

K-NET DEVELOP #2

Hyungyu Kim

K-NET

2024-04-09

쉽다고는 하는데 공부할 건 많고!

- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 위의 것은 프로그래밍의 기본 컨셉이기 때문에, 이번 학기에 잘 숙지하시면 차후 다른 언어를 배울 때 힘을 덜어낼 수 있습니다!
- 단, C와 C++의 포인터는 힘들 수도...

declaration → **void (*fp)();**
Initialization → **fp = display;**
call → **(*fp)();**

C++ developer
learning Python

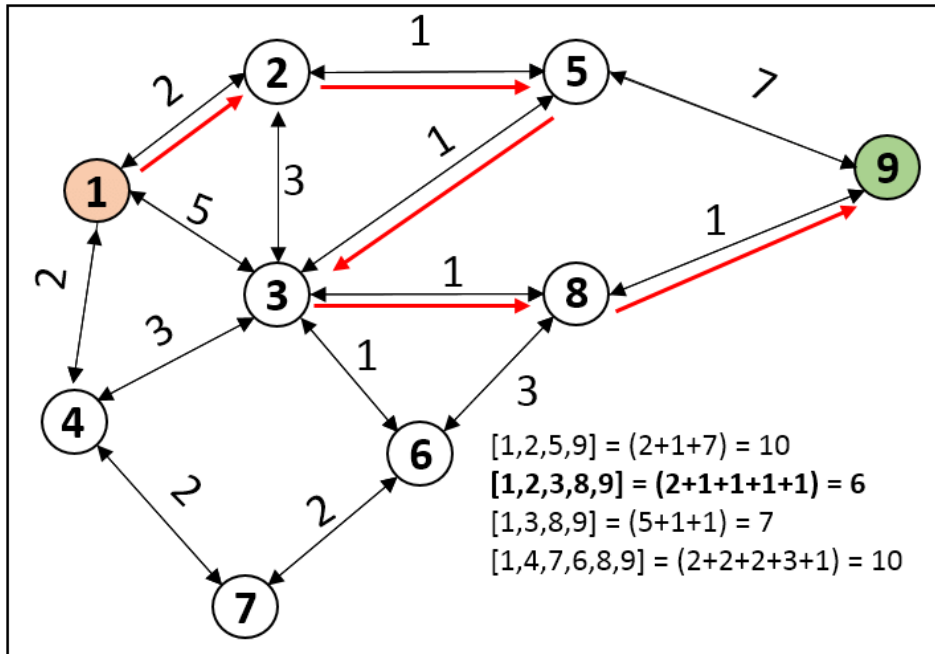


Python developer
learning C++



쉽다고는 하는데 공부할 건 많고!

- 예시: 다익스트라 알고리즘 (최단 경로 탐색 알고리즘)
- 변수, 조건문, 반복문, 배열, 함수 등을 쓴 것을 볼 수 있습니다.



Dijkstra's Algorithm Pseudocode in C++

```
function dijkstraalgorithm(G, S)
  for each node N in G
    dist[N] <- infinite
    prev[N] <- NULL
    If N != S, add N to Priority Queue Q
  dist[S] <- 0

  while Q IS NOT EMPTY
    U <- Extract MIN from Q
    for each unmarked neighbour N of U
      temporaryDist <- dist[U] + edgeWeight(U, N)
      if temporaryDist < dist[N]
        dist[N] <- temporaryDist
        prev[N] <- U
  return dist[], prev[]
```

시작하기 앞서서

- 오늘은 아직 안 알려드린 코드가 휘리릭 지나갈 것이기 때문에 주의 바랍니다.
- 코드는 대충만 보시고, **한국어 텍스트에 더 집중해주세요!**

```

      ▽ Z ← BACKWARD ARG;W;INPUT;G;DE;ETA;
      ALPHA;ΔW0;X;U;N;D;ΔW;E
[1]   W INPUT G DE ETA ALPHA ΔW0 X U ←
      ARG
[2]   D ← (E ← DE - (1 ↓ (N + 1) ⊃ X)) × ((
      N ← ρW) ⊃ U) GRADIENT G
[3]   ΔW ← C(ETA × (D ° . × N ⊃ X)) +
      ALPHA × N ⊃ ΔW0
[4]   MAIN: ⌊ (1 > N ← N - 1) / 'Z ← (⊕ ΔW)
      E ° → 0 '
[5]   D ← ((N ⊃ U) GRADIENT G) × 1 ↓ D + . × (
      N + 1) ⊃ W
[6]   ΔW ← ΔW, C(ETA × (D ° . × N ⊃ X)) +
      ALPHA × N ⊃ ΔW0
[7]   → MAIN
      ▽
```

