In [1]:

```
# image -> image (image processing)
# image -> information (computer vision) ML/DL 을 위한 이미지 처리에 집중!

```

In [ ]:

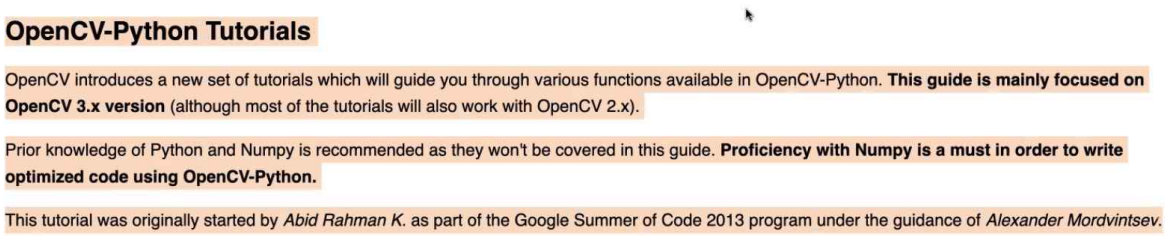
```
# Image가 뭐냐??
# 특정 좌표값에 대한 밝기나 색상을 나타내는 함수로 표현이 가능하다.
# 이미지는 직사각형 형태로 일반적으로 구성하기 때문에,
# 다음으로 나타낼 수 있다.

```

image = f(x,y) = array

In [2]:

```
from IPython.display import Image
Image('image_numpy.jpg')
# 이미지 처리는 numpy를 잘해야 한다... <공식 홈페이지>
```

Out[2]:



OpenCV-Python Tutorials

OpenCV introduces a new set of tutorials which will guide you through various functions available in OpenCV-Python. **This guide is mainly focused on OpenCV 3.x version** (although most of the tutorials will also work with OpenCV 2.x).

Prior knowledge of Python and Numpy is recommended as they won't be covered in this guide. **Proficiency with Numpy is a must in order to write optimized code using OpenCV-Python.**

This tutorial was originally started by *Abid Rahman K.* as part of the Google Summer of Code 2013 program under the guidance of *Alexander Mordvintsev*.

In [ ]:

```
다차원 배열 <연산> (= 연산량이 많아서 느릴 수 밖에 없음)
실질적으로 Numpy 를 모르면 이미지 처리 못함.
```

Numpy 왜 쓴다?

- 빠르기 때문에
- 다차원 배열 연선을 제공해주는 라이브러리인데, 다차원 배열을 elementwise 계산, 수학적인 다차원 배열 연산을 아주 빠르게 해준다.
  - 왜 Numpy가 빠르냐고 효율적이냐? ;
    - 여러개를 한꺼번에 계산시켜줌 (CPU 최대한 활용; vectorization)
    - 속도가 필요한 부분은 C로 만들고 C로 만든 것을 불러온다 -> 내부적으로 C로 만듦
    - 알고리즘과 데이터 구조를 효율적으로 한다 (Homogeneous한 효율적인 자료구조)
- 풍부한 기본 기능 -> 다 만들어놓아서 쉽게쉽게 가져다 쓸 수 있다.

현대적인 관점에서 문제!!

- DL / ML 기능은 지원하지 않는다 -> 직접 구현하면 되지 않나? -> numpy는 CPU를 쓰는데 요즘은 속도가 빠른 GPU를 사용한다.

- ++ Deep Learning, GPU 기능
  - Pytorch -> numpy 기반으로 Tensor
  - Tensorflow -> numpy 기반
- 일반적인 ML 플랫폼(scikit-learn)에서는 GPU 쓰지 않아도 지원이 가능하다.
- 어쩔 때 GPU 연산을 하냐?
- Numpy는 다음과 같은 기능을 제공해서 빠르다.
  - vectorization : 4개의 CPU -> 짝을 이루어서 연산(elementerize)을 한다. (밑 그림 참조) -> **좌측은 5번, 우측은 2번만에 끝낸다.**
  - 기본적인 CPU 기반의 vectorization(CPU 갯수만큼; 하드웨어와 결합한 vectorization)의 기능을 제공해서 속도가 빠르다.
  - 일반적인 ML에서는 CPU만 사용해도 충분!
  - Tensor, Pytorch는 DL에 주로 사용하고, 연산량이 많이 필요하므로 GPU 기반의(GPU 갯수만큼) vectorization 제공하며(처음에 리소스가 더 많이 쓰인다) 더 속도가 빠르다

