دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

پردازش تصویر
تمرین شماره ۰

آشنایی با ابزارهای برنامه‌نویسی

نگارش
زهرا سالاریان

استاد درس
دکتر حامد آذرنوش

اسفند ۰۱

سوال اول

۱.

```
rand_nums = np.random.uniform(10, 54000, 80)
```

خروجی حاصل:

array([630.51005006, 3646.30876843, 12979.41690935, 28873.11970045, 47454.34415596, 30254.9416315 , 7409.18375697, 34389.5704806 , 20362.23840516, 4764.75689231, 8206.65670055, 23853.08707532, 21926.27069459, 43070.21139602, 48527.33250611, 8060.14178857, 14977.85957418, 10677.56107675, 7232.33511965, 6256.96123948, 52295.65017427, 34514.57515895, 30070.54437482, 16113.04261733, 30025.15740842, 46870.71211332, 21428.675554 , 45291.34851143, 53893.24529434, 30813.45006939, 3430.22179943, 31010.65277053, 5626.01695924, 15109.37840609, 38789.53039708, 24740.35645084, 38704.06376058, 12728.85290987, 38292.86741632, 52009.09825029, 41999.61796869, 28328.35742422, 7838.0619894 , 36581.65963251, 18180.11158381, 24746.07334599, 25721.76220276, 39049.05091389, 11951.35147006, 38345.70225936, 27698.40330488, 7565.78437694, 29575.44206452, 25785.30505828, 28658.89463413, 31115.90334771, 28827.45767799, 32335.91534066, 9891.91264648, 35099.78892001, 2279.81657907, 14097.12341475, 6209.0160616 , 16988.5250478 , 49780.49118822, 28670.38638936, 22824.34578608, 48188.47054313, 40893.14537021, 53688.34780053, 26783.8463014 , 25027.18870528, 38509.56589946, 52428.11939347, 15048.17605075, 23936.26245666, 37232.90241049, 43539.60035522, 18762.84281414, 18153.07590656])

۲.

نوع خروجی تولید:

numpy.ndarray

نوع دادگان آرایه:

dtype('float64')

۳.

گرد کردن داده‌ها به نزدیکترین عدد صحیح:

```
nearest_int = np.rint(rand_nums)
```

خروجی حاصل:

array([631., 3646., 12979., 28873., 47454., 30255., 7409., 34390., 20362., 4765., 8207., 23853., 21926., 43070., 48527., 8060., 14978., 10678., 7232., 6257., 52296., 34515., 30071., 16113., 30025., 46871., 21429., 45291., 53893., 30813., 3430., 31011., 5626., 15109., 38790., 24740., 38704., 12729., 38293., 52009., 42000., 28328., 7838., 36582., 18180., 24746., 25722., 39049., 11951., 38346., 27698., 7566., 29575., 25785., 28659., 31116., 28827., 32336., 9892., 35100., 2280., 14097., 6209., 16989., 49780., 28670., 22824., 48188., 40893., 53688., 26784., 25027., 38510., 52428., 15048., 23936., 37233., 43540., 18763., 18153.])

۴.

بررسی محدودیت‌های نوع داده‌های عددی و میزان اشغال حافظه‌ی آن‌ها:

int:

همان int64 است که اعداد صحیح در بازه $[-۲^{۶۳}, ۲^{۶۳} - ۱]$ را پوشش می‌دهد. واضح است که ۶۴ بیت در حافظه را به اشغال خود در می‌آورد.

int8:

اعداد صحیح در بازه $[-۲^{۷}, ۲^{۷} - ۱]$ را پوشش می‌دهد. ۸ بیت در حافظه را به اشغال خود در می‌آورد.

uint8:

اعداد صحیح در بازه $[۰, ۲^{۸} - ۱]$ را پوشش می‌دهد. ۸ بیت در حافظه را به اشغال خود در می‌آورد.

int16:

اعداد صحیح در بازه $[-۲^{۱۵}, ۲^{۱۵} - ۱]$ را پوشش می‌دهد. ۱۶ بیت در حافظه را به اشغال خود در می‌آورد.