노가다꾼 죽이기

- 요즘 파이썬에서 데이터 분석이 인기가 많습니다!
- 7명의 키를 가지고 평균, 분산, 표준편차를 구하려 합니다.
- 아래 내용을 일일이 다 적어야 한다 생각하니 끔찍합니다. (노가다꾼만 좋아합니다)

```
[ ] 1  print("평균")  
    2  
    3  print((164+184+172+175+182+170+177)/7)
```

평균

174.85714285714286

노가다꾼 죽이기 (cont'd)

- 아까 출력된 값을 복붙해서 만든 표준편차 계산 코드입니다.
- 파이썬을 이렇게 밖에 쓸 수 없다면, 차라리 공학용 계산기나 쓰는 게 낫겠습니다.

```
[ ] 1 print("표준편차")
    2 print((((164-174.85714285714286)**2 + # 평균 값을 먼저 출력한 다음 복붙하기
    3         (184-174.85714285714286)**2 +
    4         (172-174.85714285714286)**2 +
    5         (175-174.85714285714286)**2 +
    6         (182-174.85714285714286)**2 +
    7         (170-174.85714285714286)**2 +
    8         (177-174.85714285714286)**2
    9         )/7)**0.5)
```

표준편차
6.423807758833461

노가다꾼 죽이기 (cont'd)

- 아까의 예시로부터 바로 변수(값을 저장하는 공간)의 유용성을 입증할 수 있습니다.
- 중요한 값들(평균, 분산, 표준편차)을 '변수에 저장'하면 이렇게 됩니다.

```
mean = (164+184+172+175+182+170+177) / 7

print("평균")
print(mean)

variance = ((164-mean)**2 + (184-mean)**2 + (172-mean)**2 + (175-mean)**2 +
            (182-mean)**2 + (170-mean)**2 + (177-mean)**2) / 7

std = variance**0.5 # 0.5제곱. 즉, 양의 제곱근

print("표준편차")
print(std)
```

노가다꾼 죽이기 (cont'd)

- 근데, 아까 코드에서도 여전히 노가다를 해야 합니다.
- 키 데이터를 일일이 164, 184, 172, ... 이렇게 매번 적어야 할까요?
- 그러나 여기서 또 변수를 만들더라도, 이름을 일일이 적어주는 건 똑같습니다.

```
height_영희 = 164
height_철수 = 184
height_맹구 = 172
height_훈이 = 175
height_짱구 = 182
height_유리 = 170
height_흰둥 = 177
```

```
mean = (height_영희 + height_철수 + height_맹구 + height_훈이 + height_짱구 + height_유리 + height_흰둥) / 7
```


노가다꾼 죽이기 (cont'd)

- 코드를 최대한 줄이기 위한 방법이 필요합니다.
- 이 예시로부터 **리스트(배열)**의 유용성을 입증할 수 있습니다.
(리스트는 변수 여러 개의 묶음이고, list[n]은 리스트의 n번째 원소입니다.)
- 하지만 아래 mean값 계산 코드도 줄줄이 다 입력해줘야 됩니다.
- 특히 같은 단어를 반복해서 적고 있습니다.



```
heights = [164, 184, 172, 175, 182, 170, 177]
```

```
mean = (heights[0] + heights[1] + heights[2] + heights[3] + heights[4] + heights[5] + heights[6]) / 7
```

노가다꾼 죽이기 (cont'd)

- 반복을 수행하는 **반복문**을 보기 전에, 유용한 **내장 함수** 몇 개를 좀 봅시다.
- `len(list)` : 리스트의 길이 계산
- `sum(list)` : 리스트 내부 원소의 합 계산
- 아주 깔끔합니다!



```
heights = [164, 184, 172, 175, 182, 170, 177]
```

```
mean = sum(heights) / len(heights)
```

```
print(mean)
```

노가다꾼 죽이기 (cont'd)

- 아까 평균 계산을 내장 함수를 이용해 깔끔히 해결했습니다.
- 그러면 분산과 표준편차는 어떻게 할까요? 설마... 이렇게 밖에 답이 없을까요?
- 주의: 파이썬에서는 수학에서처럼 “...”을 찍어 표현하면 해석을 못합니다.



```
variance = ((heights[0] - mean)**2 + (heights[1] - mean)**2 +  
(heights[2] - mean)**2 + (heights[3] - mean)**2 + ...
```

노가다꾼 죽이기 (cont'd)

