デジタルメディア処理1

担当: 井尻 敬

提出方法: 共有フォルダに 『**dm1学籍番号**』というフォルダを作成し，その中にソースコードの入ったファイルを置く．フォルダ名は全て半角.

フォルダ名の例: dm2AL150999

課題雛形: http://takashiijiri.com/classes/dm2018_1/dm1exer.zip

入出力 ： 課題では入力画像を受け取り，画像またはファイルを保存するプログラムを作る．入出力ファイル名は，以下の例のようにコマンドライン引数より与えるものとする． （※各課題の指定に従うこと）

```
$python  exer*.py  fname_in.png  fname_out.png
```

注意 ：

採点は自動化されています．フォルダ名・ファイル名やプログラムの仕様は指示に厳密に従ってください．入出力の仕様を満たさないコードは評価できず0点扱いとなることがあります．

今回は計算速度を重視しませんが，60秒以上の計算時間がかかるものは，自動採点の都合上0点とします．

各課題について，入出力例を用意するので，作成したプログラムのテストに利用してください．

締め切りと配点

課題番号	締め切り	配点
• 基礎課題01~05	**11/13 23:59**	各2.0点
• 画像課題06~16	**12/08 23:59**	各3.0点
• 発展課題17~1x	**12/08 23:59**	各x.x点

※ 全問正解で合計 約50点

※ 配点/締め切りは，上方に/後ろに修正する可能性があります

注意

- **この課題は，知人同士で相談しながら取り組んでよいです**
  - 教える立場の方は理解が補強でき，教わる方は難しい課題も解けるようになり，メリットは大きいです
  - ただし，教わる人はただ他人のコードをコピーするだけではなく，その部分にどのような意味があるかを確実に理解するようにしてください
- **この課題の解答となるコードを，『この課題の解答と分かる形』でWeb上（GitHub，SNS，個人web page)に公開することは避けてください**
- **知恵袋やteratailなどのナリッジコミュニティサイトにて，問題文をそのまま掲載し，解答を得ることは行なわないでください**
  - 上記のような活動を井尻が発見した場合は，しかるべき処理をとります
  - 分からない部分がある場合は『どこがどう分からないかを自分の言葉で明確に説明し』他者から知識を受け取ってください
- **プログラミングが不得意な方：**課題を読むと難しく感じるかもしれませんが，各課題のひな形にヒントを多く記載してあります．まずはそこを読んでみてください．
- **プログラミング得意な方:** それなりに歯応えのあるものを用意しようと思ってはいるのですが，まだ簡単なものも多いです．簡単すぎたらごめん。

10/09分 pythonの基本と画像の入出力
課題1~5

課題1. 標準出力 - 雛形 exer1.py

コマンドライン引数から2つの整数を受け取り，その積の回数だけ『hello, world』と標準出力に表示するプログラムを作成せよ

雛形(exer1.py)に途中まで作製したコードがあるので参考にすること

- 実行コマンドと実行例は以下の通り

```
$ python   exer1.py   1  4
hello, world
hello, world
hello, world
hello, world
```

```
$ python   exer1.py   2  3
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
```

※自動採点の都合上，関係ないものは標準出力に出さないで下さい

課題2. 標準出力 - 雛形 exer2.py

1からNまでの整数を順番に標準出力に表示するプログラムを作成せよ．ただし，以下の仕様を満たすこと．

- 表示する数字が4の倍数の時には ”hoge” と表示する
- 表示する数字が5の倍数の時には ”fuga” と表示する
- 表示する数字が，4と5，両方の倍数の時には ”hogefuga”と表示する
- 最大の数 *N* は，コマンドライン引数より与える

実行コマンドは以下の通り．

```
$ python   exer2.py   20
```

実行例は，後述

課題3. ファイル入力と配列 - 雛形 exer3.py

数値データをファイルから読み込み，その最大値・最小値をファイルに出力するプログラムを作成せよ

- 入力ファイル名，出力ファイル名は，コマンドライン引数より与える
- ファイルには１行に１つ数値が記載されている
- 出力ファイルの一行目に，最大値と最小値を記載する．
- 最大値と最小値の間には，スペースを一つだけ指定する