uint16:

اعداد صحیح در بازه $[۰, ۲^{۱۶} - ۱]$ را پوشش می‌دهد. ۱۶ بیت در حافظه را به اشغال خود در می‌آورد.

int32:

اعداد صحیح در بازه $[-۲^{۳۱}, ۲^{۳۱} - ۱]$ را پوشش می‌دهد. ۳۲ بیت در حافظه را به اشغال خود در می‌آورد.

int64:

اعداد صحیح در بازه $[-۲^{۶۳}, ۲^{۶۳} - ۱]$ را پوشش می‌دهد. ۶۴ بیت در حافظه را به اشغال خود در می‌آورد.

float:

همان float64 است که عدد floating point آن در بازه $[-2^{63}, 2^{63} - 1]$ قرار می‌گیرد. ۶۴ بیت در حافظه را به اشغال خود در می‌آورد.

float32:

عدد floating point آن در بازه $[-2^{31}, 2^{31} - 1]$ قرار می‌گیرد. ۳۲ بیت در حافظه را به اشغال خود در می‌آورد.

float64:

عدد floating point آن در بازه $[-2^{63}, 2^{63} - 1]$ قرار می‌گیرد. ۶۴ بیت در حافظه را به اشغال خود در می‌آورد.

با توجه به دامنه اعداد رندومی که تولید کرده‌ایم که مثبت است uint16 مناسب‌ترین انتخاب برای ما می‌باشد که کل اعداد را زیر پوشش می‌دهد.

داده‌های با نوع جدید:

array([631, 3646, 12979, 28873, 47454, 30255, 7409, 34390, 20362, 4765, 8207, 23853, 21926, 43070, 48527, 8060, 14978, 10678, 7232, 6257, 52296, 34515, 30071, 16113, 30025, 46871, 21429, 45291, 53893, 30813, 3430, 31011, 5626, 15109, 38790, 24740, 38704, 12729, 38293, 52009, 42000, 28328, 7838, 36582, 18180, 24746, 25722, 39049, 11951, 38346, 27698, 7566, 29575, 25785, 28659, 31116, 28827, 32336, 9892, 35100, 2280, 14097, 6209, 16989, 49780, 28670, 22824, 48188, 40893, 53688, 26784, 25027, 38510, 52428, 15048, 23936, 37233, 43540, 18763, 18153], dtype=uint16)

۵.

```
reshaped_conv_nums = converted_nums.reshape(8, 10)
```

خروجی حاصل:

array([[631, 3646, 12979, 28873, 47454, 30255, 7409, 34390, 20362, 4765], [8207, 23853, 21926, 43070, 48527, 8060, 14978, 10678, 7232, 6257], [52296, 34515, 30071, 16113, 30025, 46871, 21429, 45291, 53893, 30813], [3430, 31011, 5626, 15109, 38790, 24740, 38704, 12729, 38293, 52009], [42000, 28328, 7838, 36582, 18180, 24746, 25722, 39049, 11951, 38346], [27698, 7566, 29575, 25785, 28659, 31116, 28827, 32336, 9892, 35100], [2280, 14097, 6209, 16989, 49780, 28670, 22824, 48188, 40893, 53688], [26784, 25027, 38510, 52428, 15048, 23936, 37233, 43540, 18763, 18153]], dtype=uint16)

۶.

کمینه:

```
np.min(reshaped_conv_nums)
```

631

بیشینه:

```
np.max(reshaped_conv_nums)
```

53983

۷.

```
converted_int8 = reshaped_conv_nums.astype(np.int8)
```

خروجی حاصل:

array([[106, -89, 19, 117, 13, 109, -105, 119, 26, 15], [-102, 5, -89, -74, -84, 77, 87, 85, 65, -13], [8, 56, -105, 86, -55, -98, 39, -88, 8, -64], [-110, 118, -56, 71, 15, 83, -87, 108, -45, -51], [-107, -39, -53, 112, 39, -73, -31, 46, 51, 77], [37, -

