

목록

lec1.프로그래밍 기초 강좌 소개-v3.....	1
CS101 - 프로그래밍 기초 강좌 소개 Lecture 1.....	1
왜 이 수업을 듣나요?.....	2
프로그램이란?.....	3
디버깅이란?.....	4
왜 프로그래밍은 유용한가?.....	5
왜 프로그래밍을 배워야 하는가?.....	6
왜 프로그래밍은 재미있는가?.....	7
Python.....	8
Python 실습 준비 (1/4).....	9
Python 실습 준비 (graphics 예제 실습용) (2/4).....	10
Python 실습 준비 (graphics 예제 실습용) (3/4).....	11
Python 실습 준비 (graphics 예제 실습용) (4/4).....	12
MAC에서 Python 실습 준비 (graphics 예제 실습용).....	13
수업 로드맵.....	14
정리 및 예습.....	15
lec2. Python 프로그램 작성 예제.....	16
CS101 - Python 프로그램 작성 예제 Lecture 2.....	16
시작.....	17
새로운 함수 만들기.....	18
실행 순서.....	19
로봇.....	20
신문 배달.....	21
신문 배달.....	22
하향식 설계 (Top-down design).....	23
간단한 반복.....	24
반복된 코드 제거하기.....	25
정리 및 예습.....	26
lec3. if 조건문과 while 반복문.....	27
CS101 - if 조건문과 while 반복문 Lecture 3.....	27
조건문.....	28
간단한 예시.....	29
비교 감지.....	30
참의 반대는 거짓.....	31
else란.....	32
춤추고 노래하는 휴보.....	33
춤추고 노래하는 휴보.....	34
춤추고 노래하는 휴보.....	35
너무 많은 선택문.....	36

너무 많은 선택문.....	36
while 반복문.....	38
정리 및 예습.....	39
lec4.if 와 while 을 사용한 미로 탈출 예제.....	40
CS101 - if 와 while 을 사용한 미로 탈출 예제 Lecture 4.....	40
80일간의 세계일주.....	41
다른 모양의 세계.....	42
우회전이 필요한 경우.....	43
곤란한 시작점에서 빠져 나가기.....	44
곤란한 시작점에서 빠져 나가기.....	45
사람이 이해하기 좋은 코드 작성.....	46
슬라이드 번호 8.....	47
단계적 세분화.....	48
정리 및 예습.....	49
lec5.프로그램에서 사용하는 객체와 객체의 형태.....	50
CS101 - 프로그램에서 사용하는 객체와 객체의 형태 Lecture 5.....	50
객체.....	51
객체 생성.....	52
복잡한 객체 생성.....	53
형태.....	54
복잡한 형태.....	55
이름.....	56
이름 규칙.....	57
변수.....	58
멤버 변수.....	59
여러 이름을 가진 객체.....	60
여러 이름을 가진 객체.....	61
여러 이름을 가진 객체.....	62
여러 이름을 가진 객체.....	63
정리 및 예습.....	64
lec6.연산자 및 튜플.....	65
CS101 - 연산자 및 튜플 Lecture 6.....	65
연산자.....	66
식.....	67
문자열 연산.....	68
논리식.....	69
논리 연산자.....	70
튜플.....	71
정리 및 예습.....	72
lec7.튜플을 사용한 디지털 사진 변환 예제-v3.....	73
CS101 - 튜플을 사용한 디지털 사진 변환 예제 Lecture 7.....	73
색.....	74

색.....	74
for 반복문.....	76
이미지 반전.....	77
이미지 반전 예시.....	78
이미지 반전 예시.....	79
이미지 반전 예시.....	80
이미지 반전 예시.....	81
이미지 반전 예시.....	82
이미지 반전 예시.....	83
이미지 반전 예시.....	84
이미지 반전 예시.....	85
이미지 반전 예시.....	86
이미지 반전 예시.....	87
이미지 반전 예시.....	88
이미지 반전 예시.....	89
이미지 반전 예시.....	90
이미지 반전 예시.....	91
이미지 반전 예시.....	92
이미지 반전 예시.....	93
이미지 반전 예시.....	94
이미지 반전 예시.....	95
이미지 흑백 변환.....	96
정리 및 예습.....	97
lec8. 매개 변수와 반환값을 가진 함수.....	98
CS101 - 매개 변수와 반환값을 가진 함수 Lecture 8.....	98
함수.....	99
유용한 함수.....	100
수학 함수.....	101
매개 변수를 사용한 함수 정의.....	102
여러 반환문을 가진 함수.....	103
논리값을 반환하는 함수.....	104
결과값이 없는 함수.....	105
함수 호출.....	106
여러 값 반환하기.....	107
키보드 입력.....	108
정리 및 예습.....	109
lec9. 함수를 사용한 로봇 조종 및 디지털 사진 변환 프로그램.....	110
CS101 - 함수를 사용한 로봇 조종 및 디지털 사진 변환 프로그램 Lecture 9.....	110
휴보 가족.....	111
함수를 이용한 비퍼 줍기.....	112
함수를 이용한 비퍼 줍기 동영상.....	113

함수를 이용한 비퍼 줍기.....	113
흑백으로 변환하기.....	115
함수를 이용한 이미지 흑백 변환.....	116
정리 및 예습.....	117
lec10. 함수 인자와 매개 변수_v2.....	118
CS101 - 함수 인자와 매개 변수 Lecture 10.....	118
매개 변수의 대상.....	119
매개 변수의 대상.....	120
함수 인자.....	121
매개 변수의 기본값.....	122
매개 변수의 기본값 활용 예제.....	123
이름이 있는 인자.....	124
정리 및 예습.....	125
lec11. 함수가 사용하는 지역 변수와 전역 변수.....	126
CS101 - 함수가 사용하는 지역 변수와 전역 변수 Lecture 11.....	126
지역 변수.....	127
지역 변수.....	128
지역 변수를 사용하는 이유: 모듈화.....	129
전역 변수.....	130
지역 변수와 전역 변수.....	131
전역 변수의 값 쓰기.....	132
지역 변수와 전역 변수.....	133
정리 및 예습.....	134
lec12. 모듈과 그래픽 객체들_v2.....	135
CS101 - 모듈과 그래픽 객체들 Lecture 12.....	135
모듈.....	136
모듈 들여오기.....	137
Import 예시.....	138
그림 그리기.....	139
그래픽 객체.....	140
기준점.....	141
그래픽 객체의 깊이 순서.....	142
회전, 뒤집기, 크기 조정.....	143
레이어.....	144
변형.....	145
정리 및 예습.....	146
슬라이드 번호 13.....	147
슬라이드 번호 14.....	148
lec13. 그래픽 객체를 사용한 애니메이션_v4.....	149
lec14. 리스트 활용법.....	159
CS101 -리스트 활용법 Lecture 14.....	159

많은 데이터.....	149
리스트.....	161
리스트.....	151
리스트와 관련된 내장 함수들.....	152
리스트는 가변 객체이다.....	164
Aliasing.....	165
다시 보기: 여러 이름을 가진 객체.....	166
리스트 탐색하기.....	153
리스트 탐색하기.....	154
정리 및 예습.....	155
lec15. 리스트_문자열_튜플.....	170
CS101 - 시퀀스: 리스트, 문자열, 튜플 Lecture 15.....	170
많은 데이터.....	171
정렬.....	172
리스트 자르기.....	173
리스트 뒤집기.....	174
메달 순위.....	175
한 종류의 메달만 획득한 나라.....	176
많은 데이터.....	177
리스트의 멤버 함수.....	178
시퀀스.....	179
리스트, 튜플, 문자열.....	180
메달 리스트.....	181
히스토그램.....	182
히스토그램.....	183
정리 및 예습.....	184
lec16. 리스트 활용 예제_정렬과 소수 구하기.....	185
CS101 -리스트 활용 예제: 정렬과 소수 구하기 Lecture 16.....	185
다시보기 : 정렬.....	186
버블 정렬.....	187
버블 정렬.....	188
버블 정렬.....	189
버블 정렬.....	190
버블 정렬.....	191
버블 정렬.....	192
버블 정렬.....	193
소수 구하기.....	194
정리 및 예습.....	195
lec17. 자료 구조_문자열과 집합.....	196
CS101 - 자료 구조: 문자열과 집합 Lecture 17.....	196
문자열 서식화.....	197
문자열 서식화.....	198

문자열.....	198
문자열의 멤버 함수.....	200
슬라이드 번호 6.....	201
슬라이드 번호 7.....	202
슬라이드 번호 8.....	203
집합의 멤버 함수.....	204
집합의 멤버 함수.....	205
집합의 멤버 함수.....	206
정리 및 예습.....	207
lec18.자료 구조_사전.....	208
슬라이드 1: CS101 - 자료 구조: 사전 Lecture 18.....	208
슬라이드 2: 사전.....	209
슬라이드 3: 사전.....	210
슬라이드 4: 사전 관련 함수들.....	211
슬라이드 5: 사전 관련 함수들.....	212
슬라이드 6: 사전 관련 함수들.....	213
슬라이드 7: 사전과 반복문.....	214
슬라이드 8: 리스트, 집합, 사전.....	215
슬라이드 9: 리스트, 집합, 사전.....	216
슬라이드 10.....	217
lec19.이미지 프로세싱-v2.....	218
슬라이드 1: CS101 -이미지 프로세싱 Lecture 19.....	218
슬라이드 2: 복사해서 붙여넣기.....	219
슬라이드 3: 크로마키.....	220
슬라이드 4: 색 차이.....	221
슬라이드 5: 크로마키.....	222
슬라이드 6: 크로마키.....	223
슬라이드 7: 정보 은닉.....	224
슬라이드 8: 정보 은닉 예제.....	225
슬라이드 9: 정보 은닉 예제.....	226
슬라이드 10: 정보 은닉 예제.....	227
슬라이드 11: 정보 은닉 예제.....	228
슬라이드 12: 정보 은닉 예제.....	229
슬라이드 13: 정보 은닉 예제.....	230
슬라이드 14: 정리 및 예습.....	231
lec20.텍스트 프로세싱-v2.....	232
슬라이드 1: CS101 - 텍스트 프로세싱 Lecture 20.....	232
슬라이드 2: 파일.....	233
슬라이드 3: 파일에서 문자열 읽기.....	234
슬라이드 4: 파일 읽기.....	235
슬라이드 5: 단어 찾기.....	236

슬라이드 6: 빠른 단어 찾기	236
슬라이드 7: 파일의 코멘트	238
슬라이드 8: 큰 파일	239
슬라이드 9: 단어 게임	240
슬라이드 10: Abecedarian words	241
슬라이드 11: 연속된 동일한 문자 파악하기	242
슬라이드 12: 파일 쓰기	243
슬라이드 13: 금융 자료	244
슬라이드 14: 금융 자료	245
슬라이드 15: 그래프 만들기	246
슬라이드 16: 정리 및 예습	247
lec21. 객체 (object)로 블랙잭 카드 게임 만들기 (1_2)	248
슬라이드 1: CS101 - 객체 (object)로 블랙잭 카드 게임 만들기 (1/2) Lecture 21	248
슬라이드 2: 블랙잭	249
슬라이드 3: 블랙잭	250
슬라이드 4: 블랙잭 게임 인터페이스	251
슬라이드 5: 블랙잭 게임 동영상 예제	252
슬라이드 6: 튜플로 표현한 카드	253
슬라이드 7: 객체로 표현한 카드	254
슬라이드 8: 객체로 표현한 카드	255
슬라이드 9: 두 개 이상의 카드	256
슬라이드 10: 가변 객체	257
슬라이드 11: 함수는 객체이다	258
슬라이드 12: 정리 및 예습	259
lec22. 객체 (object)로 블랙잭 카드 게임 만들기 (2_2)	260
슬라이드 1: CS101 - 객체 (object)로 블랙잭 카드 게임 만들기 (2/2) Lecture 22	260
슬라이드 2: 블랙잭	261
슬라이드 3: 멤버 함수	262
슬라이드 4: 멤버 함수	263
슬라이드 5: 생성자	264
슬라이드 6: 생성자	265
슬라이드 7: 문자열 변환	266
슬라이드 8: 카드 택	267
슬라이드 9: 카드 텍	268
슬라이드 10: 블랙잭	269
슬라이드 11: 부등식	270
슬라이드 12: 부등식	271
슬라이드 13: 블랙잭 게임 인터페이스	272
슬라이드 14: Table 객체	273
슬라이드 15: 멤버 함수	274

슬라이드 16: 사용자 인터페이스 프로그래밍	274
슬라이드 17: 정리 및 예습	276
lec23. 객체 (object)로 애니메이션 만들기	277
슬라이드 1: CS101 - 객체 (object)로 애니메이션 만들기 Lecture 23	277
슬라이드 2: 객체: 상태와 동작	278
슬라이드 3: 닭 가족의 여행	279
슬라이드 4: 복사해서 붙여 넣기	280
슬라이드 5: Chicken 객체	281
슬라이드 6: Chicken 객체	282
슬라이드 7: Chicken 객체 사용하기	283
슬라이드 8: 정리 및 예습	284
lec24. 프로그램 속도 향상 방법-merge-sort-fixed	285
슬라이드 1: CS101 - 프로그램 실행 속도 향상 방법 Lecture 24	285
슬라이드 2: 속도	286
슬라이드 3: 해석기, 컴파일러	287
슬라이드 4: 해석기, 컴파일러	288
슬라이드 5: 해석기를 사용하는 이유	289
슬라이드 6: 좋은 알고리즘	290
슬라이드 7: 예제: 정렬	291
슬라이드 8: 예제: 정렬	292
슬라이드 9: 병합 정렬: 더 좋은 알고리즘	293
슬라이드 10: 병합 정렬은 재귀적이다	294
슬라이드 11: 병합 정렬: 더 좋은 알고리즘	295
슬라이드 12: 실행 시간 비교	296
슬라이드 13: 알고리즘 분석	297
슬라이드 14: 효율적인 알고리즘	298
슬라이드 15: 정리 및 예습	299
lec25. 재귀 (recursive) 함수 및 강의 마무리-v3-merge-sort-fixed	300
슬라이드 1: CS101 - 재귀 (recursive) 함수 및 강의 마무리 Lecture 25	300
슬라이드 2: 재귀	301
슬라이드 3: 재귀 함수	302
슬라이드 4: 재귀 함수	303
슬라이드 5: 재귀 함수	304
슬라이드 6: 재귀 함수	305
슬라이드 7: 재귀 함수	306
슬라이드 8: 재귀 함수	307
슬라이드 9: 재귀 함수	308
슬라이드 10: 재귀 함수	309
슬라이드 11: 재귀 함수	310
슬라이드 12: 재귀 함수	311

슬라이드 13: 재귀 함수	311
슬라이드 14: 재귀 함수	313
슬라이드 15: 재귀 함수	314
슬라이드 16: 재귀 함수	315
슬라이드 17: 재귀 함수	316
슬라이드 18: 재귀 함수	317
슬라이드 19: 재귀 함수	318
슬라이드 20: 재귀 함수	319
슬라이드 21: 재귀 함수	320
슬라이드 22: 재귀 함수	321
슬라이드 23: 재귀 함수	322
슬라이드 24: 재귀 함수	323
슬라이드 25: 재귀 함수	324
슬라이드 26: 재귀 함수	325
슬라이드 27: 재귀 함수	326
슬라이드 28: 재귀 함수	327
슬라이드 29: 재귀 함수	328
슬라이드 30: 재귀 함수	329
슬라이드 31: 재귀 함수	330
슬라이드 32: 재귀 함수	331
슬라이드 33: 재귀 함수	332
슬라이드 34: b진법으로 변환하기	333
슬라이드 35: b진법으로 변환하기	334
슬라이드 36: b진법으로 변환하기	335
슬라이드 37: b진법으로 변환하기	336
슬라이드 38: b진법으로 변환하기	337
슬라이드 39: b진법으로 변환하기	338
슬라이드 40: b진법으로 변환하기	339
슬라이드 41: b진법으로 변환하기	340
슬라이드 42: b진법으로 변환하기	341
슬라이드 43: 병합 정렬은 재귀적이다	342
슬라이드 44: 병합 정렬은 재귀적이다	343
슬라이드 45: 병합 정렬은 재귀적이다	344
슬라이드 46: 병합 정렬은 재귀적이다	345
슬라이드 47: 병합 정렬은 재귀적이다	346
슬라이드 48: 병합 정렬은 재귀적이다	347
슬라이드 49: 병합 정렬은 재귀적이다	348
슬라이드 50: 병합 정렬은 재귀적이다	349
슬라이드 51: 병합 정렬은 재귀적이다	350
슬라이드 52: 병합 정렬은 재귀적이다	351

슬라이드 53: 병합 정렬은 재귀적이다.....	351
슬라이드 54: 병합 정렬은 재귀적이다.....	353
슬라이드 55: 병합 정렬은 재귀적이다.....	354
슬라이드 56: 병합 정렬은 재귀적이다.....	355
슬라이드 57: 병합 정렬은 재귀적이다.....	356
슬라이드 58: 지금까지 배운 것들 (1/2).....	357
슬라이드 59: 지금까지 배운 것들 (2/2).....	358

CS101 - 프로그래밍 기초 강좌 소개

Lecture 1

School of Computing
KAIST

학습 목표:

- 이 수업의 목표 및 수업에서 배울 내용의 범위를 이해 할 수 있다.
- 프로그래밍의 유용성을 알 수 있다.
- Python 실습에 필요한 SW를 설치할 수 있다.

왜 이 수업을 듣나요?

모든 과학자나 기술자는 프로그래밍을 할 줄 알아야 합니다.

프로그래밍은 미적분, 선형대수, 기초 물리학, 기초화학, 영어와 같은 기초 교육의 일부입니다.



Alan Perlis 1961

컴퓨터 과학은 프로그래밍에 대한 학문이 아닙니다.

이 과목에서는 프로그래밍을 컴퓨팅 사고 (computational thinking)를 가르치기 위해 사용합니다.

- 컴퓨터를 사용한 문제 해결
- 여러 단계의 추상화에 대한 이해.
문제를 여러 문제로 나눠서 푸는 방법.
- 사람의 체계적인 사고 방법 (\neq 컴퓨터처럼 생각하기)
- 문제 해결 방법에 대한 생각 (알고리즘)

30년 전에는 과학, 엔지니어링 분야에서 수식을 통해 문제를 해결했다면, 최근에는 알고리즘을 통해 문제를 해결합니다. (DNA, 단백질 구조, 화학 반응, 논리학, 공장 구조 계획 등)



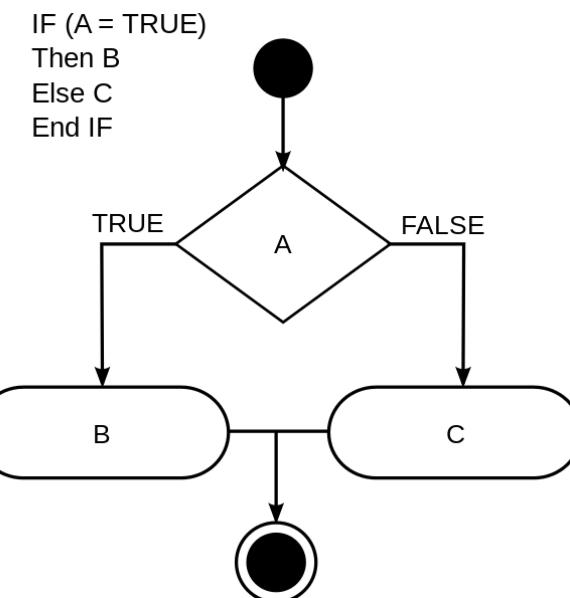
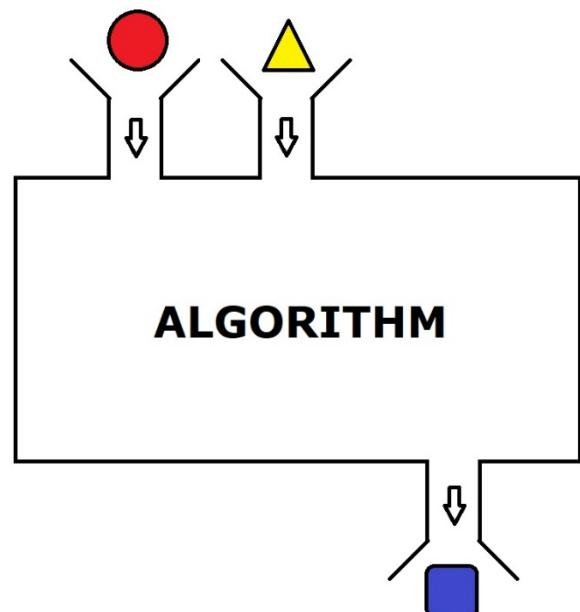
프로그램이란 문제를 해결하거나, 목표를 달성하기 위한 순차적인 명령입니다.

(예. 친구에게 전화를 걸어 빵을 굽는 방법에 대해 설명해 보세요.)

명령(Instruction)이란 컴퓨터가 수행할 수 있는 일 하나하나를 의미합니다.

하지만, 우리는 명령들을 결합하여 보다 추상적인 새로운 명령을 정의할 수 있습니다.

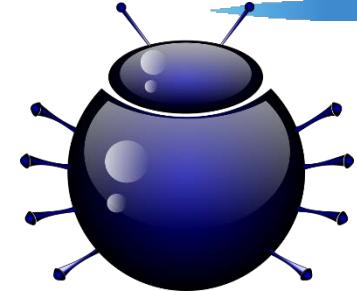
프로그램은 알고리즘을 사용합니다. (문제를 풀기 위한 방법)



디버깅이란?

버그(Bug)는 프로그램에서 잘못된 부분을 말합니다.

디버깅(Debugging)이란 이런 부분을 찾고, 고치는 일을 말합니다.



컴퓨터 프로그램은 굉장히 복잡한 구성을 가지고 있습니다. 디버깅은 과학의 실험과 비슷합니다 - 실험을 하고, 가설을 세우고, 프로그램을 변경해서 가설을 증명합니다.

오류에는 다음과 같은 종류가 있습니다.

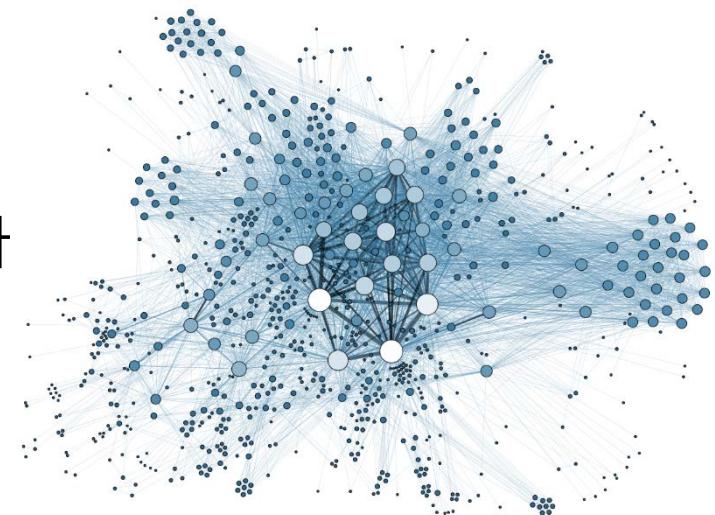
- **Syntax error.** Python이 작성한 프로그램을 이해하지 못합니다. 프로그램을 실행할 수 없습니다.
- **Runtime error.** 프로그램 실행 중에(runtime) 에러 메시지와 함께 프로그램이 갑자기 종료되는 것을 말합니다.
- **Semantic error.** 프로그램이 에러 메시지 없이 실행되지만, 사용자가 기대하지 않은 실행 결과가 나오는 것을 말합니다.

왜 프로그래밍은 유용한가?

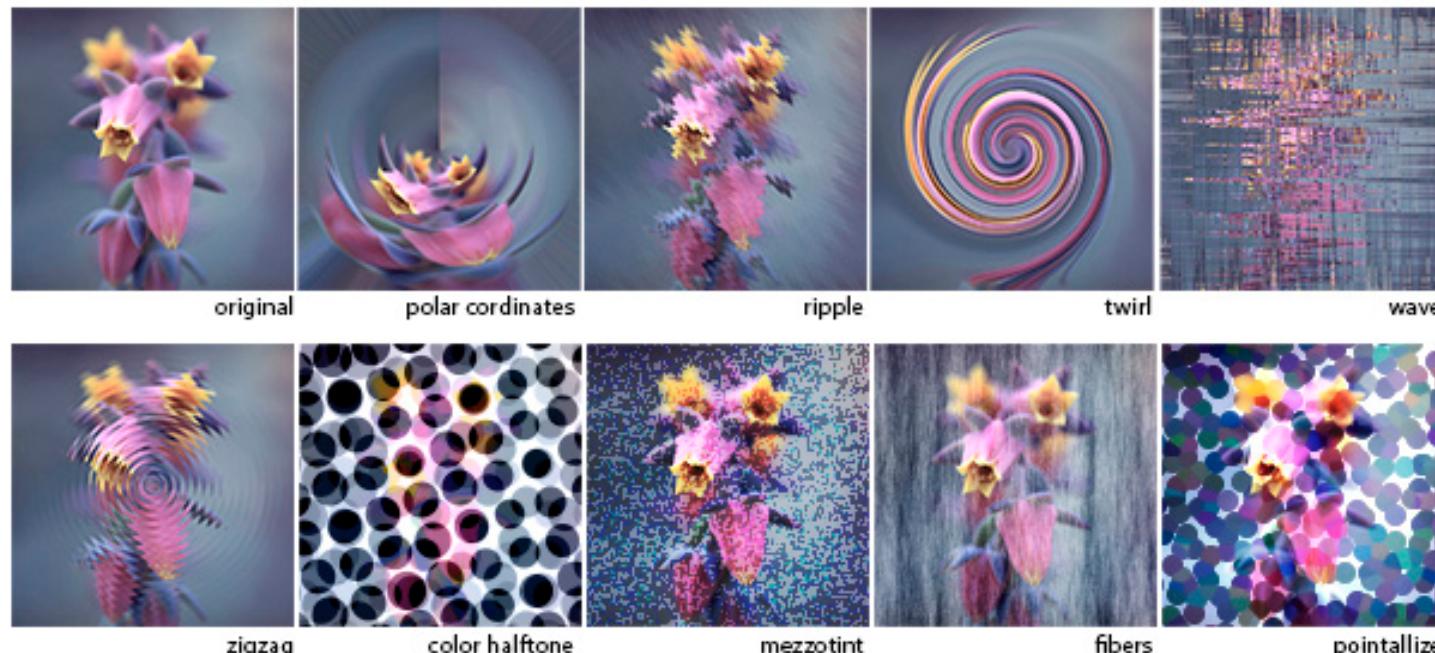
- 20년 전 전자과 학생들은 전기 회로에 대해 배웠습니다.
이제 학생들은 임베디드 시스템에 대해 배웁니다.
- 산업 공학자들은 산업 로봇을 만들기 위한 프로그램을 만들거나,
컴퓨터로만 풀 수
있는 문제들을 풀기도 합니다.
- 자동차는 반도체와 수백만 줄로 이루어진 프로그램을 사용하고
있습니다.
- 수학자들은 그래프나 집합 등의 수학적 구조들을 프로그램을
통해 만들어 연구의 단서를 찾기 위해 사용합니다.
- 실험 결과는 필요성에 맞게 가공되거나 분석될 필요가
있습니다.



Python을 사용하면 이런 일들을 훨씬 쉽게 할 수 있습니다
또한, 최근의 실험 결과들은 사람들이 직접 분석할 수 없을
만큼 커지고 있습니다.



- 자신이 원하는 일을 하는 프로그램을 만들 수 있습니다.
다른 사람이 만든 프로그램이 자신의 목적과 완전히 일치하기는 어렵습니다.
- 예를 들어 사용자가 포토샵만 사용한다면 자유자재로 영상물을 만들기는 힘듭니다.
기존 프로그램의 제한 때문에 사용자의 의도를 모두 표현할 수는 없습니다.
- 프로그래밍은 당신이 하고 싶은 일을 자유롭게 할 수 있게 해 줍니다.



왜 프로그래밍은 재미있는가?

프로그래밍은 창작 활동입니다.

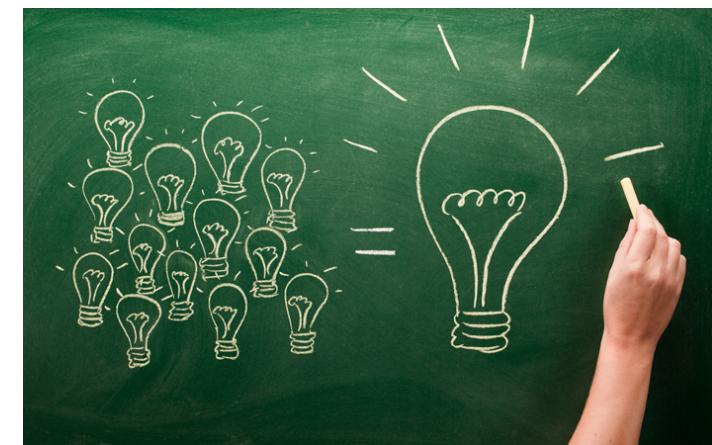
우주선의 복잡한 소프트웨어 시스템을 혼자서도 만들 수 있습니다.
다른 어떤 분야에서도 비슷한 일을 하기는 힘듭니다.

프로그래머들은 거대한 **오픈 소스 커뮤니티**를 만들고 있습니다.

프로그램을 재미 삼아 만드는 사람이나, 자신의 프로그램을 공개하고 싶은 사람들은
프로그램 코드를 무료로 인터넷에 공개하고 있습니다. 공개된 코드는 누구나 사용할 수
있고, 목적에 맞게 코드를 바꿔서 사용할 수도 있습니다.



GitHub



이 수업은 프로그래밍과 컴퓨팅 사고(Computational Thinking)에 대해 배우는 과목입니다.
프로그래밍 언어를 배우는 과목이 아닙니다.

Python은 쉽게 배울 수 있고, 유용하게 쓸 수 있는 프로그래밍 언어입니다.

- 인공지능 프로그램 작성에 많이 쓰이고 있습니다.
- NASA 등에서 수학자나 물리학자들이 과학적 계산을 위해 널리 사용합니다.
- Google에서 웹 프로그래밍을 할 때 사용하는 주요 언어입니다.
- 스마트폰과 같은 임베디드 환경에서도 사용할 수 있습니다.
- 많은 게임들의 (예. 문명 4) 개발에 Python을 사용하고 있습니다.

하나의 프로그래밍 언어를
배워두면, C++이나 Java같은
다른 프로그래밍 언어를
더 쉽게 배울 수 있습니다.



Python 실습 준비 (1/4)

- Python version 3 설치
 - 간결하고 기본적인 사용
 - <https://www.python.org/downloads/release/python-370/>
- (optional) PyCharm IDE 설치
 - 다양한 프로그래밍 및 디버깅 기능
 - <https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows&code=PCC>

The screenshot shows the Python 3.7.0 Shell window. The title bar says "Python 3.7.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, Help. The shell area contains the following code and output:

```
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> if True:
...     print("CS101 is my favorite course")
CS101 is my favorite course
>>> if False:
...     print("Every CS101 student will receive an A+")
>>>
```

Ln: 10 Col: 4

The screenshot shows the PyCharm IDE interface. The title bar says "untitled2 [C:\Users\moonzoo\PycharmProjects\untitled2] - test.py [untitled2]". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. The project tree on the left shows a single file "test.py" under the "untitled2" project. The code in "test.py" is:

```
if True:
    print("CS101 is my favorite course")
if False:
    print("Every CS101 student will receive an A+")
```

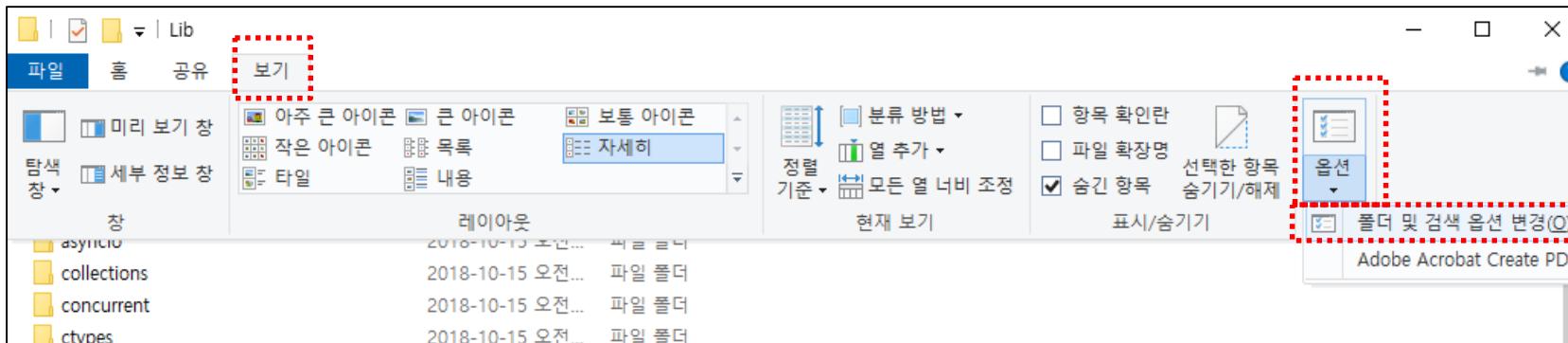
The run tool window at the bottom shows the output of the code execution:

```
C:\Users\moonzoo\PycharmProjects\untitled2\venv\Scripts\python.exe C:/Users/moonzoo.
CS101 is my favorite course
Process finished with exit code 0
```

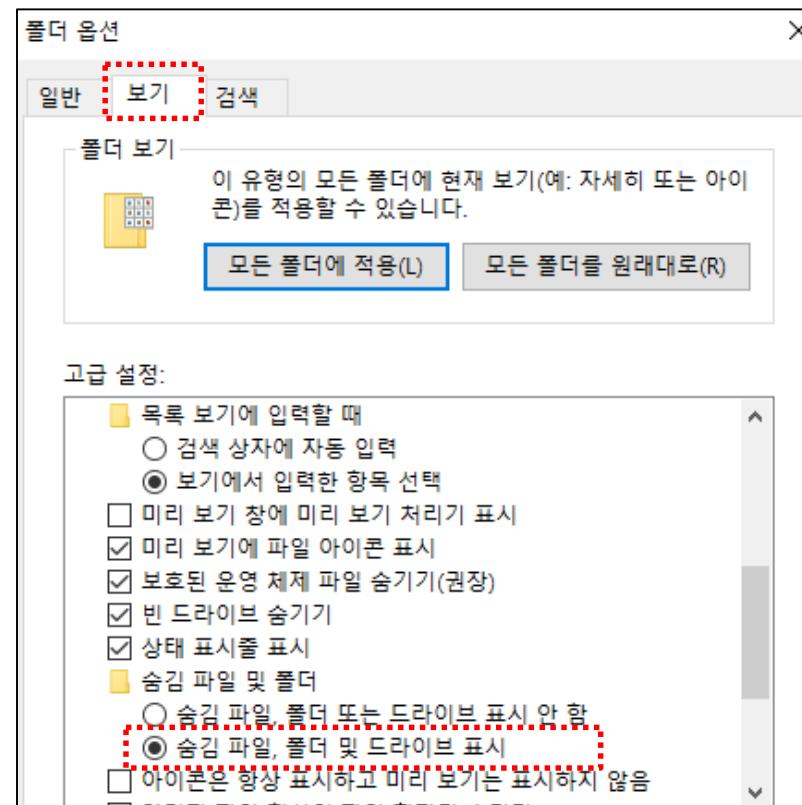
Below the run window are tabs for Python Console, Terminal, Run, TODO, and Event Log.