実行コマンドは以下の通り．(file_in.txt, file_out.txt は，適当なファイル名)

```
$ python   exer3.py   file_in.txt  file_out.txt
```

実行例は，後述．

ファイル入出力のやり方は，雛形に書いてあるので，参考にしてください

課題4. 画像のグレースケール化 - 雛形 exer4.py

画像データを読み込み，グレースケール化して保存せよ

- 入力ファイル名，出力ファイル名は，コマンドライン引数より与えられることとする
- グレースケール値は，赤・緑・青チャンネルの平均を利用することとする

  $I = (r+g+b)/3$

実行コマンドは以下の通り．(file_in.txt, file_out.txt は，適当なファイル名)

```
$ python   exer4.py   file_in.png  file_out.png
```

実行例は，後述.

画像の入出力・画素値へのアクセス方法は雛形に書いてあるので，参考にしてください

課題5. 画像の2値化 - 雛形 exer5.py

画像データを読み込み，グレースケール化した後，与えられた閾値により二値化し，その画像を保存せよ

- 入力ファイル名，出力ファイル名，閾値は，コマンドライン引数により与えられるとする
- 画素値が閾値以上のなら，その画素値を255にする
- 画素値が閾値より小さいなら，その画素値を 0 とする

実行コマンドは以下の通り．(file_in.txt/file_out.txt は適当なファイル名，150は閾値)

```
$ python   exer5.py   file_in.png  150  file_out.png
```

実行例は，後述.

画像の入出力・画素値へのアクセス方法は雛形に書いてあるので，参考にしてください

実行例(テストに利用してください)

課題2

```
$ python   exer2.py   20
1
2
3
hoge
fuga
6
7
hoge
9
fuga
11
hoge
13
14
fuga
hoge
17
18
19
hogefuga
```

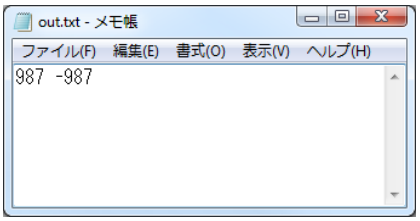
課題3

basicフォルダ内のexer2test.txt
に対して実行すると以下のファイ
ルが出力されます

実行コマンド

```
$ python   exer3.py   basic¥exer2.txt basic¥exer2out.txt
```

出力



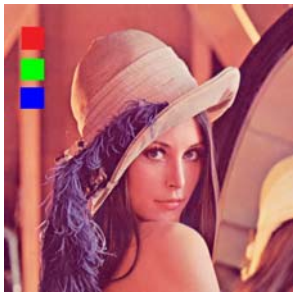
実行例

課題4

basicフォルダ内のimg.pngに対
して実行すると，同一フォルダ内
のimg_gray.pngが出力されます

実行コマンド

```
$ python   exer4.py   basic¥img.png basic¥img_gray.png
```



./basic¥img.png
入力

./basic¥img_gray.png
出力

課題5

basicフォルダ内のimg.pngに対して閾値を
150として実行すると，同一フォルダ内の
img_bin.pngが出力されます

実行コマンド

```
$ python   exer4.py   basic¥img.png 150 basic¥img_bin.png
```

./basic¥img.png
入力

./basic¥img_bin.png
出力

フィルタ処理
課題6~10

課題6. 色の変換（exer6.py）

カラー画像を読み込み，画像の赤と青チャンネルを入れ替えた
画像を保存するプログラムを作成せよ

ファイル名はexer6.pyとし，実行コマンドは以下の通りとする

```
$python   exer6.py   file_in.png   fname_out.png
```

- file_in.png, file_out.txt は，入出力ファイル名
- コマンドライン引数の与え方はひな形を参照

課題7. ガウシアンフィルタ（exer7.py）

画像を読み込み，グレースケール画像に変換後，ガウシアンフィルタを掛けた画像を保存するプログラムを作成せよ

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

フィルタの係数

ファイル名は exer7.py とし，実行コマンドは以下の通り

```
$python  exer7.py  fname_in.png  fname_out.png
```