---

61, -34, -14, 32, 75, -106, 100, 101, 114], [-46, 31, 25, 87, -63, -87, -47, 47, 50, -68], [24, -80, 33, 3, -33, -18, -32, -33, -105, -108]], dtype=int8)

به دلیل آنکه دامنه اعداد مثبتی که int8 در بر می‌گیرد بسیار کوچکتر از دامنه اعداد مثبت uint16 است، اعدادی که در خارج از آن بازه قرار دارند به یک عدد منفی یا مثبت که در بازه مجاز int8 قرار دارد مپ می‌شوند. طریقه این تبدیل به این صورت است که ابتدا ۸ بیت دوم عدد حذف شده و ۸ بیت ابتدایی آن باقی می‌ماند. سپس برای آنکه عدد علامت دار شود (منفی یا مثبت) از آن two's complement گرفته می شود و عدد نهایی به دست می‌آید.

۸.

```
C_two = tuple(converted_int8[:, 1])
C_two
```

(-89, 5, 56, 118, -39, -61, 31, -80)

```
R_three = list(converted_int8[2, 2:])
```

```
R_three
```

[-105, 86, -55, -98, 39, -88, 8, -64]

۹.

```
_dict = dict(zip(C_two, R_three))
```

خروجی حاصل:

{-89: -105, 5: 86, 56: -55, 118: -98, -39: 39, -61: -88, 31: 8, -80: -64}

سوال دوم

۱.

تابع نوشته شده برای سوال:

```
def func(seed, dims):
    # Check validity of the inputs
    if not isinstance(seed, int):
        print('seed variable must be an integer')
```

```
    return
if not isinstance(dims, tuple):
    print('dims variable must be a tuple')
    return
# Build initial matrix of zeroes
res_matrix = [ [0] * dims[1] for _ in range(dims[0])]
# Fill outthe matrix based on the calculation
for i in range(dims[0]):
    for j in range(dims[1]):
        if (i, j) == (0, 0):
            res_matrix[0][0] = seed
            continue
        if i - 1 > -1:
            res_matrix[i][j] -= res_matrix[i-1][j]
        if j - 1 > -1:
            res_matrix[i][j] += res_matrix[i][j-1]
        if i - 1 > -1 and j - 1 > -1:
            res_matrix[i][j] -= res_matrix[i-1][j-1]
return res_matrix
```

نمونه خواندن تابع و ورودی دادن به آن:

```
func(1, (3, 4))
```

خروجی حاصل:

[[1, 1, 1, 1],

[-1, -3, -5, -7],

[1, 5, 13, 25]]

سوال سوم

۱.

```
std_num = 9731089
```

۲.

تابع تعریف شده:

```
def generate_circle_matrix(r):
    if r < 3:
        print('r should be larger than 2')
        return
    r -= 1
    matrix = np.fromfunction(lambda i, j: 255*((i-r)**2 + (j-r)**2 <= r**2), shape=(2*r+1,2*r+1), dtype=np.int8)
    return matrix
```

نمونه خواندن و ورودی دادن به تابع:

```
circle_matrix = generate_circle_matrix(4)
```

خروجی حاصل:

```
array([[  0,   0,   0, 255,   0,   0,   0],
       [  0, 255, 255, 255, 255, 255,   0],
       [  0, 255, 255, 255, 255, 255,   0],
       [255, 255, 255, 255, 255, 255, 255],
       [  0, 255, 255, 255, 255, 255,   0],
       [  0, 255, 255, 255, 255, 255,   0],
       [  0,   0,   0, 255,   0,   0,   0]])
```

۳.

