# Day 0

* 컴퓨터 OS
  + 운영체제. 프로그램이 동작할 수 있는 구동 환경
* 파일 시스템
  + 파일을 저장하는 ‘트리구조’ 저장 체계
  + 파일 & 디렉토리
    - 파일은 정보를 저장하는 논리적 단위
    - 디렉토리는 파일 또는 다른 디렉토리를 포함할 수 있다
    - Root 디렉토리 : 가장 최상위의 위치. 경로의 시작점
  + 경로
    - 절대경로와 상대경로
* 터미널
  + Text를 이용하여 컴퓨터에 명령을 입력하는 인터페이스 체계
  + CMD, Windows Terminal 등등
  + 기본적인 명령어들
    - cd, mkdir, clear, cp, rm, ls ……

# Day 01

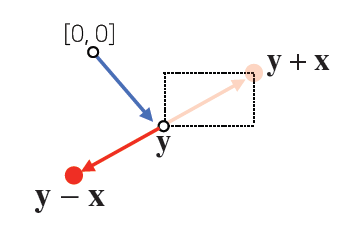
* What is Python?
  + ‘플랫폼 독립적’ ‘인터프리터’ ‘객체지향’ ‘동적 타이핑’ 언어!
    - OS에 상관없는 ‘플랫폼 독립’
    - 소스코드를 컴파일 과정 없이 바로 실행하는 ‘인터프리터 언어’
    - 컴파일러 vs 인터프리터
      * 컴파일
        + 소스코드를 기계어로 번역. 해당 플랫폼에 최적화되어 프로그램이 실행.
        + 따라서 속도가 빠르며, 메모리를 많이 차지하게 됨.
        + C, C++, JAVA, C# 등등
      * 인터프리터
        + 별도의 과정 없이 소스코드를 실행
        + 간단히 작성되며 메모리를 적게 차지. 하지만 실행시점에서 해석하게 되므로 속도는 느림
        + 파이썬, 스칼라 등등
  + 객체 지향
    - 실행 순서가 아닌, 모듈 단위 중심으로 프로그램을 작성하는 방법론
    - 객체는 method와 attribute를 가짐
  + 동적 타이핑
    - 프로그램이 실행하는 시점에서 사용하는 데이터 타입을 결정함
* 개발환경 세팅
  + Python 설치
  + Python miniconda 설치
    - Anaconda prompt 실행하여 시작
  + VS Code 설치
    - Python extension 추가하기
  + Jupyter
    - IPython 커널 기반의 대화형 파이썬 셸(Shell)
    - Shell + 코드 편집기 개념
    - jupyter notebook 으로 실행
    - 관련 단축키 참고 링크: <https://www.youtube.com/watch?v=Q7XMSDpBb7g>
  + Colab
    - Jupyter + 구글 드라이브 + GCP
    - GPU 무료 지원, VSC 지원 등등의 장점