Python 실습 준비 (graphics 예제 실습용) (2/4)

- CS101 라이브러리를 다운받는다
 - http://cs101.kaist.ac.kr/assets/files/cs101_libraries_py35.zip
- 파일 탐색기 -> 보기 탭 -> 옵션 -> 폴더 및 검색 옵션 변경 선택

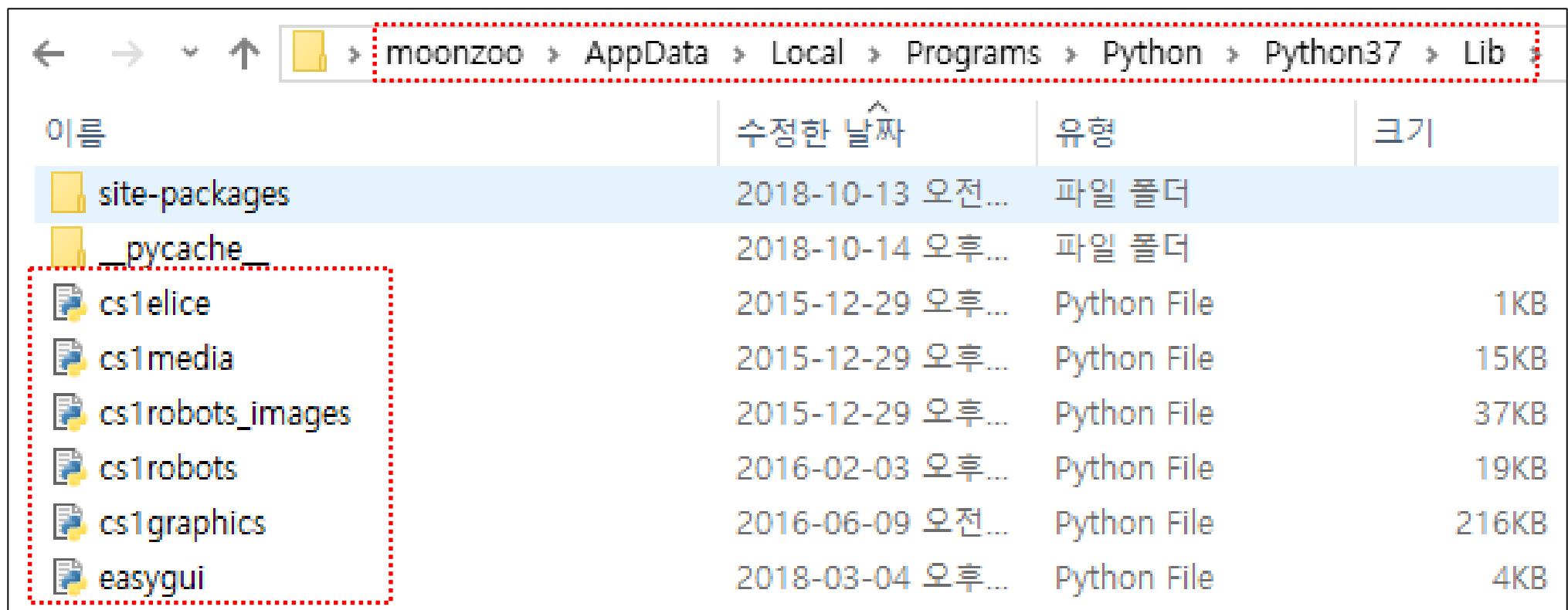


- 오른쪽 팝업 윈도우 -> 보기 탭-> 고급 설정 -> “숨김 파일, 폴더 및 드라이브 표시” 항목 선택



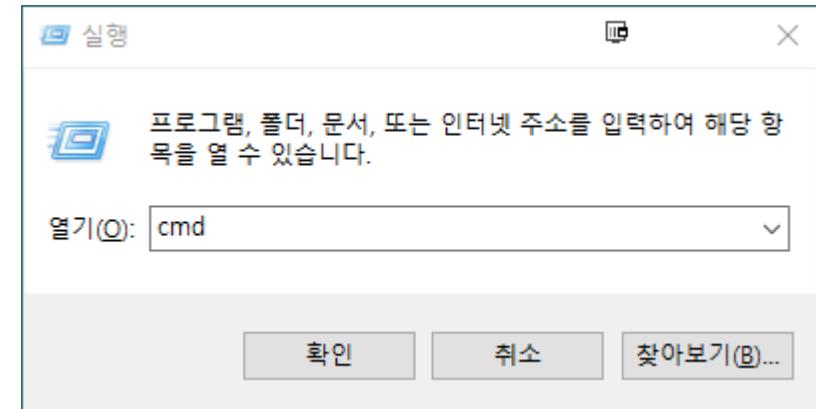
Python 실습 준비 (graphics 예제 실습용) (3/4)

- 다운 받은 압축화일을 다음의 폴더에 푼다
 - C:\Users\[username]\AppData\Local\Programs\Python\[pythonversion]\Lib
 - 또는
C:\사용자\[username]\AppData\Local\Programs\Python\[pythonversion]\Lib



Python 실습 준비 (graphics 예제 실습용) (4/4)

- Pillow 설치 (python imaging library)
 - 운영체제의 “명령 프롬프트”를 연다
 - 예. 윈도우 키 + R 을 누른후 cmd 실행



- 'py -m pip install Pillow' 혹은 'python -m pip install Pillow'를 입력해 Pillow 라이브러리를 설치한다

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\moonzoo>py -m pip install Pillow
Collecting Pillow
  Downloading https://files.pythonhosted.org/packages/55/ea/305f61258278790706e69f01c53e10
  7b0830ea5a4a69aa1f2c11fe605ed3/Pillow-5.3.0-cp37-cp37m-win_amd64.whl (1.6MB)
    100% |████████████████████████████████| 1.6MB 1.0MB/s
Installing collected packages: Pillow
Successfully installed Pillow-5.3.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

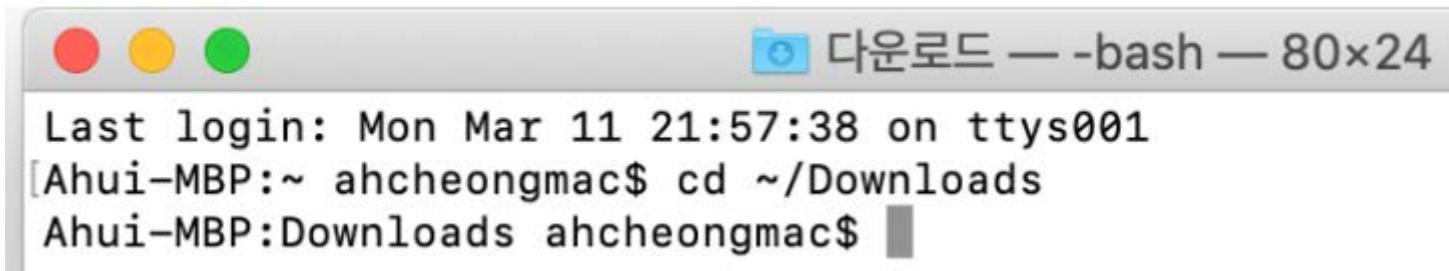
C:\Users\moonzoo>
```

A screenshot of a Windows Command Prompt window. The title bar shows "C:\WINDOWS\system32\cmd.exe". The window displays the output of the command "py -m pip install Pillow". It shows the package being downloaded from "https://files.pythonhosted.org" and successfully installed. A message at the end encourages upgrading the pip version. The command prompt prompt is "C:\Users\moonzoo>".

MAC에서 Python 실습 준비 (graphics 예제 실습용)

1. 터미널 실행 후 cd ~/Downloads 입력 후 Enter키

- cs101관련 파일을 다운 받은 디렉토리로 이동



```
Last login: Mon Mar 11 21:57:38 on ttys001
[Ahui-MBP:~ ahcheongmac$ cd ~/Downloads
Ahui-MBP:Downloads ahcheongmac$
```

2. cp -r cs101_libraries_py35/

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7

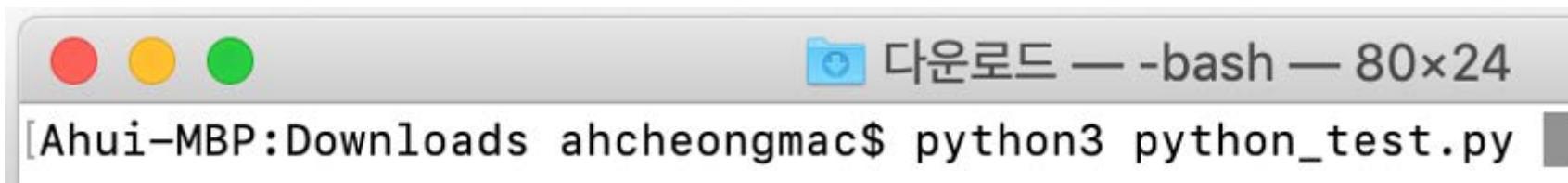
- 실행하여 다운 받은 library 파일들을 python library 기본 폴더로 복사

3. pip3 install Pillow 실행하여 Pillow 설치



```
Ahui-MBP:~ ahcheongmac$ pip3 install Pillow
```

4. python3 [filename].py로 작성한 python 코드 실행



```
[Ahui-MBP:Downloads ahcheongmac$ python3 python_test.py
```

수업 로드맵

1. CS101 강의 소개
2. 조건문과 while 반복문
3. 변수와 기초 자료형
4. 매개 변수와 반환값을 가진 함수
5. 지역/전역 변수와 그래픽 객체들
6. 시퀀스: 리스트, 문자열, 튜플
7. 다양한 자료구조와 (문자열, 집합, 사전)
8. 자료구조를 활용한 이미지 및 텍스트 프로세싱
9. 객체: 생성과 속성
10. 프로그램 작성 고급 기술

정리 및 예습

본 강의 학습 목표:

- 이 수업의 목표 및 수업에서 배울 내용의 범위를 이해 할 수 있다.
- 프로그래밍의 유용성을 알 수 있다.
- Python 실습에 필요한 SW를 설치할 수 있다.

다음 강의 학습 목표:

- Python 프로그램의 형태 및 동작을 이해할 수 있다.
- 하향식 (Top-down) 프로그램 설계 방식을 이해 할 수 있다.

CS101 – Python 프로그램 작성 예제

Lecture 2

School of Computing
KAIST

학습 목표:

- Python 프로그램의 형태 및 동작을 이해할 수 있다.
- 하향식 (Top-down) 프로그램 설계 방식을 이해 할 수 있다.

이제 몇 가지 Python 코드를 살펴봅시다.

- Python 상호작용
- Python 프로그램 (스크립트)
- 주석
- 함수
- 키워드
- for 반복문
- 들여쓰기

함수는 여러 개의 프로그램 명령어들을 모아 놓은 것입니다.

즉, 함수는 새로운 함수의 이름과 함수가 호출될 때 실행될 명령들로 만듭니다
(함수의 정의란, 함수의 이름과 실행할 명령들을 모아놓은 코드입니다)

```
def print_message():
    print("CS101 is fantastic!")
    print("Programming is fun!")

    ↗ 키워드
    ↗ 콜론 (:)
    ↗ 들여쓰기
```

함수 안에서 다른 함수를 호출할 수 있습니다.

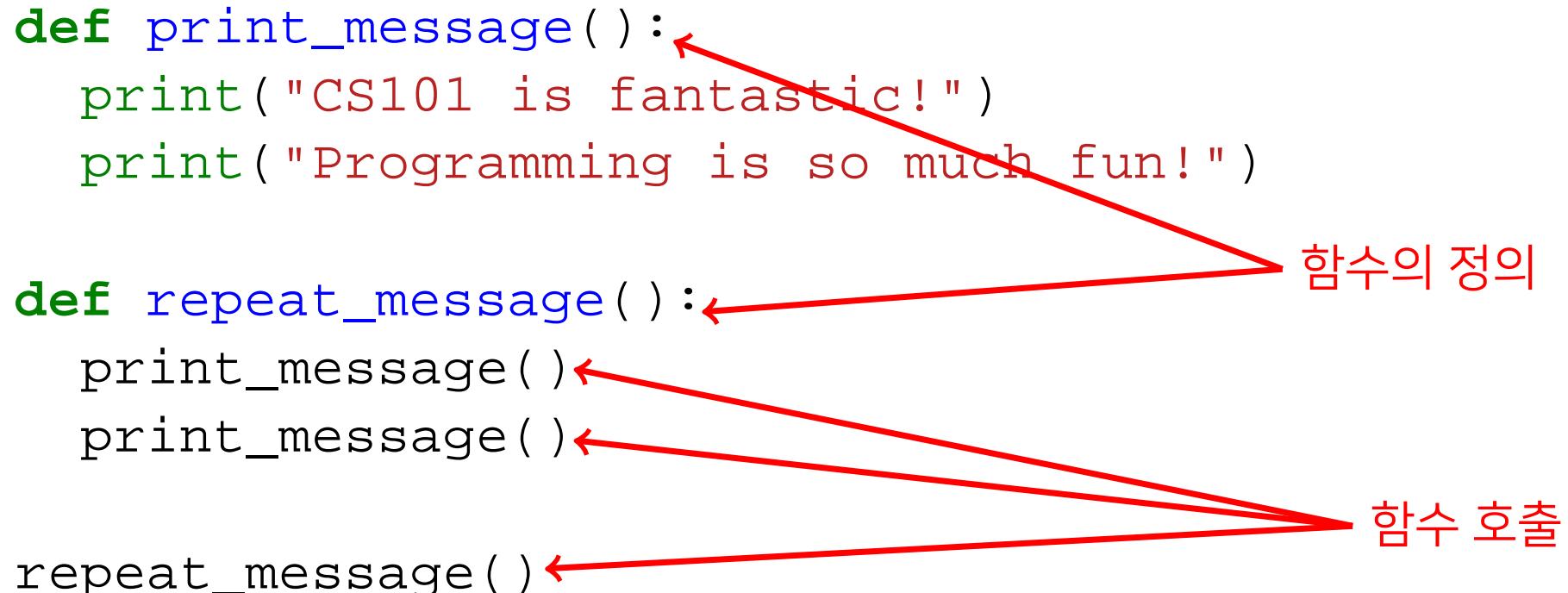
```
def repeat_message():
    print_message()
    print_message()
```

실행 순서

```
def print_message():  
    print("CS101 is fantastic!")  
    print("Programming is so much fun!")  
  
def repeat_message():  
    print_message()  
    print_message()  
  
repeat_message()
```

함수의 정의

함수 호출



프로그램은 첫 번째 명령부터 실행합니다. 명령은 위에서 아래로 하나씩 실행합니다.
함수의 정의는 함수를 정의할 뿐, 정의한 함수가 자동으로 실행 되지는 않습니다.
함수를 호출하면, 그 함수의 정의대로 실행한 후 함수 호출을 종료합니다.

실습

```
# create a robot with one beeper
hubo = Robot(beepers = 1) ← 인자의 기본값을 가진 객체

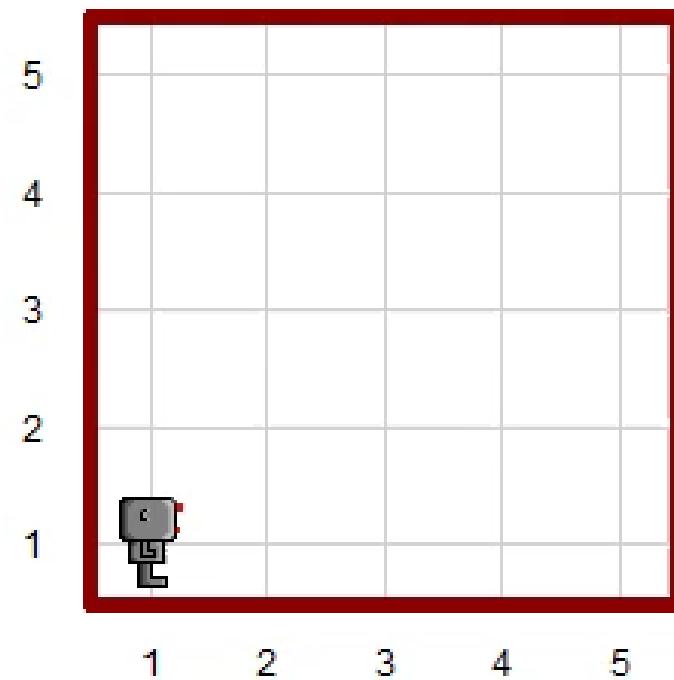
# move one step forward
hubo.move() ← 멤버 함수: 점(.)을 사용하여 표기

# turn left 90 degrees
hubo.turn_left() ←
```

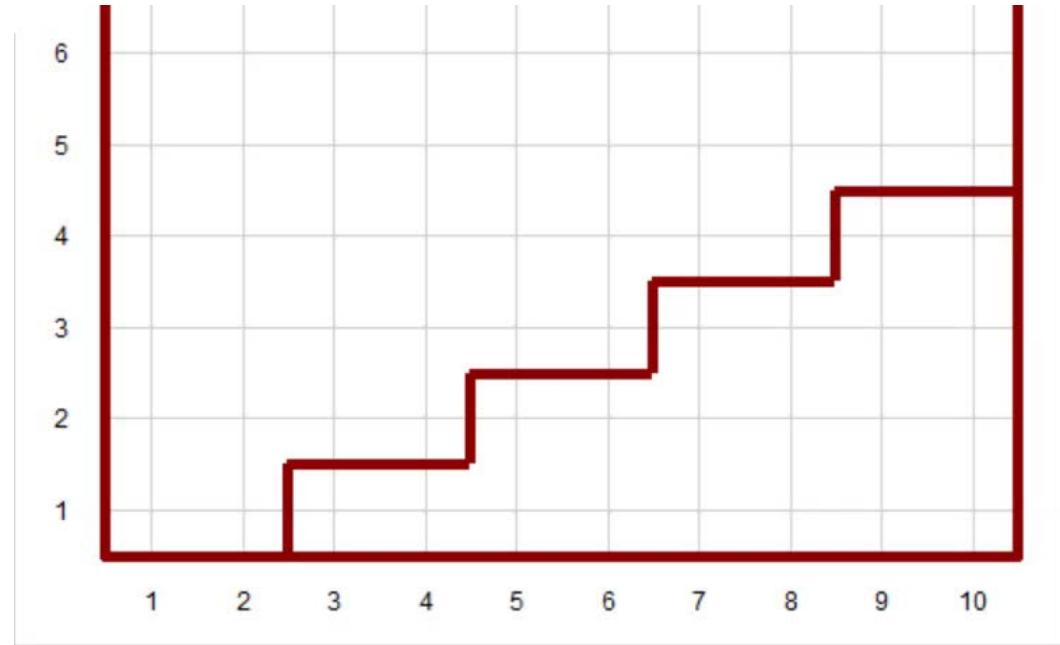
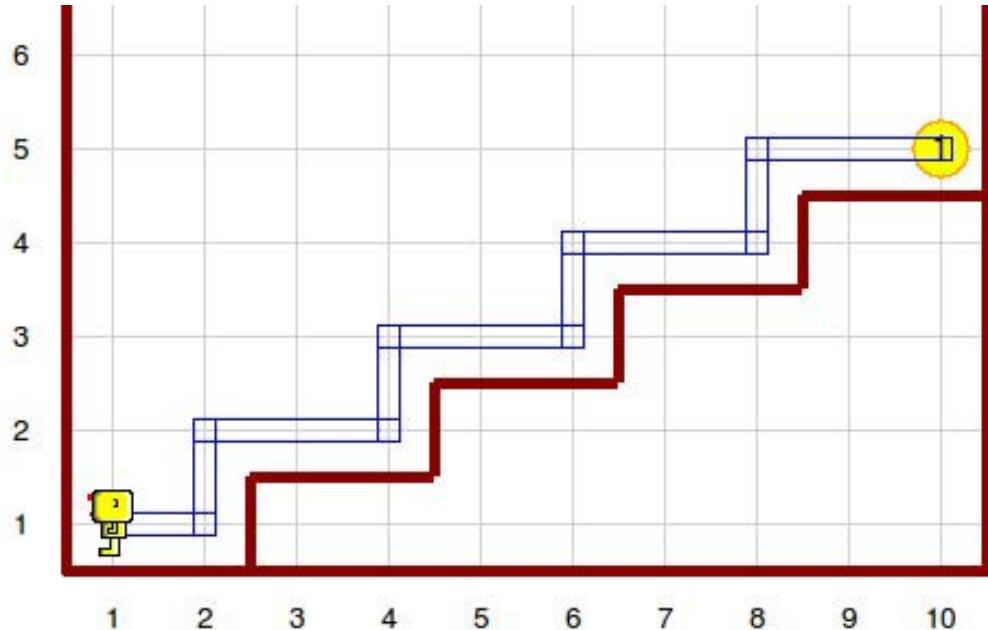
hubo는 어떻게 우회전을 할 수 있을까요?

새로운 함수를 만들어서 사용합시다.

```
def turn_right():
    hubo.turn_left()
    hubo.turn_left()
    hubo.turn_left()
```



휴보가 계단을 올라가서 문 앞에 신문을 놓고, 처음 위치로 돌아가게 하고 싶습니다.



문제의 개요:

- 계단을 4칸 올라가서
- 신문을 놓고
- 돌아서서
- 계단을 4칸 내려간다

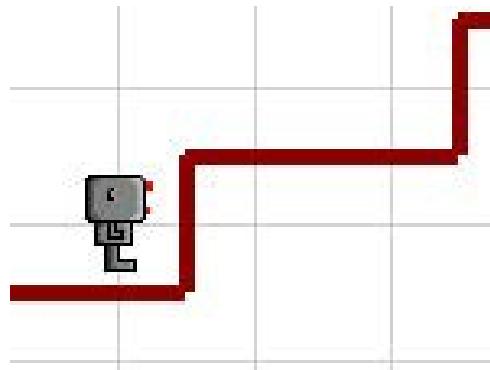
Python으로 만들면

```
climb_up_four_stairs()
hubo.drop_beeper()
turn_around()
climb_down_four_stairs()
```

```
def turn_around():
    hubo.turn_left()
    hubo.turn_left()

def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()

def climb_up_one_stair():
    hubo.turn_left()
    hubo.move()
    turn_right()
    hubo.move()
    hubo.move()
```



하향식 설계 (Top-down design)

하나의 큰 문제를 중간 크기의 여러 문제로 나누고, 중간 크기의 문제 하나 하나를 어떻게 해결할 지 파악합니다.

그리고, 중간 크기의 문제에 대한 해결책을 작성하기 위해 더 작은 크기의 문제들로 나눕니다.

문제가 쉽게 풀수 있을 만큼 작아지면, 작은 문제에 대한 해결책을 작성한 뒤, 그 해결 책들을 모아서 보다 큰 문제에 대한 해결책으로 활용합니다.

동일한 명령을 4번 반복해서 실행해봅시다.

```
for i in range(4):  
    print("CS101 is fantastic!")
```

for-반복문
들여쓰기를 잊지 마세요!

다음 코드와

```
for i in range(4):  
    print("CS101 is great!")  
    print("I love programming!")
```

다음 코드의 차이점은 무엇일까요?

```
for i in range(4):  
    print("CS101 is great!")  
print("I love programming!")
```

실습

```
def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
```

프로그램을 작성 할 때, 동일한 내용의 코드를 여러 번 쓰는 것은 피해야 합니다.
for 반복문은 위 코드를 좀 더 깔끔하게 쓸 수 있게 도와줍니다.

```
def climb_up_four_stairs():
    for i in range(4):
        climb_up_one_stair()
```

정리 및 예습

본 강의 학습 목표:

- Python 프로그램의 형태 및 동작을 이해할 수 있다.
- 하향식 (Top-down) 프로그램 설계 방식을 이해 할 수 있다.

다음 강의 학습 목표:

- if 조건문의 형태 및 동작을 이해 할 수 있다.
- while 반복문의 형태 및 동작을 이해할 수 있다.

CS101 - if 조건문과 while 반복문

Lecture 3

School of Computing
KAIST

학습 목표:

- if 조건문의 형태 및 동작을 이해 할 수 있다.
- while 반복문의 형태 및 동작을 이해할 수 있다.

지금까지 우리가 만든 프로그램은 실행될 때마다 동일한 작업을 수행했습니다.

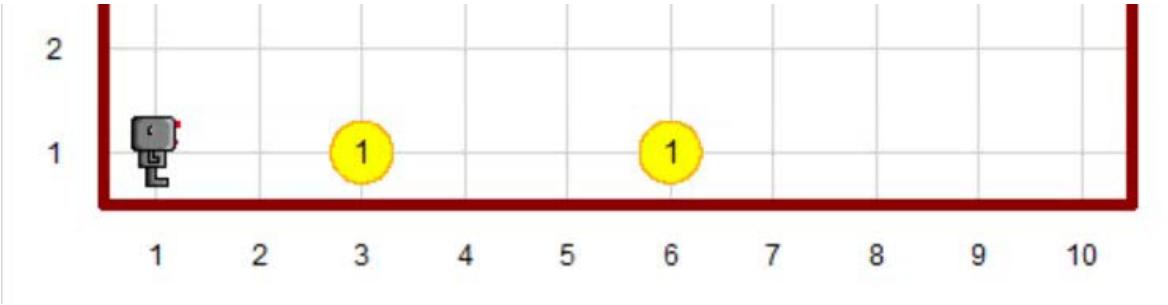
하지만, 로봇은 환경(상황)에 의존해서 움직여야 할 때가 자주 있습니다.

```
if it rains: ← 조건  
    listen_to_cs101_lecture()  
else:           ← 조건이 참이면, 이 작업을 수행한다  
    eat_strawberries_in_the_sun()  ← 조건이 거짓이면, 이 작업을 수행한다
```

조건은 참(**True**)이나 거짓(**False**) 값을 가질 수 있습니다.

```
if True:  
    print("CS101 is my favorite course")  
  
if False:  
    print("Every CS101 student will receive an A+")  
  
if 3 < 5:  
    print("3 is less than 5")  
else:  
    print("3 is larger than 5")
```

로봇을 9칸 전진시키면서 경로에 있는 모든 비퍼를 줍도록 하려고 합니다.



hubo.pick_beeper()는 비퍼가 없을 때 에러가 발생합니다.

다음 과정을 9회 반복합니다 :

- 한 칸 전진한다
- 비퍼가 있는지 확인한다
- 비퍼가 있으면, 비퍼를 줍는다

```
def move_and_pick():
    hubo.move()
    if hubo.on_beeper():
        hubo.pick_beeper()

for i in range(9):
    move_and_pick()
```

방금 과정을 반대로 해 봅시다

: 현재 위치에 비퍼가 **없을** 때만 비퍼를 떨어뜨리고 싶습니다.

```
if not hubo.on_beeper():  
    hubo.drop_beeper()
```

not 키워드는 조건을 반대로 바꿉니다.

: **not True**는 **False**이고, **not False**는 **True**입니다.

다음 코드의 결과는 어떻게 될까요?

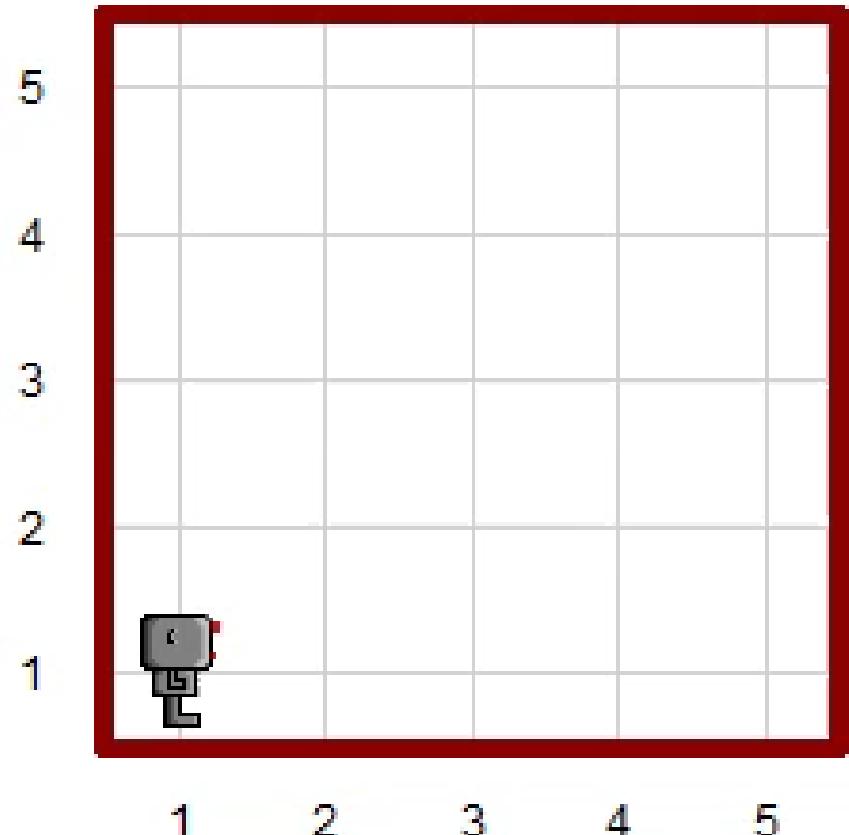
```
print(not 3 < 5)
```

else란

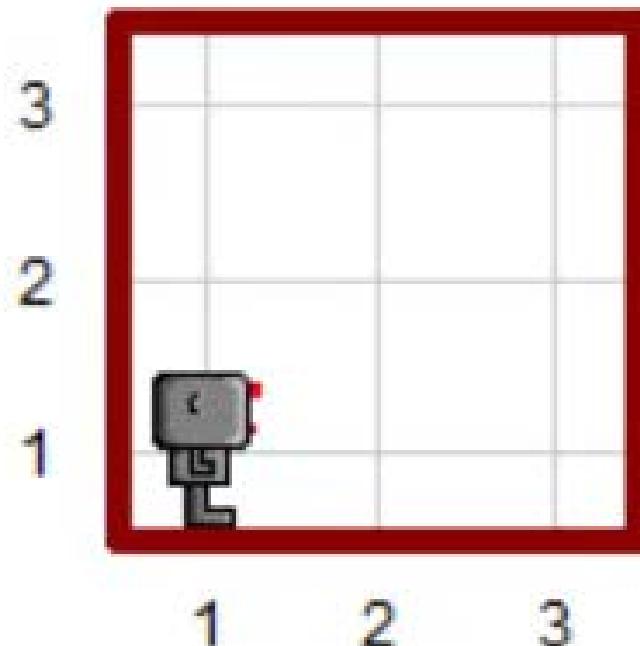
로봇이 세계의 경계선을 따라서 움직이게 해 봅시다.

: 전방에 벽이 없으면 전진하고, 벽이 있으면 좌회전합니다.

```
def move_or_turn( ) :  
    if hubo.front_is_clear( ) :  
        hubo.move( )  
    else :  
        hubo.turn_left( )  
  
for i in range(20) :  
    move_or_turn( )
```



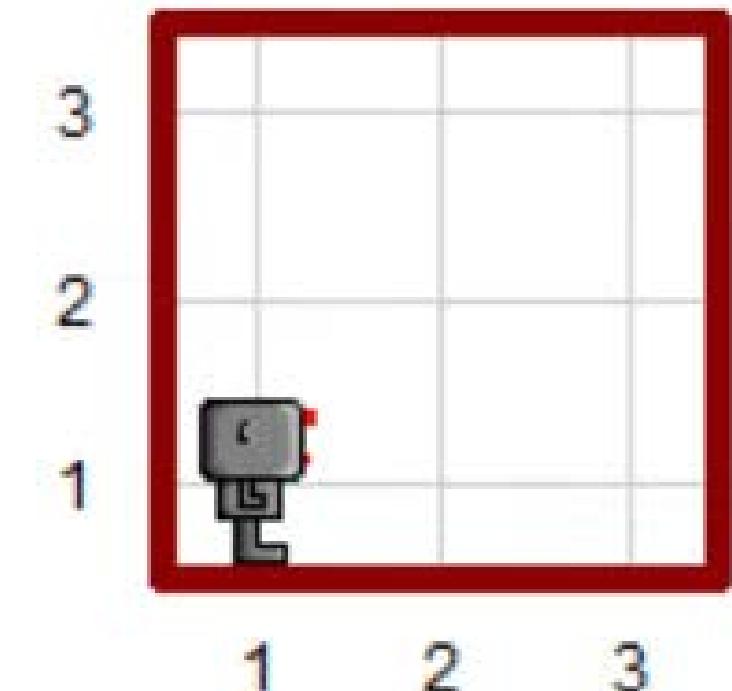
```
def dance( ):  
    for i in range(4):  
        hubo.turn_left()  
  
def move_or_turn( ):  
    if hubo.front_is_clear():  
        dance()  
        hubo.move()  
    else:  
        hubo.turn_left()  
        hubo.drop_beeper()  
  
for i in range(18):  
    move_or_turn()
```



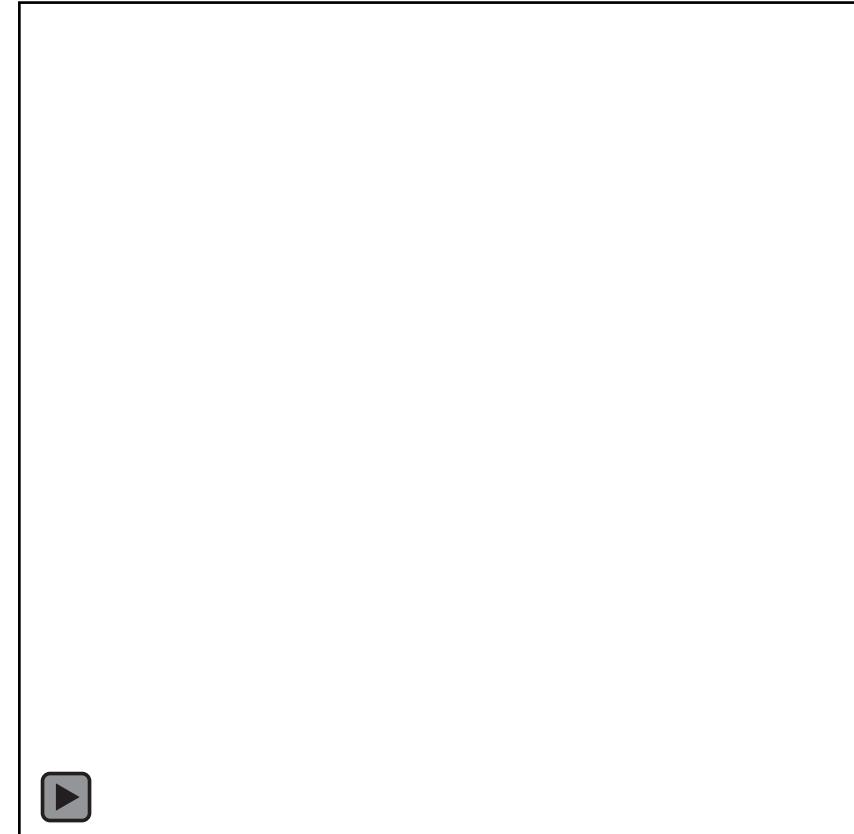
```
def dance( ):  
    for i in range(4):  
        hubo.turn_left()  
  
def move_or_turn( ):  
    if hubo.front_is_clear():  
        dance()  
        hubo.move()  
    else:  
        hubo.turn_left()  
    hubo.drop_beeper()  
  
for i in range(18):  
    move_or_turn()
```

들여쓰기에 주의하세요!

이제 로봇이 어떻게 행동할까요?



```
def dance( ):  
    for i in range(4):  
        hubo.turn_left()  
  
def move_or_turn( ):  
    if hubo.front_is_clear():  
        dance()  
        hubo.move()  
    else:  
        hubo.turn_left()  
hubo.drop_beeper()  
  
for i in range(18):  
    move_or_turn()
```



... 이렇게 바꾸면요?

```
if hubo.on_beeper():
    hubo.pick_beeper()
else:
    if hubo.front_is_clear():
        hubo.move()
    else:
        if hubo.left_is_clear():
            hubo.turn_left()
        else:
            if hubo.right_is_clear():
                turn_right()
            else:
                turn_around()
```

문제점) 이 코드는 읽고 이해하기가 너무 힘들어요!