- 여기서는 기교를 부리지 말고 우직하게 반복문으로 해결해봅시다.
- **반복문**은 패턴이 같은 코드를 컴퓨터로 하여금 반복 수행시키는 것입니다.
- 반복문을 이용하면 평균, 표준편차 구하기와 같은 유사한 일을 전부 해결할 수 있습니다.

```
for (var i = 0; i < 2; i++) {  
    console.log(i);  
}
```

Execution flow annotations:

- ① i=0 (before loop start)
- ② true (condition i < 2)
- ③ i=0 (before log)
- ④ i=1 (before loop start)
- ⑤ true (condition i < 2)
- ⑥ i=1 (before log)
- ⑦ i=2 (before loop start)
- ⑧ false (condition i < 2)

for 문의 실행 순서

노가다꾼 죽이기 (cont'd)

- 반복문, 그 중에서도 **for문**을 이용하여 평균과 분산, 표준편차를 계산해보면...
- 여기에 관련해서 지금 모두 이해할 필요는 없습니다. 이에 관한 이야기는 5주차(조건문과 반복문)에 이뤄질 예정입니다.



```
heights = [164, 184, 172, 175, 182, 170, 177]
```

```
# 평균 계산
```

```
mean = 0
```

```
for height in heights:
```

```
    mean += height
```

```
mean /= len(heights)
```

노가다꾼 죽이기 (cont'd)

- previous code (cont'd)
- 참고: 이 코드에서 이제 뭔가 줄줄이 길게 노가다 식으로 쓰는 일은 없어졌습니다.

```
# 분산 계산
var = 0
for height in heights:
    var += (height - mean)**2
var /= len(heights)

# 표준편차 계산
std = var ** 0.5

print(mean)
print(std)
```

노가다꾼 죽이기 (cont'd)

- 참고: 파이썬에서는 리스트를 다룰 때 for문(반복문)과 비슷하게 생긴 “리스트 컴프리헨션”을 써서 코드를 간단히 만들 수 있습니다.



```
heights = [164, 184, 172, 175, 182, 170, 177]

mean = sum(heights) / len(heights)
sq_dev = [(height - mean)**2 for height in heights] # squared deviation: 편차 제곱
var = sum(sq_dev) / len(sq_dev)
std = var ** 0.5

print(mean)
print(std)
```

노가다꾼 죽이기 (cont'd)

- 꼬투리 좀 잡자면, 이전 코드에서 `sum()/len()`이 계속 반복되고 있습니다.
- 여러 번 중복하여 쓰인 코드는 함수를 이용해 묶어버리면 좋습니다.
- 그 함수를 `average`라 (우리 마음대로) 이름 붙였습니다.

```
def average(list):  
    return sum(list) / len(list)  
  
heights = [164, 184, 172, 175, 182, 170, 177]  
  
mean = average(heights)  
sq_dev = [(height - mean)**2 for height in heights]  
std = average(sq_dev) ** 0.5  
  
print(mean)  
print(std)
```


노가다꾼 죽이기 (cont'd)

- 분산 구하는 과정 역시 복잡하니까, 그것 역시 한 개의 함수로 묶어버립시다.
- 참고: 오늘 이 코드를 모두 이해할 필요는 없습니다. 텍스트를 위주로 보세요.



```
def average(list): # 평균을 계산
    return sum(list)/len(list)

def st_dev(list): # 분산을 계산
    mean = average(heights) # 함수 내에서 다른 함수는 언제든지 다시 가져올 수 있습니다.
    sq_dev = [(x - mean)**2 for x in list]
    std = average(sq_dev) ** 0.5
    return std
```

노가다꾼 죽이기 (cont'd)

- previous code (cont'd)
- 복잡한 과정을 함수로 묶어버리니까 코드가 더 명확하고 깔끔해졌습니다.

```
heights = [164, 184, 172, 175, 182, 170, 177]
```

```
avg = average(heights) # 함수 호출  
std = st_dev(heights)
```

```
print(avg)  
print(std)
```

무리수 두기!

- 너무 뜬금없지만, 그룹을 남녀로 나눠서 따로 평균과 표준편차를 계산해봐야 하는 상황이 되었다고 가정합시다.
- 어떡하죠? 위에서 본 리스트에는 남녀에 대한 정보를 담기가 쉽지 않아보입니다.

```
heights = [164, 184, 172, 175, 182, 170, 177]
```

- 가장 깔끔한 방법은 **이차원 리스트**, **딕셔너리**를 이용하는 것입니다.
- 여기부터는 **어려운 이야기**이니, 코드 자체에 집착하지 마시길 바랍니다.