In [4]:

```
1 Image('vectoriztion.JPG')
```

Out[4]:

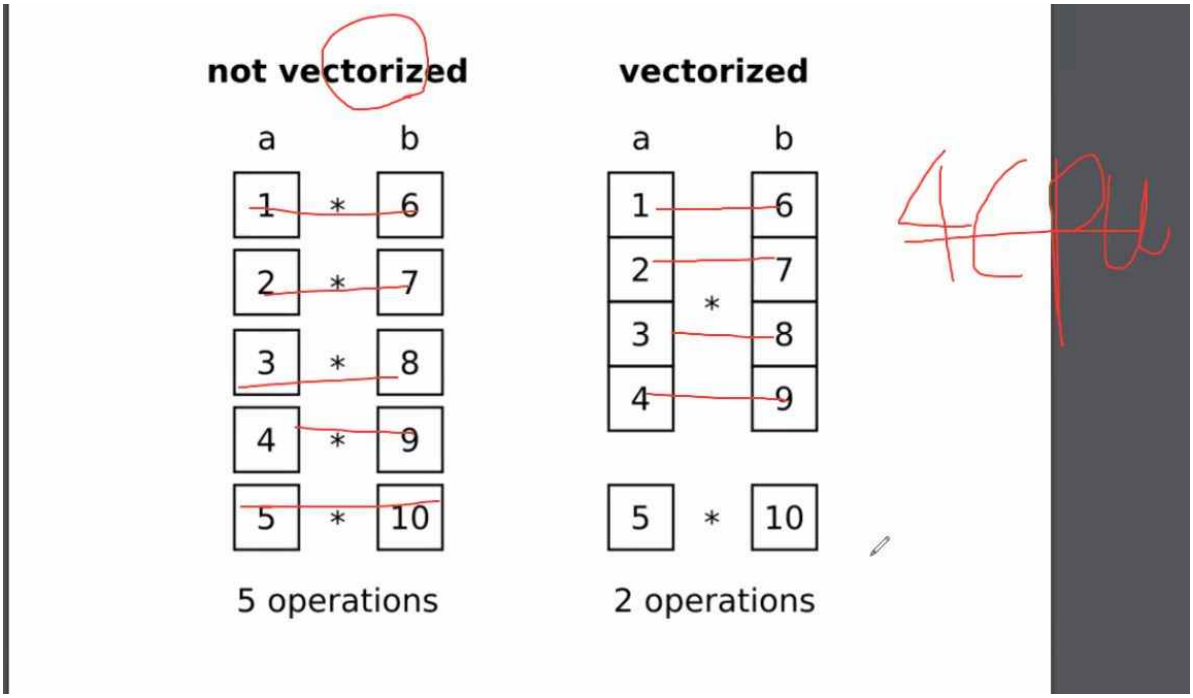


image = file > array(numpy)기반으로 만들어주는 것이 중요하다.