- グレースケール化の方法についてはひな形を参照
- 右図のガウシアンフィルタを利用すること
- 画像の周囲1pixelは計算せず0を入れること
- 今回はプログラミング練習が目的なので，次のフィルタ関数『cv2.filter2D() / cv2.GaussianBlur() / cv2.Sobel() / np.convolve() 』は利用しないこと

課題8. ソーベルフィルタ（exer8.py）

画像を読み込み，グレースケール画像に変換後，横ソーベルフィルタを掛けた画像を保存するプログラムを作成せよ

-1	0	1
-2	0	2
-1	0	1

- ファイル名は exer8.py とし，実行コマンドは以下の通り

```
$python  exer8.py  fname_in.png  fname_out.png
```

- 画像の周囲1pixelは計算せず0を入れること
- フィルタ適用後，負値となる画素は －1 倍して正値に変換すること
- フィルタ適用後，値が255を超える画素には255を代入すること
- 今回はプログラミング練習が目的なので，次のフィルタ関数『cv2.filter2D() / cv2.GaussianBlur() / cv2.Sobel() / np.convolve()』は利用しないこと

課題9. 勾配強度画像の作成（exer9.py）

画像を読み込み，グレースケール画像に変換後，勾配強度画像を計算し保存するプログラムを作成せよ

ファイル名は exer9.py とし，実行コマンドは以下の通りとする

```
$python  exer9.py  fname_in.png  fname_out.png
```

-1	0	1
-2	0	2
-1	0	1

横方向
ソーベルフィルタ

1	2	1
0	0	0
-1	-2	-1

縦方向
ソーベルフィルタ

- 画像の周囲1pixelは計算せず0を入れること
- ある画素の勾配強度は $I = \sqrt{f_x^2 + f_y^2}$ とする．ただし，f_xとf_yはそれぞれ横方向・縦方向のソーベルフィルタの応答である
- 勾配強度を計算後、画素値が255を越えていたら255を代入すること

※今回はプログラミング練習が目的なので，次のフィルタ関数『cv2.filter2D() / cv2.GaussianBlur() / cv2.Sobel() / np.convolve()』は利用しないこと

課題10. メディアンフィルタ（exer10.py）

画像を読み込み，グレースケール画像に変換後，メディアンフィルタを掛けた画像を保存するプログラムを作成せよ

- ファイル名は exer10.py とし，実行コマンドは以下の通り

```
$python  exer10.py  fname_in.png  fname_out.png
```

- np.arrayには，平均・分散・中央値を求める関数があるので活用すること
- フィルタサイズは**5x5**とする
- 画像周囲の2画素分は，計算せず0を入れておくこと
- 今回はプログラミング練習が目的なので，次のフィルタ関数『cv2.medianBlur』は利用しないこと

参考 (出力結果の例) ※ファイルは ./filter ディレクトリにあります

課題6

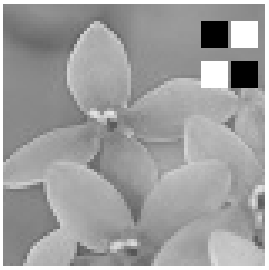
```
$python  exer6.py  filter/img.png  filter/img_flip.png
```