```
if hubo.on_beeper():
    hubo.pick_beeper()
elif hubo.front_is_clear():
    hubo.move()
elif hubo.left_is_clear():
    hubo.turn_left()
elif hubo.right_is_clear():
    turn_right()
else:
    turn_around()
```

elif 는 **else** 와 **if** 를 결합시킨 것으로,
복잡한 들여쓰기 없이 많은 연관된 조건문들을 표현할 수 있습니다.

while 반복문

for 반복문은 정해진 횟수만큼 명령을 반복합니다.

while 반복문은 주어진 조건이 참이라면 명령을 계속 반복합니다.

비퍼를 발견하기 전까지 계속 전진하려면

```
while not hubo.on_beeper( ) :
```

```
    hubo.move( )
```

정리 및 예습

본 강의 학습 목표:

- if 조건문의 형태 및 동작을 이해할 수 있다.
- while 반복문의 형태 및 동작을 이해 할 수 있다.

다음 강의 학습 목표:

- if 조건문과 while 반복문을 사용하여 복잡한 미로를 탈출하는 프로그램을 작성할 수 있다.
- 프로그램을 작성할 때 따라야 하는 과정을 이해 할 수 있다.

CS101 - if 와 while 을 사용한 미로 탈출 예제

Lecture 4

School of Computing
KAIST

학습 목표:

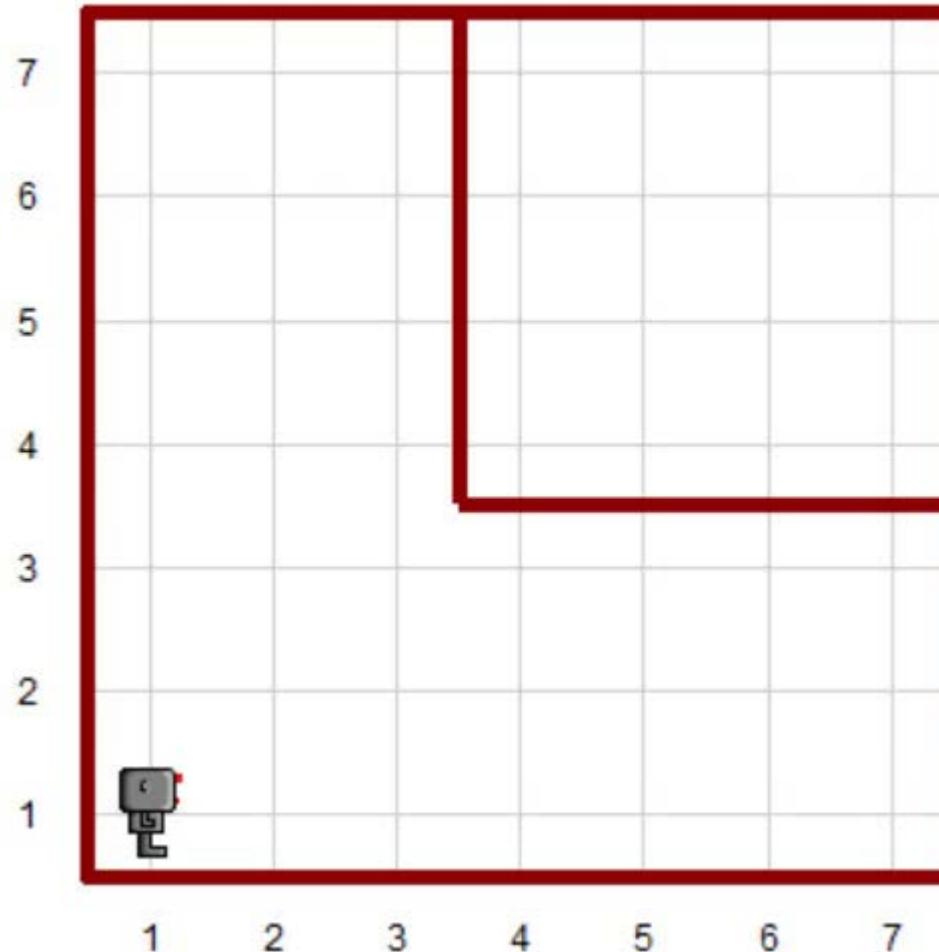
- if 조건문과 while 반복문을 사용하여 복잡한 미로를 탈출하는 프로그램을 작성할 수 있다.
- 프로그램을 작성할 때 따라야 하는 과정을 이해 할 수 있다.

로봇이 시작점에 다시 도착할 때까지 세계의 경계선을 따라 움직이는 프로그램을 만들어 봅시다.

해결 과정 :

- ① 시작점에 비퍼를 놓는다
- ② 벽을 만날 때까지 전진한다
- ③ 좌회전한다
- ④ 2, 3번 과정을 비퍼를 발견할 때까지 반복한다
- ⑤ 비퍼를 발견하면 종료한다

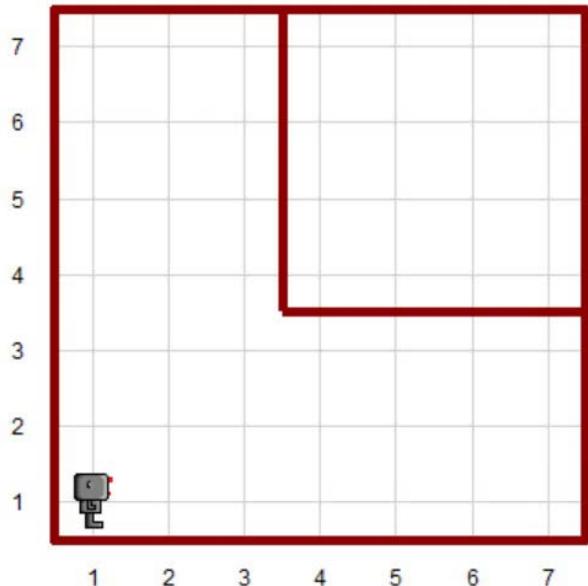
```
hubo.drop_beeper()
while not hubo.on_beeper():
    if hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
```



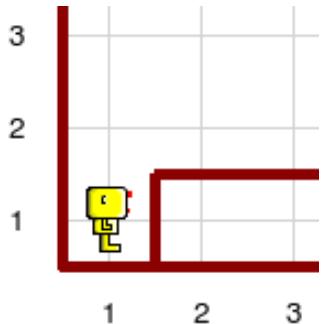
이전 페이지에서 만든 코드를 “amzing2.wld”에서 실행시켜보고,
프로그램이 제대로 동작하는지 살펴봅시다.

```
hubo.drop_beeper()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
```

이 코드는
무한 루프에
빠질 수 있어요!

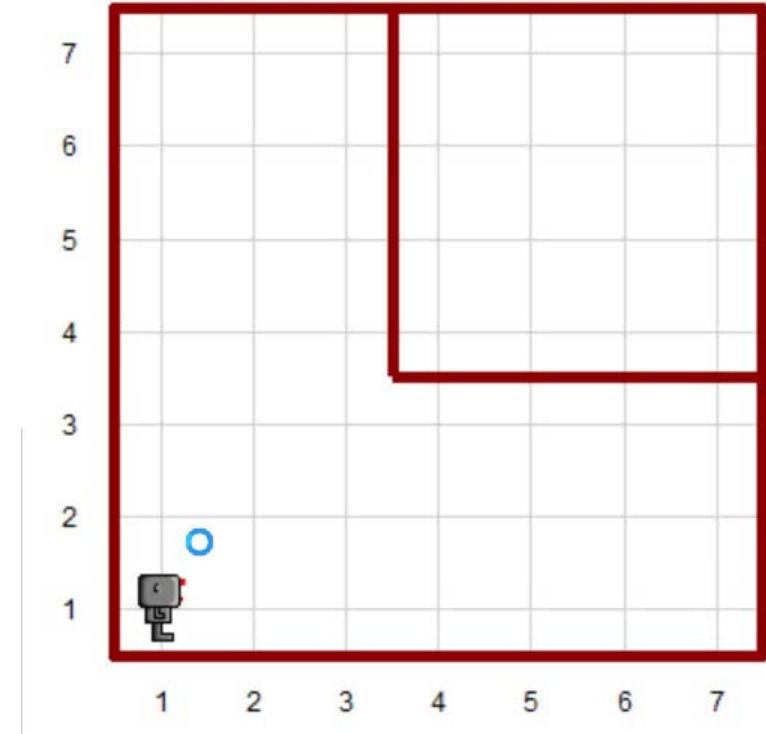


그리고 시작점
바로 앞에 벽이
있는 경우에는
제대로 움직이지
않아요!

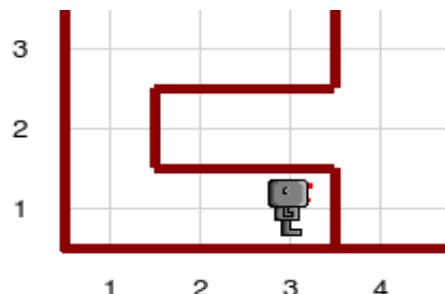


곤란한 시작점에서 빠져 나가기

```
hubo.drop_beeper()
if not hubo.front_is_clear():
    hubo.turn_left()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
```

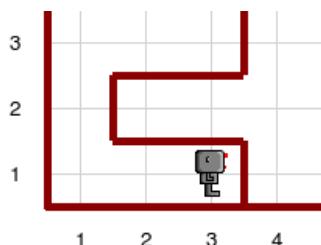


하지만 시작점이 (1,1)이 아닐 때는 여전히 문제가 있어요.



곤란한 시작점에서 빠져 나가기

```
hubo.drop_beeper()
while not hubo.front_is_clear():
    hubo.turn_left()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
```



사람이 이해하기 좋은 코드 작성

정확하고 품격 있는 프로그램을 작성하기 위한 한 가지 방법은,
컴퓨터가 아닌 사람이 읽을 것이라 생각하고 작성하는 것입니다.
지금까지 만든 프로그램을 정리해봅시다.

```
# This program lets the robot go around her world
# counter clockwise, stopping when he returns
# to the starting point.
from cs1robots import *
load_world()
hubo = Robot(beepers=1)

def turn_right():
    for i in range(3):
        hubo.turn_left()

def mark_starting_point_and_move():
    hubo.drop_beeper()
    while not hubo.front_is_clear():
        hubo.turn_left()
    hubo.move()
```

```
def follow_right_wall():
    if hubo.right_is_clear():
        # Keep to the right
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        # move following the right wall
        hubo.move()
    else:
        # follow the wall
        hubo.turn_left()

# end of definitions, begin solution

mark_starting_point_and_move()

while not hubo.on_beeper():
    follow_right_wall()
```

프로그램 작성시 추천 방법:

- 간단하게 시작하자
- 한 번에 하나의 작은 작업만 수행하자
- 각각의 작업들이 이전에 했던 작업에 영향을 주지 않도록 하자
- 알기 쉬운 유용한 주석을 달자
(프로그램 명령들을 있는 그대로 글로 쓰지는 말자)
- 의미를 잘 전달하는 이름을 고르자

정리 및 예습

본 강의 학습 목표:

- if 조건문과 while 반복문을 사용하여 복잡한 미로를 탈출하는 프로그램을 작성할 수 있다.
- 프로그램을 작성할 때 따라야 하는 과정을 이해 할 수 있다.

다음 강의 학습 목표:

- 객체 (objects)와 형태를(types) 이해할 수 있다.
- 변수 (variables)를 이해할 수 있다.

CS101 - 프로그램에서 사용하는 객체와 객체의 형태

Lecture 5

School of Computing
KAIST

학습 목표:

- 객체 (objects)와 형태를(types) 이해할 수 있다.
- 변수 (variables)를 이해할 수 있다.

프로그램은 실행 중에 여러 데이터를 사용합니다.

Python 프로그램에서 사용하는 각각의 데이터는 객체(Object)라 부릅니다.

객체의 크기는 아주 작을 수도 있고 (숫자 3), 매우 클 수도 있습니다 (사진 파일).

모든 객체는 형태(Type)를 가지고 있습니다.

형태는 객체를 이용해 할 수 있는 일을 결정합니다.

Python 동물원

Python 프로그램이 동물원이라고 생각해 봅시다.

객체를 만드는 일은, 동물을 만드는 일에 빗대어 말할 수 있습니다.

동물이 무엇을 할 수 있는지는 동물의 종류(형태)에 따라 결정됩니다.

(예. 새는 날 수 있고, 물고기는 헤엄칠 수 있고, 코끼리는 무거운 물건을 들 수 있습니다.)

동물이 더 이상 사용되지 않으면, 그 동물은 죽습니다 (사라집니다).

객체는 다음과 같은 방법으로 만들 수 있습니다.

숫자: 숫자 그대로 적습니다.

13

3.14159265

-5

3 + 6j

문자열: 문자열을 따옴표(“, ”) 사이에 적습니다.

"CS101 is wonderful"

'The instructor said: "Well done!" and smiled'

논리값(Boolean): **True** 또는 **False**로 적습니다.

복잡한 객체 생성

복잡한 객체는 그 객체를 만드는 함수를 부르는 방법으로 만듭니다.

```
from cs1robots import *
Robot()

from cs1media import *
load_picture("photos/geowi.jpg")
```

튜플(Tuple)은 다른 객체들을 포함하는 객체입니다.
여러 객체들을 쉼표(,)를 사이에 두고 적어서 만들 수 있습니다.

```
( 3, 2.5, 7 )
( "red", "yellow", "green" )
( 20100001, "Hong Gildong" )
```

모든 객체는 **형태(Type)**를 가지고 있습니다.

객체의 형태는 객체가 할 수 있는 일과, 객체를 이용해 할 수 있는 일을 결정합니다.
(예. 두 숫자는 더할 수 있지만, 두 로봇은 더할 수 없습니다)

Python에서는 다음과 같은 방법으로 객체가 어떤 형태를 가지고 있는지 알 수 있습니다.

>>> type(3)	정수: int
<class 'int'>	
>>> type(3.1415)	실수: float
<class 'float'>	
>>> type("CS101 is fantastic")	문자열: str
<class 'str'>	
>>> type(3 + 7j)	복소수: complex
<class 'complex'>	
>>> type(True)	논리값: bool
<class 'bool'>	

복잡한 형태

복잡한 객체들의 형태는 다음과 같이 표시됩니다.

```
>>> type(Robot())
<class 'cs1robots.Robot'>
>>> type((3, -1.5, 7))
<class 'tuple'>
>>> type(load_picture("geowi.jpg"))
<class 'cs1media.Picture'>
```

객체에는 이름을 줄 수 있습니다.

```
message = "CS101 is fantastic"  
n = 17  
hubo = Robot()  
pi = 3.1415926535897931  
finished = True  
img = load_picture("geowi.jpg")
```

n = 17과 같은 문장은 **대입문(Assignment)**이라고 부릅니다.

n이라는 이름이 숫자 17에 붙여져서, n을 숫자 17처럼 사용할 수 있기 때문입니다.



Python 동물원에서, 이름은 동물 우리 앞의 팻말과 같습니다.

변수와 함수의 이름을 지을 때는 다음과 같은 규칙을 따라야 합니다.

- 영어 문자, 숫자, 그리고 밑줄 문자(_)로만 이루어져야 합니다.
- 숫자는 이름의 첫 글자로 올 수 없습니다.
- 파이썬에서 등록된 키워드(예약어)와 동일한 이름은 지을 수 없습니다.
e.g.) **def, if, else, while**
- 이름은 대소문자를 구분합니다
e.g.) Pi와 pi는 다른 이름입니다.

좋은 예시:

```
my_message = "CS101 is fantastic"  
a13 = 13.0
```

나쁜 예시:

```
more@ = "illegal character"  
13a = 13.0  
def = "Definition 1"
```

이름이 가리키는 객체는 바꿀 수 있기에, 이름은 변수(Variable)라고도 불립니다.
다시 말해, 프로그램 실행 중에 변수가 가리키는 객체는 바뀔 수 있습니다.

```
n = 17  
n = "Seventeen"  
n = 17.0
```

변수에 대입된 객체는 변수의 값이라고 부릅니다.
변수의 값은 바뀔 수 있습니다.

None이라는 이름의 특별한 객체는 비었다는 것을 의미하기 위해 사용됩니다.

```
n = None  
>>> type(n)  
<class 'NoneType'>
```

멤버 변수

객체가 할 수 있는 일은 객체의 형태에 따라 결정됩니다.

(예. 새는 날 수 있고, 물고기는 헤엄칠 수 있습니다)

객체는 **멤버 함수(Method)**를 통해 이러한 일들을 할 수 있습니다.

멤버 함수는 점(.) 연산자를 통해 실행할 수 있습니다.

```
>>> hubo = Robot()
>>> hubo.move()
>>> hubo.turn_left()

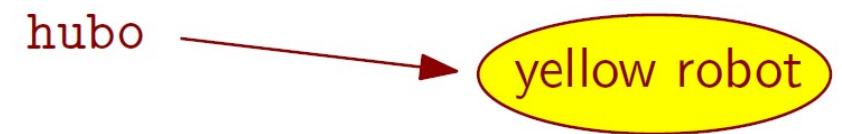
>>> img = load_picture("geowi.jpg")
>>> print(img.size()) # width and height in pixels
(58, 50)
>>> img.show()          # display the image

>>> b = "banana"
>>> print(b.upper())
BANANA
```

여러 이름을 가진 객체

하나의 객체는 여러 이름을 가질 수 있습니다.

```
hubo = Robot( "yellow" )
```



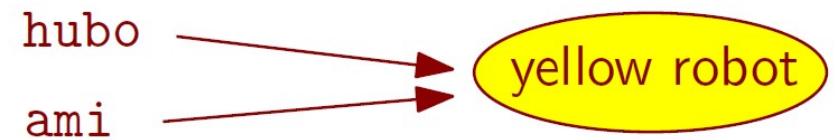
여러 이름을 가진 객체

하나의 객체는 여러 이름을 가질 수 있습니다.

```
hubo = Robot( "yellow" )
```

```
hubo.move()
```

```
ami = hubo
```



여러 이름을 가진 객체

하나의 객체는 여러 이름을 가질 수 있습니다.

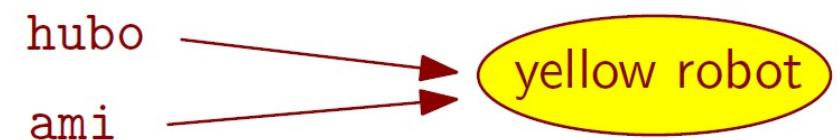
```
hubo = Robot( "yellow" )
```

```
hubo.move()
```

```
ami = hubo
```

```
ami.turn_left()
```

```
hubo.move()
```



여러 이름을 가진 객체

하나의 객체는 여러 이름을 가질 수도 있습니다.

```
hubo = Robot( "yellow" )
```

```
hubo.move( )
```

```
ami = hubo
```

```
ami.turn_left( )
```

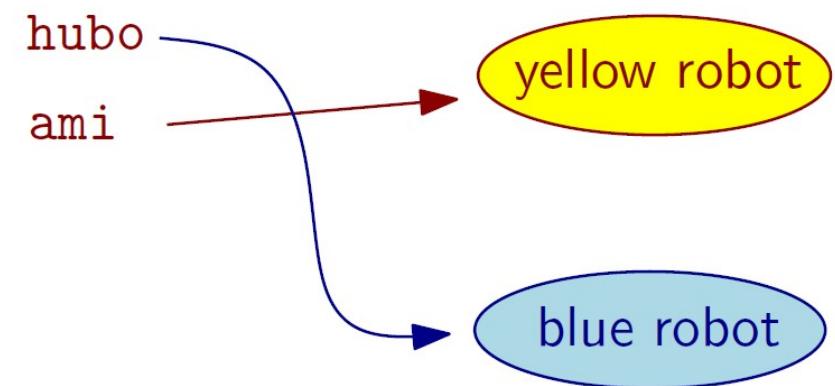
```
hubo.move( )
```

```
hubo = Robot( "blue" )
```

```
hubo.move( )
```

```
ami.turn_left( )
```

```
ami.move( )
```



정리 및 예습

본 강의 학습 목표:

- 객체 (objects)와 형태를(types) 이해할 수 있다.
- 변수 (variables)를 이해할 수 있다.

다음 강의 학습 목표:

- 연산자를 통한 식을 이해하고 작성할 수 있다.
- 기초 자료 구조인 튜플을 이해하고 작성할 수 있다.

CS101 - 연산자 및 튜플

Lecture 6

School of Computing
KAIST

학습 목표:

- 연산자를 통한 식을 이해하고 작성할 수 있다.
- 기초 자료 구조인 튜플을 이해하고 작성할 수 있다.

숫자의 연산에는 +, -, *, /, //, %, ** 연산자를 사용할 수 있습니다.

** 는 거듭제곱 연산입니다. ($a^{**} b = a^b$)

```
>>> 2**16
```

```
65536
```

% 는 나눗셈의 나머지 연산입니다.

```
>>> 7 % 3
```

```
1
```

// 는 정수 나눗셈(소수 부분을 제외한 나눗셈) 연산입니다.

```
>>> 13.0 // 4.0
```

```
3.0
```

```
>>> 9 / 7
```

```
1.2857142857142858
```

식은 객체, 변수, 연산자, 함수 호출의 조합으로 이루어집니다.

3.0 * (2 ** 15 - 12 / 4) + 4 ** 3

수학에서 사용하는 것과 같이, 연산자는 우선순위에 따라 적용됩니다.

- ① 거듭제곱 (**)
- ② 곱셈, 나눗셈 (*, /, //, %)
- ③ 덧셈, 뺄셈 (+, -)

연산자의 계산 순서가 헷갈릴 때는 괄호를 쓰세요.

예시) $\frac{a}{2\pi}$ 는 $a/2*pi$ 가 아닙니다.

$a/(2*pi)$ 나, $a/2/pi$ 와 같이 써야 합니다.

모든 연산자는 복소수에서도 사용할 수 있습니다.

- + 와 * 연산자는 문자열에서도 사용할 수 있습니다.

```
>>> "Hello" + "CS101"
```

```
'HelloCS101'
```

```
>>> "CS101" * 8
```

```
'CS101 CS101 CS101 CS101 CS101 CS101 CS101 CS101 '
```

논리식

논리식은 계산 결과가 논리값인 식입니다.

이 식은 **if** 와 **while** 문에서 사용됩니다.

다음 연산자들의 연산 결과는 논리값입니다.

`==, !=, >, <, <=, >=`

```
>>> 3 < 5
```

True

```
>>> 27 == 14 ← '=='은 두 값이 같은지 비교하는 연산자입니다.  
False 대입 연산자인 = 와 헷갈리지 마세요!
```

```
>>> 3.14 != 3.14
```

False

```
>>> 3.14 >= 3.14
```

True

```
>>> "Cheong" < "Choe"
```

True

```
>>> "3" == 3
```

False

논리 연산자

`not`, `and`, `or` 는 논리 연산자입니다.

```
( not True ) == False
```

```
( not False ) == True
```

```
( False and False ) == False
```

```
( False and True ) == False
```

```
( True and False ) == False
```

```
( True and True ) == True
```

```
( False or False ) == False
```

```
( False or True ) == True
```

```
( True or False ) == True
```

```
( True or True ) == True
```

주의: 만약 논리 연산자 왼쪽 값으로 연산의 결과가 결정된다면

(예. `False and ...` 또는 `True or ...`)

Python은 논리 연산자 오른쪽 값을 계산하지 않습니다.

튜플

튜플은 다른 객체들을 포함하는 객체입니다.

```
>>> position = (3.14, -5, 7.5)
>>> profs = ("In-Young Ko", "Sunghee Choi", "Lee
YoungHee", "Duksan Ryu", "Key-Sun Choi")
```

튜플은 **tuple** 형태를 가진 하나의 객체입니다.

```
>>> print(position, type(position))
(3.14, -5, 7.5) <class 'tuple'>
```

튜플이 포함하는 객체들은 풀 수 있습니다.

```
>>> x, y, z = position
>>> print(x)
3.14
```

튜플의 값을 한 번에 풀고, 다시 묶으려면

```
>>> a, b = ("aa", "bb")
>>> a, b = b, a
>>> print(b)
aa
```

정리 및 예습

본 강의 학습 목표:

- 연산자를 통한 식을 이해하고 작성할 수 있다.
- 기초 자료 구조인 튜플을 이해하고 작성할 수 있다.

다음 강의 학습 목표:

- 튜플을 활용하여 디지털 사진을 표현하는 방식을 이해할 수 있다.
- 디지털 사진을 색 반전이나 흑백 모드로 변환 할 수 있다.

CS101 - 튜플을 사용한 디지털 사진 변환 예제

Lecture 7

School of Computing
KAIST

학습 목표:

- 튜플을 활용하여 디지털 사진을 표현하는 방식을 이해할 수 있다.
- 디지털 사진을 색 반전이나 흑백 모드로 변환할 수 있다.

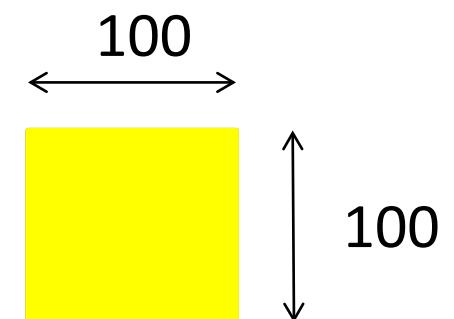
색

색은 3개의 값을 가진 튜플로 표현됩니다.

이 튜플이 가진 3개의 값들은 해당 색의 빨간색, 초록색, 파란색 세기/강도를 의미합니다.

```
red = ( 255, 0, 0 )
blue = ( 0, 0, 255 )
white = ( 255, 255, 255 )
black = ( 0, 0, 0 )
yellow = ( 255, 255, 0 )
purple = ( 128, 0, 128 )
```

```
from cs1media import *
img = create_picture(100, 100, purple)
img.show()
img.set_pixels(yellow)
img.show()
```

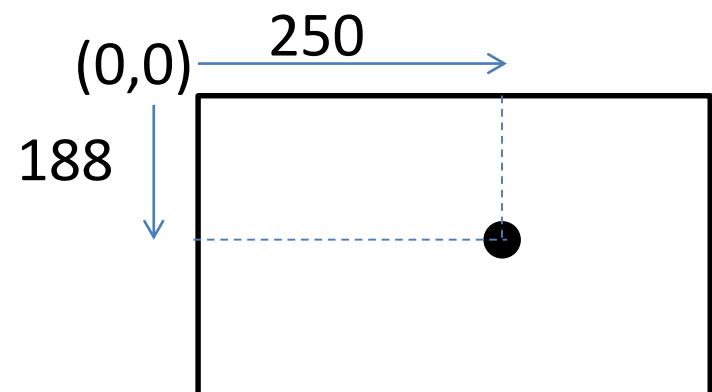


너비 w 픽셀, 높이 h 픽셀을 가진 디지털 이미지는
h개의 행과 w개의 열을 가진 직사각형 행렬로 표현됩니다.

0, 0	1, 0	2, 0	3, 0	4, 0
0, 1	1, 1	2, 1	3, 1	4, 1
0, 2	1, 2	2, 2	3, 2	4, 2

이미지의 각 픽셀은 행렬의 x와 y좌표를 이용해서 접근할 수 있습니다.
x좌표의 범위는 0부터 w-1까지, y좌표의 범위는 0부터 h-1까지입니다.

```
>>> img.get(250, 188)  
(101, 104, 51) ← 빨간색, 초록색, 파란색 값  
>>> img.set(250, 188, (255, 0, 0))
```



for 반복문

for 반복문은 변수에 정수 값을 대입합니다.

```
>>> for i in range(4):  
...     print(i)
```

```
0  
1  
2  
3
```

```
>>> for i in range(7):  
...     print ("*" * i)
```

```
*
```



```
**
```



```
***
```



```
****
```



```
*****
```



```
******
```

```
from cs1media import *
```

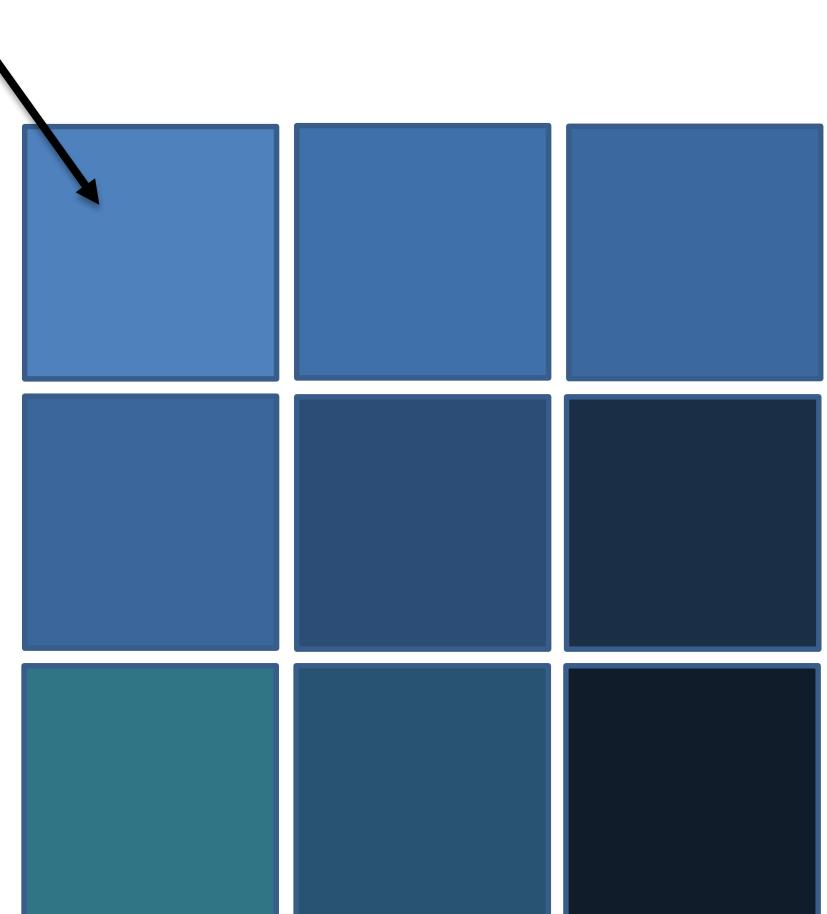
```
img = load_picture( "../photos/geowi.jpg" )
w, h = img.size()
for y in range(h):
    for x in range(w):
        r, g, b = img.get(x, y)
        r, g, b = 255 - r, 255 - g, 255 - b
        img.set(x, y, (r, g, b))
img.show()
```



```
from cs1media import *

img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```

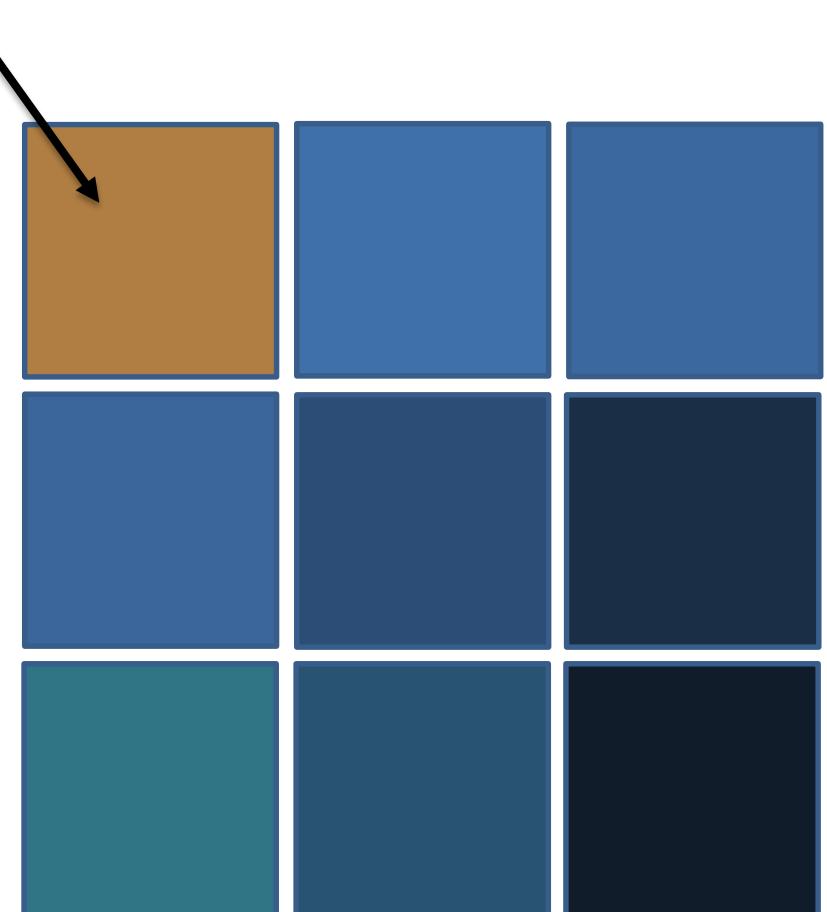
(x = 0, y = 0): 79, 129, 189



```
from cs1media import *

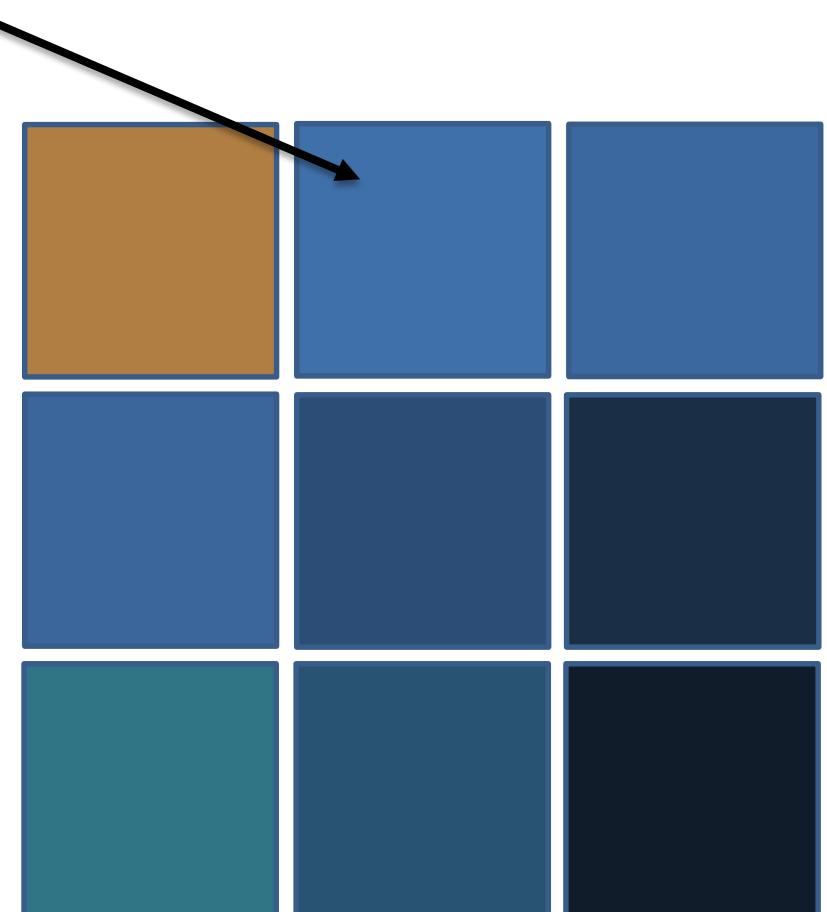
img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```

(x = 0, y = 0): 176, 126, 66



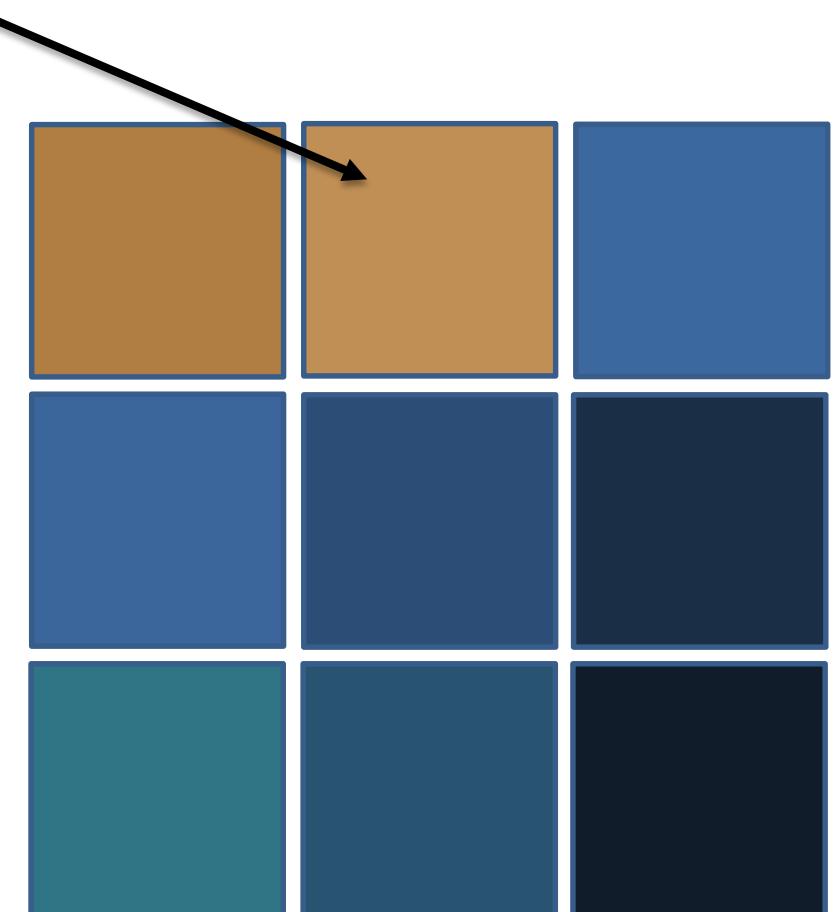
(x = 1, y = 0): 64, 112, 170

