**Numpy**

* np.pi : 원주율 (상수)
* np.exp(x) : 𝑒^𝑥 를 구한다
* np.repeat(array, [times], axis = 0) : array를 times 만큼 반복한다.
  + array : 반복할 배열(이미지)
  + 주의 : 슬라이스 사용시, img[0:, 0] : X, img[0:,0:1] : O
  + [times] : 반복할 횟수, [] 미 입력시 요소 단위로 반복
  + axis : 0 : 행 단위 반복, 1: 열 단위 반복. default 0
* np.flip(array, axis = None) : array를 뒤집는다.
  + array : 뒤집을 배열
  + axis
    - 미 입력시 : 행과 열을 모두 뒤집는다
    - Axis = 0 : 행을 뒤집는다.
    - Axis = 1 : 열을 뒤집는다.
* np.mgrid[-m:m+1, -n:n+1] : 두 개의 array를 반환한다.
  + 하나는 행이 -m부터 m까지 1씩 증가하는 배열
  + 하나는 열이 -n부터 n까지 1씩 증가하는 배열.
  + 꼭 알아야 할 필요는 없지만, 알고 있으면 좋다.
* Np.ogrid[-m:m+] : -m~m까지 증가하는 array를 반환한다.
* np.median(array) : array의 median 값을 반환한다.

**Array member**

* arr.shape : array의 shape를 나타낸다. 512x512의 BGR image : (512,512,3)
* arr = arr.reshape(row, col)
  + array를 row \* col으로 shape를 바꾼다.
  + Row나 col 위치에 -1을 입력하게되면, 나머지 인자들을 이용해 자동으로 계산한다.
  + Array[10]을 Array.reshape[2, -1] 하게되면 Array[2,5]가 된다.
* arr.copy() : 해당 array를 복사한다.
* arr.sum() : array의 값을 모두 더해 반환한다.

**OpenCV**

* cv2.blur(img, (val1, val2)) : box filter를 적용
  + img : filter를 적용할 이미지
  + (val1, val2) : kernel 크기. Val1 x val2 크기의 kernel 사용.
* cv2.GaussianBlur(img, (val1, val2), sigma) : Gaussian filter를 적용.
  + img : filter를 적용할 이미지
  + (val1, val2) : kernel 크기. Val1 x val2 크기의 kernel 사용.
  + Sigma : 가우시안 분포에서의 표준 편차
* cv2.medianBlur(img, val) : median filter를 적용.
  + Img : filter를 적용할 이미지
  + Val : kernel 크기. Val x val 크기의 kernel 사용.
* cv2.Sobel(img, ddepth, dx, dy, val): sobel 미분을 수행
  + img : filter를 적용할 이미지
  + ddepth : 결과 이미지의 data type. Cv2.CV\_8U, cv2.CV\_32F 등으로 지정
  + dx, dy : x축 y축에 대한 미분 차수.
  + Val : kernel 크기. Val x val 크기의 kernel 사용.

**Time**

* Import time : time 패키지를 import 한다.
* time.perf\_counter() : 현재 시간을 return 한다.

참고만 하세요

* cv2.sepFilter2D(src, ddepth, kernelX, kernelY) : kernelX,와 kernelY을 순차적으로 사용해 filtering한 이미지를 반환한다.
  + Src : 대상 이미지
  + Ddepth : 결과 이미지 data type
    - cv2.CV\_8U, cv2.CV\_16U, cv2.CV\_32F 등
    - -1 입력시 원본과 동일한 자료형으로 반환
  + kernelX, kernelY : 각각 행에 사용될 Kernel, 열에 사용될 Kernel
* cv2.getGaussianKernel(size, sigma): sigma를 표준편차로 가지는 size 길이의 1D gaussian kernel을 반환한다.
  + Shape : 얻어 올 가우시안 커널의 크기.
  + Sigma : 가우시안 분포의 표준편차
  + Return : (size, 1)
* 아래 페이지에 과제 주의사항이 있습니다. -

과제 주의사항

* 과제 자체는 단순히 Kernel의 shape와 시그마를 받아 Kernel을 생성하는 함수, 해당 Kernel과 이미지를 받아 Filtering을 수행하는 함수 두 가지만 구현하면 됩니다.
* 커널의 크기와 이미지의 크기가 충분히 커야합니다.
  + (row, col) 크기의 image, (ksize, ksize) 크기의 Gaussian kerenl을 적용한다고 했을 때.
  + 1D x 1D filtering(1 x ksize) (ksize x 1)
    - 곱셈 연산 횟수 : row \* col \* 2 \* ksize
    - Filtering 연산을 해야하는 횟수 : 2회. (1xksize) filtering 수행 후, (ksizex1) filtering 수행. (for loop, itemset, padding 등 모든 작업을 두 번 수행.)

**(Python 에서는 for loop 속도가 매우 느립니다.)**

* + 2D filtering
    - 곱셈 연산 횟수 : row \* col \* ksize \* ksize
    - Filtering 연산을 해야하는 횟수 : 1회 (ksize x ksize) filtering 수행.
  + 따라서 1D x 1D filtering이 2D filtering보다 연산량이 줄어들기 위해서는 ksize가 충분히 증가해야한다.
  + Ksize만 크면 이미지가 너무 과하게 bluring, 적당히 큰 이미지 사용 권장.
* TA와 같은 방법으로 코드 구성 시, Ksize가 작을수록 2D filtering > 1D x 1D filtering
* Ksize가 커질수록 속도는 1D x 1D filtering > 2D filtering (커널이 커질수록 차이가 커짐)