入力画像
./filter/img.png

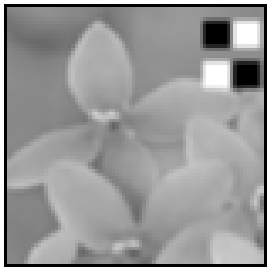
出力画像
./fliter/img_flip.png

課題7

```
$python  exer7.py  filter/img_small.png  filter/img_small_gauss.png
```



入力画像
./filter/img_small.png

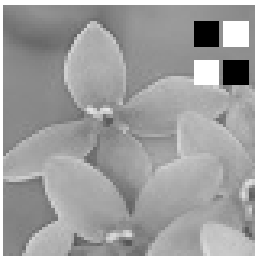


出力画像
./filter/img_small_gauss.png

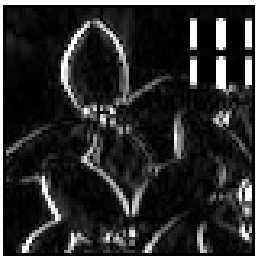
参考 (出力結果の例) ※ファイルは ./filter ディレクトリにあります

課題8

```
$python  exer8.py  filter/img_small.png  filter/img_small_sobel.png
```



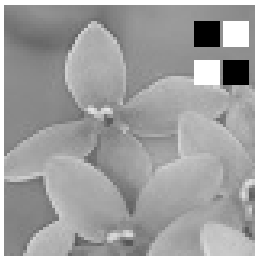
入力画像
./filter/img_small.png



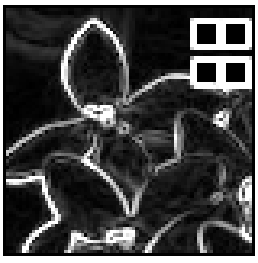
出力画像
./filter/img_small_sobel.png

課題9

```
$python  exer9.py  filter/img_small.png  filter/img_small_gm.png
```



入力画像
./filter/img_small.png



出力画像
./filter/img_small_gm.png

参考 (出力結果の例) ※ファイルは ./filter ディレクトリにあります

課題10

```
$python   exer10.py   filter/img_noise.png   filter/img_median.png
```

入力画像

./filter/img_noise.png

出力画像

./filter/img_median.png

Salt and pepper noiseのようなノイズに対して
Median filterは堅固に動きます。
この例では画像解像度が低いので、
5x5の窓は大きすぎるかもしれないですね。。

ハーフトーン処理
課題11~13

課題11. モザイク画像作成（exer11.py）

カラー画像を読み込み，モザイク画像を作成するプログラムを作成せよ

- ファイル名はexer11.pyとし，ファイル名・モザイクサイズ*N*を以下の通りコマンドライン引数として取得せよ

```
$python  exer11.py  fname_in.png  N  fname_out.png
```

- モザイク画像生成の際，画像を*N* x *N*のブロックに分割し，各ブロックをその平均画素値で塗るものとする.
- 画像の右端・下端に割り切れないブロック（サイズが*N*に満たない領域）が発生する場合も，その領域を平均画素値で塗ること.

課題12. ハーフトーン – ティザ法（exer12.py）

画像を読み込み，グレースケール画像に変換後，ティザ法により
濃淡を維持した二値画像を作成・保存せよ

- ファイル名は exer12.py とし，実行コマンドは以下の通り

```
$python  exer12.py  fname_in.png  fname_out.png
```

- 出力画像は2値画像（0か255）とする
- ブロックサイズは４とし，右図のティザパターンを利用すること
- 画素値xは y=x*16/255 と値を変換してからティザパターンと比較すること
- 画像の幅・高さが4の倍数でない場合，画像の右端・下端に余る領域が発生する．この領域は計算せず0を代入するか，そのままティザパターンを重ね合わせて計算すること

15	7	13	5
3	11	1	9
12	4	14	6
0	8	2	10

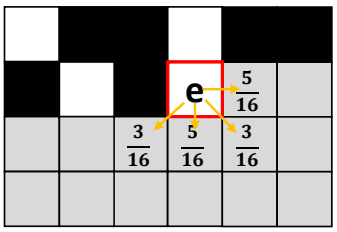
ティザパターン
講義資料とは少し違うので注意

課題13. ハーフトーン – 誤差拡散法（exer13.py）

画像を読み込み，グレースケール画像に変換後，誤差拡散法により濃淡を維持した二値画像を作成・保存せよ

- ファイル名は exer13.py とし，実行コマンドは以下の通り

```
$python  exer13.py  fname_in.png  fname_out.png
```



誤差拡散する隣接画素

- 出力画像は2値画像(0か255)とする
- 画素値+誤差値が **127より大きい**時にその画素を白（255）とせよ
- 拡散する誤差の割合は右図の通りとする
- 右端の画素を計算する際，その右隣の画素が存在しないため右へ誤差を拡散できない．この場合は，単純に右隣への誤差拡散を省略せよ．拡散係数を変化させるなどは行わなくてよい．
- 下端の画素の誤差拡散についても同様．