# Variable & List

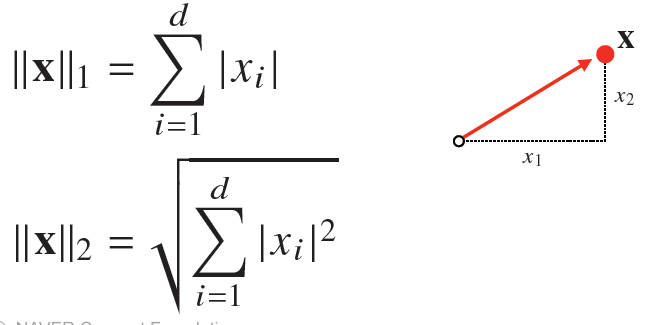
* 데이터를 저장하기 위한 메모리 공간 (메모리 주소)
  + 변수는 메모리주소, 값은 메모리 주소에 할당됨
* 변수 작명법
  + 의미있는 단어 사용. 예약어는 쓰지 않는다.
* 기본 자료형
  + 수치 자료형: Int, float
  + 문자형: string
  + 논리 자료형: Boolean
  + Dynamic typing
* Operator, operand
  + 사칙연산 연산자
  + 기타 연산자들
    - \*\*: 제곱, & 나머지, +=, -=, …
    - Float(), int()를 이용한 형변환
* List
  + 시퀀스 자료형. 여러 데이터의 집합
  + indexing
    - 리스트에 있는 값들은 모두 주소(offset)을 가짐.
    - Colors[0], colors[1], … 으로 접근
  + slicing
    - 리스트의 값들은 적절하게 잘라내서 사용할 수 있음
    - Cities[0:6] // 0번부터 5번까지
    - Cities[-9:] // -9부터 끝까지
    - [::2] // 2칸단위로 뽑아냄
    - [::-1] // 역순으로 슬라이싱
  + 리스트 연산
    - +, \*로 리스트간 Concat도 가능하다.
    - in : 해당 요소가 리스트에 있는지 여부를 반환
    - append(“abc”) // 요소 추가
    - extend( [“a”, “b”] ) // 리스트에 새로운 리스트를 추가
    - insert(2, “abc”) // 2번 주소에 요소 추가
    - remove(“abc”) // “abc” 요소를 삭제함
    - del color[0] // 0번 요소가 삭제됨
  + 파이썬 리스트의 특징
    - 다양한 데이터 타입이 한 리스트에서 공존할 수 있음
    - 리스트 안에 리스트도 가능!
    - 리스트 변수는 리스트 ‘주소’값 만이 저장된다. 즉, a = b를 하면 두 변수는 모두 같은 리스트를 가리키게 된다.
      * 복사를 위해서는 copy의 deepcopy를 이용
  + 패킹 & 언패킹
    - 패킹: 한 변수에 여러 개의 데이터를 넣는 것
      * T = [1, 2, 3]
    - 언패킹: 한 변수의 데이터를 각각의 변수로 반환
      * a, b, c = t
      * print(t, a, b, c) // [1, 2, 3] 1 2 3
  + 이차원 리스트
    - 리스트 안의 리스트 (행렬)
    - 이차원 리스트를 복사하려면?

# AI Math #1

* Vector
  + 벡터는 공간에서 한 점을 의미.
  + 숫자를 원소로 가지는 list 또는 array. 같은 모양의 벡터 간에는 덧셈 뺄셈 계산이 가능하다.
    - 같은 모양 벡터 간에는 Hadamard Product (성분곱) 계산을 할 수 있음
  + 벡터에 숫자(상수)를 곱하면 길이만 변화함. 🡪 스칼라 곱
* 벡터의 덧셈
  + 두 벡터의 덧셈(뺄셈)은 상대적 위치 이동을 표현함



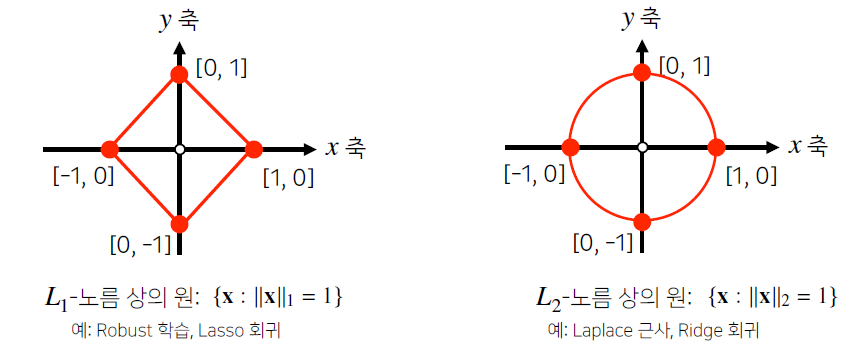
* 벡터의 Norm (노름)
  + 원점으로부터의 거리



* + d는 차원을 의미함
  + L1 노름: 각 성분의 변화량의 절대값을 모두 더함
  + L2 노름: 피타고라스 정리를 이용한 ‘유클리드 거리’ 계산
  + 파이썬에서 노름 계산하기