```
from cs1media import *
img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```



```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

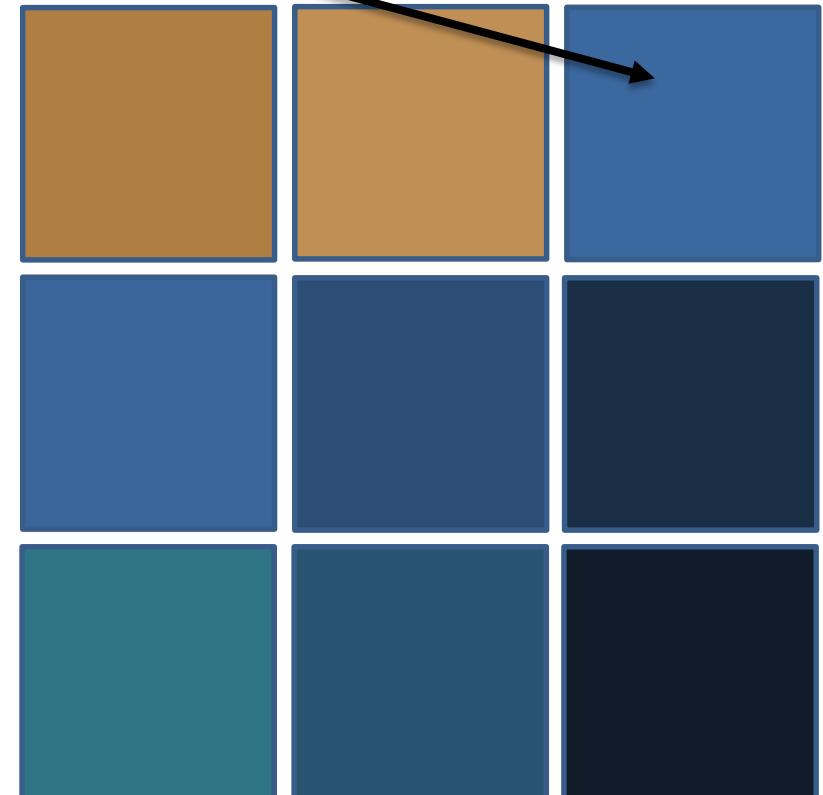
(x = 1, y = 0): 191, 143, 85



(x = 2, y = 0): 59, 104, 159

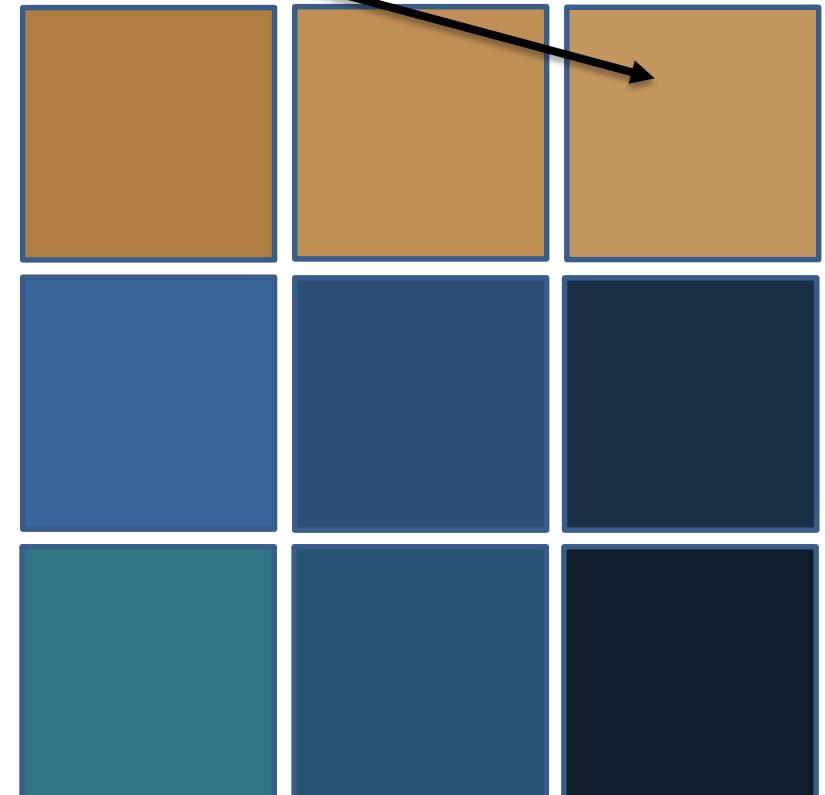
```
from cs1media import *

img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```



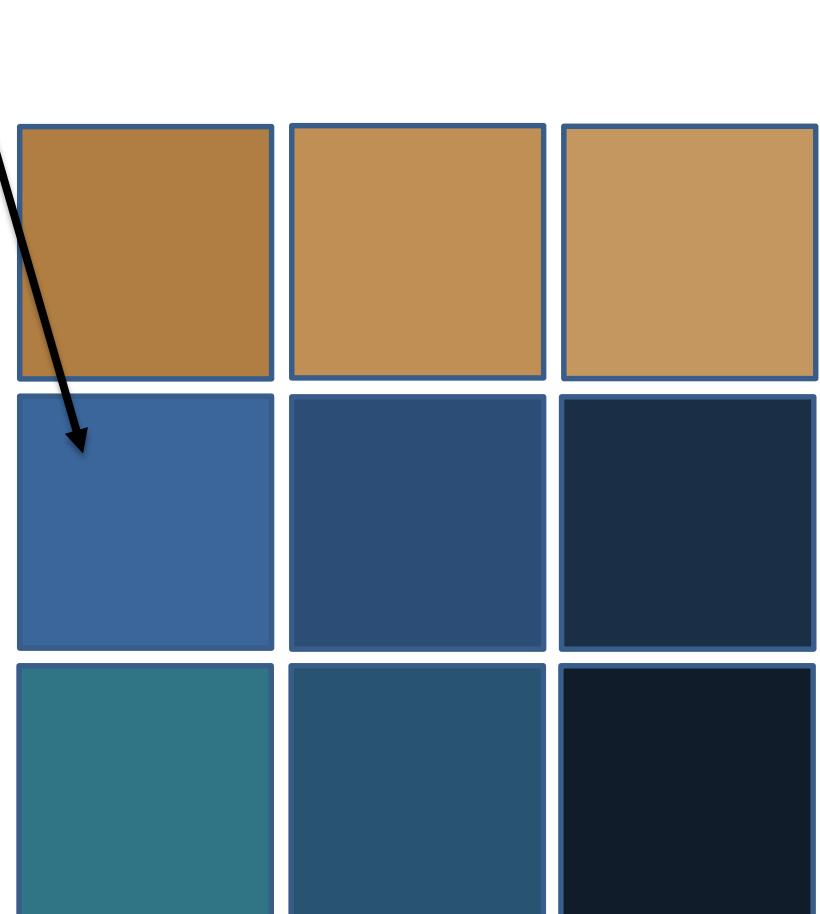
(x = 2, y = 0): 196, 151, 96

```
from cs1media import *
img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```



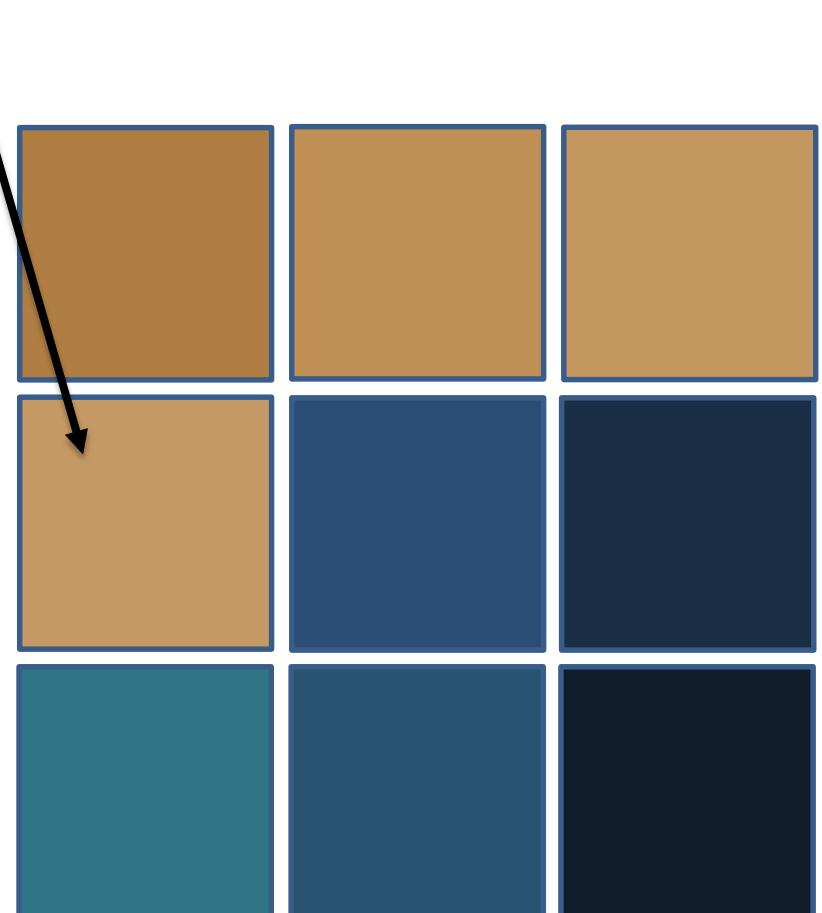
($x = 0, y = 1$): 58, 102, 156

```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```



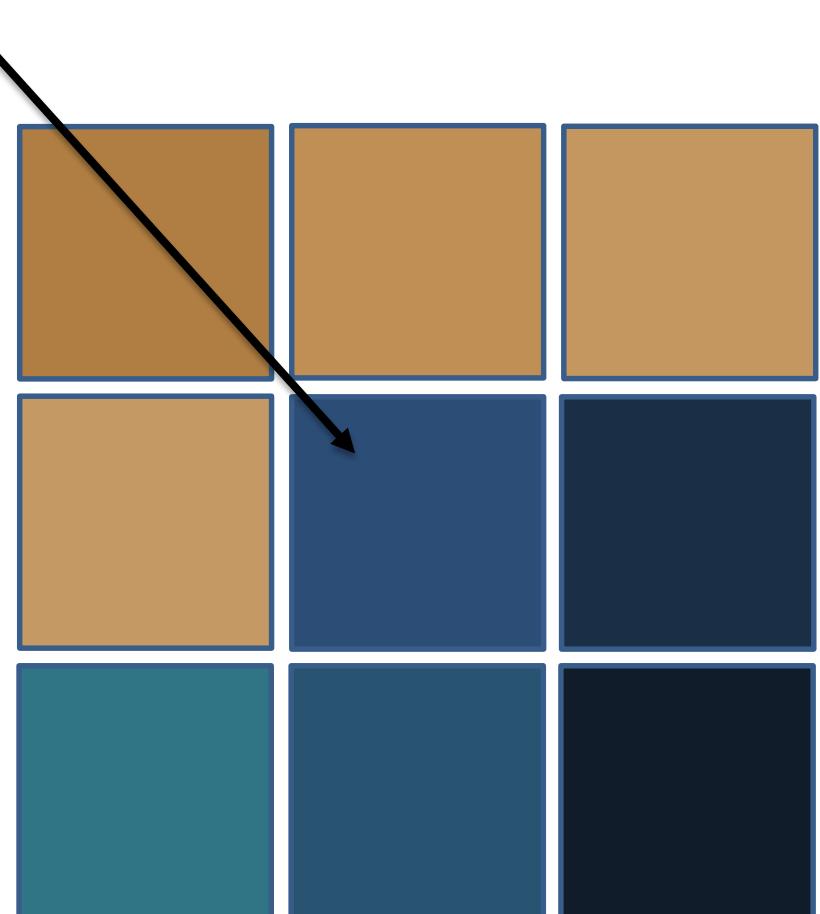
(x = 0, y = 1): 197, 153, 99

```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```



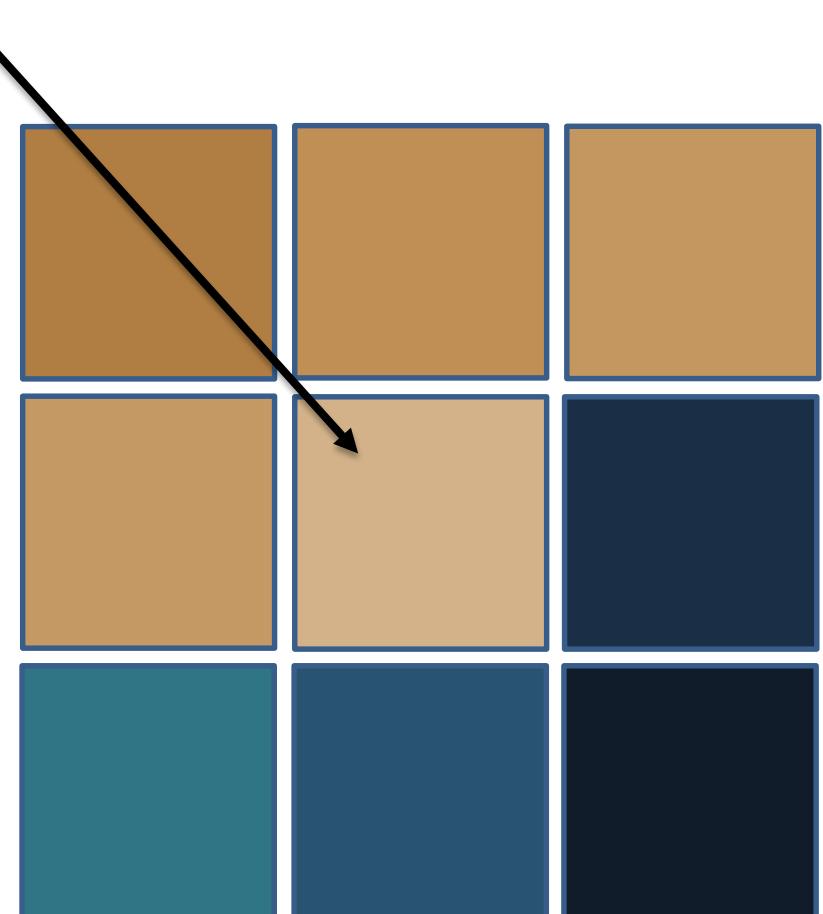
```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

(x = 1, y = 1): 44, 77, 118



```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

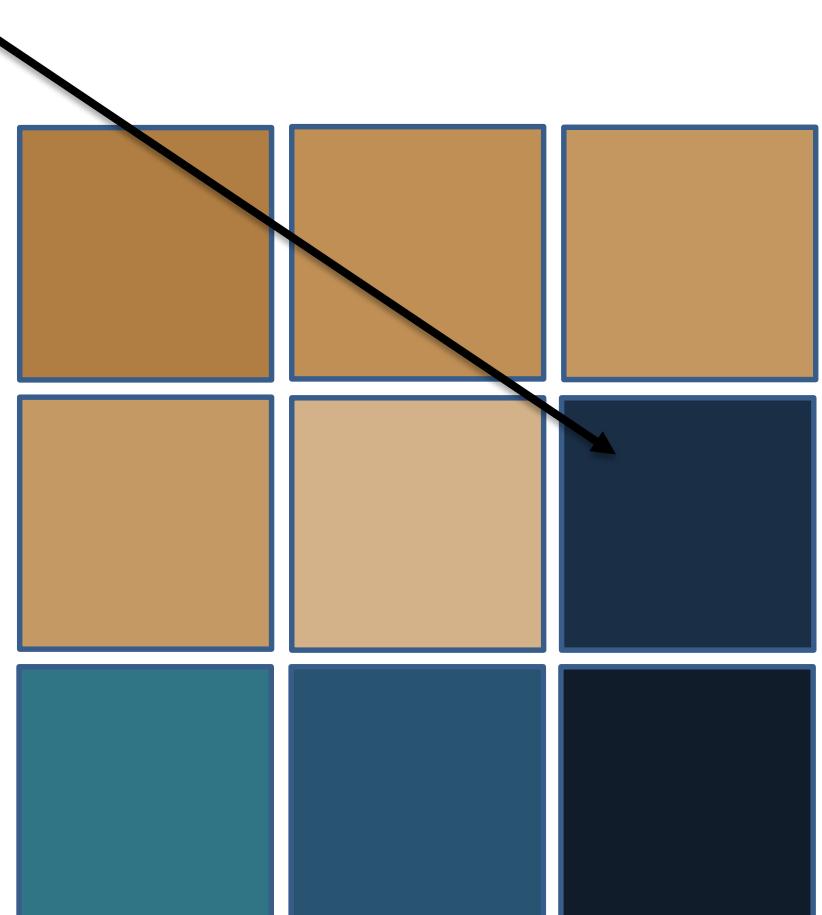
(x = 1, y = 1): 211, 178, 137



($x = 2, y = 1$): 26, 46, 70

```
from cs1media import *

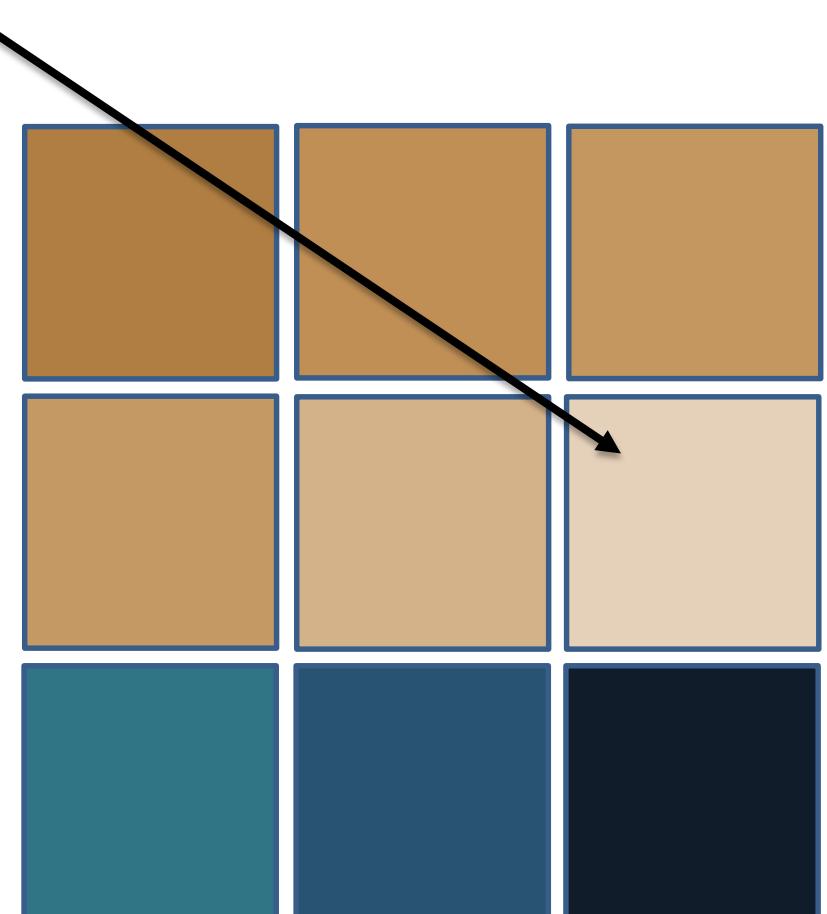
img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```



```
from cs1media import *

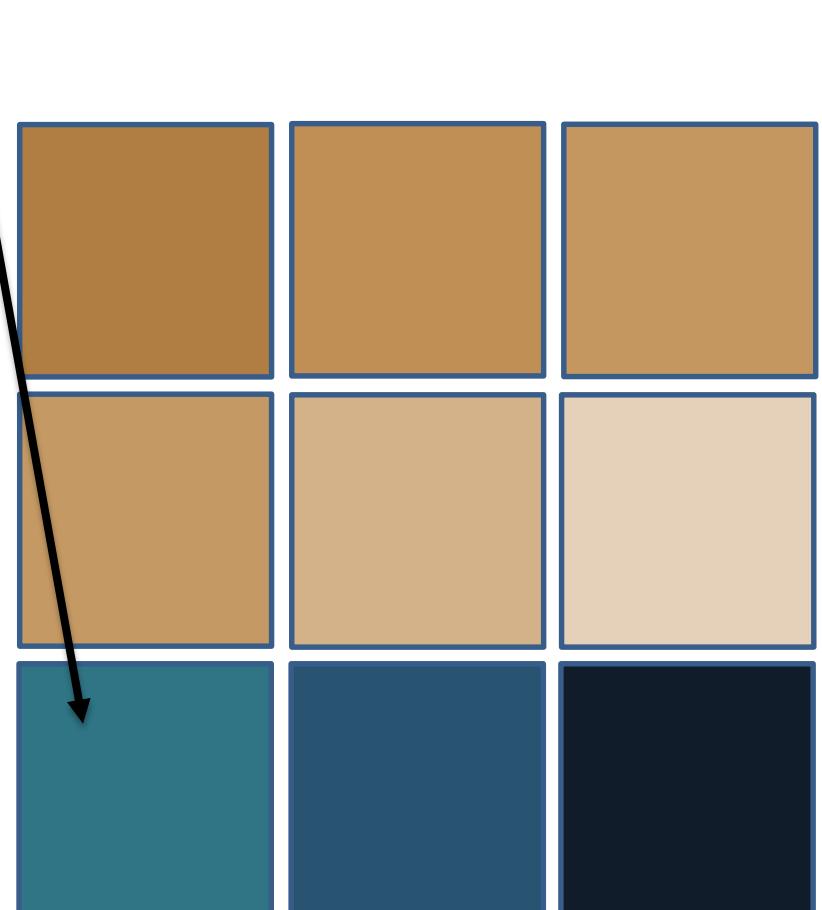
img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```

(x = 2, y = 1): 229, 209, 185



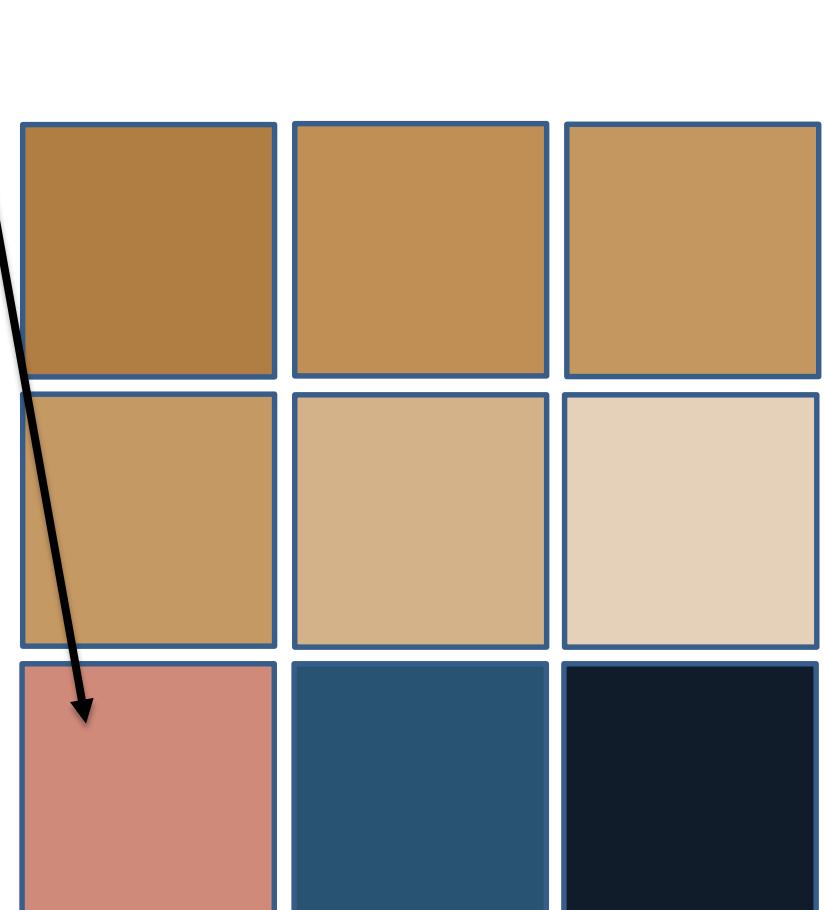
($x = 0, y = 2$): 47, 117, 133

```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```



(x = 0, y = 2): 208, 138, 122

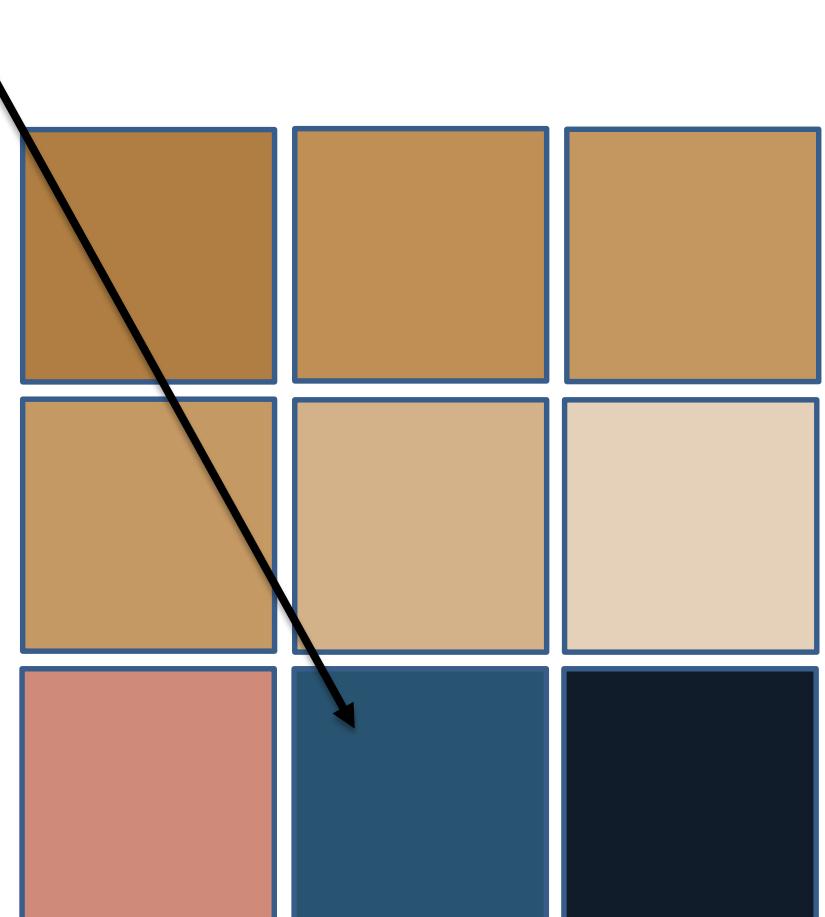
```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```



(x = 1, y = 2): 41, 84, 113

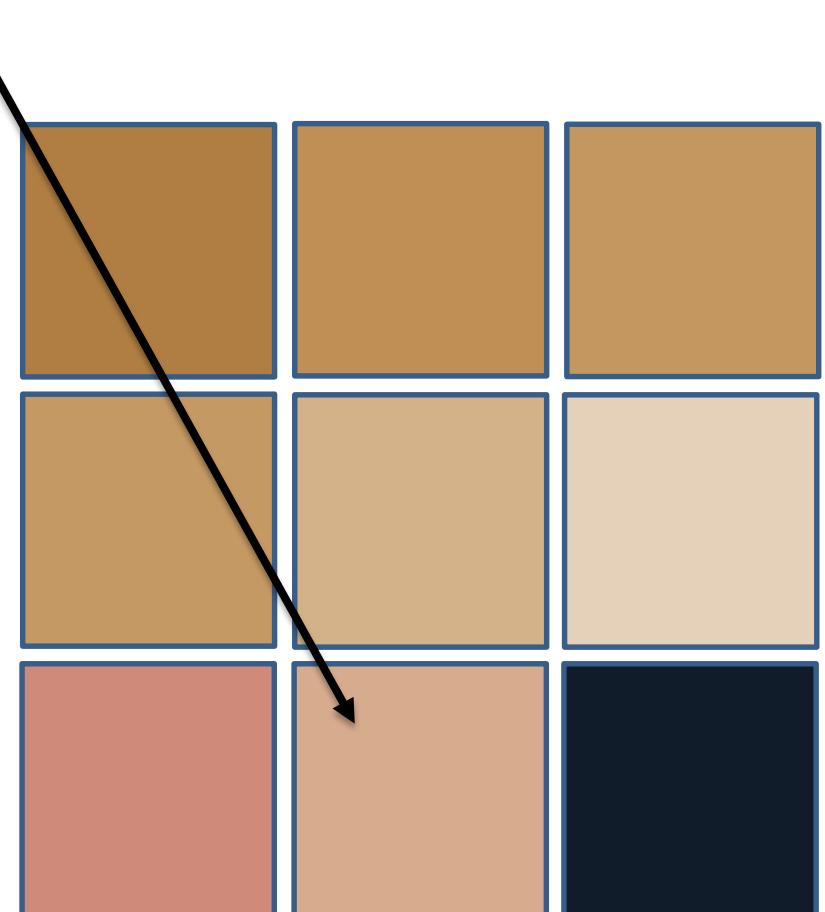
```
from cs1media import *

img = load_picture(...)
w, h = img.size()
for y in range(h):
    for x in range(w):
        r,g,b = img.get(x, y)
        r,g,b = 255-r,255-g,255-b
        img.set(x, y, (r, g, b))
img.show()
```



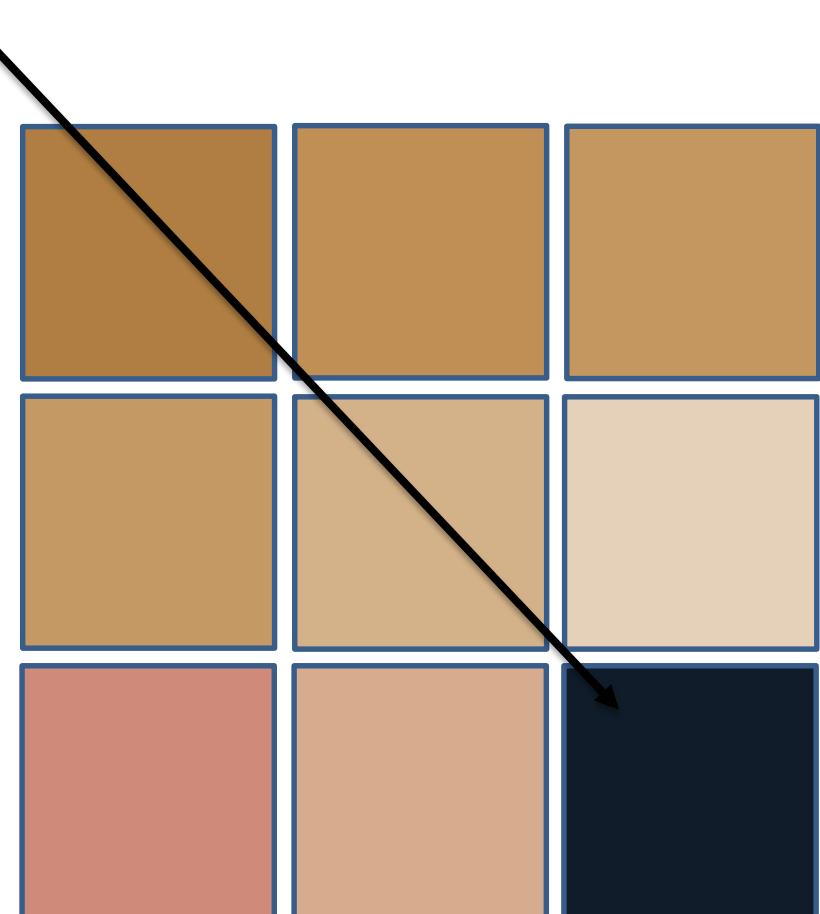
```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

(x = 1, y = 2): 214, 171, 142



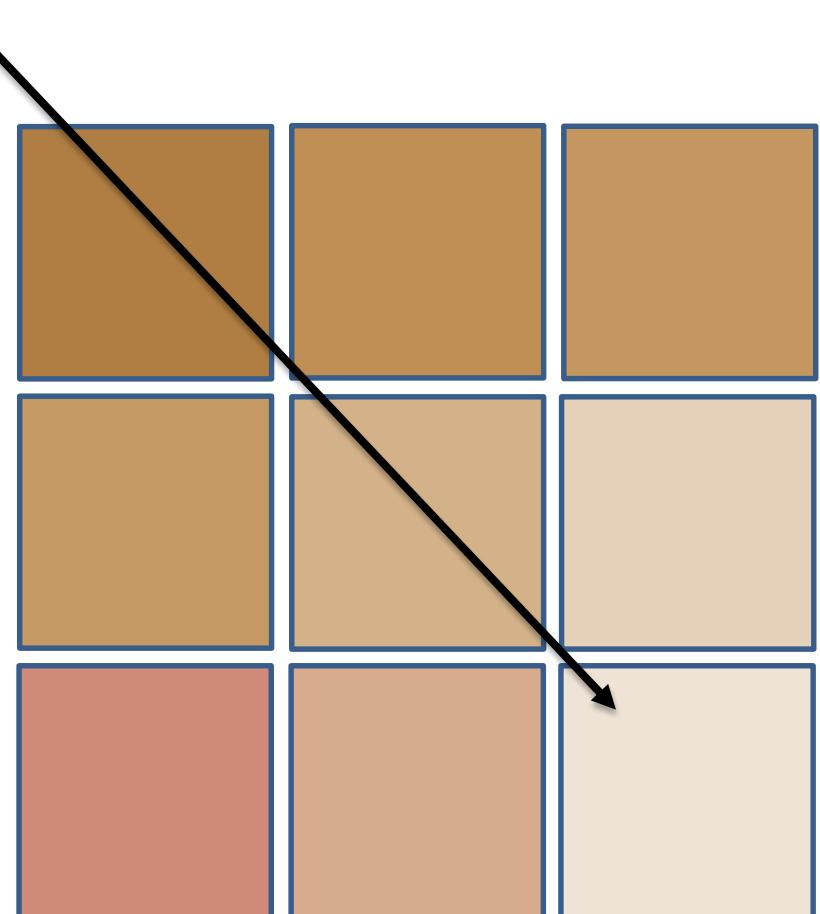
```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

(x = 2, y = 2): 16, 28, 42



```
from cs1media import *\n\nimg = load_picture(...)\nw, h = img.size()\nfor y in range(h):\n    for x in range(w):\n        r,g,b = img.get(x, y)\n        r,g,b = 255-r,255-g,255-b\n        img.set(x, y, (r, g, b))\nimg.show()
```

(x = 2, y = 2): 239, 227, 213



```
from cs1media import *

threshold = 100
white = ( 255 , 255 , 255 )
black = ( 0 , 0 , 0 )

img = load_picture( "../photos/yunal.jpg" )
w, h = img.size()
for y in range(h):
    for x in range(w):
        r, g, b = img.get(x, y)
        v = (r + g + b) // 3          # average of r,g,b
        if v > threshold:
            img.set(x, y, white)
        else:
            img.set(x, y, black)
img.show()
```

정리 및 예습

본 강의 학습 목표:

- 튜플을 활용하여 디지털 사진을 표현하는 방식을 이해할 수 있다.
- 디지털 사진을 색 반전이나 흑백 모드로 변환 할 수 있다.

다음 강의 학습 목표:

- 매개 변수와 반환값을 가진 함수를 이해하고 사용할 수 있다.

CS101 – 매개 변수와 반환값을 가진 함수

Lecture 8

School of Computing
KAIST

학습 목표:

- 매개 변수와 반환값을 가진 함수를 이해하고 사용할 수 있다.

함수(Function)라는 이름은 수학에서 비롯되었습니다.

수학에서의 함수의 정의는 한 집합의 임의의 한 원소를, 다른 집합의 오직 한 원소에 대응시키는 대응 관계입니다.

$$\begin{aligned}f : \mathbb{R} &\rightarrow \mathbb{R} \\x &\rightarrow \pi \times \frac{x}{180.0}\end{aligned}$$

이 예시에서 x 는 함수의 인자(Argument)이고, $f(x)$ 는 함수의 결과입니다.

Python에서의 함수 역시 인자를 전달받아 결과를 반환합니다.

```
def to_radians(deg):
    return (deg / 180.0) * math.pi

>>> a = to_radians(90)
>>> print(a)
1.5707963267948966
```

Python은 많은 내장 함수를 제공하고 있습니다.

형 변환 함수는 객체의 형태를 다른 형태로 바꿔주는 함수입니다.

```
>>> int( "32" )  
32  
>>> int( 17.3 )  
17  
>>> float( 17 )  
17.0  
>>> float( "3.1415" )  
3.1415  
>>> str( 17 ) + " " + str( 3.1415 )  
'17 3.1415'  
>>> complex( 17 )  
( 17 + 0j )
```

수학 함수를 사용하려면, Python에서 math 모듈을 사용하고 싶다고 선언해야 합니다.

```
import math  
degrees = 45  
radians = degrees / 360.0 * 2 * math.pi  
print(math.sin(radians))  
print(math.sqrt(2) / 2)
```

함수들을 자주 사용한다면 함수에 더 짧은 이름을 지어줄 수 있습니다.

