==3:
        wave = np.abs(t*2.0-1.0)*2.0-1.0
    else:
        wave = np.sin(2.0*np.pi*t)

    if keyon == 1:
        vels = velosity + x * ((sl[1]/1000)**3+0.00001)
        vels[vels>0.6] = 0.6
    else:
        vels = velosity - x * ((sl[2]/1000)**3+0.00001)
        vels[vels<0.0] = 0.0
    velosity = vels[-1]
    wave = vels * wave

    wave = lowpass(wave)

    return wave
```

```
playing = 1
def audioplay():
    print ("Start Streaming")
    p=pyaudio.PyAudio()
    stream=p.open(format = pyaudio.paInt16,
            channels = 1,
            rate = RATE,
            frames_per_buffer = bufsize,
            output = True)
    while stream.is_active():
        buf = synthesize()

        buf = (buf * 32768.0).astype(np.int16)
        buf = struct.pack("h" * len(buf), *buf)
        stream.write(buf)
        if playing == 0:
            break
    stream.stop_stream()
    stream.close()
    p.terminate()
    print ("Stop Streaming")


if __name__ == "__main__":
    thread = threading.Thread(target=audioplay)
    thread.start()
    while (True):
        cv2.imshow("keyboard", keyboard)
        k = cv2.waitKey(100) & 0xFF
        if k == ord('q'):
            playing = 0
            break

    cv2.destroyAllWindows()
```

# Pythonでシンセ作成



今回の添付ファイル

python-09.ipynb

画像：_1098-15079.jpg

画像：4976660182232.jpg

subtractive-synth.py