参考 (出力結果の例) ※ファイルはhtディレクトリにあります

課題11

```
$python  exer11.py  ht/img1.png  ht/img_mosaic.png
```

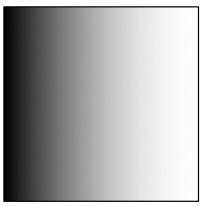
入力画像
./ht/img1.png

出力画像
./ht/img_mosaic.png

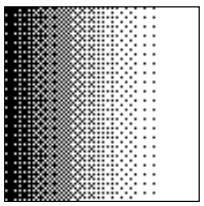
参考 (出力結果の例) ※ファイルはhtディレクトリにあります

課題12, 例1

```
$python  exer12.py  ht/grd.png  ht/grd_tiza.png
```



入力画像
./ht/grd.png

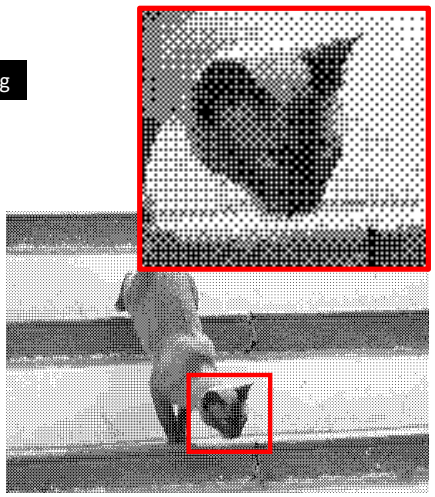


出力画像
./ht/grd_tiza.png

課題12, 例2

```
$python  exer12.py  ht/cat.png  ht/cat_tiza.png
```

入力画像
./ht/cat.png

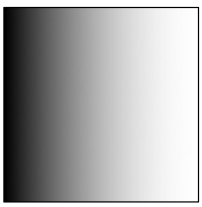


出力画像
./ht/cat_tiza.png

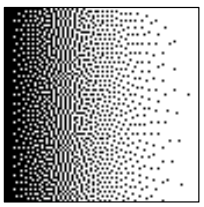
参考 (出力結果の例) ※ファイルはhtディレクトリにあります

課題13, 例1

```
$python  exer13.py  ht/grd.png  ht/grd_err.png
```



入力画像
./ht/grd.png

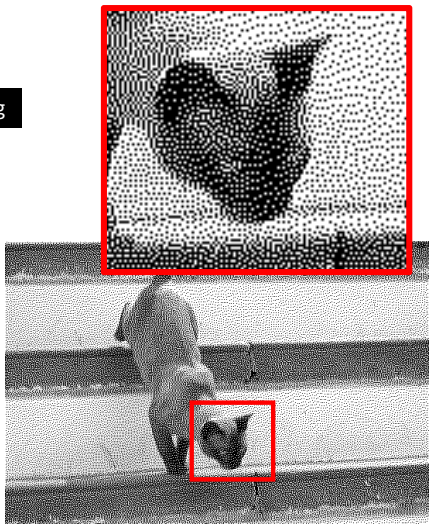


出力画像
./ht/grd_err.png

課題13, 例2

```
$python  exer13.py  ht/cat.png  ht/cat_err.png
```

入力画像
./ht/cat.png



出力画像
./ht/cat_err.png

※誤差拡散法ではグリッドパターンが見えないので，よりきれいな結果が得られます

周波数フィルタ
課題14~16

課題14. フーリエ変換1D（exer14.py）

実数列が書き込まれたテキストファイルを読み込み，その実数列をフーリエ変換した結果をテキストファイルとして出力せよ

ファイル名はexer14.pyとし，入出力の詳細は雛形と出力例を参照せよ

```
$python   exer9.py   sample_fl.txt     fname_out.txt
```

- 得られる周波数係数 F_k は複素数となる．Pythonには複素数型が存在するがこれは利用せず，$F_k = R_k + I\ I_k$ と実部R_kと虚部I_kに分けて保持し，それぞれをテキスト形式で出力せよ
- フーリエ変換には複数の定義が存在するが以下のものを利用すること