```
import math  
sin = math.sin  
pi = math.pi  
radians = degrees / 360.0 * 2 * pi  
print(sin(radians))
```

매개 변수를 사용한 함수 정의

함수는 인자들을 가리키는 변수들을 이용하여 정의됩니다.
이 이름들은 **매개 변수 (Parameter)**라고 부릅니다.

```
def compute_interest(amount, rate, years):
```

함수 내부에서 매개 변수는 다른 변수와 동일하게 사용됩니다.

```
    value = amount * (1 + rate/100.0) ** years
```

함수의 결과 계산이 끝나면 결과값을 **반환 (Return)**해야 합니다.
함수는 함수가 반환되는 시점에 종료되며, 함수의 결과는 반환값을 통해 전달됩니다.

```
    return value
```

이제 우리는 같은 함수를 다른 인자 값들을 이용하여 부를 수 있습니다.

```
>>> s1 = compute_interest(200, 7, 1)
>>> s2 = compute_interest(500, 1, 20)
```

다음 함수는 절대값을 계산하는 함수입니다 (내장 함수 `abs`와 동일한 기능):

```
def absolute(x):  
    if x < 0:  
        return -x  
    else:  
        return x
```

같은 함수를 다음처럼 쓸 수도 있습니다.

```
def absolute(x):  
    if x < 0:  
        return -x  
    return x
```

하지만, 다음처럼 쓰면 안 됩니다.

```
def absolute(x):  
    if x < 0:  
        return -x  
    if x > 0 :  
        return x
```

조건을 검사해서 **True**나 **False**를 반환하는 함수는 **조건함수(Predicate)**라고 부릅니다.

```
# is integer a divisible by b?  
def is_divisible(a, b):  
    if a % b == 0:  
        return True  
    else:  
        return False
```

조건 함수는 **if** 문이나 **while** 문에서 바로 사용할 수 있습니다.

```
if is_divisible(x, y):  
    print('x is divisible by y')
```

위 함수는 다음처럼 간단하게 정의할 수도 있습니다.

```
def is_divisible(a, b):  
    return a % b == 0
```

지금까지 사용한 함수들 중에는 **반환문**이 없는 함수들도 많이 있습니다.

```
def turn_right():
    for i in range(3):
        hubo.turn_left()
```

반환문이 없는 함수는 자동으로 **None**을 반환합니다.

```
>>> s = turn_right()
>>> print(s)
```

None

함수가 호출되면, 호출될 때의 함수 인자 (argument) 들은 함수의 매개 변수 (parameter)로 대입됩니다.

```
def print_twice(text):  
    print(text)           ↑ 매개 변수  
    print(text)
```

함수 호출에 사용하는 인자의 수는 함수의 매개 변수의 수와 동일해야 합니다.

```
>>> print_twice("I love CS101")
```

```
I love CS101
```

```
I love CS101
```

```
>>> print_twice(math.pi)
```

```
3.14159265359
```

```
3.14159265359
```

여러 값 반환하기

함수는 하나의 값만 반환할 수 있습니다.

하지만 함수는 튜플을 반환할 수도 있습니다.

Python은 함수가 여러 값을 반환할 때, 자동으로 이 값을 튜플로 만들어 반환합니다.

```
def student():
    name = "Hong, Gildong"
    id = 20101234
    return name, id
```

튜플로 된 반환값은 다음처럼 바로 풀어낼 수 있습니다.

```
>>> name, id = student()
```

```
>>> print(name)
```

```
"Hong, Gildong"
```

```
>>> print(id)
```

```
20101234
```

`input` 함수는 메시지를 출력하고, 키보드를 통한 사용자의 문자열 입력을 기다립니다. 사용자가 Enter 키를 입력하면, 사용자가 입력한 전체 문자열이 반환됩니다.

```
name = input("What is your name? ")  
print("Welcome to CS101, " + name)
```

숫자 입력이 필요하다면, 문자열을 숫자로 변환해야 합니다.

```
raw_n = input("Enter a positive integer> ")  
n = int(raw_n)  
for i in range(n):  
    print("*" * i)
```

정리 및 예습

본 강의 학습 목표:

- 매개 변수와 반환값을 가진 함수를 이해하고 사용할 수 있다.

다음 강의 학습 목표:

- 함수를 사용하여 휴보 로봇이 비퍼들을 줍는 프로그램을 모듈러하게 작성할 수 있다.
- 함수를 사용하여 색의 밝기 측정 및 디지털 사진을 흑백 모드로 변환할 수 있다.

CS101 - 함수를 사용한 로봇 조종 및 디지털 사진 변환 프로그램

Lecture 9

School of Computing
KAIST

학습 목표:

- 함수를 사용하여 휴보 로봇이 비퍼들을 줍는 프로그램을 모듈러하게 작성할 수 있다.
- 함수를 사용하여 색의 밝기 측정 및 디지털 사진을 흑백 모드로 변환할 수 있다.

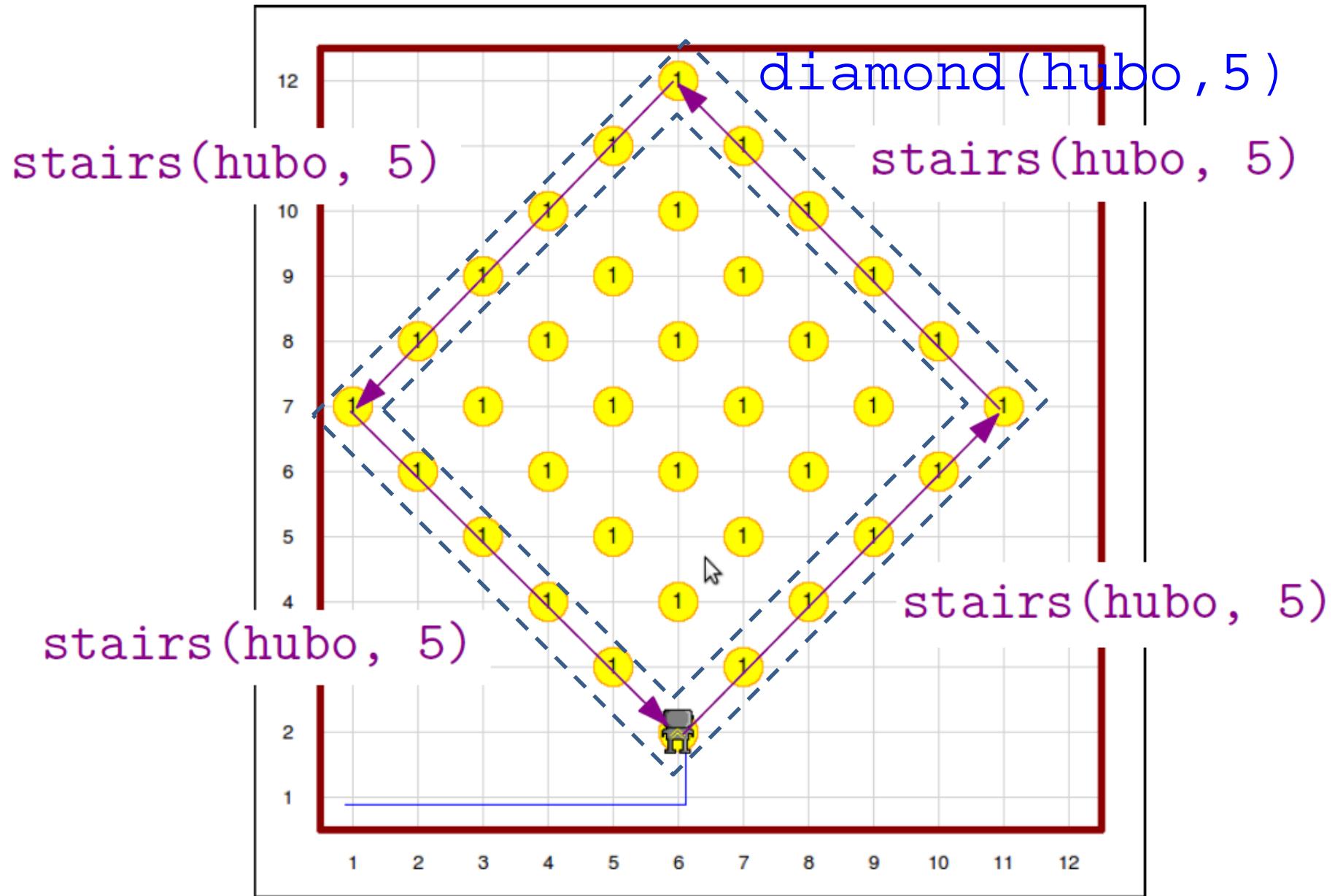
이제 우리는 휴보만 사용할 수 있었던 `turn_right` 함수를 모든 로봇이 사용할 수 있게 만들 수 있습니다.

```
def turn_right(robot):
    for i in range(3):
        robot.turn_left()

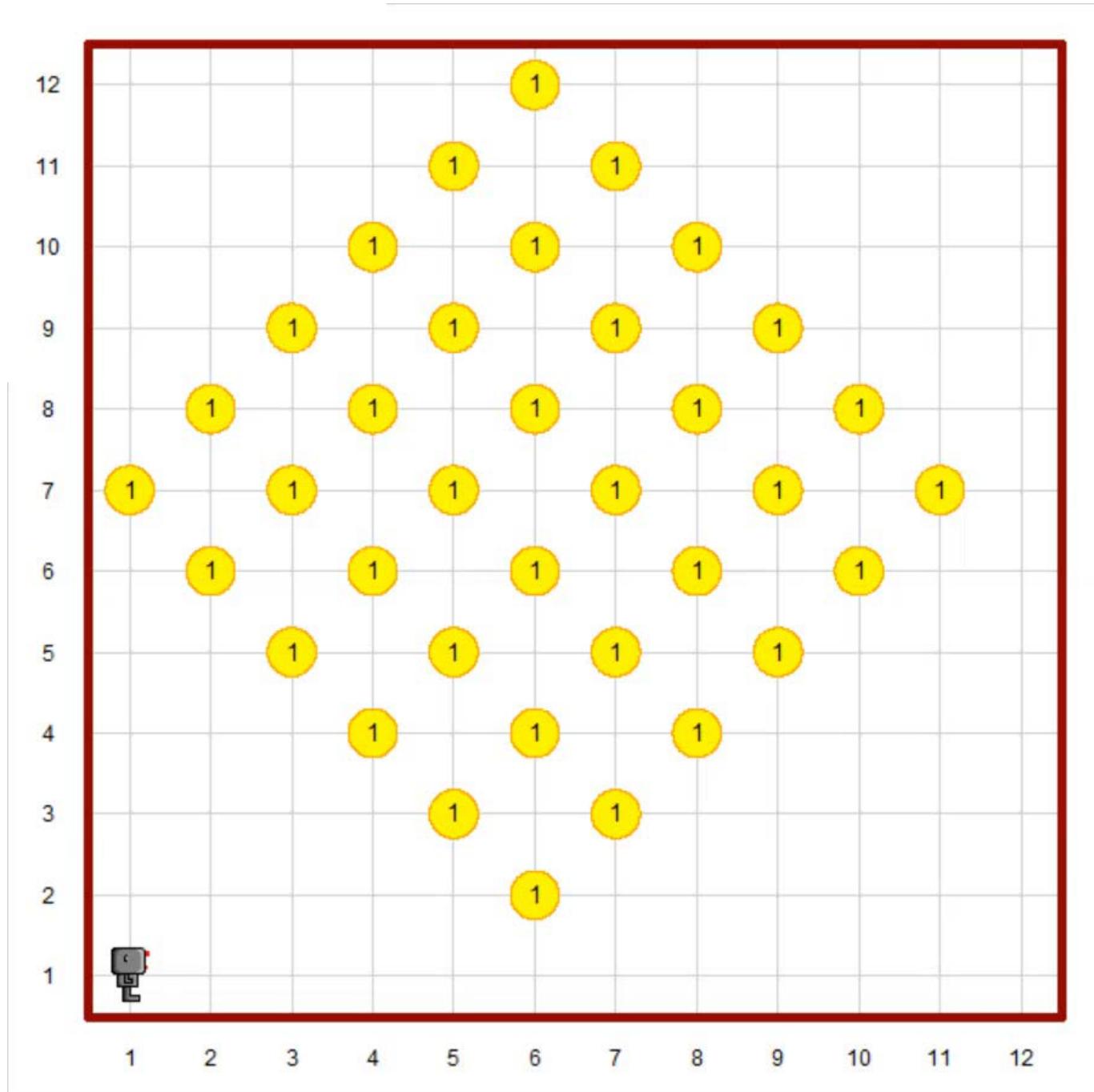
ami = Robot("yellow")
hubo = Robot("blue")
turn_right(ami)
turn_right(hubo)
```

주의: 매개 변수는 함수 내부에서만 사용할 수 있습니다.

harvest_all(hubo)



함수를 이용한 비퍼 줍기 동영상



```
def stairs(robot, n):
    for i in range(n):
        robot.pick_beeper()
        robot.move()
        turn_right(robot)
        robot.move()
        robot.turn_left()
```

```
def diamond(robot, n):
    for i in range(4):
        stairs(robot, n)
        robot.turn_left()
```

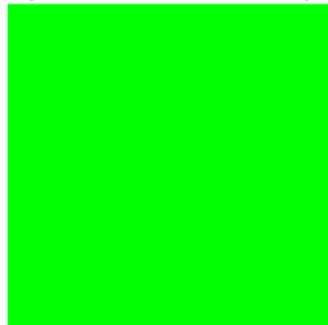
```
def harvest_all(robot):
    for i in range(3):
        n = 5 - 2 * i
        diamond(robot, n)
        robot.move()
        robot.move()
```

(r, g, b)로 표현된 색의 밝기(luma)는 어떻게 계산해야 할까요?

(255, 0, 0)



(0, 255, 0)



(0, 0, 255)



색의 밝기를 계산하는 함수는 다음과 같습니다.

```
def luma(p):
    r, g, b = p
    return int(0.213 * r + 0.715 * g + 0.072 * b)
```

```
white = ( 255 , 255 , 255 )
```

```
black = ( 0 , 0 , 0 )
```

```
def blackwhite(img, threshold):  
    w, h = img.size()  
    for y in range(h):  
        for x in range(w):  
            v = luma(img.get(x, y))  
            if v > threshold:  
                img.set(x, y, white)  
            else:  
                img.set(x, y, black)
```

```
pict = load_picture("../photos/yunal.jpg")  
blackwhite(pict, 100)  
pict.show()
```

정리 및 예습

본 강의 학습 목표:

- 함수를 사용하여 휴보 로봇이 비퍼들을 줍는 프로그램을 모듈러하게 작성할 수 있다.
- 함수를 사용하여 색의 밝기 측정 및 디지털 사진을 흑백 모드로 변환할 수 있다.

다음 강의 학습 목표:

- 함수에서 사용하는 인자와 매개 변수의 다양한 사용방법을 이해할 수 있다.

CS101 - 함수 인자와 매개 변수

Lecture 10

School of Computing
KAIST

학습 목표:

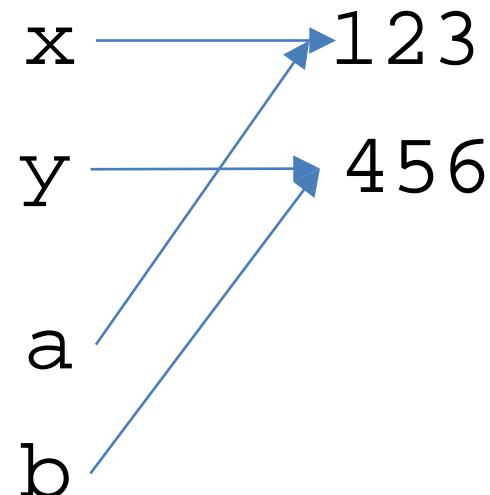
- 함수에서 사용하는 인자와 매개 변수의 다양한 사용방법을 이해할 수 있다.

매개 변수의 대상

다음 코드의 실행 결과는 무엇일까요?

```
def swap(a, b):  
    a, b = b, a
```

```
x, y = 123, 456  
swap(x, y)  
print(x, y)
```

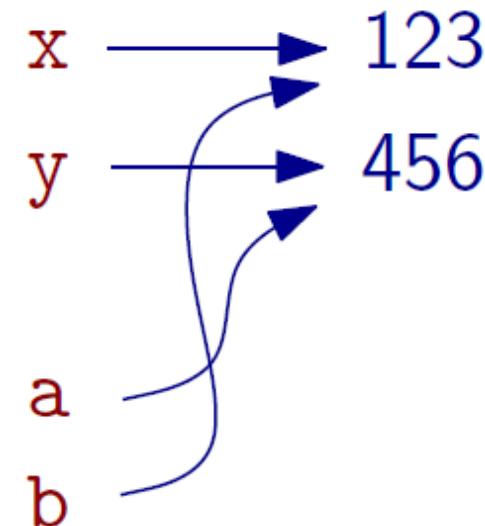


`a`는 변수 `x`를 가리키는 이름이 아니라, 객체 123의 새로운 이름입니다.

다음 코드의 실행 결과는 무엇일까요?

```
def swap(a, b):  
    a, b = b, a
```

```
x, y = 123, 456  
swap(x, y) ←  
print(x, y)
```



`a`는 변수 `x`를 가리키는 이름이 아니라, 객체 123의 새로운 이름입니다.

함수 인자

우리는 함수의 인자와 매개 변수에 대해 배웠습니다.

```
def create_sun(radius, color):  
    sun = Circle(radius)  
    sun.setFillColor(color)  
    sun.setBorderColor(color)  
    sun.moveTo(100, 100)  
    return sun
```

```
sun = create_sun(30, "yellow")
```

함수의 인자는 함수의 매개 변수에 하나씩, 왼쪽부터 차례대로 대입됩니다.

함수의 매개 변수에는 기본값을 설정할 수 있습니다.

```
def create_sun(radius = 30, color = "yellow"):  
    sun = Circle(radius)  
    sun.setFillColor(color)  
    sun.setBorderColor(color)  
    sun.moveTo(100, 100)  
    return sun
```

이제 우리는 이렇게 함수를 부를 수 있습니다.

```
moon = create_sun(28, "silver")  
star = create_sun(2) # create_sun(2, "yellow")과 동일  
sun = create_sun() # create_sun(30, "yellow")과 동일
```

```
def avg(data, start = 0, end = None):
    if not end: # (not None)==True, (not 4)==False
        end = len(data)
    return sum(data[start:end]) / float(end-start)

>>> d = (3, 4, 5, 6, 7)
>>> avg(d, 1, 4)
5.0
>>> avg(d, 2) # avg(d,2,None) 또는 avg(d,2,5) 와 동일
6.0
>>> avg(d)      # avg(d,0,None) 또는 avg(d,0,5) 와 동일
5.0
```

참고 :기본값을 가지는 매개 변수는 다른 매개 변수의 뒤에 와야 합니다.

```
def f1(x, y=0):    # 옳은 함수 정의
    return x+y
```

```
Def f2(x=0, y):   # 틀린 함수 정의
    return x+y
```

이름이 있는 인자

함수를 부를 때 인자에 이름을 붙여서 부를 수 있습니다.

인자에 이름을 붙여 주면 함수를 부르는 코드를 좀 더 명확하게 할 수 있고,
함수를 부를 때 인자의 순서를 고려하지 않아도 됩니다.

```
moon = create_sun(color = "silver")
moon = create_sun(color = "silver", radius = 28)

>>> avg(d, end=3)
5.0
>>> avg(data=d, end=3)
5.0
>>> avg(end=3, data=d)
5.0
>>> avg(end=3, d)
SyntaxError: non-keyword arg after keyword arg
```

정리 및 예습

본 강의 학습 목표:

- 함수에서 사용하는 인자와 매개 변수의 다양한 사용방법을 이해할 수 있다.

다음 강의 학습 목표:

- 함수에서 사용하는 지역 변수와 전역변수의 차이를 이해할 수 있다.
- 지역 변수와 전역 변수의 장단점을 이해하고 활용할 수 있다

CS101 - 함수가 사용하는 지역 변수와 전역 변수

Lecture 11

School of Computing
KAIST

학습 목표:

- 함수에서 사용하는 지역 변수와 전역 변수의 차이를 이해할 수 있다.
- 지역 변수와 전역 변수의 장단점을 이해하고 활용할 수 있다

$ax^2 + bx + c$ 식을 계산하기 위한 함수는 다음과 같이 만들 수 있습니다.

```
def quadratic(a, b, c, x):  
    quad_term = a * x ** 2  
    lin_term = b * x  
    return quad_term + lin_term + c
```

quad_term과 lin_term 변수는 quadratic 함수 안에서만 사용할 수 있습니다.
이러한 변수들은 지역 변수 (local variable)라고 부릅니다.

함수의 매개 변수 또한 지역 변수입니다.

함수가 호출될 때, 함수 호출에 사용된 인자들이 매개 변수로 대입됩니다.

```
def quadratic(a, b, c, x):  
    quad_term = a * x ** 2  
    lin_term = b * x  
    return quad_term + lin_term + c  
  
result = quadratic(2, 4, 5, 3)
```

지역 변수는 함수 안에서만 사용할 수 있는 변수입니다.

a → 2

b → 4

c → 5

x → 3

quad _term → 18

lin_term → 12

지역 변수를 사용하는 이유: 모듈화



사람은 한 번에 너무 많은 내용들을 기억하기 힘듭니다.

큰 소프트웨어가 있을 때, 소프트웨어의 세세한 부분들이 어떻게 만들어졌는지 몰라도 각 부분들이 하는 역할만 알고 있다면 우리는 이 소프트웨어를 이해할 수 있습니다.

`quadratic` 함수를 사용하기 위해서는 다음 내용만 기억하면 됩니다.

```
def quadratic(a, b, c, x):  
    # implemented somehow
```

모듈화란 소프트웨어를 여러 부분으로 나눠서 개발하는 것을 의미합니다. 소프트웨어의 한 기능을 사용하기 위해서는, 그 기능의 역할만 알면 되고 그 기능이 어떻게 구현되어 있는지는 몰라도 됩니다.

`cs1robots`는 `Robot` 타입의 객체를 구현하는 모듈입니다.

로봇의 각 기능이 어떻게 구현되었는지는 모르더라도 로봇은 움직일 수 있습니다.

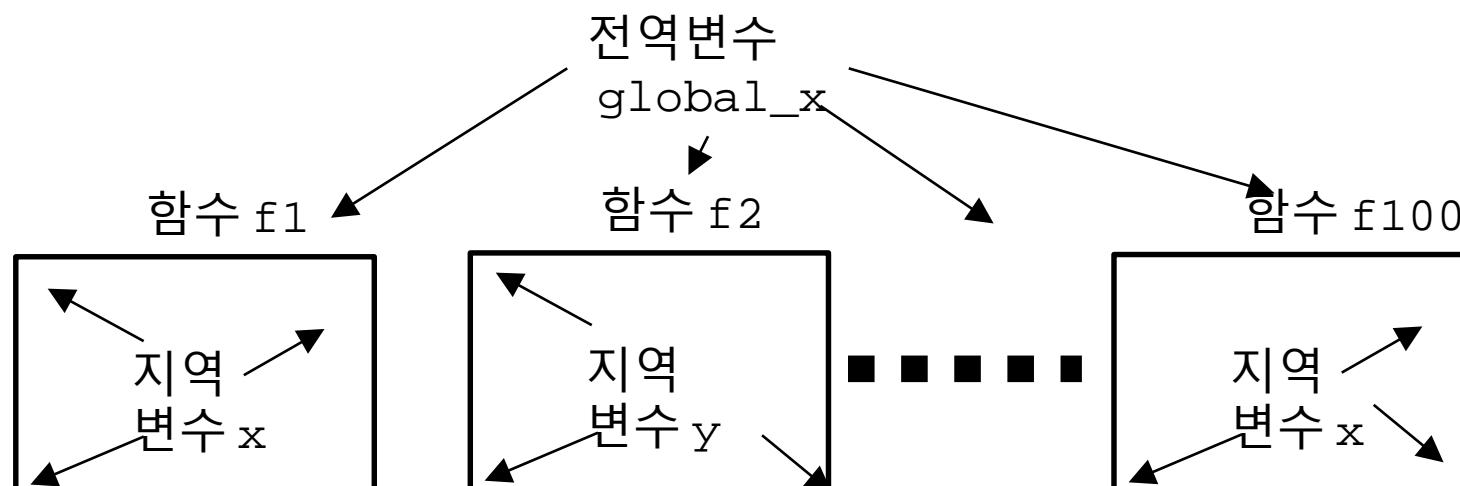
함수 밖에서 정의된 변수들은 전역 변수 (global variable)라 부릅니다.

전역 변수는 함수 안에서도 읽거나 쓸 수 있습니다.

```
hubo = Robot( ) ← 전역 변수 hubo
```

```
def turn_right( ):  
    for i in range(3):  
        hubo.turn_left() ← 전역 변수 사용
```

전역 변수는 어느 함수에서나 사용할 수 있기 때문에 실수를 일으키기 쉽습니다.
이 때문에, 큰 프로그램일수록 전역 변수를 조심해서 사용해야 합니다.



함수 안에서 값을 읽기만 하는 변수는 전역 변수입니다.

```
def f1( ) :  
    return 3 * a + 5
```

함수 안에서 값을 쓰는 (write) 변수는 지역 변수입니다.

```
def f2( x ) :  
    a = 3 * x + 17  
    return a * 3 + 5 * a
```

다음 test 함수의 결과는 무엇일까요?

```
a = 17  
def test( ) :  
    print(a)  
    a = 13  
    print(a)  
test()
```

오류 발생!

변수 a는 test 함수 안에서 값을 썼기 (write) 때문에 지역 변수지만, 첫 번째 print문이 불린 시점에서 지역 변수 값이 정의되지 않았기 때문에 오류가 발생합니다.

전역 변수의 값 쓰기

함수 안에서 전역 변수의 값을 바꿔야 할 때가 있습니다.

```
hubo = Robot()
hubo_direction = 0
```

```
def turn_left():
    hubo.turn_left()
    global hubo_direction
    hubo_direction += 90
```

```
def turn_right():
    for i in range(3):
        hubo.turn_left()
    global hubo_direction
    hubo_direction -= 90
```

```
a = "Letter a"
```

```
def f(a):  
    print("A = ", a)
```

```
def g():  
    a = 7  
    f(a + 1)  
    print("A = ", a)
```

```
print("A = ", a)  
f(3.14)  
print("A = ", a)  
g()  
print("A = ", a)
```

정리 및 예습

본 강의 학습 목표:

- 함수에서 사용하는 지역 변수와 전역변수의 차이를 이해할 수 있다.
- 지역 변수와 전역 변수의 장단점을 이해하고 활용할 수 있다

다음 강의 학습 목표:

- 다양한 기능을 불러서 사용하기 위한 모듈 기능을 이해할 수 있다.
- 다양한 그래픽 객체를 생성하고 변형시킬 수 있다.

CS101 - 모듈과 그래픽 객체들

Lecture 12

School of Computing
KAIST

학습 목표:

- 다양한 기능을 불러서 사용하기 위한 모듈 기능을 이해할 수 있다.
- 다양한 그래픽 객체를 생성하고 변형 시킬 수 있다.

Python의 모듈은 여러 함수들을 묶어서 파일로 만든 것입니다.
Python은 기본적으로 수많은 유용한 모듈을 제공하고 있습니다.
또한, 새로운 모듈을 직접 만들 수도 있습니다.

- math 모듈은 수학적 함수들을 제공합니다.
- random 모듈은 난수, 무작위 섞기 등의 함수들을 제공합니다.
- sys와 os 모듈은 운영체제와 관련된 함수들을 제공합니다.
- urllib 모듈은 웹에서 파일을 다운받는 함수들을 제공합니다.
- cs1robots 모듈은 휴보에 관련된 함수들을 제공합니다.
- cs1graphics 모듈은 그래픽과 관련된 함수들을 제공합니다.
- cs1media 모듈은 사진의 처리와 관련된 함수들을 제공합니다.

모듈의 정보는 help 함수를 이용해서 볼 수 있습니다.

```
>>> help("cs1media")
>>> help("cs1media.picture_tool")
```

모듈을 사용하려면 **import**를 이용해 모듈을 들여와야 합니다.

```
import math  
print(math.sin(math.pi))
```

다음처럼 모듈의 함수를 모듈 이름을 붙이지 않고 사용할 수 있습니다.

```
from math import *  
print(sin(pi))      # OK  
print(math.pi)     # NameError: name 'math'
```

필요한 함수들만 모듈에서 들여오는 것도 가능합니다.

```
from math import sin  
print(sin(3.14))      # OK  
print(pi)              # NameError: name 'pi'  
print(cos(3.14))      # NameError: name 'cos'  
print(math.cos(3.14))  # NameError: name 'math'
```

다음 코드는

```
from cs1robots import *
create_world()
hubo = Robot()
hubo.move()
hubo.turn_left()
```

다음 코드와 동일한 의미를 가집니다.

```
import cs1robots
cs1robots.create_world()
hubo = cs1robots.Robot()
hubo.move()
hubo.turn_left()
```

일반적으로, **import ***는 사용하지 않는 것이 좋습니다.

그림을 그리기 위해서는, 먼저 그림을 그릴 캔버스를 만들어야 합니다.

```
from cs1graphics import *
```

```
canvas = Canvas(400, 300)
canvas.setBackgroundColor("light blue")
canvas.setTitle("CS101 Drawing exercise")
```

캔버스의 좌표계는

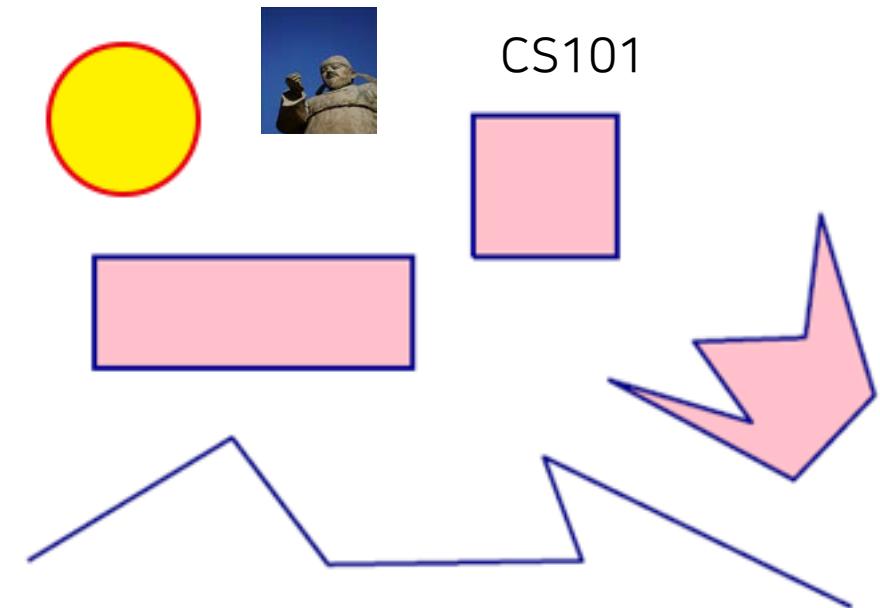
x좌표는 0부터 399까지 오른쪽으로,

y좌표는 0부터 299까지 위에서 아래로 증가합니다.



그래픽 객체를 캔버스에 추가하는 방식으로 캔버스에 그림을 그릴 수 있습니다.

- 원: Circle(radius)
- 정사각형: Square(side)
- 직사각형: Rectangle(width, height)
- 다각형: Polygon
- 선: Path
- 글자: Text(message, font_size)
- 이미지: (image_filename)



객체의 외곽선 색을 바꾸거나 알아보려면 (color는 문자열 또는 (r,g,b) 튜플)

```
obj.setBorderColor(color)
```

```
obj.getBorderColor()
```

객체 내부의 색을 바꾸거나 알아보려면 (color는 문자열 또는 (r,g,b) 튜플)

: 원, 정사각형, 직사각형, 다각형 객체에만 사용할 수 있습니다.

```
obj.setFillColor(color)
```

```
obj.getFillColor()
```

모든 그래픽 객체는 기준점을 가지고 있습니다.

캔버스상의 객체의 기준점 위치는 `move(dx, dy)`나 `moveTo(x, y)` 함수를 이용해 바꿀 수 있습니다.

```
sq = Square(100)
canvas.add(sq)
sq.setFillColor("blue")
sq.setBorderColor("red")
sq.setBorderWidth(5)
sq.moveTo(200, 200)
```

절대 좌표

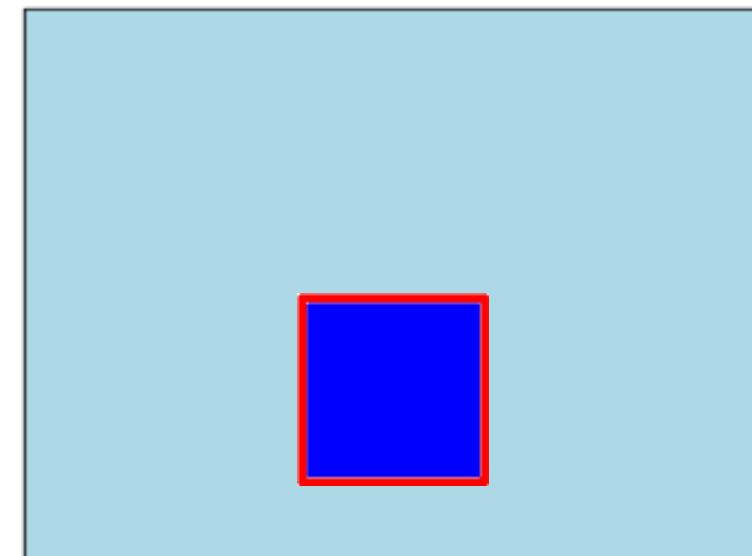
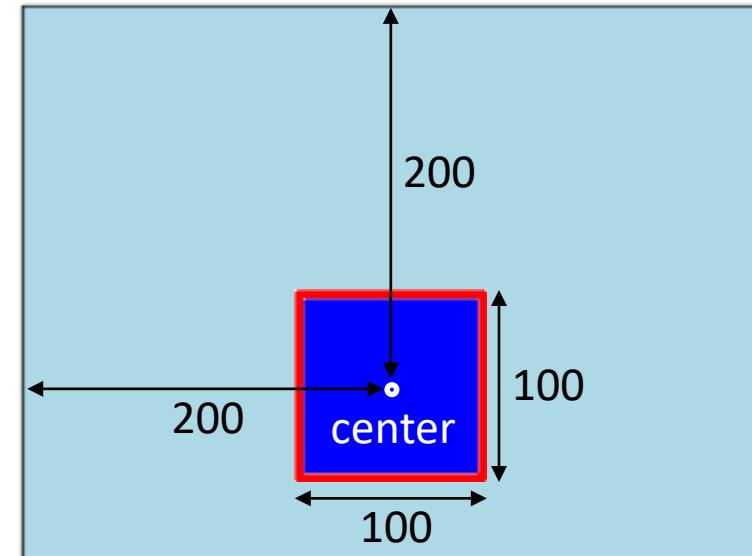
다음과 같이 애니메이션을 만들 수도 있습니다.

```
for i in range(100):
    sq.move(1, 0)
```

상대 좌표

```
    time.sleep(0.1)
```

눈에 보이도록
애니메이션 속도 조절
(`import time` 필요)



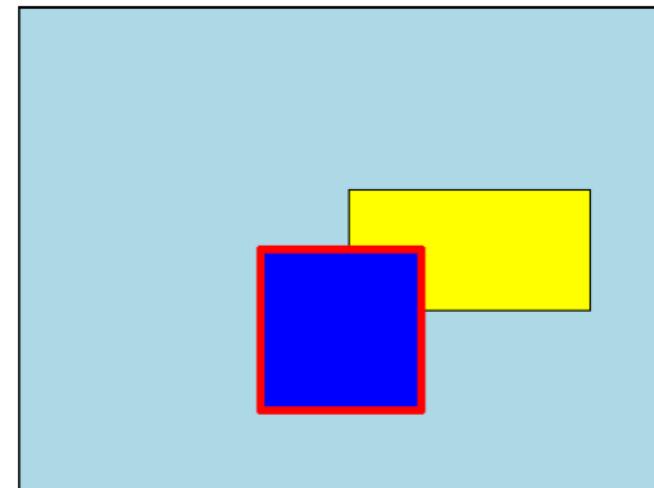
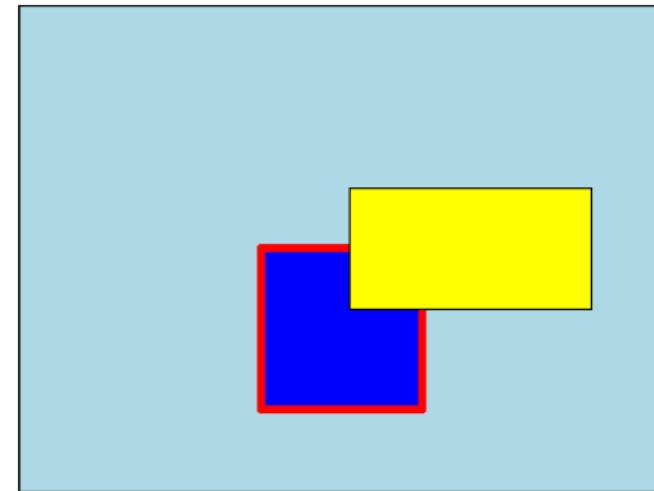
그래픽 객체의 깊이 순서

```
r = Rectangle(150, 75)  
canvas.add(r)  
r.setFillColor("yellow")  
r.moveTo(280, 150)
```

깊이를 변경하려면

```
sq.setDepth(10)  
r.setDepth(20)
```

작은 깊이를 가진 객체일수록 앞쪽에 나타납니다.



각 객체는 기준점을 기준으로 회전시킬 수 있습니다.