무리수 두기! (cont'd)

- 참고: 간단한 설명
- 2차원 리스트: 리스트 내에 리스트가 들어갑니다. 행렬과 비슷하게 쓸 수도 있습니다.
- 딕셔너리: Key-Value의 쌍을 저장합니다. 전화번호부 등과 비슷하게 쓰일 수 있습니다.

```
1 a[0][2]=99
2 a
[[0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 99, 0, 0, 0, 0, 0, 0, 0]]
```

```
1 # Create a dictionary
2
3 my_dict = {'Alex': 5,
4            'Ben' : 10,
5            'Carly': 12,
6            'Danielle': 7,
7            'Evan' : 6}
8 my_dict
{'Alex': 5, 'Ben': 10, 'Carly': 12, 'Danielle': 7, 'Evan': 6}
```

무리수 두기! (cont'd)

- 2차원 배열을 이용하여 데이터를 나타내기
- 참고: 파이썬의 리스트 내에는 서로 다른 타입이 들어갈 수 있습니다.

```
heights = [[164, "영희", "여자"], # 파이썬의 문자열은 따옴표가 필요합니다!  
            [184, "철수", "남자"],  
            [172, "맹구", "남자"],  
            [175, "훈이", "남자"],  
            [182, "짱구", "남자"],  
            [170, "유리", "여자"],  
            [177, "흰둥", "여자"]]  
  
print(heights[4]) # 출력 결과: [182, "짱구", "남자"]  
print(heights[4][0]) # 출력 결과: 182
```

무리수 두기! (cont'd)


- (중첩된) 딕셔너리를 이용하여 데이터를 나타내기

```
data = {
    "영희": {"키": 164, "성별": "여자"},
    "철수": {"키": 184, "성별": "남자"},
    "맹구": {"키": 172, "성별": "남자"},
    "훈이": {"키": 175, "성별": "남자"},
    "짱구": {"키": 182, "성별": "남자"},
    "유리": {"키": 170, "성별": "여자"},
    "흰둥": {"키": 177, "성별": "여자"},
}
# 딕셔너리 안에 딕셔너리가 들어간 것이 보이시나요?

print(data["짱구"]) # 출력 결과: {"키": 182, "성별": "남자"}
print(data["짱구"]["키"]) # 출력 결과: 182
```

무리수 두기! (cont'd)

- 참고: 딕셔너리에도 리스트처럼 반복문(for문)을 적용할 수 있습니다.
- 더 자세한 이야기는 반복문 챕터에서 이야기하겠습니다.



```
# tmi: 딕셔너리도 반복문 쓸 수 있어요!  
heights = [data[key]["키"] for key in data.keys()]  
print(sum(heights)/len(heights))
```

무리수 두기! (cont'd)

- 이제 약간의 조건문(if문)을 이용해 성별에 따라 평균값을 구해봅시다.
- 데이터는 아까와 같이 똑같이 주어집니다.

```
data = {  
    "영희": {"키": 164, "성별": "여자"},  
    "철수": {"키": 184, "성별": "남자"},  
    "맹구": {"키": 172, "성별": "남자"},  
    "훈이": {"키": 175, "성별": "남자"},  
    "짱구": {"키": 182, "성별": "남자"},  
    "유리": {"키": 170, "성별": "여자"},  
    "현등": {"키": 177, "성별": "여자"},  
}
```


무리수 두기! (cont'd)

- previous code (cont'd)
- 리스트 컴프리헨션 안에 조건문(if문)이 들어갔습니다.
- if문 안의 “==”는 좌변과 우변이 같은 것인지 검사합니다.

```
heights = [value["키"] for key, value in data.items()]
heights_men = [value["키"] for key, value in data.items() if value["성별"]=="남자"]
heights_women = [value["키"] for key, value in data.items() if value["성별"]=="여자"]

print(sum(heights)/len(heights))
print(sum(heights_men)/len(heights_men))
print(sum(heights_women)/len(heights_women))
```

무리수 두기! (cont'd)