フーリエ変換　$F_k = \frac{1}{N}\sum_{l=0}^{N-1} f_l \left(\cos\frac{2\pi kl}{N} - i \sin\frac{2\pi kl}{N} \right)$

課題15. 逆フーリエ変換1D（exer15.py）

複素数列が書き込まれたテキストファイルを読み込み，その配列を逆フーリエ変換した結果をテキストファイルとして出力せよ

ファイル名はexer10.pyとし，入出力ファイル形式は雛形と入出力例を参照のこと

```
$python  exer10.py  sample_Fk.txt  fname_out.txt
```

- フーリエ変換には複数の定義が存在するが以下のものを利用すること

逆フーリエ変換　$f_l = \sum_{k=0}^{N-1} F_k \left(\cos\frac{2\pi kl}{N} + i\sin\frac{2\pi kl}{N} \right)$

- 入力される配列F_kと，得られる配列 f_l は複素数となる．Pythonには複素数型が存在するがこれは利用せず，$f_l = r_l + i_l$ と実部r_lと虚部i_lに分けて保持し，それぞれをテキスト形式で出力せよ
- 課題9で作成したデータを逆フーリエ変換し，ほぼ元に戻ることを確認せよ（虚部はほぼ 0 になる）

課題16. フーリエ変換2D（exer16.py）

画像を読み込み，グレースケール変換後，画像 f_{ij} をフーリエ変換し，フーリエ係数F_{kl}を画像として出力せよ

ファイル名はexer16.pyとし，入出力ファイルの詳細は雛形を参照せよ

```
$python  exer16.py  fname_in.png  Rkl.png  Ikl.png
```

- 以下の式を利用すること

フーリエ変換：　$F_{uv} = \frac{1}{WH} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} f_{xy} e^{-\frac{2\pi xu}{W}i} e^{-\frac{2\pi yv}{H}i}$

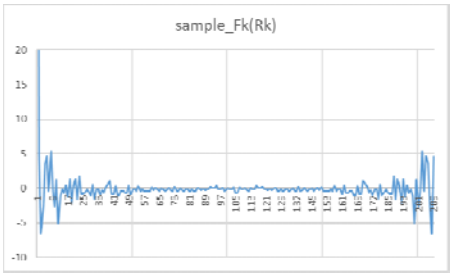
- 結果は$F_{uv} = R_{uv} + i\, I_{uv}$ と実部と虚部に分け，それぞれを画像2枚として出力せよ
- 実部R_{uv}と虚部I_{kl} は，値域[0,255]の範囲に収まらないため，最小値と最大値を用いて，[0,255]に正規化すること（※ R_{uv}とI_{kl}は個別に正規化すること）
- 素朴な実装は処理時間が長くなるので，小さな画像でテストするとよい

参考 (出力結果の例) ※ファイルはfourieディレクトリにあります

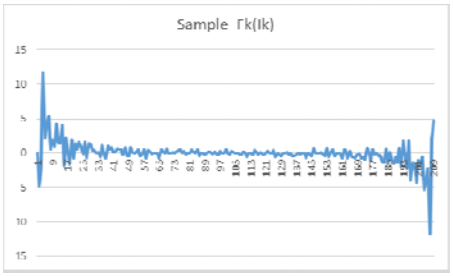
課題14, sample_fl.txt 内の実数配列をフーリエ変換すると，sample_Fk.txt(複素数配列)が得られる

```
$python   exer14.py   fourie/sample_fl.txt   fourie/sample_Fk.txt
```