- 예시사진은 위와 동일

In [6]:

```
1 import PIL # (Python Image Library)
```

In [7]:

```
1 from PIL import Image
```

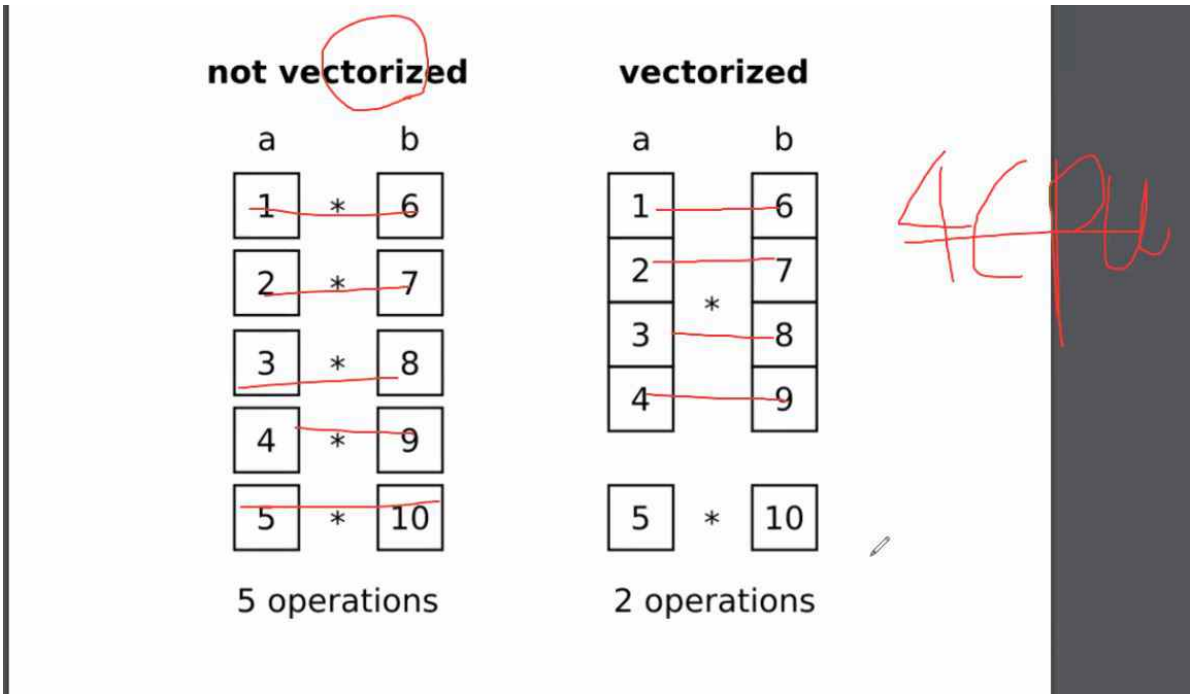
In [13]:

```
1 im = Image.open('vectoriztion.JPG')
```

In [19]:

```
1 im
```

Out[19]:



In [20]:

```
1 dir(im) # __array_interface__ 가 있으므로, numpy 와 호환된다
```

Out[20]:

```
['_Image__transformer',
 '__array_interface__',
 '__class__',
 '__copy__',
 '__del__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__enter__',
 '__eq__',
 '__exit__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
```

In [23]:

```
import numpy as np
import tensorflow as tf
import torch
```

In [26]:

```
b = tf.constant([1,2,3])
b
```

Out[26]:

```
<tf.Tensor: shape=(3,), dtype=int32, numpy=array([1, 2, 3])>
```

In [27]:

```
c = torch.Tensor([1,2,3])
dir(c)
```

Out[27]:

```
['H',
 'T',
 '__abs__',
 '__add__',
 '__and__',
 '__array__',
 '__array_priority__',
 '__array_wrap__',
 '__bool__',
 '__class__',
 '__complex__',
 '__contains__',
 '__deepcopy__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__div__',
```

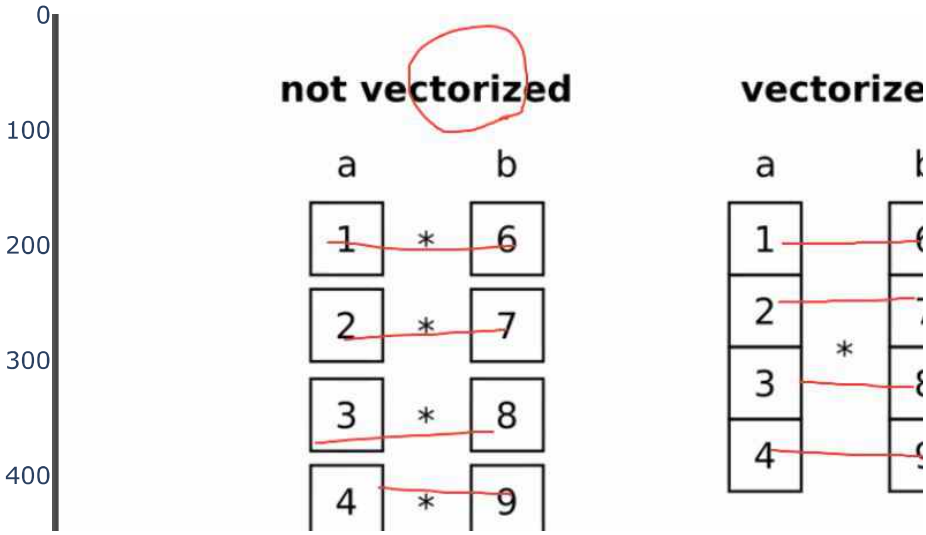
dir()로 확인해서 array, array_interface 가 있고, 호출 시 객체에 array가 있으면 numpy와 호환되어진다! >> image의 데이터를 numpy로 np.array() 로 만들 수 있다