- 저렇게 입력이 주어진 상황에서도 평균과 표준편차를 구할 수는 있습니다.
- 그런데, 갈수록 코드가 점점 길어져서 오늘 보여드리기엔 부담이 되네요...

```
data = {
    "영희": {"키": 164, "성별": "여자"},
    "철수": {"키": 184, "성별": "남자"},
    "맹구": {"키": 172, "성별": "남자"},
    "훈이": {"키": 175, "성별": "남자"},
    "짱구": {"키": 182, "성별": "남자"},
    "유리": {"키": 170, "성별": "여자"},
    "원동": {"키": 177, "성별": "여자"},
}

avg = lambda list: sum(list)/len(list)

def st_dev(list):
    mean = avg(list)
    sq_dev = [(height - mean)**2 for height in list]
    return avg(sq_dev) ** 0.5

heights_men = [data[key]["키"] for key in data.keys() if data[key]["성별"]=="남자"]
heights_women = [data[key]["키"] for key in data.keys() if data[key]["성별"]=="여자"]

mean_men, mean_women = avg(heights_men), avg(heights_women)
std_men, std_women = st_dev(heights_men), st_dev(heights_women)

print(mean_men, mean_women, std_men, std_women)
```

```
heights = [[164, "영희", "여자"],
            [184, "철수", "남자"],
            [172, "맹구", "남자"],
            [175, "훈이", "남자"],
            [182, "짱구", "남자"],
            [170, "유리", "여자"],
            [177, "원동", "여자"]]

num_men = 0
num_women = 0
mean_men = 0
mean_women = 0
for data in heights:
    if data[2] == "남자":
        num_men += 1
        mean_men += data[0]
    if data[2] == "여자":
        num_women += 1
        mean_women += data[0]
mean_men /= num_men
mean_women /= num_women

var_men = 0
var_women = 0
for data in heights:
    if data[2] == "남자":
        var_men += (data[0] - mean_men)**2
    if data[2] == "여자":
        var_women += (data[0] - mean_women)**2
var_men /= num_men
var_women /= num_women

std_men = var_men ** 0.5
std_women = var_women ** 0.5

print(mean_men)
print(mean_women)
print(std_men)
print(std_women)
```

무리수 두기! (cont'd)

- 실제 데이터 분석에서는 pandas라고 하는 라이브러리를 사용하여 엑셀처럼 유연하게 다룰 수 있습니다. (설치 필요)

```
import pandas as pd

data = {
    "영희": {"키": 164, "성별": "여자"},
    "철수": {"키": 184, "성별": "남자"},
    "맹구": {"키": 172, "성별": "남자"},
    "훈이": {"키": 175, "성별": "남자"},
    "짱구": {"키": 182, "성별": "남자"},
    "유리": {"키": 170, "성별": "여자"},
    "흰둥": {"키": 177, "성별": "여자"},
}

df = pd.DataFrame(data)

df = df.T # 우리 보기 편한대로 행과 열 바꾸기(Transpose)

print(df)
```

무리수 두기! (cont'd)

- 그러면 이렇게 엑셀 스프레드시트와 비슷한 것이 나옵니다.
- 참고: 주피터 노트북은 셀의 마지막 줄에 `print()`를 적지 않아도 출력이 됩니다.

```
13 df = pd.DataFrame(data)
14
15 df = df.T # 우리 보기 편한대로 행과 열 바꾸기
16
17 df
```

| | 키 | 성별 |
|----|-----|----|
| 영희 | 164 | 여자 |
| 철수 | 184 | 남자 |
| 맹구 | 172 | 남자 |
| 훈이 | 175 | 남자 |
| 짱구 | 182 | 남자 |
| 유리 | 170 | 여자 |
| 흰둥 | 177 | 여자 |

무리수 두기! (cont'd)

- 그리고 아까 우리가 어렵게 구현했던 것을 **단 한 줄로** 구할 수 있게 됩니다.
- 이렇듯, 파이썬 라이브러리는 강력합니다!

```
[ ] 1 df.groupby("성별").mean()
```

키

성별

남자 178.250000

여자 170.333333

```
[ ] 1 df.groupby("성별").std() # 표본 분산과 모분산이 다름에 유의하세요!
```

키

성별

남자 5.678908

여자 6.506407

화석 김현규의 끈대질 시간...

- 데이터 분석 라이브러리가 잘 만들어져 있는데, 평균, 중앙값, 최빈값 등의 기능을 직접 만들어보는 것이 딱히 의미 없는 것 아니냐고요?
- 절대적으로 틀린 말입니다! 특히나 여러분께서 나중에 코딩 테스트를 칠 것이라면 저런 구현을 잘 해야 합니다.
- 프로그래밍 초심자 분들이라면, 그런 것을 직접 만들어보시면서 ‘컴퓨팅적 사고’에 익숙해지셨으면 하는 바람입니다. (프로그래머스 0레벨 추천)
- 프로젝트 도중, 일일이 구현하기엔 시간 소모가 큰 기능에 대해서는 라이브러리에 의존하시면 됩니다.

우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들) – 미정
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

다음 시간에 다룰 주제

- 변수와 리스트