入力実数列



出力の実部
（見やすさのため値域を[-20,20]にした）

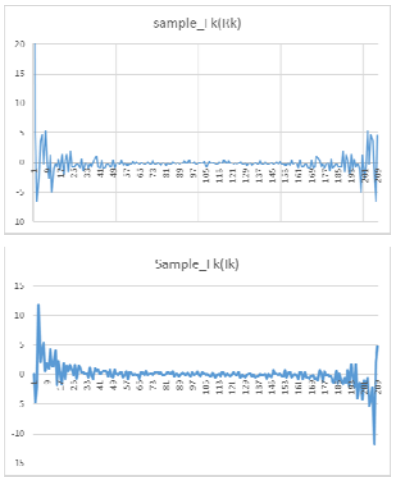


出力の虚部
（見やすさのため値域を[-15,15]にした）

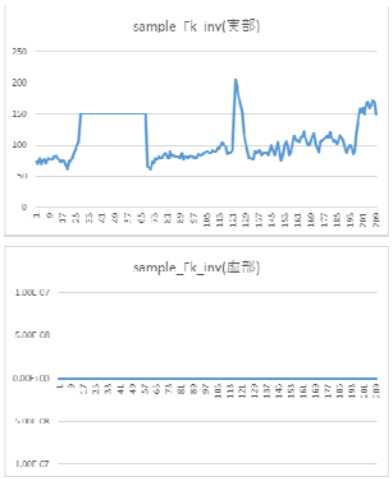
参考 (出力結果の例) ※ファイルはfourieディレクトリにあります

課題15, 問14で得られたsample_Fk.txt 内の複素数配列を逆フーリエ変換すると，
sample_Fk_inv.txt (複素数配列)が得られる

```
$python   exer15.py   fourie/sample_Fk.txt   fourie/sample_Fk_inv.txt
```



入力複素数列
（実部（上）と虚部（下））



出力複素数列
（実部（上）と虚部（下））

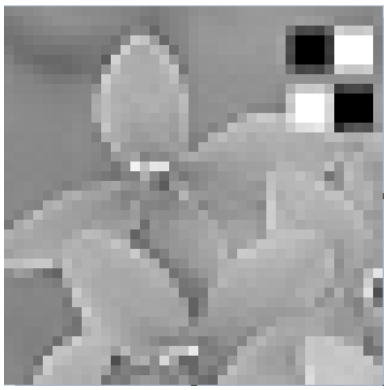
元の実数列（sample_fl）を
フーリエ変換したもの(sample_Fk)を
逆フーリエ変換すると(sample_Fk_inv)
元の関数に戻ります。

Sample_Fk_invの虚部には非常に小さな
値が入ります

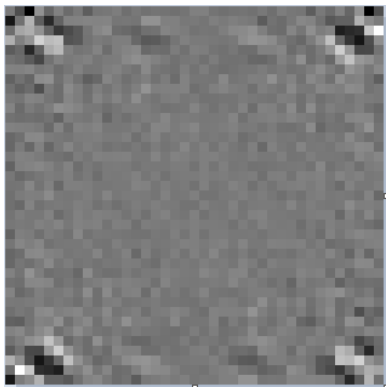
参考 (出力結果の例) ※ファイルはfourieディレクトリにあります

課題16, fourie/img.png フーリエ変換すると，複素数画像 (Rvu + i Ivu)が得られる．実部と虚部それぞれを画像として出力したものが Rvu.pngとIvu.png

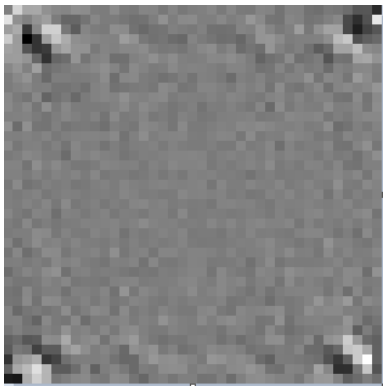
```
$python   exer15.py   fourie/img.png   fourie/Rvu.png  fourie/Ivu.png
```



img.png



Rvu.png



Ivu.png

素朴な実装をすると4重ループになり，pythonではそれなりの時間がかかります。

発展課題
課題17~18

発展問題（作成中）

- 周波数フィルタリング （大切&面白いが，数学の知識がないと面白くない）
- csv名簿のフォーマット変換（すごい役に立つけど面白くはない）
- 自由に美顔フィルタを作る（採点つらい）
- Flood fillで迷路を解く（簡単で面白い、発展課題ではないか…）
- 最短路探索（Dijikstra法）
- 二分探索関連
- ソート関連