```
sq.rotate(45)
```

scale을 통해 객체의 크기를 키우거나 줄일 수 있습니다.

```
sq.scale(1.5)
```

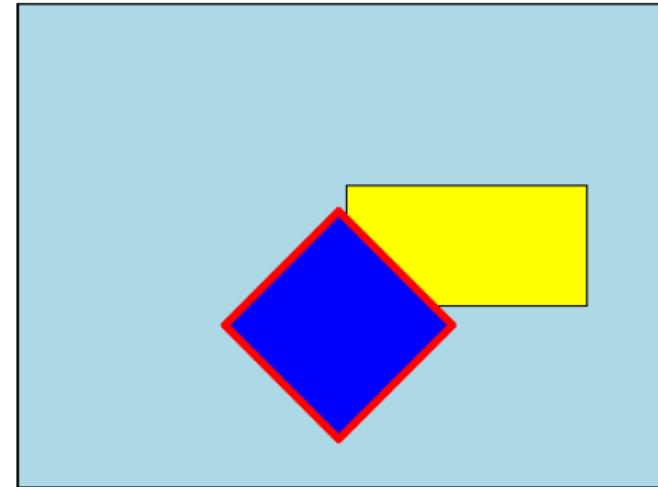
```
r.scale(0.5)
```

Fade-out 효과를 주기 위해서는

```
for i in range(80):
```

```
    sq.scale(0.95)
```

```
canvas.remove(sq)
```

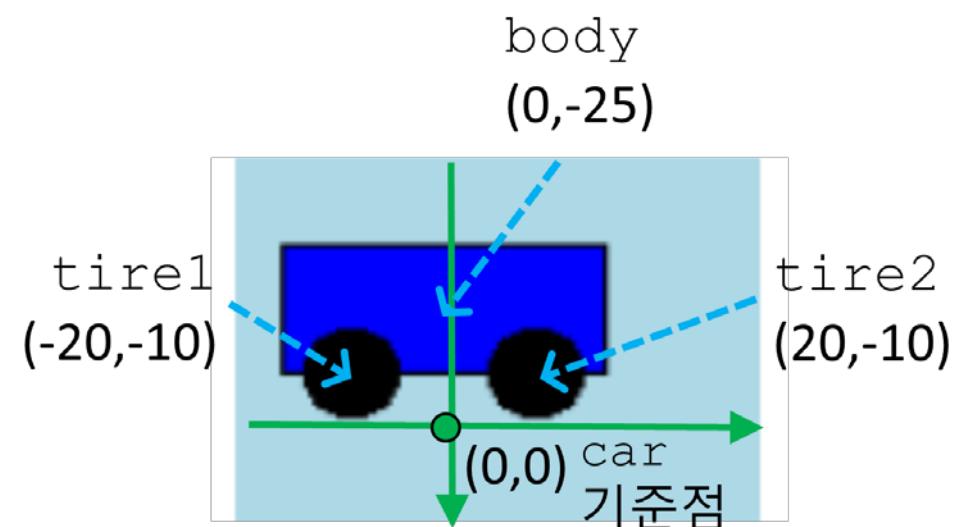


레이어는 여러 개의 그래픽 객체들을 묶어서 같이 변형할 수 있게 합니다.

```
car = Layer()
tire1 = Circle(10, Point(-20,-10))
tire1.setFillColor('black')
car.add(tire1)
tire2 = Circle(10, Point(20,-10))
tire2.setFillColor('black')
car.add(tire2)
body = Rectangle(70, 30, Point(0, -25))
body.setFillColor('blue')
body.setDepth(60)
car.add(body)
```

차에 애니메이션을 주기 위해서는

```
for i in range(250):
    car.move(2, 0)
```



변형

하나의 레이어는 하나의 객체처럼 변형할 수 있습니다.

```
for i in range(50):
    car.move(2, 0)
for i in range(22):
    car.rotate(-1)
for i in range(50):
    car.move(2,-1)
for i in range(22):
    car.rotate(1)
for i in range(50):
    car.move(2,0)
for i in range(10):
    car.scale(1.05)
```



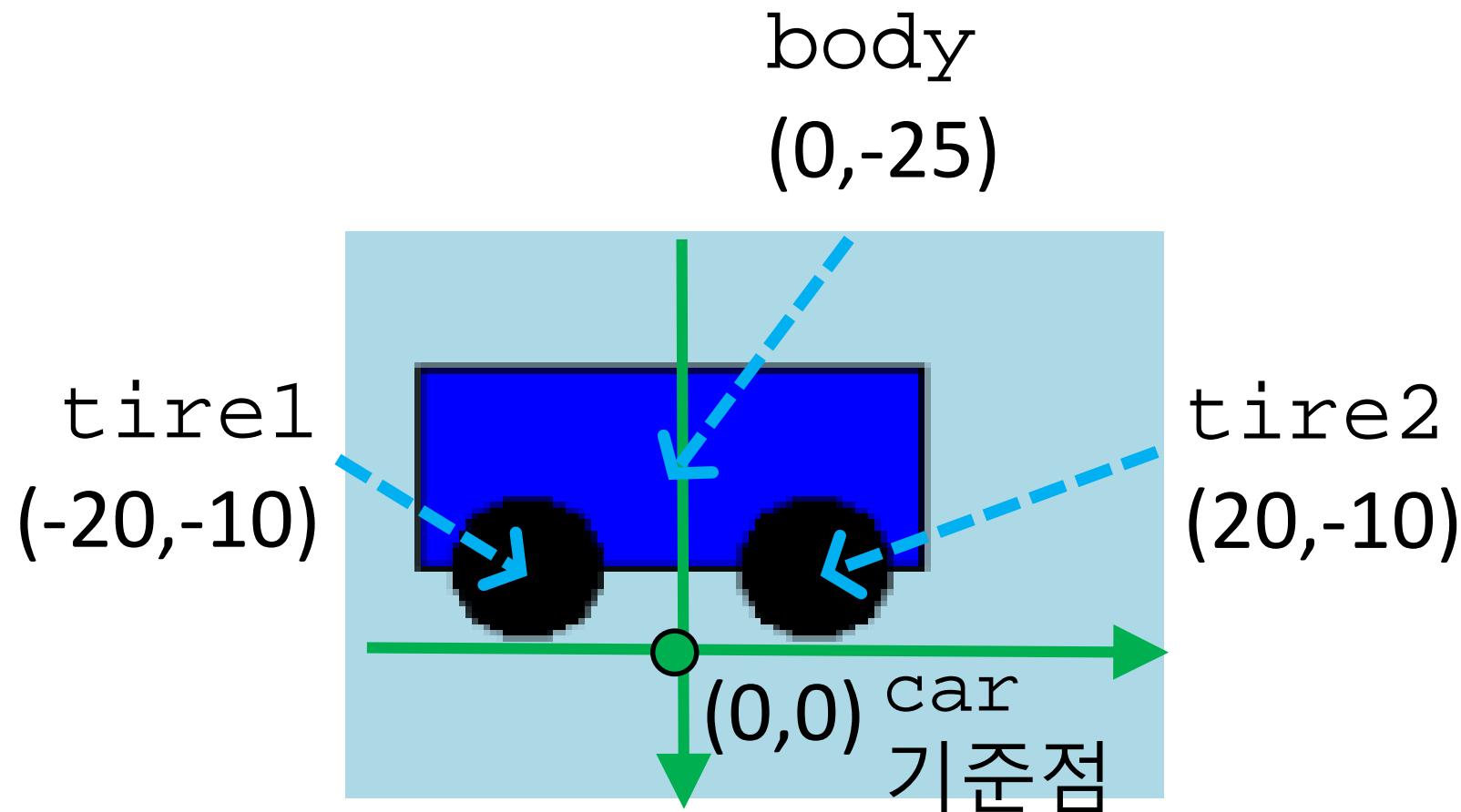
정리 및 예습

본 강의 학습 목표:

- 다양한 기능을 불러서 사용하기 위한 모듈 기능을 이해할 수 있다.
- 다양한 그래픽 객체를 생성하고 변형시킬 수 있다.

다음 강의 학습 목표:

- 그래픽 객체의 특성을 (예. 위치와 색깔)변화 하여 일출 및 일몰 애니메이션을 만들 수 있다.



많은 데이터

다음 표는 2014 소치 동계올림픽의 메달 집계 결과입니다.

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

이 데이터를 Python에서 어떻게 저장할 수 있을까요?
하나하나 변수로 만들려면 총 4×26 개의 변수가 필요하네요..

리스트(List)를 사용하면 여러 값들을 모아서 보관할 수 있습니다.

리스트의 길이는 `len`을 사용해서 구할 수 있습니다.

```
>>> len(countries)
```

26

빈 리스트는 []로 표기할 수 있습니다. 빈 리스트의 길이는 0입니다.

하나의 리스트는 여러 다른 종류의 객체를 담을 수도 있습니다.

```
>>> korea = [ 'Korea' , 'KR' , 3 , 3 , 2 ]
```

```
>>> korea[1]
```

'KR'

```
>>> korea[2]
```

3

튜플을 담을 수도 있습니다.

```
>>> korea = [ "Korea" , 'KR' , (3 , 3 , 2) ]
```

`len`은 리스트의 길이를 반환합니다.

`sum`은 리스트의 각 원소의 합을 반환합니다.

`max`은 리스트에서 가장 큰 원소를, `min`은 가장 작은 원소를 반환합니다.

```
>>> len(gold), sum(gold), max(gold), min(gold)
```

```
(26, 99, 13, 0)
```

```
>>> len(silver), sum(silver), max(silver)
```

```
(26, 97, 11)
```

```
>>> len(bronze), sum(bronze), max(bronze)
```

```
(26, 99, 12)
```

반복문을 이용해 리스트의 각 원소를 탐색할 수 있습니다.

```
for country in countries:  
    print(country)
```

함수 **range**는 range 타입의 객체를 만들어주는 함수입니다.

```
>>> range(10)  
range(0, 10)  
>>> type(range(10))  
<class 'range'>  
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> list(range(10, 15))  
[10, 11, 12, 13, 14]
```

리스트의 원소 값을 바꾸고 싶다면, 그 원소의 위치를 알아야 합니다.

```
>>> l = list(range(1, 11))  
>>> for i in range(len(l)):  
...     l[i] = l[i] ** 2  
>>> l  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

나라별 총 메달 수를 출력해봅시다.

```
>>> for i in range(len(countries)):
...     print(countries[i], gold[i]+silver[i]+bronze[i])
```

위에서 구한 값을 저장하는 새로운 리스트를 만들 수 있습니다.

```
>>> totals = []
>>> for i in range(len(countries)):
...     medals = gold[i]+silver[i]+bronze[i]
...     totals.append( (medals, countries[i]) )
```

새로 만든 totals 리스트는 튜플 (medals, country) 의 리스트입니다.

```
>>> totals
[(3, 'Australia'), (17, 'Austria'), (6, 'Belarus'),
 ..., (4, 'Latvia'), (24, 'Netherlands'), ...,
 (8, 'South Korea'), ..., (2, 'Ukraine'), (28,
 'United States')]
```

정리 및 예습

본 강의 학습 목표:

- 리스트를 통해 복잡한 자료를 프로그램으로 분석하는 방법을 이해할 수 있다.

다음 강의 학습 목표:

- 리스트를 활용하는 다양한 방법을 이해할 수 있다.
- 시퀀스를 표현하는 리스트, 문자열, 튜플의 차이점을 이해한다.

CS101 -리스트 활용법

Lecture 14

School of Computing
KAIST

학습 목표:

- 리스트를 통해 복잡한 자료를 프로그램으로 분석하는 방법을 이해할 수 있다.

많은 데이터

다음 표는 2014 소치 동계올림픽의 메달 집계 결과입니다.

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

이 데이터를 Python에서 어떻게 저장할 수 있을까요?
하나하나 변수로 만들려면 총 4×26 개의 변수가 필요하네요..

리스트(List)를 사용하면 여러 값들을 모아서 보관할 수 있습니다.

리스트는 여러 값을 대괄호 안에 나열해서 적는 방법으로 만들 수 있습니다.

```
>>> countries = [ "Australia", ... , "United States" ]  
>>> gold = [ 0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0,  
           8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9 ]
```

리스트는 `list` 타입의 객체입니다.

리스트의 각 원소는 위치 값을 사용해서 접근할 수 있습니다.

첫 번째 원소는 0번째 위치에, 두 번째 원소는 1번째 위치에 있습니다.

```
>>> countries[0]  
'Australia'  
>>> countries[21]  
'South Korea'  
>>> gold[21]  
3
```

음수 위치를 사용하면 리스트의 끝에서부터 접근할 수 있습니다.

```
>>> countries[-1]  
'United States'  
>>> countries[-5]  
'South Korea'
```

리스트의 길이는 `len`을 사용해서 구할 수 있습니다.

```
>>> len(countries)
```

26

빈 리스트는 []로 표기할 수 있습니다. 빈 리스트의 길이는 0입니다.

하나의 리스트는 여러 다른 종류의 객체를 담을 수도 있습니다.

```
>>> korea = [ 'Korea' , 'KR' , 3 , 3 , 2 ]
```

```
>>> korea[1]
```

'KR'

```
>>> korea[2]
```

3

튜플을 담을 수도 있습니다.

```
>>> korea = [ "Korea" , 'KR' , (3 , 3 , 2) ]
```

`len`은 리스트의 길이를 반환합니다.

`sum`은 리스트의 각 원소의 합을 반환합니다.

`max`은 리스트에서 가장 큰 원소를, `min`은 가장 작은 원소를 반환합니다.

```
>>> len(gold), sum(gold), max(gold), min(gold)
```

```
(26, 99, 13, 0)
```

```
>>> len(silver), sum(silver), max(silver)
```

```
(26, 97, 11)
```

```
>>> len(bronze), sum(bronze), max(bronze)
```

```
(26, 99, 12)
```

리스트는 가변 객체이다

다음 리스트는 비활성기체의 이름을 담은 리스트입니다.

```
>>> nobles = [ 'helium', 'none', 'argon', 'krypton',  
    'xenon' ]
```

저런, 오타가 있네요. 오타를 고쳐볼게요.

```
>>> nobles[1] = "neon"  
>>> nobles  
[ 'helium', 'neon', 'argon', 'krypton', 'xenon' ]
```

이번에는 라돈을 빠뜨렸네요.

```
>>> nobles.append( 'radon' )  
>>> nobles  
[ 'helium', 'neon', 'argon', 'krypton', 'xenon',  
 'radon' ]
```

다시 보기: 하나의 객체는 여러 이름을 가질 수 있습니다.

이를 **Aliasing**이라 부릅니다.

가변 객체를 사용할 때는 객체가 잘못 바뀌지 않도록 조심해서 사용해야 합니다.

```
>>> list1 = [ "A" , "B" , "C" ]      >>> list1 = [ "A" , "B" , "C" ]
>>> list2 = list1                      >>> list2 = [ "A" , "B" , "C" ]
>>> len(list1)                         >>> len(list1)
3                                         3
>>> list2.append( "D" )                >>> list2.append( "D" )
>>> len(list1)                         >>> len(list1)
4                                         3
>>> list1[1] = "X"                     >>> list1[1] = "X"
>>> list2
[ 'A' , 'X' , 'C' , 'D' ]              >>> list2
                                         [ 'A' , 'B' , 'C' , 'D' ]
>>> list1 is list2                    >>> list1 is list2
True                                     False
```

다시 보기: 여러 이름을 가진 객체

하나의 객체는 여러 이름을 가질 수 있습니다.

```
hubo = Robot( "yellow" )
```

```
hubo.move()
```

```
ami = hubo
```

```
ami.turn_left()
```

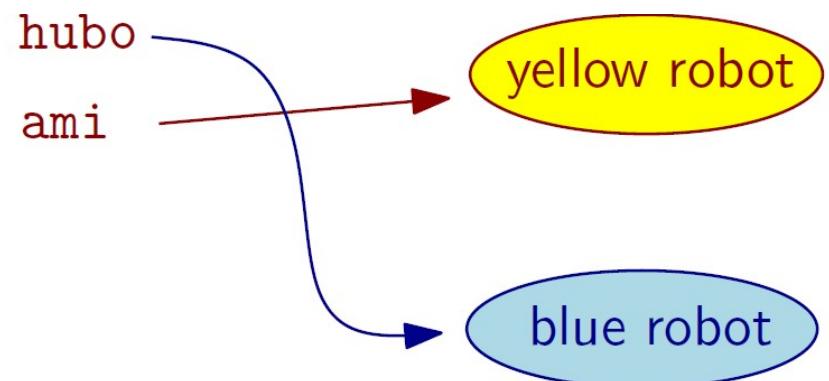
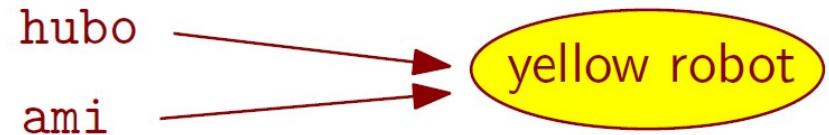
```
hubo.move()
```

```
hubo = Robot( "blue" )
```

```
hubo.move()
```

```
ami.turn_left()
```

```
ami.move()
```



반복문을 이용해 리스트의 각 원소를 탐색할 수 있습니다.

```
for country in countries:  
    print(country)
```

함수 **range**는 range 타입의 객체를 만들어주는 함수입니다.

```
>>> range(10)  
range(0, 10)  
>>> type(range(10))  
<class 'range'>  
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> list(range(10, 15))  
[10, 11, 12, 13, 14]
```

리스트의 원소 값을 바꾸고 싶다면, 그 원소의 위치를 알아야 합니다.

```
>>> l = list(range(1, 11))  
>>> for i in range(len(l)):  
...     l[i] = l[i] ** 2  
>>> l  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

나라별 총 메달 수를 출력해봅시다.

```
>>> for i in range(len(countries)):
...     print(countries[i], gold[i]+silver[i]+bronze[i])
```

위에서 구한 값을 저장하는 새로운 리스트를 만들 수 있습니다.

```
>>> totals = []
>>> for i in range(len(countries)):
...     medals = gold[i]+silver[i]+bronze[i]
...     totals.append( (medals, countries[i]) )
```

새로 만든 totals 리스트는 튜플 (medals, country) 의 리스트입니다.

```
>>> totals
[(3, 'Australia'), (17, 'Austria'), (6, 'Belarus'),
 ..., (4, 'Latvia'), (24, 'Netherlands'), ...,
 (8, 'South Korea'), ..., (2, 'Ukraine'), (28,
 'United States')]
```

정리 및 예습

본 강의 학습 목표:

- 리스트를 통해 복잡한 자료를 프로그램으로 분석하는 방법을 이해할 수 있다.

다음 강의 학습 목표:

- 리스트를 활용하는 다양한 방법을 이해할 수 있다.
- 시퀀스를 표현하는 리스트, 문자열, 튜플의 차이점을 이해한다.

CS101 – 시퀀스: 리스트, 문자열, 튜플

Lecture 15

School of Computing
KAIST

학습 목표:

- 리스트를 활용하는 다양한 방법을 이해할 수 있다.
- 시퀀스를 표현하는 리스트, 문자열, 튜플의 차이점을 이해한다.

많은 데이터

다음 표는 2014 소치 동계올림픽의 메달 집계 결과입니다.

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

이 데이터를 Python에서 어떻게 저장할 수 있을까요?
하나하나 변수로 만들려면 총 4×26 개의 변수가 필요하네요..

리스트(List)를 사용하면 여러 값들을 모아서 보관할 수 있습니다.

리스트는 sort 함수를 이용해 원소들을 정렬할 수 있습니다.

```
>>> ta = [ "JinYeong" , "Jeongmin" , "Minsuk" ,
           "Dohoo" , "Sangjae" , "Byung-Jun" ]
>>> ta.sort( )
>>> ta
[ 'Byung-Jun' , 'Dohoo' , 'Jeongmin' , 'JinYeong' ,
  'Minsuk' , 'Sangjae' ]
```

totals 리스트를 정렬해봅시다.

```
>>> totals.sort( )
>>> totals
[ (1, 'Croatia') , (1, 'Kazakhstan') , (1, 'Slovakia') ,
  (2, 'Ukraine') , (3, 'Australia') , . . . , (8,
  'Japan') , (8, 'Slovenia') , (8, 'South Korea') ,
  . . . , (33, 'Russia') ]
```

리스트의 특정 부분을 잘라내서 새로운 리스트로 만들 수 있습니다 (slicing)

```
sublist = mylist[i:j]
```

sublist는 mylist의 인덱스 $i, i+1, \dots, j-1$ 에 해당하는 원소를 포함한 리스트입니다.

i 를 생략하면, sublist는 mylist의 첫 번째 원소부터 가지게 됩니다.

j 를 생략하면, sublist는 mylist의 마지막 원소까지 가지게 됩니다.

다음처럼 쓰면 리스트를 복사할 수 있습니다.

```
list2 = list1[:]
```

다음처럼 리스트를 뒤집어, 메달이 많은 나라가 앞쪽에 오게 할 수 있습니다.

```
>>> totals.reverse()  
>>> totals  
[(33, 'Russia'), (28, 'United States'), ..., (8,  
'South Korea'), ..., (1, 'Kazakhstan'), (1,  
'Croatia')]
```

메달이 가장 많은 10개의 나라만 따로 볼 수도 있습니다.

```
>>> top_ten = totals[:10]  
>>> for p in top_ten:  
...     medals, country = p  
...     print(medals, country)
```

리스트의 원소는 다음처럼 풀어낼 수 있습니다.

```
>>> for medals, country in top_ten:  
...     print(medals, country)
```

메달 수 상위 10개의 나라를 메달수에 관해 정렬해봅시다.

```
table = [ ]
for i in range(len(countries)):
    table.append( (gold[i], silver[i], bronze[i],
                  countries[i]) )
table.sort()
top_ten = table[-10:]
top_ten.reverse()
for g, s, b, country in top_ten:
    print(country, g, s, b)
```

Russia	13	11	9
Norway	11	5	10
Canada	10	10	5
United States	9	7	12
Netherlands	8	7	9
Germany	8	6	5
Switzerland	6	3	2
Belarus	5	0	1
Austria	4	8	5
France	4	4	7

한 종류의 메달만 획득한 나라

한 종류의 메달만 획득한 나라들을 모두 찾아봅시다.
(메달을 하나도 획득하지 못한 나라는 없다고 가정합니다)

```
def no_medals(countries, al, bl):
    result = []
    for i in range(len(countries)):
        if al[i] == 0 and bl[i] == 0:
            result.append(countries[i])
    return result
```

```
only_gold = no_medals(countries, silver, bronze)
only_silver = no_medals(countries, gold, bronze)
only_bronze = no_medals(countries, gold, silver)
```

```
only_one = only_gold + only_silver + only_bronze
```

많은 데이터

다음 표는 2014 소치 동계올림픽의 메달 집계 결과입니다.

	Australia	0	2	1
Austria	4	8	5	
Belarus	5	0	1	
Canada	10	10	5	
China	3	4	2	
Croatia	0	1	0	
Czech Republic	2	4	2	
Finland	1	3	1	
France	4	4	7	
Germany	8	6	5	
Great Britain	1	1	2	
Italy	0	2	6	
Japan	1	4	3	
Kazakhstan	0	0	1	
Latvia	0	2	2	
Netherlands	8	7	9	
Norway	11	5	10	
Poland	4	1	1	
Russia	13	11	9	
Slovakia	1	0	0	
Slovenia	2	2	4	
South Korea	3	3	2	
Sweden	2	7	6	
Switzerland	6	3	2	
Ukraine	1	0	1	
United States	9	7	12	

리스트 객체 L은 다음과 같은 멤버 함수들을 가지고 있습니다.

- L.append(v) v 객체를 리스트 끝에 추가
- L.insert(i, v) 객체를 리스트의 i번째 위치에 추가
- L.pop() 리스트의 마지막 원소를 삭제하고, 그 값을 반환
- L.pop(i) i번째 원소를 삭제하고, 그 값을 반환
- L.remove(v) v와 일치하는 첫 번째 원소를 삭제
- L.index(v) v와 일치하는 첫 번째 원소의 위치를 반환
- L.count(v) v와 일치하는 원소들의 개수를 반환
- L.extend(K) K의 모든 원소를 리스트 L 끝에 추가
- L.reverse() 리스트의 모든 원소를 역순으로 재배열
- L.sort() 리스트 정렬

다음 두 코드는 어떤 차이가 있을까요?

L.append(13)

L + [13]

리스트는 시퀀스(Sequence)의 한 종류입니다.

문자열과 튜플 또한 시퀀스의 한 종류입니다.

문자열

```
>>> a = "CS101"  
>>> a[-1]  
'1'  
>>> a[2:]  
'101'  
>>> for i in a:  
...     print(i)  
C  
S  
1  
0  
1
```

튜플

```
>>> t = ("CS101", "A+", 13)  
>>> t[0]  
'CS101'  
>>> t[-1]  
13  
>>> t[1:]  
( 'A+' , 13 )  
>>> for i in t:  
...     print(i)  
CS101  
A+  
13
```

리스트, 튜플, 문자열

리스트과 튜플은 비슷하지만 큰 차이점이 있습니다.

리스트는 **가변 객체**지만, 튜플(과 문자열)은 **불변 객체**입니다.

```
>>> t[0] = "CS206"
```

```
TypeError: 'tuple' object does not support item assignment
```

시퀀스는 `list`, `tuple` 함수를 이용해 리스트이나 튜플로 만들 수 있습니다.

```
>>> list(t)
```

```
[ 'CS101', 'A+', 13 ]
```

```
>>> tuple(gold)
```

```
( 0, 4, 5, 10, 3, 0, 2, 1, 4, ..., 2, 6, 1, 9 )
```

```
>>> list("CS101")
```

```
[ 'C', 'S', '1', '0', '1' ]
```

Python에서는 올림픽 메달 집계 정보를 리스트 4개로 만드는 것보다, 튜플들의 리스트로 만드는 방법을 더 많이 사용합니다.

```
medals = [ ('Australia', 0, 2, 1),
            ('Austria', 4, 8, 5),
            ...
            ('United States', 9, 7, 12) ]
```

나라별 총 메달 수는 다음처럼 출력할 수 있습니다.

```
def print_totals1():
    for country, g, s, b in medals:
        print(country + ":", g + s + b)

def print_totals2():
    for item in medals:
        print(item[0] + ":", sum(item[1:]))
```

메달 집계 결과로 히스토그램을 만들어 봅시다.

```
def histogram( ) :  
    t = [ 0 ] * 13  
  
    for item in medals :  
        total = sum(item[1: ] )  
        t[total // 3] += 1  
  
    for i in range(13) :  
        print (str(3*i) + " ~ " +  
               str(3*i+2)+": \t" + ("*" * t[ i ] ))
```

실행 결과

0~2:	*****
3~5:	*****
6~8:	*****
9~11:	**
12~14:	
15~17:	***
18~20:	*
21~23:	
24~26:	***
27~29:	*
30~32:	
33~35:	*
36~38:	

히스토그램

메달 집계 결과로 히스토그램을 만들어 봅시다.

```
def histogram():
    t = [0] * 13
    for item in medals:
        total = sum(item[1:])
        t[total // 3] += 1
    for i in range(13):
        print(str(3*i) + "~" +
              str(3*i+2)+":\t"+("*"*t[i]))
```

실행 결과

KAIST

t[0]	0~2:	****
t[1]	3~5:	****
t[2]	6~8:	*****
t[3]	9~11:	**
t[4]	12~14:	
t[5]	15~17:	***
t[6]	18~20:	*
t[7]	21~23:	
t[8]	24~26:	***
t[9]	27~29:	*
t[10]	30~32:	
t[11]	33~35:	*
t[12]	36~38:	

item	total	total // 3
('Australia', 0, 2, 1)	3	1
('Austria', 4, 8, 5)	17	5
('Belarus', 5, 0, 1)	6	2
('Canada', 10, 10, 5)	25	8
('China', 3, 4, 2)	9	3
('Croatia', 0, 1, 0)	1	0
('Czech Republic', 2, 4, 2)	8	2



t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	...
0	1	0	0	0	0	0	0	0	...
0	1	0	0	0	1	0	0	0	...
0	1	1	0	0	1	0	0	0	...
0	1	1	0	0	1	0	0	1	...
0	1	1	1	0	1	0	0	1	...
1	1	1	1	0	1	0	0	1	...
1	1	2	1	0	1	0	0	1	...

정리 및 예습

본 강의 학습 목표:

- 리스트를 활용하는 다양한 방법을 이해할 수 있다.
- 시퀀스를 표현하는 리스트, 문자열, 튜플의 차이점을 이해한다.

다음 강의 학습 목표:

- 2중 반복문을 통해 리스트를 정렬할 수 있다.
- 소수 (prime number)를 구하는 알고리즘을 구현할 수 있다.

CS101 -리스트 활용 예제: 정렬과 소수 구하기

Lecture 16

School of Computing
KAIST

학습 목표:

- 2중 반복문을 통해 리스트를 정렬할 수 있다.
- 소수 (prime number)를 구하는 알고리즘을 구현할 수 있다.

리스트는 sort 함수를 이용해 원소들을 정렬할 수 있습니다.

```
>>> ta = [ "JinYeong" , "Jeongmin" , "Minsuk" ,  
        "Dohoo" , "Sangjae" , "Byung-Jun" ]  
>>> ta.sort()  
>>> ta  
[ 'Byung-Jun' , 'Dohoo' , 'Jeongmin' , 'JinYeong' ,  
  'Minsuk' , 'Sangjae' ]
```

totals 리스트를 정렬해봅시다.

```
>>> totals.sort()  
>>> totals  
[ (1, 'Croatia') , (1, 'Kazakhstan') , (1, 'Slovakia') ,  
  (2, 'Ukraine') , (3, 'Australia') , . . . , (8,  
  'Japan') , (8, 'Slovenia') , (8, 'South Korea') ,  
  . . . , (33, 'Russia') ]
```

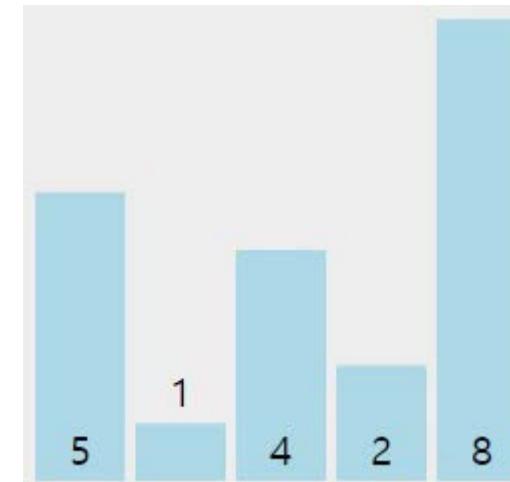
버블 정렬

Sort 시작화 참고

<https://visualgo.net/en/sorting>

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```



예. bubbleSort는
[5, 1, 4, 2, 8]을
[1, 2, 4, 5, 8]로
정렬합니다.

버블 정렬

i	a [i-1]	a [i]	a[i-1] > a[i]
1	5	1	True

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):  
    sorted = False  
    while(not sorted):  
        sorted = True  
        for i in range(1, len(a)):  
            if(a[i-1] > a[i]):  
                a[i-1], a[i]=a[i], a[i-1]  
                sorted = False
```

예. bubbleSort는
[5, 1, 4, 2, 8]을
[1, 2, 4, 5, 8]로
정렬합니다.

버블 정렬

i	a[i-1]	a[i]	a[i-1] > a[i]
1	5	1	True
2	5	4	True

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

(1 5 4 2 8) -> (1 4 5 2 8)

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```

예. bubbleSort는
[5 , 1 , 4 , 2 , 8] 을
[1 , 2 , 4 , 5 , 8] 로
정렬합니다.

버블 정렬

i	a[i-1]	a[i]	a[i-1] > a[i]
1	5	1	True
2	5	4	True
3	5	2	True

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

(1 5 4 2 8) -> (1 4 5 2 8)

(1 4 5 2 8) -> (1 4 2 5 8)

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```

예. bubbleSort는
[5 , 1 , 4 , 2 , 8] 을
[1 , 2 , 4 , 5 , 8] 로
정렬합니다.

버블 정렬

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

(1 5 4 2 8) -> (1 4 5 2 8)

(1 4 5 2 8) -> (1 4 2 5 8)

(1 4 2 5 8) = (1 4 2 5 8)

sorted = False

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```

i	a [i-1]	a [i]	a[i-1] > a[i]
1	5	1	True
2	5	4	True
3	5	2	True
4	5	8	False

예. bubbleSort는
[5, 1, 4, 2, 8]을
[1, 2, 4, 5, 8]로
정렬합니다.

버블 정렬

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```

예. bubbleSort는
[5, 1, 4, 2, 8]을
[1, 2, 4, 5, 8]로
정렬합니다.

i	a [i-1]	a [i]	a[i-1] > a[i]
1	5	1	True
2	5	4	True
3	5	2	True
4	5	8	False

i	a [i-1]	a [i]	a[i-1] > a[i]
1	1	4	False
2	4	2	True
3	4	5	False
4	5	8	False

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

(1 5 4 2 8) -> (1 4 5 2 8)

(1 4 5 2 8) -> (1 4 2 5 8)

(1 4 2 5 8) = (1 4 2 5 8)

sorted = False



두번째 반복

(1 4 2 5 8) = (1 4 2 5 8)

(1 4 2 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

sorted = False

버블 정렬

리스트 a의 원소들을 크기 순으로 정렬

```
def bubbleSort(a):
    sorted = False
    while(not sorted):
        sorted = True
        for i in range(1, len(a)):
            if(a[i-1] > a[i]):
                a[i-1], a[i]=a[i], a[i-1]
                sorted = False
```

예. bubbleSort는
[5, 1, 4, 2, 8]을
[1, 2, 4, 5, 8]로
정렬합니다.

i	a [i-1]	a [i]	a[i-1] > a[i]
1	5	1	True
2	5	4	True
3	5	2	True
4	5	8	False

i	a [i-1]	a [i]	a[i-1] > a[i]
1	1	4	False
2	4	2	True
3	4	5	False
4	5	8	False

i	a [i-1]	a [i]	a[i-1] > a[i]
1	1	2	False
2	2	4	False
3	4	5	False
4	5	8	False

첫번째 반복

(5 1 4 2 8) -> (1 5 4 2 8)

(1 5 4 2 8) -> (1 4 5 2 8)

(1 4 5 2 8) -> (1 4 2 5 8)

(1 4 2 5 8) = (1 4 2 5 8)

sorted = False



두번째 반복

(1 4 2 5 8) = (1 4 2 5 8)

(1 4 2 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

sorted = False



세번째 반복

(1 2 4 5 8) = (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

(1 2 4 5 8) = (1 2 4 5 8)

sorted = True : 프로그램 종료

소수 구하기

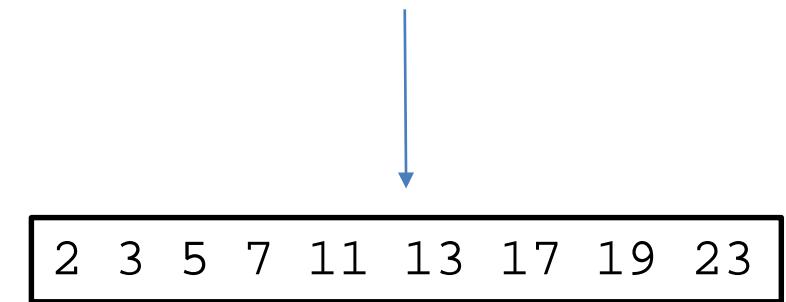
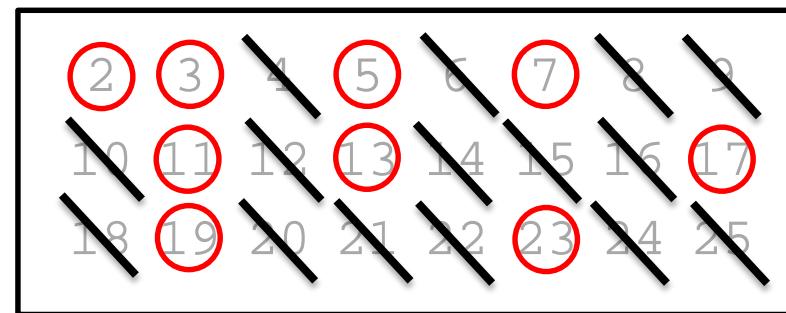
에라토스테네스의 체를 사용해봅시다.

```
# n 보다 작은 소수 리스트 구하기
def sieve(n):
    candidates = list(range(2,n))
    i = 0
    # i: 확인된 소수에 대한 인덱스
    while i < len(candidates):
        prime = candidates[i]
        # j: 소수임을 확인해야 하는
        # 숫자에 대한 인덱스
        j = i + 1
        while j < len(candidates):
            if candidates[j] % prime == 0:
                candidates.pop(j)
            else :
                j= j+1
        i = i + 1
    return candidates
```

sieve(26) :

n이 26일 때 반환되는 리스트

candidates=



정리 및 예습

본 강의 학습 목표:

- 2중 반복문을 통해 리스트를 정렬할 수 있다.
- 소수 (prime number)를 구하는 알고리즘을 구현할 수 있다.