In [31]:

```
import cufflinks as cf
import chart_studio.plotly as py
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import plotly.express as px

cf.go_offline(connected= True)
```

In [32]:

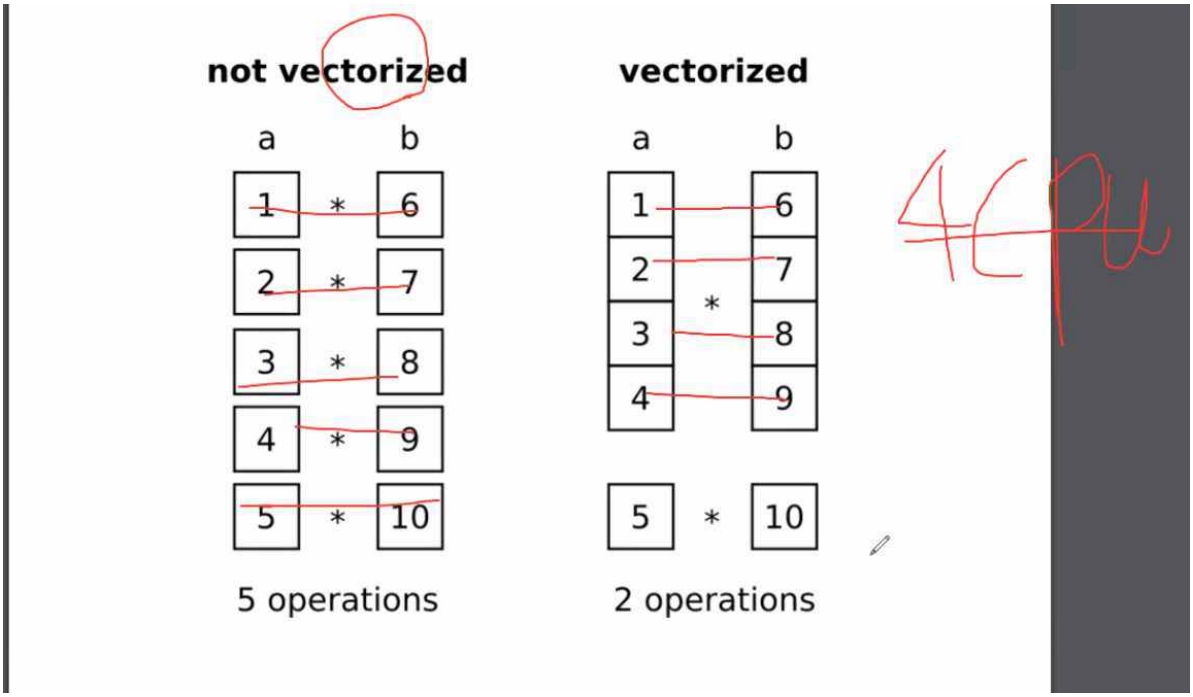
```
1 fig = px.imshow(im)
2 fig.show()
```



In [93]:

```
1 im
```

Out[93]:

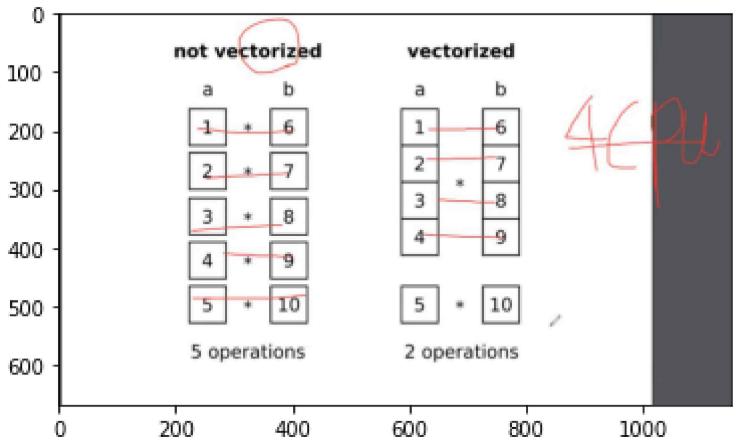


In [33]:

```
1 plt.imshow(im)
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x18bd6160e08>
```



In [36]:

```
1 a = np.array(im)
```

In [38]:

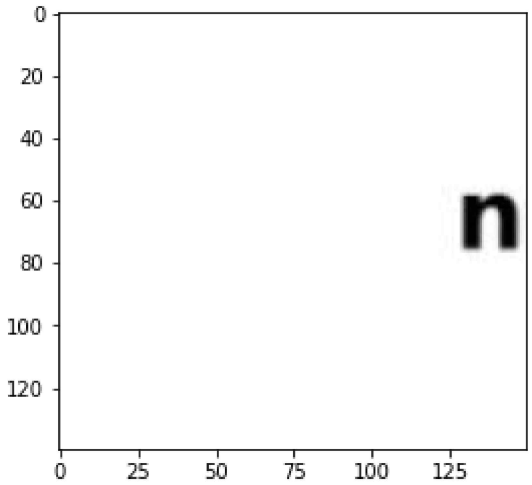
```
1 b = a[:140,70:220:,:]# slicing
```

In [39]:

```
1 plt.imshow(b)
```

Out[39]:

<matplotlib.image.AxesImage at 0x18bd5aea948>



In [65]:

```
fig = px.imshow(b)
fig.show()
```

In [70]:

```
a.shape # 가로 * 세로 * 내부
```

Out[70]:

(669, 1151, 3)

In [71]:

```
a.dtype # dtype('uint8') ; unsigned , 0 ~ 2^8-1 데이터 타입이다.
>> 상황에 맞는 데이터 타입을 결정하는 것이 중요하다 ; 속도와 메모리!!!!
```

Out[71]:

dtype('uint8')

Numpy의 데이터 타입은 np.sctypeDict.keys() 로 확인하자

- 데이터 타입이 속도를 결정!
- 데이터 공간이 작으면 작을 수록 빠르고 데이터가 작게 든다
- 상황에 맞는 데이터 타입을 정하는 것이 중요하다

In [75]:

```
np.sctypes.keys()
# 기본적인 데이터 타입은 파이썬에서 가져온 것도 있고,  C기반으로 가지고 온 것도 있다.
```

Out[75]:

```
dict_keys(['int', 'uint', 'float', 'complex', 'others'])
```

In [77]:

```
np.sctypeDict.keys() # C + python 데이터 타입 다 쓸 수 있음.
# 단축 표현! 너무 데이터 타입이 많으니깐 사용 ('b', 'B' 등등....)
```