تابع نوشته شده:

```
def add_noise(_matrix, high_noise):
    # Generate noises
    noises = np.random.uniform(0, high_noise, size=_matrix.shape)
    # Multipy noise in -1 for 255 values
    adjusted_noises = np.where(_matrix != 0, noises*(-1), noises)
    # Add noises to the original matrix
    noisy_matrix = np.add(_matrix, adjusted_noises)
    # Calc the floor of the values
    res = np.floor(noisy_matrix).astype(_matrix.dtype)
```

```
    return res
```

محاسبه جمع ارقام شماره دانشجویی:

```
std_str = str(std_num)
std_digits_sum = sum(list(map(int, std_str.strip())))
```

مقدار حاصل:

37

فراخوانی و مقداردهی تابع:

```
noisy_matrix = add_noise(circle_matrix, (std_digits_sum % 15) + 20)
```

خروجی حاصل:

```
array([[ 17,   0,  13, 253,   6,  17,  15],
       [ 13, 241, 231, 243, 239, 240,   9],
       [ 21, 240, 242, 242, 249, 228,  26],
       [254, 241, 247, 230, 230, 232, 233],
       [  5, 245, 239, 248, 251, 237,  11],
       [  8, 237, 248, 252, 228, 242,   3],
       [  2,  25,   3, 250,  16,  14,  15]])
```

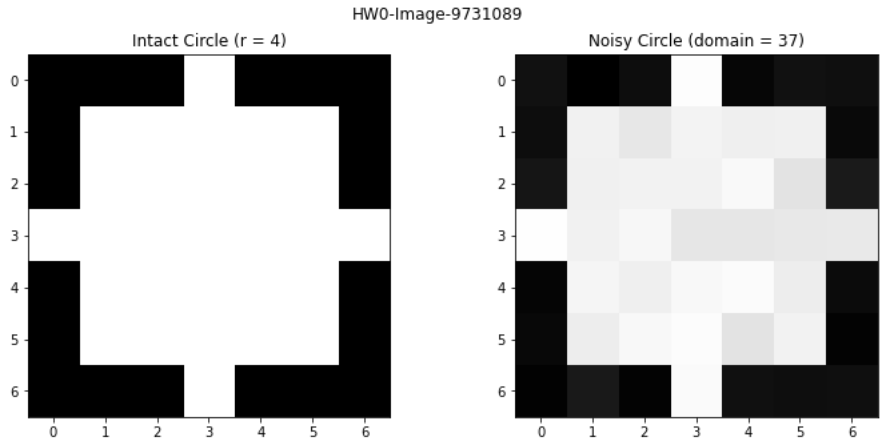
۴.

کد رسم نمودارها:

```
fig, ax = plt.subplots(1, 2, figsize = (12, 5))
fig.suptitle(f'HW0-Image-{std_num}')
ax[0].set_title(f'Intact Circle (r = {r})')
ax[0].imshow(circle_matrix, cmap='gray')

ax[1].set_title(f'Noisy Circle (domain = {std_digits_sum})')
ax[1].imshow(noisy_matrix, cmap='gray', vmin = 0, vmax = 255)
plt.show()
```

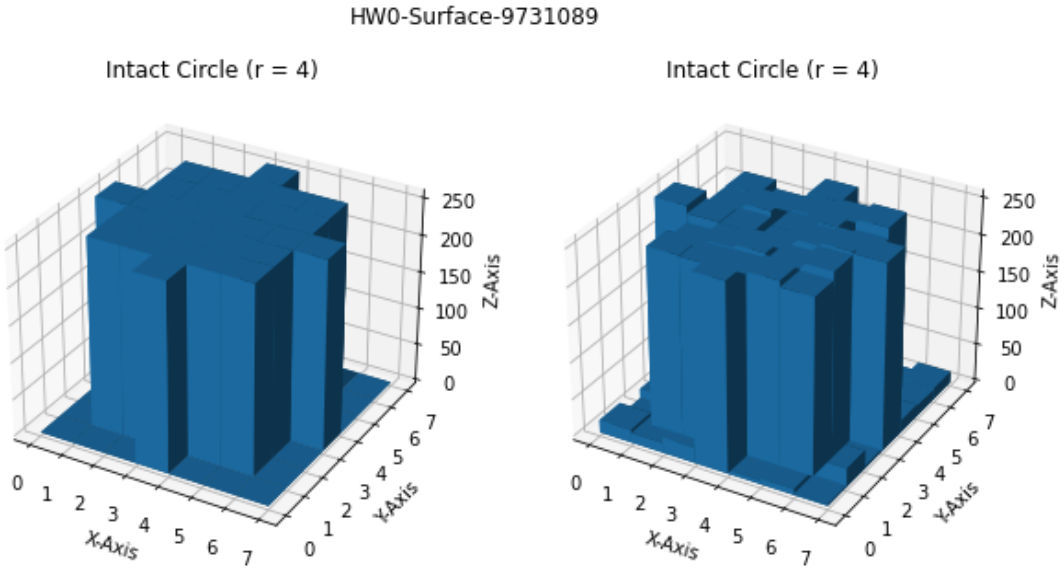
نمودارهای حاصل:



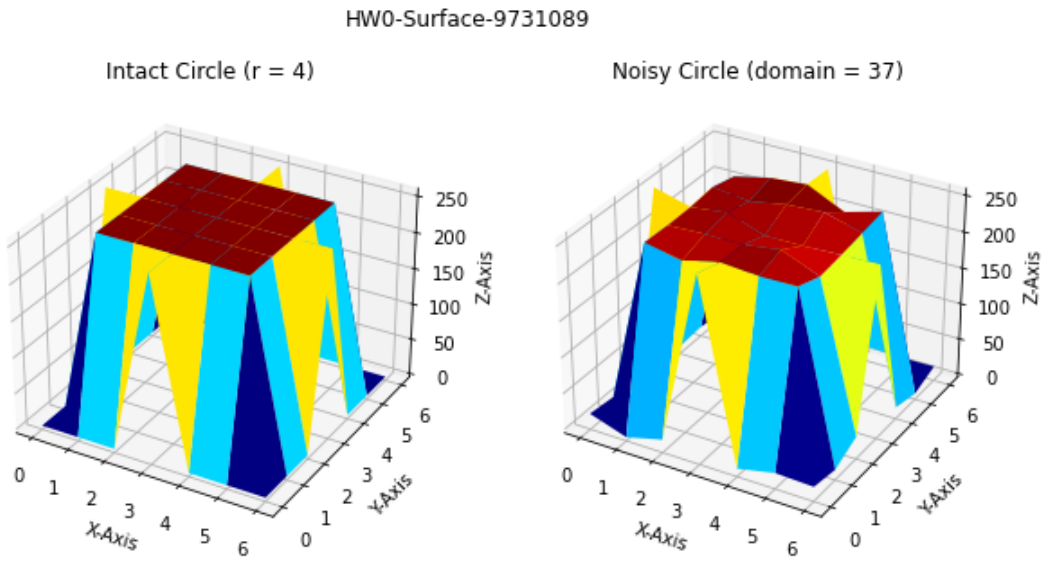
۵.

دو سری نمودار رسم شده است.

نمودارهای سری اول:



نمودارهای سری دوم:



سوال چهارم

۱.

خواندن تصویر:

```
img = cv.imread('chest-xray.png', cv.IMREAD_UNCHANGED)
```

ابعاد تصویر:

(493, 600)

همانطور که میبینیم تصویر از ابتدا دو بعدی بوده است و این یعنی که از ابتدا خاکستری می باشد و نیاز به تغییری ندارد.

۲.

نوع داده ی هر پیکسل:

dtype('uint8')

۳.

میزان حافظه اشغال شده تصویر خاکستری بدون فشرده سازی:

```
print(f'Total memory used by the grayscale image: {img.nbytes / 1000} KB')
```

خروجی حاصل:

```
Total memory used by the grayscale image: 295.8 KB
```

۴.

برش تصویر:

```
cropped_gray_image = img[:, :img.shape[1] // 2]
```

تصویر حاصل:

۵.

تصویر قرینه شده:

```
flipped_x_axis = cv.flip(cropped_gray_image, 0)
```

۶ و ۷.