다음 강의 학습 목표:

- 문자열을 다양한 방법으로 활용할 수 있다.
- 집합 (set) 자료구조를 활용할 수 있다.

CS101 - 자료 구조: 문자열과 집합

Lecture 17

School of Computing
KAIST

학습 목표:

- 문자열을 다양한 방법으로 활용할 수 있다.
- 집합 (set) 자료구조를 활용할 수 있다.

문자열을 출력할 때 여러 변수의 값을 이용해야 하는 경우가 있습니다.

```
print( "Max between " + str(x0) + " and " +  
      str(x1) + " is " + str(val))
```

문자열 포맷 연산자 %를 사용하면 더 쉽고 간단하게 문자열을 출력할 수 있습니다.

```
print( "Max between %d and %d is %g" % (x0, x1, val))
```

포맷 연산자는 다음처럼 사용합니다.

```
format_string % (arg0, arg1, .... )
```

포맷 연산자가 사용하는 튜플의 원소들이 format_string 안의 문자열 포맷코드들의 값에 일대일로 지정됩니다. 자료형에 따라 다른 문자열 포맷코드를 사용합니다.

- %d: 10진법 정수
- %g: 실수
- %.2f : 소수점 자리수가 설정된 실수 (이 경우는 소수점 둘째 자리까지)
- %s: 모든 자료형 (문자열 등)

하나의 문자열 포맷 코드만 사용하는 데는 튜플을 쓰지 않아도 됩니다.

```
print( "Maximum is %g" % val)
```

각 문자열 포맷 코드가 나타낼 문자열이 차지하는 영역의 크기를 설정할 수 있습니다.

```
>>> (x0,x1,x2) = (1,2,3)
```

```
>>> print ("%3d~%3d: %10g" % (x0, x1, x2))
```

1 ~ 2: 3

다음처럼 문자열 포맷 코드가 나타낼 문자열을 왼쪽으로 정렬시킬 수도 있습니다.

```
>>> print ("% -3d~% -3d: % -12g" % (x0, x1, x2))
```

1 ~2 :3

문자열은 시퀀스입니다.

```
def is_palindrome (s):
    for i in range (len(s) // 2):
        if s[ i ] != s[ len(s) - i - 1]:
            return False
    return True
```

문자열은 불변 객체입니다.

문자열에는 `in` 연산자를 사용할 수 있습니다.

```
>>> "abc" in "01234abcdefg"
```

True

```
>>> "abce" in "01234abcdefg"
```

False

리스트와 튜플에서는 `in` 연산자가 리스트/튜플의 원소를 대상으로 하지만,
문자열에서는 부분 문자열을 대상으로 합니다.

문자열 객체에서는 다음과 같은 멤버 함수를 사용할 수 있습니다.

- upper() , lower(), capitalize()
- isalpha() , isdigit()
- startswith(prefix) , endswith(suffix)
- find(str1) , find(str1, start),
find(str1, start, end)
- replace(str1, str2)
- rstrip() , lstrip(), strip()
- split(), split(sep)
- join(list1)

모든 멤버 함수는 Python에서 제공하는 문서에도 설명되어 있습니다.

집합(Set)은 수학의 집합과 관련된 자료들을 쉽게 처리하기 위한 자료 구조입니다.

집합은 서로 다른 여러 객체들로 이루어져 있습니다 (중복된 자료가 없습니다).

집합은 중괄호 {} 나 `set()` 함수를 이용해서 만들 수 있습니다.

```
>>> odds = {1, 3, 5, 7, 9}
>>> evens = {2, 4, 6, 8, 10}
>>> emptyset = set() # {} creates an empty dictionary
>>> randomset = {4, 6, 2, 7, 5, 2, 3} # Duplicated ele.

>>> odds
{9, 3, 5, 1, 7}
>>> evens
{8, 10, 2, 4, 6}
>>> emptyset
set()
>>> randomset
{2, 3, 4, 5, 6, 7}
```

리스트는 집합으로 변환할 수 있습니다.

```
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,  
          0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]  
>>> gold  
[0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,  
 0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]  
>>> goldset = set(gold)  
>>> goldset  
{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13}  
>>> type(goldset)  
<class 'set'>
```

문자열 또한 집합으로 변환할 수 있습니다.

```
>>> set("Good morning!")  
{'G', 'm', 'o', 'd', ' ', 'r', 'n', 'g', '!', 'o', 'G'}
```

집합의 원소에는 순서가 없기 때문에 원소의 위치를 특정할 수 없습니다.

```
>>> odds[ 1]
```

```
TypeError: 'set' object does not support indexing
```

하지만, `in` 연산자는 사용할 수 있습니다.

```
>>> 3 in odds
```

```
True
```

```
>>> 2 in odds
```

```
False
```

```
>>> for num in odds:
```

```
...     print( num)
```

```
9
```

```
3
```

```
5
```

```
1
```

```
7
```

집합 객체 s 에는 다음과 같은 멤버 함수들을 사용할 수 있습니다.

- $s.add(v)$: 원소 v 를 집합 s 에 추가한다.
- $s.remove(v)$: 원소 v 를 집합 s 에서 제거한다.
- $s.pop()$: 무작위 원소를 집합 s 에서 제거하고 그 원소를 반환한다.
- $s.intersection(k)$: 집합 s, k 의 공통 원소를 반환한다. ($s \cap k$)
- $s.union(k)$: 집합 s, k 의 합집합을 반환한다. ($s \cup k$)
- $s.difference(k)$: 집합 k 에 있는 원소들을 s 에서 제거한다. ($s \cap k^c$)

다음은 멤버 함수들의 사용 예시입니다.

```
>>> randomset
```

```
{2, 3, 4, 5, 6, 7}
```

```
>>> randomset. add(9)
```

```
>>> randomset
```

```
{2, 3, 4, 5, 6, 7, 9}
```

```
>>> randomset. remove(7)
```

```
>>> randomset
```

```
{2, 3, 4, 5, 6, 9}
```

```
>>> randomset. pop()
```

```
2
```

```
>>> randomset
```

```
{3, 4, 5, 6, 9}
```

다음은 멤버 함수들의 사용 예시입니다.

```
>>> randomset
```

```
{3, 4, 5, 6, 9}
```

```
>>> randomset. intersection(odds)
```

```
{9, 3, 5}
```

```
>>> randomset. union(evens)
```

```
{2, 3, 4, 5, 6, 8, 9, 10}
```

```
>>> randomset. difference(odds)
```

```
{4, 6}
```

```
>>> odds. difference(randomset)
```

```
{1, 7}
```

```
>>> randomset. difference(odds, evens)
```

```
set()
```

정리 및 예습

본 강의 학습 목표:

- 문자열을 다양한 방법으로 활용할 수 있다.
- 집합 (set) 자료구조를 활용할 수 있다.

다음 강의 학습 목표:

- 사전 (dictionary) 자료구조를 활용할 수 있다.
- 리스트, 집합, 사전의 장단점을 이해하여
목적에 적합한 자료구조를 사용할 수 있다.

CS101 - 자료 구조: 사전

Lecture 18

School of Computing
KAIST

학습 목표:

- 사전 (dictionary) 자료구조를 활용할 수 있다.
- 리스트, 집합, 사전의 장단점을 이해하여 목적에 적합한 자료구조를 사용할 수 있다.

사전(Dictionary) 또한 Python에서 유용하게 쓰일 수 있는 자료 구조입니다.

사전은 리스트나 집합처럼 여러 객체를 모아서 만든 객체입니다.

다른 자료 구조와의 가장 큰 차이점은, 다양한 종류의 불변 객체를 사용해서 자료에 접근할 수 있다는 점입니다 (예. 숫자, 문자열을 사용해서 자료에 접근이 가능).

사전에서 사용되는 인덱스는 **키(key)** 라고 부르고, 키가 가리키는 대상 객체는 **값(value)** 이라고 부릅니다. 키와 값의 쌍을 key-value pair라 합니다.

사전은 중괄호 {}나 `dict()` 함수를 이용해서 만들 수 있습니다.

```
majors = {"CS": "Computer Science",
           "EE": "Electrical Engineering",
           "MAS": "Mathematical Sciences", "ME":
           "Mechanical Engineering"}
```

```
d1 = dict() # an empty dictionary
```

```
d2 = {} # an empty dictionary
```

사전

사전의 원소들에는 순서가 없습니다.

사전의 원소에는 키로만 접근할 수 있습니다.

```
>>> majors[0]
```

```
KeyError: 0
```

다음과 같이 사전에 새로운 키와 값을 넣을 수 있습니다.

```
>>> majors["PH"] = "Physic"
```

```
>>> majors["PH"]
```

```
'Physic'
```

다음처럼 키가 가리키는 값을 바꿀 수도 있습니다.

```
>>> majors["PH"] = "Physics"
```

```
>>> majors["PH"]
```

```
'Physics'
```

사전 객체 d는 다음과 같은 함수들과 연산자들을 사용할 수 있습니다.

- `len(d)`: d의 원소의 수를 반환한다.
- `key in d`: d가 key를 가지고 있으면 `True`를, 아니면 `False`를 반환한다.
- `d.get(key, default=None)`: d에서 key가 가리키는 값을 반환한다.
d가 key를 가지고 있지 않으면 default 값을 반환한다.
- `d.keys()`: d의 모든 key 객체 목록을 반환한다.
- `d.values()`: d의 모든 value 객체 목록을 반환한다.
- `d.items()`: d의 모든 key-value pair 목록을 반환한다.
- `del d[key]`: key에 해당하는 key-value pair를 사전 d에서 제거한다.

`keys()`, `values()`, `items()`로 반환되는 객체는 리스트 객체가 아닙니다.

이 객체들은 리스트처럼 원소들을 가지고 있지만, 객체를 변경할 수는 없습니다.
(예. `append()`와 같이 원소를 추가할 수 없습니다.)

다음은 함수들의 사용 예시입니다.

```
>>> majors[0]=0.001
```

```
>>> majors
```

```
{0: 0.001, 'CS': 'Computer Science', 'PH': 'Physics',  
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',  
'MAS': 'Mathematical Sciences' , 0:0.001}
```

```
>>> len(majors)
```

```
6
```

```
>>> del majors[0]
```

```
>>> majors
```

```
{'CS': 'Computer Science', 'PH': 'Physics',  
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',  
'MAS': 'Mathematical Sciences'}
```

```
>>> len(majors)
```

```
5
```

```
>>> "CS" in majors
```

```
True
```

```
>>> "AI" in majors
```

```
False
```

다음은 함수들의 사용 예시입니다.

```
>>> majors.keys()  
dict_keys(['CS', 'PH', 'ME', 'EE', 'MAS'])
```

```
>>> majors.values()  
dict_values(['Computer Science', 'Physics',  
'Mechanical Engineering', 'Electrical Engineering',  
'Mathematical Sciences'])
```

```
>>> majors.items()  
dict_items([('CS', 'Computer Science'), ('PH', 'Physics'),  
(('ME', 'Mechanical Engineering'), ('EE', 'Electrical Engineering'),  
(('MAS', 'Mathematical Sciences'))])
```

반복문과 **in** 연산자를 사용하면, 사전의 키들에 해당하는 값들을 찾을 수 있습니다.

```
>>> for key in majors:  
...     print("%s is %s." % (key, majors[key]))  
CS is Computer Science.  
PH is Physics.  
ME is Mechanical Engineering.  
EE is Electrical Engineering.  
MAS is Mathematical Sciences.
```

반복문과 `items()` 함수를 사용하면 key-value pair들을 찾을 수 있습니다.

```
>>> for key, value in majors.items():  
...     print("%s is %s." % (key, value))  
CS is Computer Science.  
PH is Physics.  
ME is Mechanical Engineering.  
EE is Electrical Engineering.  
MAS is Mathematical Sciences.
```

언제 어떤 자료 구조를 사용하면 좋을까요?

- 순서가 있는 객체들을 다룰 때
 - > 리스트 자료 구조
- 순서가 없는 객체들을 다룰 때
 - > 집합 자료 구조
- 키를 통해 키가 가리키는 값을 쉽게 찾고 싶을 때
 - > 사전 자료 구조

리스트보다 집합에서 원소의 포함 여부를 더 빠르게 계산할 수 있습니다.

```
import time
```

```
large_list = list(range(10000000))
```

```
large_set = set(large_list)
```

```
st = time.time()
```

```
for num in range(100000):
```

```
    if num not in large_list:
```

```
        print("What?!")
```

```
print("Running time for list: %f sec" % (time.time() - st))
```

```
st = time.time()
```

```
for num in range(100000):
```

```
    if num not in large_set:
```

```
        print("What?!")
```

```
print("Running time for set: %f sec" % (time.time() - st))
```

Result:

```
Running time for list: 78.066966 sec
```

```
Running time for set: 0.010978 sec
```

본 강의 학습 목표:

- 사전 (dictionary) 자료구조를 활용할 수 있다.
- 리스트, 집합, 사전의 장단점을 이해하여 목적에 적합한 자료구조를 사용할 수 있다.

다음 강의 학습목표:

- 자료구조를 활용하여, 영상을 합성할 수 있다 (크로마키).
- 자료구조를 활용하여, 사진에 비밀정보를 숨길 수 있다.

CS101 -이미지 프로세싱

Lecture 19

School of Computing
KAIST

학습 목표:

- 자료구조를 활용하여, 영상을 합성 할 수 있다 (크로마키).
- 자료구조를 활용하여, 사진에 비밀 정보를 숨길 수 있다.

왼쪽의 장영실 동상 그림을 오른쪽 배경 위에 올려봅시다.

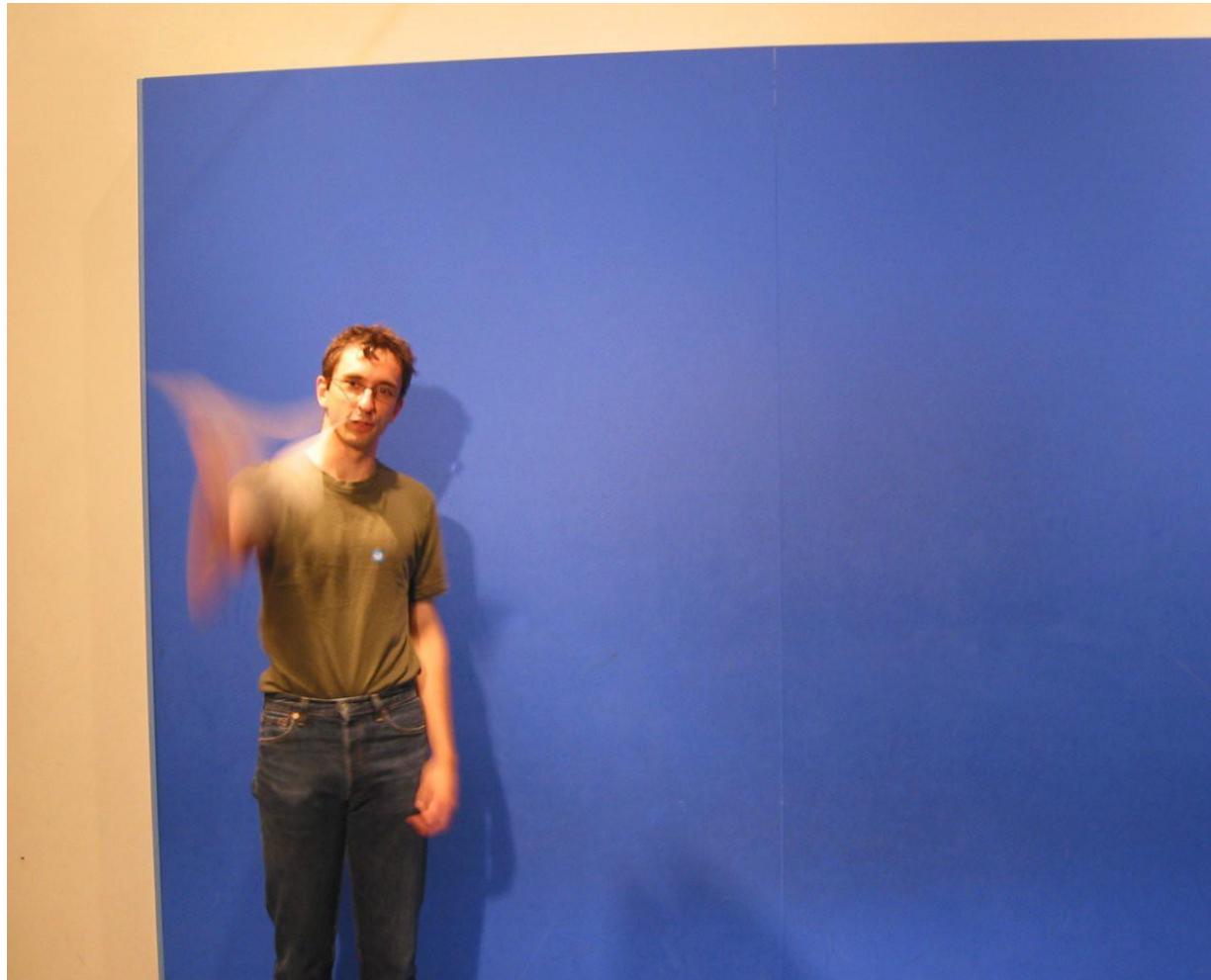


```
def paste(canvas, img, x1, y1):  
    w, h = img.size()  
    for y in range(h):  
        for x in range(w):  
            canvas.set(x1 + x, y1 + y, img.get(x, y))
```

크로마키는 두 개의 영상을 합성하는 기술입니다.

한 영상의 특정 색을 투명하게 만들어서 뒤의 배경 영상을 비치게 할 수 있습니다.

이 기술은 일기 예보에서 많이 사용합니다.



오른쪽 사진의 배경은 정확히 한 가지 색이 아닙니다

- 파란색을 띠는 비슷한 색일 뿐입니다.

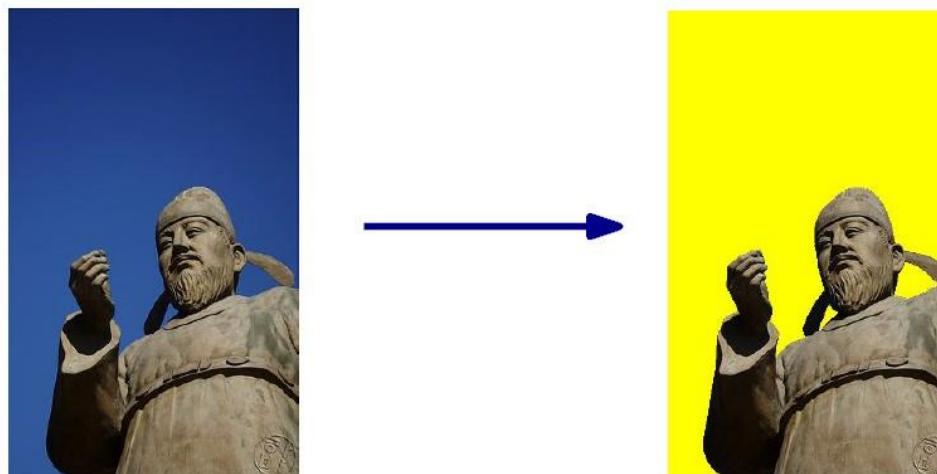
두 색이 얼마나 비슷한지 알 수 있는 함수가 있으면, 함수를 통해 어느 부분이 사진의 배경인지 알 수 있습니다.

```
def dist(c1, c2):  
    r1, g1, b1 = c1  
    r2, g2, b2 = c2  
    return math.sqrt((r1-r2)**2 + (g1-g2)**2 +  
                     (b1-b2)**2)
```



이 수식은 3차원 공간에서 두 점의 거리를 구하는 식과 같습니다.

```
def chroma(img, key, threshold):  
    w, h = img.size()  
    for y in range(h):  
        for x in range(w):  
            p = img.get(x, y)  
            if dist(p, key) < threshold:  
                img.set(x, y, Color.yellow)
```



이제 노란색 배경 대신, 배경 사진의 색을 이용하는 함수를 만들어 봅시다.

```
def chroma_paste(canvas, img, x1, y1, key):  
    w, h = img.size()  
    for y in range(h):  
        for x in range(w):  
            p = img.get(x, y)  
            if p != key:  
                canvas.set(x1 + x, y1 + y, p)
```



사람은 색의 작은 차이를 거의 인지하지 못합니다.
이 현상은 사진 안에 어떤 정보를 숨기는 데 사용할 수 있습니다.

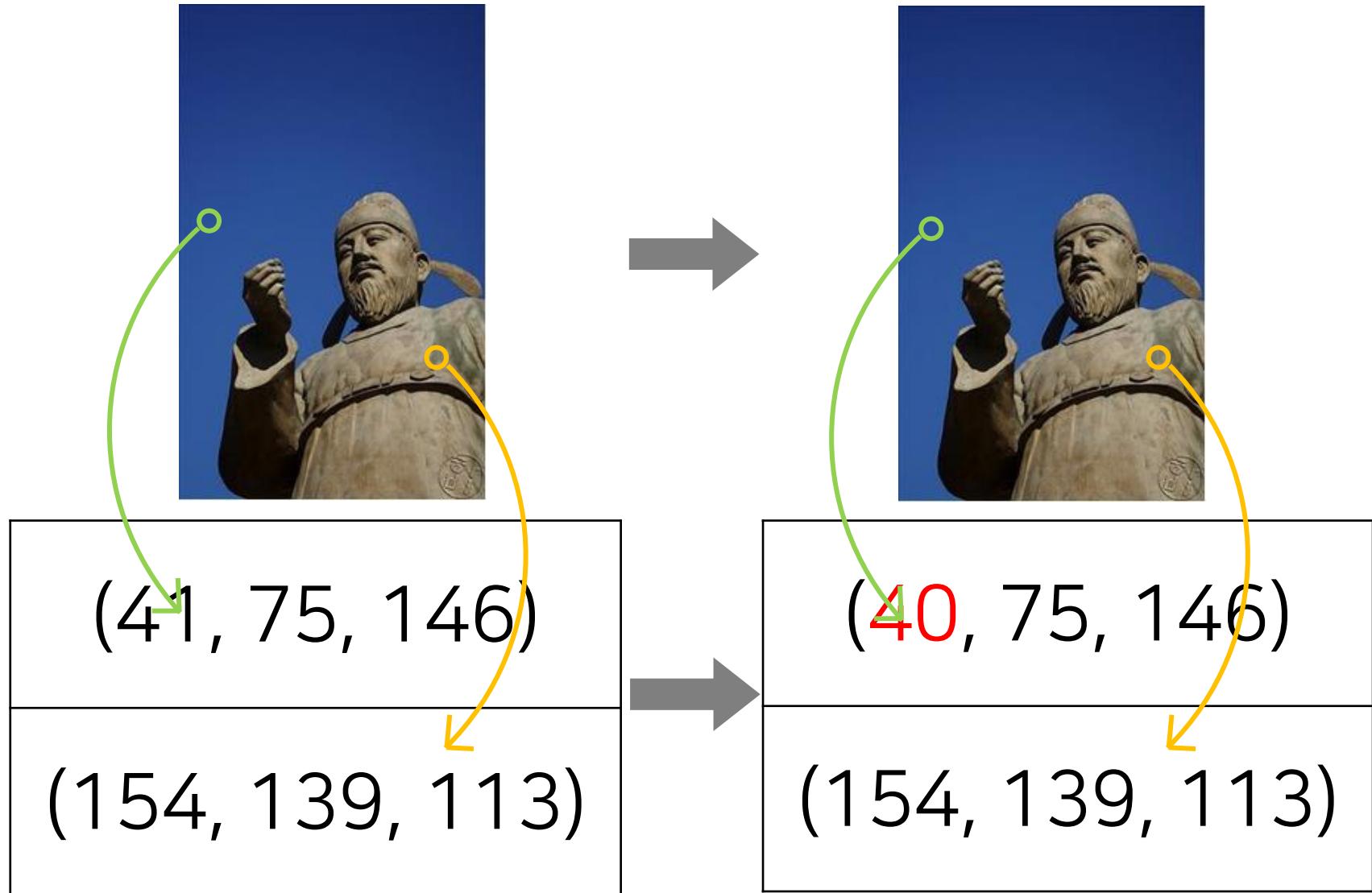
다음은 사진 `img` 안에 흑백 사진을 숨기는 알고리즘입니다.

- `img`의 모든 픽셀 (r, g, b)에서, r 이 홀수면 1을 뺍니다.
- 흑백 사진의 검정색 픽셀과 동일한 위치에 있는 `img`의 모든 픽셀을 찾아, 픽셀의 r 에 1을 더합니다.

이 숨겨진 흑백 사진은 `img`의 모든 픽셀 (r, g, b)에서 r 이 홀수면 해당 픽셀을 검정색으로, 짹수면 하얀색으로 바꿔서 얻을 수 있습니다.

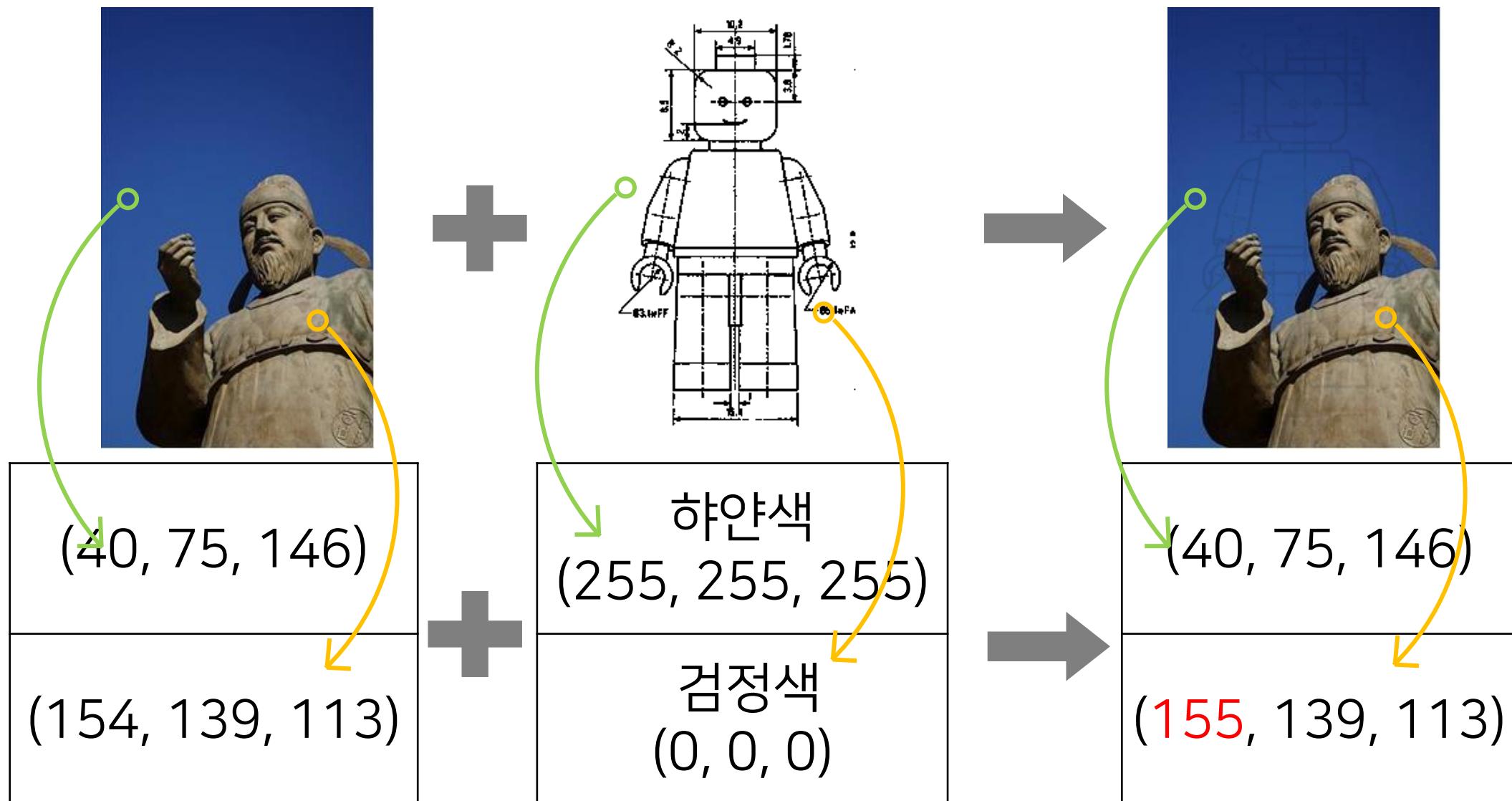
정보 은닉 예제

정보 숨기기(1/2)



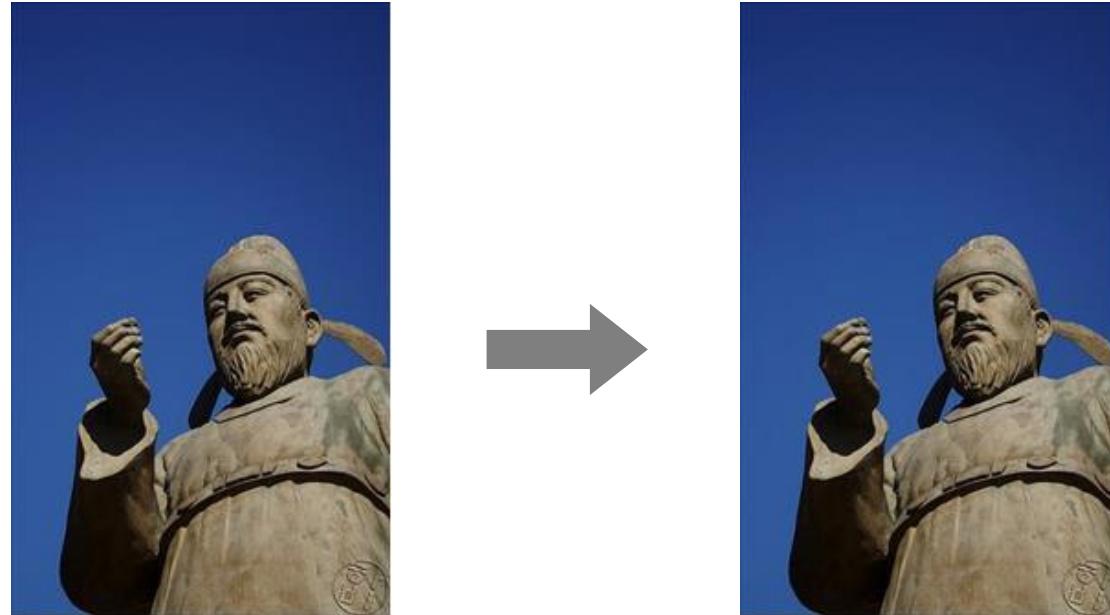
정보 은닉 예제

정보 숨기기(2/2)



정보 은닉 예제

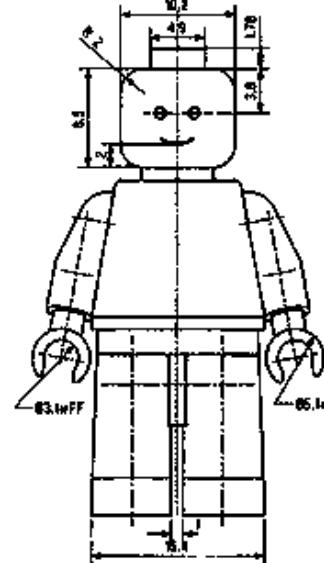
정보 숨기기 코드 (1/2)



```
# hide bwimg into img
def hide_picture(img, bwimg):
    w, h = img.size()
    for y in range(h):
        for x in range(w):
            r, g, b = img.get(x, y)
            if r % 2 == 1:
                r = r - 1
            img.set(x, y, (r, g, b))
    ...
    return img
```

정보 은닉 예제

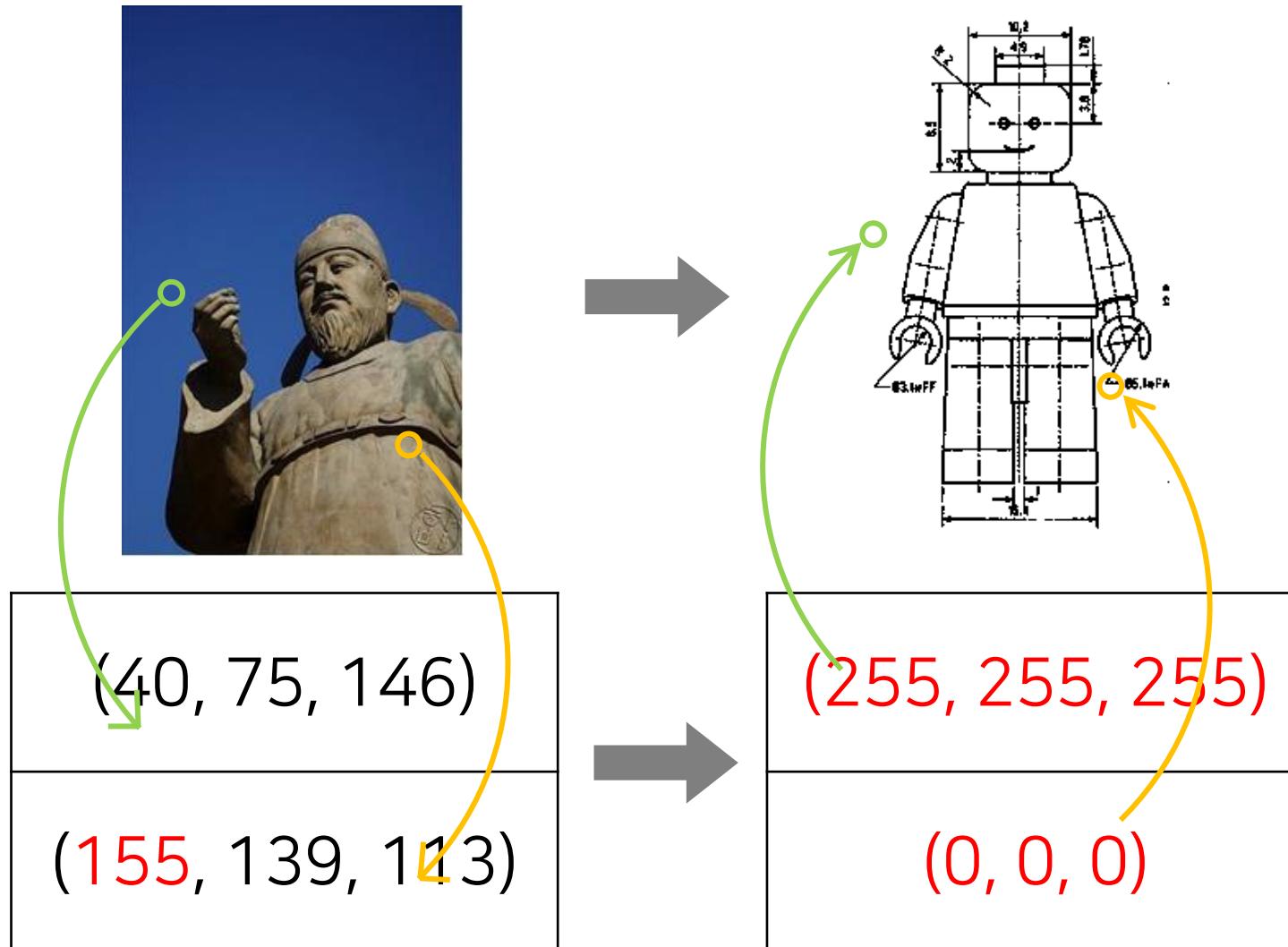
정보 숨기기 코드 (2/2)



```
...  
black = (0, 0, 0)  
w1, h1 = bwimg.size()  
for y in range(h1):  
    for x in range(w1):  
        r, g, b = img.get(x, y)  
        if bwimg.get(x, y) == black  
            r = r + 1  
        img.set(x, y, (r, g, b))
```

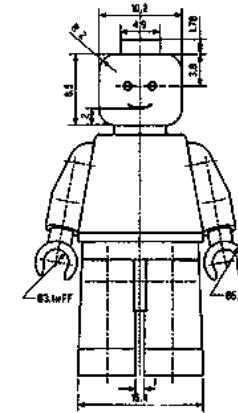
정보 은닉 예제

정보 복원하기



정보 은닉 예제

정보 복원하기 코드



```
white = (255, 255, 255)
```

```
black = (0, 0, 0)
```

```
def restore_picture(img):
```

```
    w, h = img.size()
```

```
    for y in range(h):
```

```
        for x in range(w):
```

```
            r, g, b = img.get(x, y)
```

```
            if r % 2 == 1:
```

```
                img.set(x, y, black)
```

```
            else:
```

```
                img.set(x, y, white)
```

정리 및 예습

본 강의 학습목표:

- 자료구조를 활용하여, 영상을 합성할 수 있다 (크로마키).
- 자료구조를 활용하여, 사진에 비밀정보를 숨길 수 있다.