Out[77]:

```
dict_keys(['?', 0, 'byte', 'b', 1, 'ubyte', 'B', 2, 'short', 'h', 3, 'ushort', 'H',
4, 'i', 5, 'uint', 'I', 6, 'intp', 'p', 9, 'uintp', 'P', 10, 'long', 'l', 7, 'L', 8,
'longlong', 'q', 'ulonglong', 'Q', 'half', 'e', 23, 'f', 11, 'double', 'd', 12, 'lon
gdouble', 'g', 13, 'cfloat', 'F', 14, 'cdouble', 'D', 15, 'clongdouble', 'G', 16,
'O', 17, 'S', 18, 'unicode', 'U', 19, 'void', 'V', 20, 'M', 21, 'm', 22, 'bool8', 'b
1', 'int64', 'i8', 'uint64', 'u8', 'float16', 'f2', 'float32', 'f4', 'float64', 'f
8', 'complex64', 'c8', 'complex128', 'c16', 'object0', 'bytes0', 'str0', 'void0', 'd
atetime64', 'M8', 'timedelta64', 'm8', 'Bytes0', 'Datetime64', 'Str0', 'Uint64', 'in
t32', 'i4', 'uint32', 'u4', 'int16', 'i2', 'uint16', 'u2', 'int8', 'i1', 'uint8', 'u
1', 'complex_', 'int0', 'uint0', 'single', 'csingle', 'singlecomplex', 'float_', 'in
tc', 'uintc', 'int_', 'longfloat', 'clongfloat', 'longcomplex', 'bool_', 'bytes_',
'string_', 'str_', 'unicode_', 'object_', 'int', 'float', 'complex', 'bool', 'objec
t', 'str', 'bytes', 'a'])
```

In [83]:

```
b.itemsize # 8 bit = 1 byte
```

Out[83]:

```
1
```

In [84]:

```
# image 에서 주로 사용하는 데이터는 uint8 , float32를 많이 쓴다.
```

In [87]:

```
aa = a/255
```

In [89]:

```
aa.dtype
```

Out[89]:

```
dtype('float64')
```

data type의 중요성

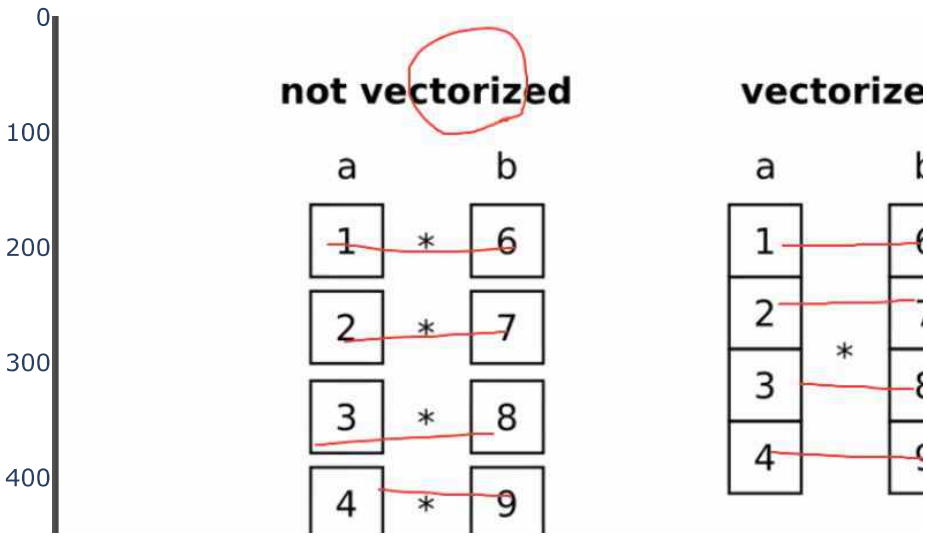
- uint8 , float32 ; 상대적인 intensity로 보기 때문에 위와 똑같이 볼 수 있다.
- 이때 값들은 상대적인 값으로 이미지만 봐서는 사람은 똑같이 볼 수 있다. (255;rgb 갯수 로 나눈 값과 그냥 값)

- 그러나 컴퓨터에게 uint8와 float32의 데이터는 다르다. 딥러닝에서는 연속적인 데이터를 다루기 때문에 float32를 사용한다.
- float 32가 더 효율적이다. 똑같이 보인다

```
- uint8 , float32 ; 상대적인 intensity로 봄
- 이때 값들은 상대적인 값으로 이미지만 봐서는 사람은 똑같이 볼 수 있다.
- 그러나 컴퓨터에게 uint8와 float32의 데이터는 다르다. 딥러닝에서는 연속적인 데이터를 다루기 때문에 float32를 사용한다.
- float 32가 더 효율적이다. 똑같이 보인다
```

In [94]:

```


fig =px.imshow(aa)
fig.show()
# 255로 나눈 것도 똑같이 나온다.
```



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
1
```

In [ ]:

```
1
```