می دانیم که کمترین مقدار یک پیکسل ۰ است و vmax را هم از روی دامنه نوع داده ها به شکل زیر تعیین می کنیم:

```
vmax = 2^8 - 1
```

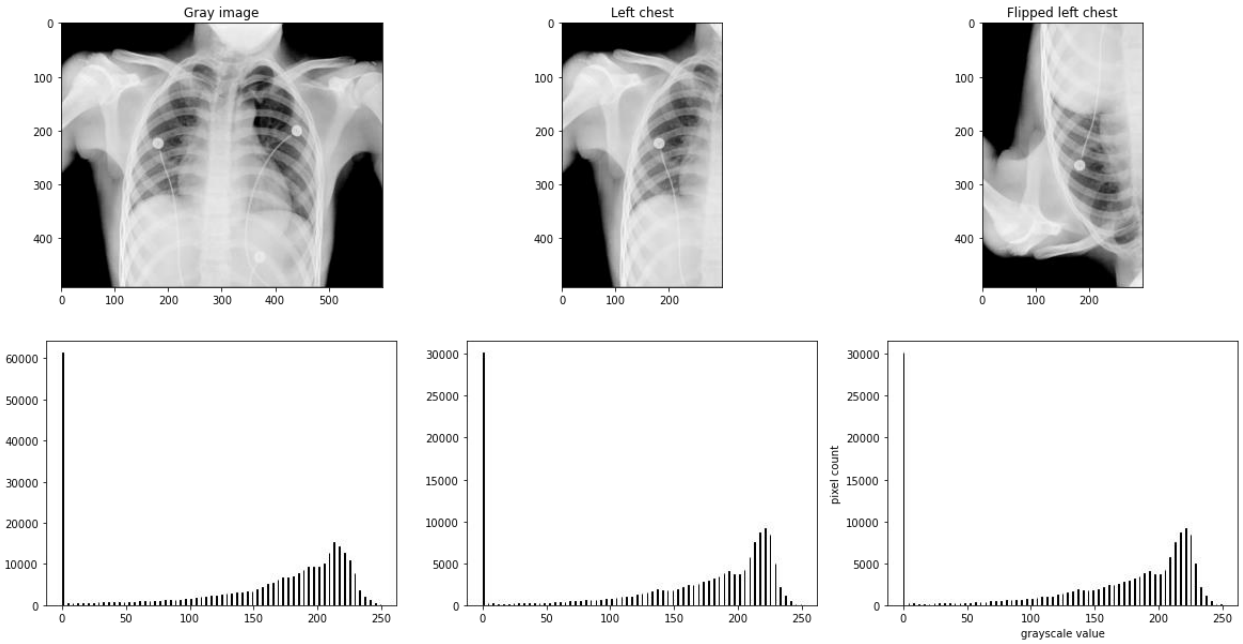
در کد زیر مقدار bins را طوری تعیین کرده ایم که هر ۴ شدت در یک دسته شمرده شوند:

```
hist, bins = np.histogram(img, bins=int((img.max() - img.min())/4), range= (0, (img.max() - img.min())))
```

و در تکه کد زیر نیز عرض نسبی هر میله را ۰.۶ مشخص کردیم:

```
ax[1][0].bar(bins[:-1], hist, width=0.6, align="edge", ec="k", color='red')
```

نمودارهای حاصل:



از نمودارهای حاصل نتیجه میگیریم که تعداد پیکسل هایی که یک مقدار شدت خاص را دارند، با تغییر مکان پیکسل ها تغییری نمیکند و به همین خاطر است که نمودارهای هیستوگرام عکس های دوم و سوم یکسان هستند. همچنین نتیجه می گیریم که تصویر اصلی چون به طور حدودی دو برابر تصاویر بریده شده است، پس فراوانی تعداد پیکسل ها با شدت های مختلف نیز در آن حدودا ۲ برابر عکس های دوم و سوم است. همانطور که میبینید اعداد روی محور y در نمودار تصویر اول دو برابر همین اعداد در نمودارهای تصاویر دوم و سوم است.