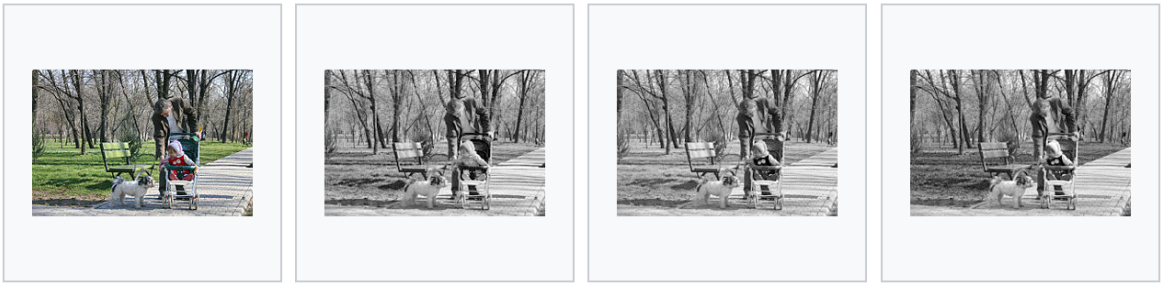
Image Processing

Not really image processing.

Intro

> “
>
> Color digital images are made of pixels, and pixels are made of combinations of primary colors represented by a series of code. A channel in this context is the grayscale image of the same size as a color image, made of just one of these primary colors. For instance, an image from a standard digital camera will have a red, green and blue channel. A grayscale image has just one channel.

For example,



A 24-bit RGB image | The red channel, displayed as grayscale | The green channel, displayed as grayscale | The blue channel, displayed as grayscale

For convenience, we only deal with grayscale images, which can be represented by a matrix of grayscale values. Each grayscale value indicates a pixel of the image.