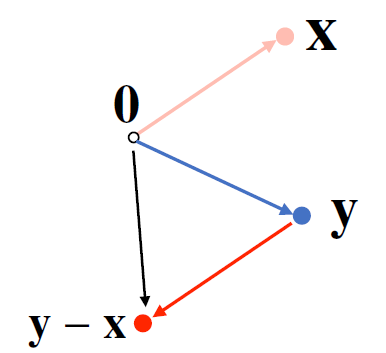


* + 두가지 노름을 쓰는 이유?
    - ‘기하학적 성질’의 차이가 있음
    - 머신러닝에서 각 성질들이 필요한 경우가 있어서 둘 다 사용함

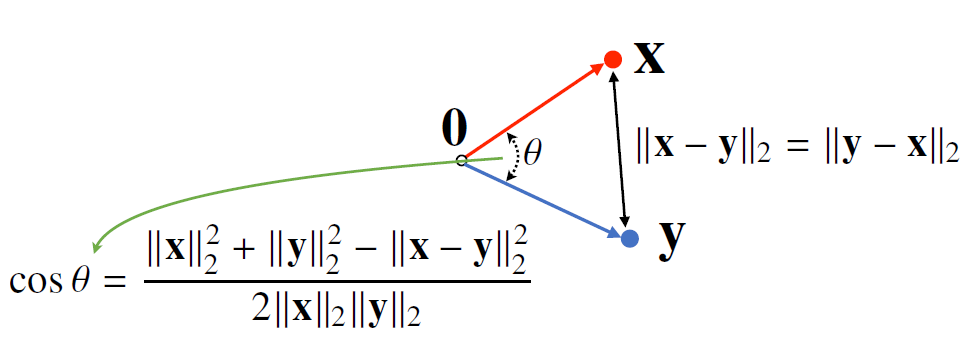


-3 -4

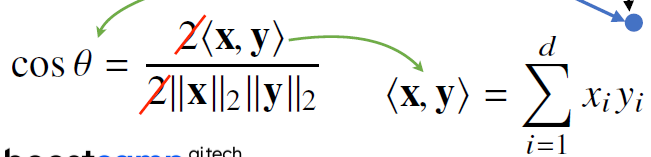
* 벡터 사이의 거리 계산 with Norm
  + L1, L2 노름을 이용하여 거리 계산을 할 수 있음
  + 사실 ‘벡터의 뺄셈’을 이용하는 것임



* + - 0에서 y-x 벡터까지의 거리가 x와 y 사이의 거리가 된다.
* 벡터 사이의 각도 계산



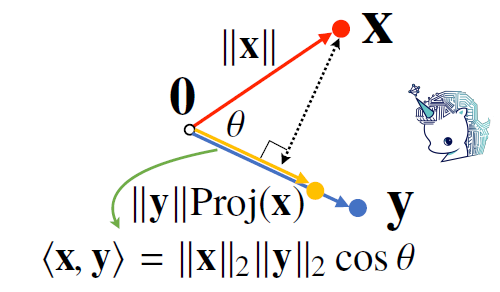
* + L2 노름과 제2 코사인법칙을 이용한다.
* 내적으로 각도를 계산하는 방법



* + 분모: x와 y의 거리의 곱으로 표현됨
  + 분자: x와 y의 내적
  + **같은 차원, 내적<x, y>는 각 원소끼리 곱의 합!**



* + 파이썬에서는 np.inner(x, y)를 이용하여 내적을 계산함
* 주의점
  + 각도는 L2 노름에서만 구할 수 있다.
* 내적의 해석
  + 내적은 ‘정사영’된 벡터의 길이와 관련이 있다.



* + Proj(x)의 길이는 코사인법칙에 의해 ||x|| cos0가 된다.
  + 내적은 정사영의 길이를 벡터 y의 길이 ||y|| 만큼 조정한 값이다. 따라서 ||y||Proj(x). 즉, 두 벡터의 ‘유사도’를 측정하는데 사용 할 수 있다.