For example, this image

can be represented by

```
253 253 253 253 253 253 253 253 254 254 247 254 254 254 252 252 248 179 129 155 180 196 234 241 249 254 248 251 254 249 254 247
253 253 253 253 253 253 253 253 247 254 254 254 250 246 254 243 183  85  73 117 128 144 149 197 200 229 249 250 254 250 250 254
253 253 253 253 253 253 253 253 254 248 251 254 251 254 230 131 112 113  57 110 127  93  86 121 115 117 191 247 247 254 251 247
253 253 253 253 253 253 253 253 248 254 254 251 252 226 147  59 110 127 136 197 194 207 211 170  99 101  70 138 254 249 251 254
253 253 253 253 253 253 253 253 254 247 250 254 230 135  53  69 141 192 233 250 247 252 247 248 228 165 115  77 174 249 250 252
253 253 253 253 253 253 253 253 252 254 253 249 215  94  28 107 212 254 244 251 254 245 253 248 240 200 156  67  70 242 253 250
253 253 253 253 253 253 253 248 254 244 254 237  73  57 214 243 254 254 246 251 248 254 252 242 218 175  84  28 211 243 251
253 253 253 253 253 253 253 253 254 245 254 253 181  67 146 242 252 249 254 253 252 254 254 245 246 234 196 125  52 120 254 252
253 253 253 253 253 253 251 249 252 254  88  62 212 249 251 254 246 247 254 250 246 242 231 216 195  94  48  38 251 251
253 253 253 253 253 253 253 253 254 254 254 188  32  93 226 238 233 214 188 185 226 251 224 128  76  96  99  63  30  31 198 248
253 253 253 253 253 253 253 253 254 253 252 170  34  76 222 245 229 224 184 151 203 244 164  31  89 129  52  29  22  12 161 249
254 254 254 254 254 254 254 254 252 254 246 137  26  70 227 231 153 109  97 141 222 253 158  97 116  61  31  29  31  16 127 254
254 254 254 254 254 254 254 254 254 254 250 254 109  25  74 229 244 204 186 211 238 252 254 210 175 226 194 131 135 103  12 134 251
254 254 254 254 254 254 254 254 254 254 248 113  18  95 249 251 254 254 252 245 250 236 189 128 240 244 239 211 110  45 136 252
254 254 254 254 254 254 254 254 254 249 254 254 114 168 131 225 248 251 251 253 251 196 121  53  68 222 236 229 152  57  25  37 224
254 254 254 254 254 254 254 254 254 247 247 111 135 167 217 234 244 243 254 249 187 195  80  49 189 222 171  80  24  26  26 205
253 253 254 254 254 254 254 254 251 254 221  99 222 186 199 214 231 244 254 248 245 210 178 174 207 191 127  41  26  21  49 238
253 254 254 254 254 254 254 254 249 254 224 121 196 250 210 215 226 235 250 230 115  61  28  44 143 153  96  50  25  21 107 250
253 254 254 254 254 254 254 254 254 247 175  97 144 230 227 227 240 246 241 139  84  90  90  40  51 133  75  47  31  23 181 254
254 254 254 254 254 254 254 254 251 244 121  73 107  78  91 230 242 233 239 235 125  84  51  33  73 105  73  34  37 157 254 246
254 254 254 254 254 254 254 254 253 213 100  87 111  77  24 184 230 242 237 233 217 178 163 158 127  97  41  32  76 240 246 254
254 254 254 254 254 254 254 253 254 218 151 115  80  93  62 136 191 210 236 243 248 253 248 209 127  66  26  51 140 252 254 247
254 254 254 254 254 254 254 253 253 233 151 113  85  73  97 158 130 129 165 191 194 169 131  99  41  43  57  45 190 251 252 253
254 254 254 254 254 254 253 253 230 200 107  94  72  57  93 149 183  73  73  78  75  64  62  51  55  50  35  86 250 254 250 250
254 253 253 245 254 241 233 230 186 152 110  81  26  50  75 119 228 149  70  55  49  48  48  48  46  36  42 201 253 252 253 251
248 252 254 254 246 194 213 214 201 153 120  49  29  60  92  80 232 222 147  75  46  45  40  40  34  36 121 254 246 247 254 249
254 254 254 246 230 189 195 164 142 134  74  31  25 103 106  90 189 225 207 148  81  48  34  29  32  54 223 250 254 254 245 254
254 249 243 229 228 221 192 148 173 129  62  90 137 183 171 136 207 206 179 150 101  63  43  41  28 107 254 249 248 254 250 249
251 243 222 214 202 198 192 204 176 156 135 183 180 195 224 177 222 216 159 117  94  62  32  35  33  51 214 253 248 254 252 254
250 227 176 199 220 223 232 240 233 228 221 211 193 161 218 164 207 216 180 138  99  67  39  30  30  22  58 123 226 250 249 252
249 242 220 239 247 230 237 229 247 238 203 218 171 133 162 170 204 189 156 123  79  60  48  30  21  17  24  31  65 208 249 254
```

Again, for convenience's sake, we only deal with raw matrices (or 2-dimensional arrays). In this assignment, you need to implement a couple of operations on these matrices. However, you can easily process real images using `PIL` together with `numpy` if you want😄, but unfortunately not in this assignment😢.

⚠️ **The matrices are given as `list`s. However, you should use `numpy` and only `numpy` to deal with matrices in this assignment.**

⚠️ **All numbers in the given matrices are guaranteed to be integers $\in [0, 255]$ and the given matrices are guaranteed to be valid.**

⚠️ **In the outputs, make sure you print your answer as a `list` and the numbers in the `list` are integers.**

⚠️ ⚠️ ⚠️ **You should code in " `numpy` style" to deal with matrices. In other words, you should utilize as much `numpy` features as you can. A criterium is that you should use no more than 2 " `for` loop"s in Task 1 and no more than 4 " `for` loop"s in Task 2. Otherwise you will be graded 0 in the corresponding task. A `while` is also counted as a `for` .**

Task 1

⚠️ **Number of `for` s** $\in [0, 2]$.

In this task, you are going to **rotate**, **crop**, **flip** and **inverse** a given matrix.

Details

Given a matrix, you need to

- **rotate** the original matrix and print the result
- **crop** the original matrix and print the result
- **flip** the original matrix and print the result
- **invert** the original matrix and print the result

Rotate

By "**rotate**", we mean rotating a matrix 90 degrees clockwise.

For example,

```
1   1 2 3                   7 4 1
2   4 5 6         -->       8 5 2
3   7 8 9                   9 6 3
```

Crop

By "**crop**", we mean cropping a matrix by quarter. To put it more straightforward, given a $m \times n$ matrix, we want the **top-left** submatrix with size $\lceil \frac{m}{2} \rceil \times \lceil \frac{n}{2} \rceil$.

For example,

```
1   1 2 3                   1 2
2   4 5 6         -->       4 5
3   7 8 9
```

Flip

By "**flip**", we mean flipping the matrix horizontally.

For example,

```
1   1 2 3                   3 2 1
2   4 5 6         -->       6 5 4
3   7 8 9                   9 8 7
```

Invert

By "**invert**", we mean inverting a matrix in the sense that each element is subtracted by 255.

For example,

```
1    1  2  3                    254 253 252
2    4  5  6          -->       251 250 249
3    7  8  9                    248 247 246
```

In & Out Format

- You should use `eval` to read in the matrix as a `list` .
- There are four lines in the output, which are the results of the four operations. Each line is the output of a `list` .

Examples

Input:

```
1    [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Output:

```
1    [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
2    [[1, 2], [4, 5]]
3    [[3, 2, 1], [6, 5, 4], [9, 8, 7]]
4    [[254, 253, 252], [251, 250, 249], [248, 247, 246]]
```

Task 2

⚠️ **Number of `for` s** $\in [0, 4]$.

In this task, you are going to **sample**, **apply a convolution to** and **compute the histogram of** a given matrix.

Details

Given a matrix, you need to

- **sample** the original matrix and print the result
- **apply a convolution to** the original matrix and print the result
- **compute the histogram of** the original matrix and print the result

Sample

By "**sample**", we mean generating a new matrix from the original one by discarding certain elements. More specifically, we divide a $m \times n$ matrix into many small submatrices of size 2×2, if possible, and only preserve the top-left elements of these submatrices so that we generate a new matrix of size $\lceil \frac{m}{2} \rceil \times \lceil \frac{n}{2} \rceil$.

For example,

```
1     1   2   3   4   5                    1   3   5
2     6   7   8   9   10         -->       11  13  15
3     11  12  13  14  15
4     16  17  18  19  20
```

Convolution

By "**applying a convolution to**", we mean generating a new matrix from the original one by applying a very simple convolution. More specifically, given a $m \times n$ matrix A, the result is matrix B of size $(m-2) \times (n-2)$ such that

$$B_{i,j} = \left\lfloor \frac{A_{i,j} + A_{i,j+1} + A_{i,j+2} + A_{i+1,j} + A_{i+1,j+1} + A_{i+1,j+2} + A_{i+2,j} + A_{i+2,j+1} + A_{i+2,j+2}}{9} \right\rfloor$$

For example,

```
1     1   2   3   4   5                    7   8   9
2     6   7   8   9   10         -->       12  13  14
3     11  12  13  14  15
4     16  17  18  19  20
```

Histogram

By "**compute the histogram of**", we mean generating an 1-dimensional array from the original matrix that stores the numbers of elements in the matrix falling into certain intervals. More specifically, given a matrix A, the result is a list B of length 16 such that

$$B_k = |\{x | x \in [16k, 16k + 16)\}|$$

where x is an element in A and $k = 0, 1, \ldots, 15$.

For example,

```
1    1   2   3   4   5
2    6   7   8   9   10      -->      15  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0
3    11  12  13  14  15
4    16  17  18  19  20
```

In & Out Format

- You should use `eval` to read in the matrix as a `list`.
- There are three lines in the output, which are the results of the three operations. Each line is the output of a `list`.

Examples

Input:

```
1    [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20]]
```

Output:

```
1    [[1, 3, 5], [11, 13, 15]]
2    [[7, 8, 9], [12, 13, 14]]
3    [15, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```