다음 강의 학습 목표:

- 대량의 자료를 포함하는 텍스트 파일을 만들어서 읽고 쓸 수 있다.
- 프로그램 구문 실행 흐름을 바꾸는
break와 continue를 활용할 수 있다.

CS101 – 텍스트 프로세싱

Lecture 20

School of Computing
KAIST

학습 목표:

- 대량의 자료를 포함하는 텍스트 파일을 만들어서 읽고 쓸 수 있다.
- 프로그램 구문 실행 흐름을 바꾸는 break와 continue를 활용할 수 있다.

파일

"planets.txt" 파일은 다음 내용을 가지고 있습니다.

Mercury

Venus

Earth

Mars

Jupiter

Saturn

Uranus

Neptune

```
>>> f = open("planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

f는 파일의 내용이 아니라 파일 객체입니다. (type: <class '_io.TextIOWrapper'>)

\n은 파일에서 읽어온 개행(줄 바꿈) 문자를 의미합니다.

파일에서 읽어온 문자열 앞뒤의 공백 문자(줄 바꿈 문자, 띄어쓰기 등)를 제거하기 위해서 `strip()`, `rstrip()` 함수를 사용합니다.

`print()` 문에 `end=" "` 인자를 추가하면 문자열을 출력한 후 줄을 바꾸는 대신 한 칸을 띄어 쓸 수 있습니다.

```
>>> for l in f:  
...     s = l.strip()  
...     print(s, end=" ")
```

Venus Earth Mars Jupiter Saturn Uranus Neptune

파일 객체에 `for` 반복문을 사용하면 반복할 때마다 `readline()` 함수를 실행합니다. 반복문은 파일의 마지막 줄을 읽은 후에 종료됩니다.

파일 객체의 사용이 끝나면 `f.close()` 함수를 실행해야 합니다.

다음은 파일 전체의 내용을 읽어 리스트에 저장하는 프로그램입니다.

```
planets = []
f = open("planets.txt", "r")
for line in f:
    planets.append(line.strip())
f.close()
print(planets)
```

파일 객체는 위와 비슷한 일을 하는 멤버 함수를 가지고 있습니다.
(이 함수는 공백 문자를 따로 제거하지는 않습니다)

```
planets = f.readlines()
```

파일에서 earth라는 단어가 포함된 줄의 위치를 찾고 싶습니다.

```
f = open("planets.txt", "r")
current = 0
earth = 0
for line in f:
    current += 1
    planet = line.strip().lower()
    if planet == "earth":
        earth = current
print("Earth is planet #%" + str(earth))
```

이 프로그램은 earth 단어의 위치와 관계 없이 항상 파일 전체 내용을 읽지만,
단어를 찾은 이후에는 더 이상 파일을 읽을 필요가 없습니다.

break 키워드는 현재 실행중인 반복문을 중지하고 빠져나옵니다.

```
f = open("planets.txt", "r")
earth = 0
for line in f:
    earth += 1
    planet = line.strip().lower()
    if planet == "earth":
        break
print("Earth is planet #%d" % earth)
```

break 를 사용하면 가장 안쪽의 반복문만 빠져나옵니다.

```
>>> for x in range(2):
...     for y in range(5):
...         print(y, end=" ")
...         if y == 3:
...             break
0 1 2 3 0 1 2 3
```

파일의 내용에는 주석이 있을 수도 있습니다.

파일의 모든 주석이 #으로 시작한다고 하면, 다음과 같이 주석의 내용을 읽지 않고 건너뛸 수 있습니다.

```
f = open("planetsc.txt", "r")
earth = 0
for line in f:
    planet = line.strip().lower()
    if planet[0] == "#":
        continue
    earth += 1
    if planet == "earth":
        break
print("Earth is planet #%d" % earth)
```

continue 키워드는 현재 실행중인 반복을 건너뜁니다.

113,809개의 영어 단어로 이루어진 words.txt 파일의 자료를 사용해서 단어 게임을 해 봅시다 (https://en.wikipedia.org/wiki/Moby_Project)

단어의 길이가 18보다 긴 모든 영어 단어를 출력해 봅시다.
(예시: counterdemonstrations, 21글자)

```
f = open("words.txt", "r")  
  
for line in f:  
    word = line.strip()  
    if len(word) > 18:  
        print(word)  
  
f.close()
```

글자 'e'가 포함되지 않은 단어의 수를 세 봅시다.

```
f = open("words.txt", "r")  
  
count = 0  
  
for line in f:  
    word = line.strip()  
    if not "e" in word:  
        count += 1  
  
print("%d words have no 'e'" % count)  
f.close()
```

Abecedarian words

단어의 모든 글자가 알파벳 순서로 정렬된 단어들을 찾아봅시다.
(예시: art, allow, cello)

```
def is_abecedarian(word):  
    for i in range(1, len(word)):  
        if word[i-1] > word[i]:  
            return False  
    return True
```

```
f = open("words.txt", "r")
```

```
for line in f:  
    word = line.strip()  
    if is_abecedarian(word):  
        print(word)
```

```
f.close()
```

동일한 두 글자가 연속해서 3번 이상 이어지는 단어는 얼마나 있을까요?

(예시: bookkeeper)

(유사하지만 해당되지 않는 경우: Committee, Mississippi)

```
def three_doubles(word):
    s = ""
    for i in range(1, len(word)):
        if word[i-1] == word[i]:
            s = s + "*"
        else:
            s = s + " "
    return "***" in s
```

파일을 새로 만들고, 내용을 쓰는 것도 가능합니다.

```
f = open("./test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

파일을 쓰기 위해서는 "`w`" 모드를 사용해야 합니다.

파일 객체에는 파일에 내용을 쓰기 위한 `write(text)` 멤버 함수가 있습니다.
`print()` 와는 달리, `write()` 함수는 `text` 내용 출력 후 자동으로 줄을 바꾸거나 공백 문자를 넣지 않습니다. 줄을 바꾸고 싶다면 개행 문자 '`\n`'을 추가로 출력해야 합니다.

파일 사용이 끝나면 `close()`를 호출해야 합니다.
(호출하지 않으면 파일이 불완전하게 저장될 수 있습니다)

환율 자료를 이용해 실습을 해 봅시다. 1994.txt ... 2009.txt 파일은 해당 연도의 일별 KRW-USD 환율 정보를 가지고 있습니다. (www.oanda.com)

2009/05/11 0.00080110

2009/05/12 0.00083010

...

즉, 2009년 5월 11일에는 1 미국 달러가 약 1248.28 원이었습니다.
($1248.28 \div 1 / 0.00080110$)

먼저, 모든 16개 파일을 읽어서, 각 라인의 문자열에 대해 `split` 함수를 사용하여, 날짜, 환율 정보가 담긴 튜플의 리스트로 만듭니다.

```
[... (20091227, 1154), (20091228, 1154),  
(20091229, 1167), (20091230, 1167),  
(20091231, 1163)]
```

최고, 최저 환율과 각 연도별 평균 환율을 구해봅시다.

Minimum: (19950705, 755)

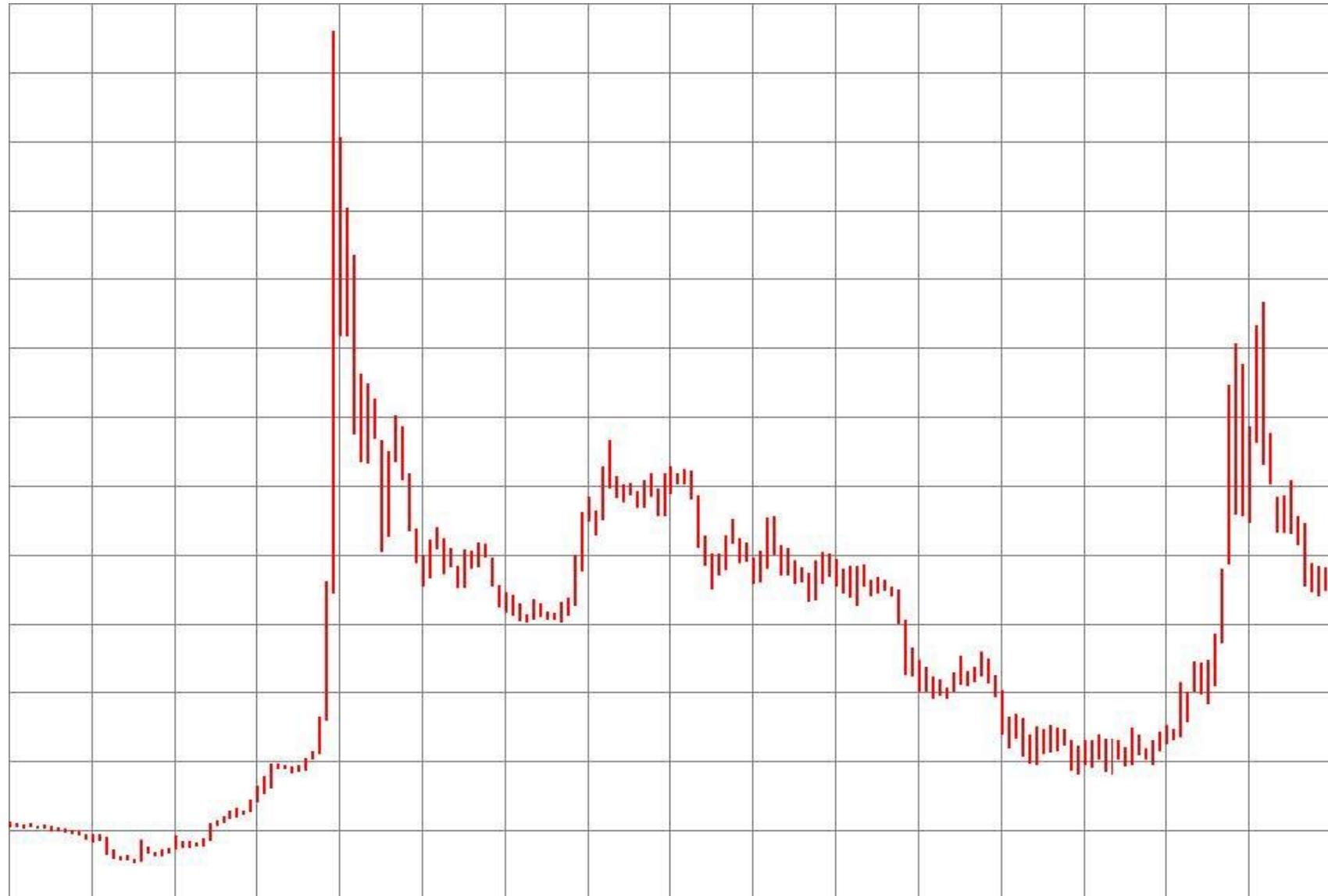
Maximum: (19971223, 1960)

월별 최대, 최소 환율을 구해봅시다.

```
def find_minmax(yr):
    minmax = [ (9999, 0) ] * 12
    data = read_year(yr)
    for d, v in data:
        # make month 0 .. 11
        month = (d // 100) % 100 - 1
        minr, maxr = minmax[month]
        if v < minr:
            minr = v
        if v > maxr:
            maxr = v
        minmax[month] = minr, maxr
    return minmax
```

그래프 만들기

cs1media 모듈을 이용해 환율 변화 그래프를 만들어 봅시다.



정리 및 예습

본 강의 학습목표:

- 대량의 자료를 포함하는 텍스트 파일을 만들어서 읽고 쓸 수 있다.
- 프로그램 구문 실행 흐름을 바꾸는 break와 continue를 활용할 수 있다.

다음 강의 학습 목표:

- 블랙잭 카드 게임을 프로그램으로 만들기 위해 블랙잭 규칙을 이해 할 수 있다.
- 블랙잭 카드 게임에 사용되는 카드를 객체로 표현할 수 있다.

CS101 - 객체 (object)로 블랙잭 카드 게임 만들기 (1/2)

Lecture 21

School of Computing
KAIST

학습 목표:

- 블랙잭 카드 게임을 프로그램으로 만들기 위해 블랙잭 규칙을 이해할 수 있다.
- 블랙잭 카드 게임에 사용되는 카드를 객체로 표현할 수 있다.

블랙잭 게임은 총 52장으로 이루어진 플레잉 카드를 사용합니다.
각각의 카드는 무늬와 끊수를 가지고 있습니다.

- 무늬 (suit)
 - 클럽
 - 스페이드
 - 하트
 - 다이아몬드

- 끊수 (face)

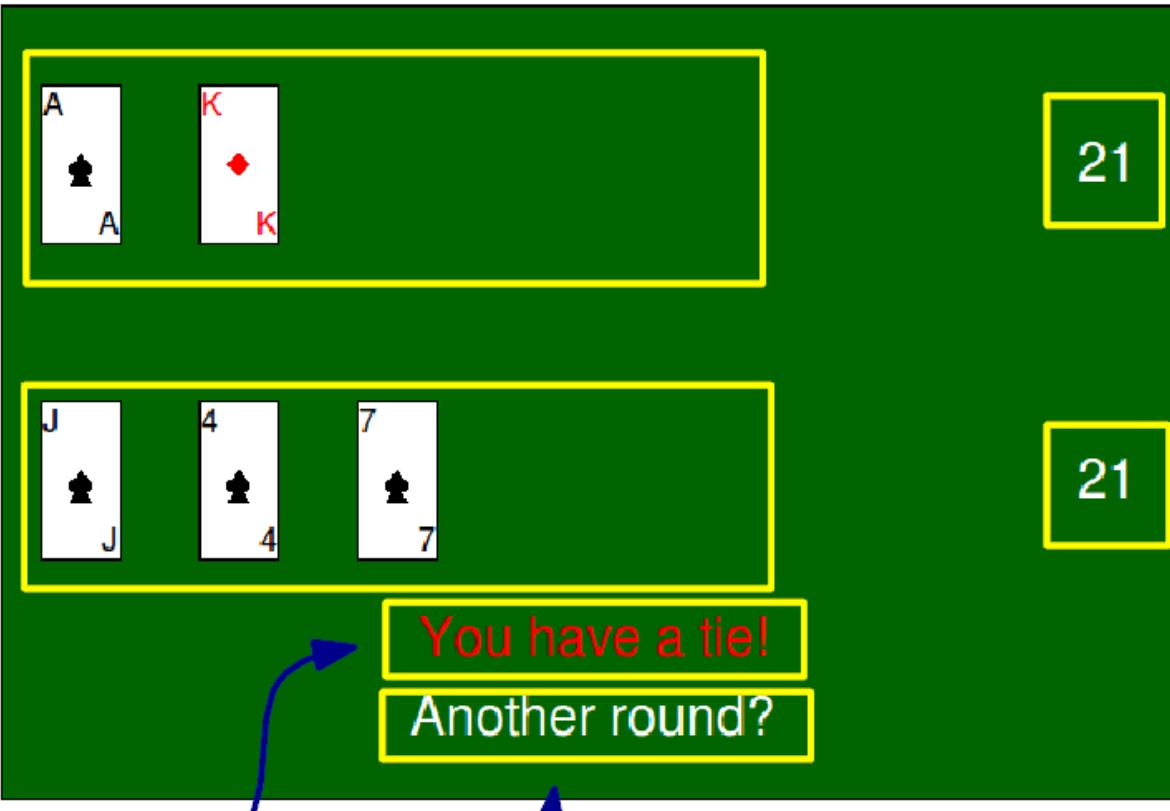
- 2
- 3
- ...
- 10
- J (Jack)
- Q (Queen)
- K (King)
- A (Ace)



- 블랙잭 게임에서는 각 카드가 값(value)을 하나씩 가집니다.
- 숫자 카드는 카드의 끗수가, A카드는 11, J,Q,K 카드는 10이 카드의 값을 합니다.
- 각각의 카드는 (끗수, 무늬, 값) 튜플로 표현할 수 있습니다.
- 카드 튜플 card에서 card[0]은 카드의 끗수를, card[1]은 무늬를, card[2]는 카드의 값을 의미합니다.
- 처음에 2장의 카드를 받고 카드를 한장 씩 더 받을지 그만 받을지 선택합니다.
- 승리 조건
 - 보유한 모든 카드의 값이 상대방보다 21에 가까우면 게임에서 이깁니다.
 - 보유한 모든 카드의 값이 21보다 크면 게임에서 집니다.



플레이어의 손패

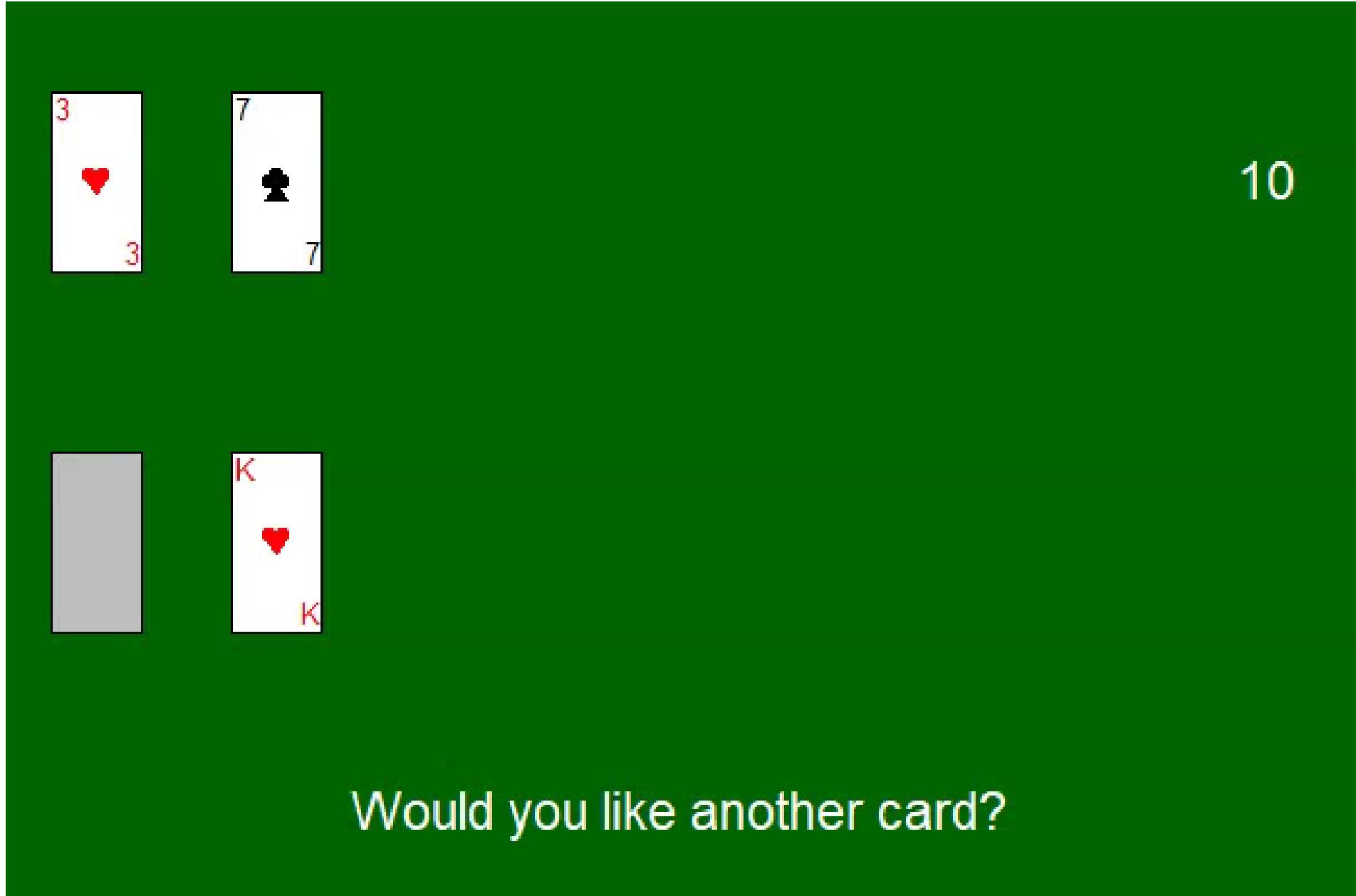


플레이어의 점수

딜러의 손패

사용자의 Y/N 입력을 기다리는 메시지

블랙잭 게임 동영상 예제



튜플로 표현한 카드

자신이 가지고 있는 카드들의 값을 합해 봅시다.

```
def hand_value(hand):  
    total = 0  
    for card in hand:  
        total += card[2]  
    return total
```

카드의 내용을 출력해봅시다.

```
def card_string(card):  
    article = "a "  
    if card[0] in [8, "Ace"]:  
        article = "an "  
    return article + str(card[0]) + " of " + card[1]
```

카드를 튜플로 표현하면 다음과 같은 실수를 할 수 있습니다.

- `card[2]`는 무엇을 의미하나요? `card[1]`과 의미가 헷갈리면 어떡하죠?
- 실수로 ("Ace", "Spades", 5)라는 튜플을 만들면 어떻게 되나요?

객체로 표현한 카드

카드를 표현하기 위한 새로운 객체 타입을 만들어봅시다.
Card 객체는 끊수, 무늬, 값을 속성 (Attribute) 으로 가집니다.

```
class Card(object):  
    """A Blackjack card."""  
    pass
```

```
c = Card()      # Create Card object  
c.face = "Ace" # Set attributes of the card  
c.suit = "Spades"  
c.value = 11
```

c 객체의 형태는 사용자가 정의한 Card 입니다.

```
>>> type(c)  
<class '__main__.Card'>
```

객체로 표현한 카드

자신이 가지고 있는 카드들의 값을 합해봅시다.

```
def hand_value(hand):  
    total = 0  
    for card in hand:  
        total += card.value  
    return total
```

카드의 내용을 출력해봅시다.

```
def card_string(card):  
    article = "a "  
    if card.face in [8, "Ace"]:  
        article = "an "  
    return article+str(card.face)+" of "+card.suit
```

카드를 객체로 표현하면 다음 실수들을 방지할 수 있습니다.

- ~~card[2]~~는 무엇을 의미하나요? ~~card[1]~~과 의미가 헷갈리면 어떡하죠?
- 실수로 (~~"Ace"~~, ~~"Spades"~~, 5)라는 튜플을 만들면 어떻게 되나요?

두 개 이상의 카드

Card 클래스를 이용해서 여러 카드들을 만들 수 있습니다.

```
Card1 = Card()
```

```
card1.face = "Ace"
```

```
card1.suit = "Spades"
```

```
card1.value = 11
```

```
card2 = Card()
```

```
card2.face = 2
```

```
card2.suit = "Clubs"
```

```
card2.value = 2
```

```
...
```

```
>>> print(card_string(card1))
```

```
an Ace of Spades
```

```
>>> print(card_string(card2))
```

```
a 2 of Clubs
```

튜플과 Card 객체의 큰 차이점 중 하나는
튜플은 불변 객체이고, Card 객체는 가변 객체라는 점입니다.

```
c = Card()  
c.face = "Ace"  
c.suit = "Spades"  
c.value = 11  
# ... AND LATER ...  
c.suit = "Hearts"
```

가능? 불가능?

함수는 객체이다

함수 역시 객체입니다.

```
>>> def f(x):
...     return math.sin(x / 3.0 + math.pi/4.0)
>>> print(f)
<function f at 0xb7539a3c>
>>> print(type(f))
<class 'function'>
```

함수의 인자로 함수를 사용할 수 있습니다.

```
def print_table(func, x0, x1, step):
    x = x0
    while x <= x1:
        print(x, func(x))
        x += step
print_table(f, -math.pi, 3 * math.pi, math.pi/8)
```

정리 및 예습

본 강의 학습목표:

- 블랙잭 카드 게임을 프로그램으로 만들기 위해 블랙잭 규칙을 이해할 수 있다.
- 블랙잭 카드 게임에 사용되는 카드를 객체로 표현할 수 있다.

다음 강의 학습 목표:

- 블랙잭 카드 게임에 필요한 자료구조들을 객체를 사용해서 만들 수 있다.
- 블랙잭 카드 게임에 필요한 사용자 인터페이스 프로그래밍을 할 수 있다.

CS101 - 객체 (object)로 블랙잭 카드 게임 만들기 (2/2)

Lecture 22

School of Computing
KAIST

학습 목표:

- 블랙잭 카드 게임에 필요한 자료구조들을 객체를 사용해서 만들 수 있다.
- 블랙잭 카드 게임에 필요한 사용자 인터페이스 프로그래밍을 할 수 있다.

블랙잭 게임은 총 52장으로 이루어진 플레잉 카드를 사용합니다.

각각의 카드는 무늬와 끗수를 가지고 있습니다.

(무늬: 클럽, 스페이드, 하트, 다이아몬드)

(끗수: 2, 3, ..., 10, J, Q, K, A)

```
class Card(object):
```

```
    """A Blackjack card."""
```

```
pass
```

```
card = Card()
```

```
card.face = "Ace"
```

```
card.suit = "Spades"
```

```
card.value = 11
```



카드의 값은 끈수에 의해서 결정되기 때문에, 카드의 값을 의미하는 속성이 따로 있을 필요는 없습니다.

대신 값을 계산해주는 멤버 함수 `value()`를 만들어서 사용합시다.

```
class Card(object):
    """A Blackjack card."""
    def value(self): # method of Card
        if type(self.face) == int:
            return self.face
        elif self.face == "Ace":
            return 11
        else:
            return 10
```

`self` 는 객체의 멤버 함수에서 객체 자신을 가리킵니다.

멤버 함수

Card 객체를 만들어서 사용해 봅시다.

```
>>> card1 = Card()
>>> card1.face = "Ace"
>>> card1.suit = "Spades"
>>> card2 = Card()
>>> card2.face = 2
>>> card2.suit = "Clubs"
>>> card_string(card1)
'an Ace of Spades'
>>> card1.value()
11
>>> card_string(card2)
a 2 of Clubs
>>> card2.value()
2
```

Card 객체를 만드는 과정이 너무 비직관적이네요.

Card(8, "Clubs") 처럼 좀 더 근사하게 객체를 만들고 싶습니다.

card_string() 함수도 Card 객체의 멤버 함수로 만들고 싶어요.

객체는 `__init__`이라는 특별한 멤버 함수를 가지고 있습니다.

이 멤버 함수는 **생성자(Constructor)** 라고 불립니다.

객체가 생성될 때 생성자는 자동으로 호출됩니다.

```
FACES = list(range(2, 11)) +
        ['Jack', 'Queen', 'King', 'Ace']
SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
```

Class Card(object) :

```
"""A Blackjack card."""
def __init__(self, face, suit):
    assert face in FACES and suit in SUITS
    self.face = face
    self.suit = suit
```

이제 Card 객체를 훨씬 간단하게 만들 수 있습니다.

```
hand = [Card("Ace", "Spades"), Card(8, "Diamonds"),
        Card("Jack", "Hearts"), Card(10, "Clubs")]
```

card_string(card) 함수를 Card 객체의 멤버 함수로 만들어 봅시다.

```
class Card(object):
    """A Blackjack card."""
    """Already defined __init__ and value methods"""
    def string(self):
        article = "a "
        if self.face in [8, "Ace"]:
            article = "an "
        return (article + str(self.face) + " of " + self.suit)
```

이제 다음처럼 카드 내용을 출력할 수 있습니다.

```
>>> for card in hand:
...     print(card.string(), "has value", card.value())
```

card_string() 함수를 사용하지 않고도 카드 내용을 문자열로 바꾸는 방법이 있습니다.
str(card) 는 card의 특별한 멤버 함수 `__str__` 를 호출합니다.

```
class Card(object):
    """A Blackjack card."""
    """Already defined __init__ and value methods"""

    def __str__(self):
        article = "a "
        if self.face in [8, "Ace"]:
            article = "an "
        return (article + str(self.face) + " of"
                + self.suit)
```

이제 다음처럼 카드 내용을 출력할 수 있습니다.

```
>>> for card in hand:
...     print(card, "has value", card.value())
```

print 함수는 인자 card 를 `__str__` 를 사용해서 자동으로 문자열 형태로 바꿉니다.

블랙잭 게임에는 52장의 모든 카드를 섞어서 만든 카드 덱(deck)이 필요합니다.
카드 덱을 의미하는 객체를 만들어봅시다.
이 객체에는 덱에서 카드를 한 장 뽑는 역할의 멤버 함수가 필요합니다.

```
class Deck(object):
    """A deck of cards."""
    def __init__(self):
        "Create a deck of 52 cards and shuffle them."
        self.cards = []
        for suit in SUITS:
            for face in FACES:
                self.cards.append(Card(face, suit))
        random.shuffle(self.cards)

    def draw(self):
        """Draw the top card from the deck."""
        return self.cards.pop()
```

```
num_players = 3
num_cards = 5
deck = Deck()
hands = [] # A list of lists (one for each player)
for j in range(num_players):
    hands.append([])
    for i in range(num_cards):
        for j in range(num_players):
            card = deck.draw()
            hands[j].append(card)
            print("Player", j+1, "draws", card)

for j in range(num_players):
    print ("Player %d's hand (value %d):" %
          (j+1, hand_value(hands[j])))
    for card in hands[j]:
        print (" ", card)
```

이제 블랙잭 게임을 만들어 봅시다.

You are dealt a 6 of Hearts

Dealer is dealt a hidden card

You are dealt a 3 of Spades

Dealer is dealt a 9 of Hearts

Your total is 9

Would you like another card? (y/n) y

You are dealt an Ace of Clubs

Your total is 20

Would you like another card? (y/n) n

The dealer's hidden card was a 10 of Spades

The dealer's total is 19

Your total is 20

The dealer's total is 19

You win!

객체에서 비교 연산자 (`==`, `!=`, `<` 등)를 사용하면 예상과 다른 결과가 나올 수 있습니다.

```
>>> Card(8, "Diamonds") == Card(9, "Diamonds")
```

```
False
```

```
>>> Card(8, "Diamonds") == Card(8, "Diamonds")
```

```
False
```

사용자의 생각대로 비교 연산자를 통해 객체를 비교하기 위해서는 다음과 같이 정의를 해야 합니다.

```
class Card(object):  
    """A Blackjack card."  
    """Already defined other methods"  
    def __eq__(self, rhs):  
        return (self.face == rhs.face and  
                self.suit == rhs.suit)  
    def __ne__(self, rhs):  
        return not self == rhs
```

Q __eq__만 정의하고 __ne__ 멤버함수를 정의해주지 않으면?

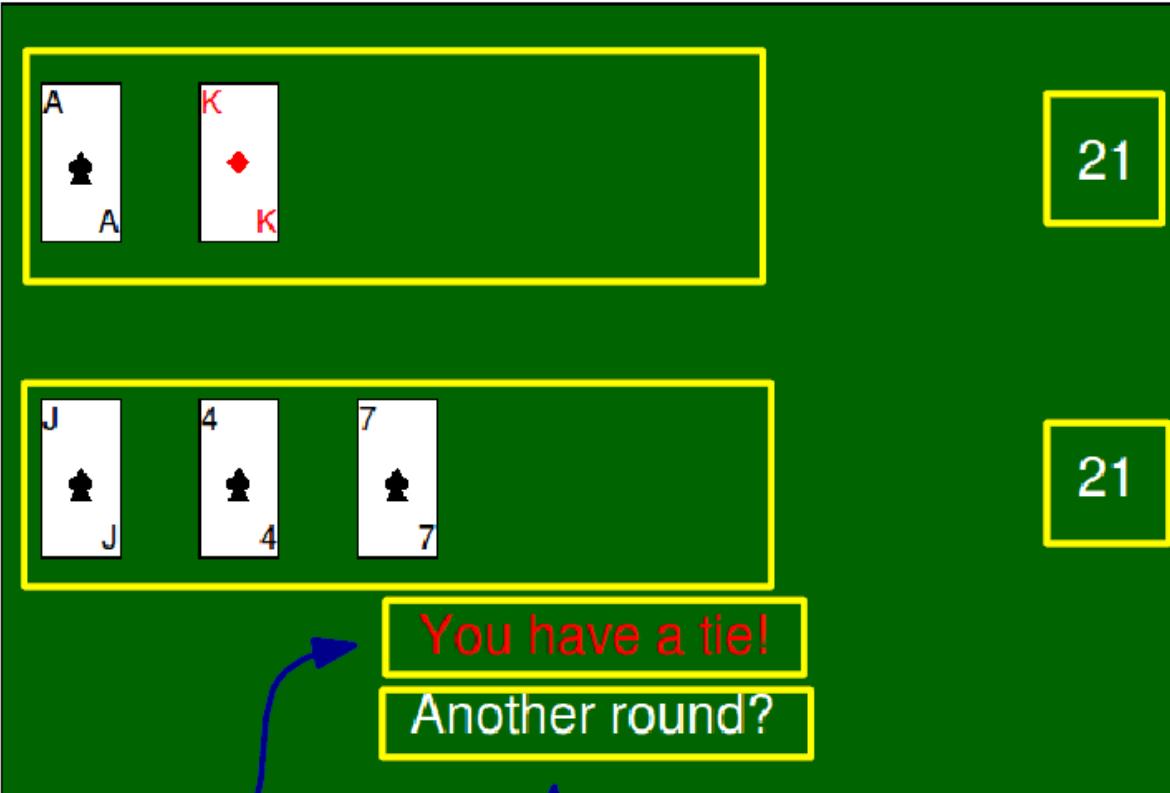
```
>>> Card(8, "Diamonds") == Card(8, "Diamonds")
```

True

```
>>> Card(8, "Diamonds") != Card(8, "Diamonds")
```

True

플레이어의 손패



플레이어의 점수

딜러의 손패

딜러의 점수

메시지

사용자의 Y/N 입력을 기다리는 메시지

Table 객체

Table 객체는 블랙잭 게임 판을 의미하는 객체입니다.

이 객체는 다음과 같은 멤버 함수들을 가지고 있습니다.

- clear() : 게임 초기화
- close() : 창을 닫고 게임을 종료
- set_score(which, text)
: 0,1 중 하나의 값을 가진 which에 해당하는 대상의 점수를 설정
- set_message(text)
- ask(prompt) : 사용자의 입력(y, n)을 받아서 **True** 나 **False** 를 반환

Table 객체는 dealer, player라는 두 개의 속성을 가지고 있습니다.

이 속성들은 각각 플레이어의 패를 의미하는 Hand 객체입니다.

Hand 객체는 다음과 같은 멤버 함수들을 가지고 있습니다.

- clear()
- add(card, hidden = **False**)
- show() : 모든 숨겨진 카드를 공개
- value() : 손에 있는 카드들의 값의 합을 반환

Table.ask(prompt) 함수는 사용자의 입력을 기다려야 합니다.

```
def ask(self, prompt):
    self.question.setMessage(prompt)
    while True:
        e = self.canvas.wait()
        d = e.getDescription()
        if d == "canvas close":
            sys.exit(1)
        if d == "keyboard":
            key = e.getKey()
            if key == 'y':
                return True
            if key == 'n':
                return False
```

그래픽 유저 인터페이스(GUI)를 사용하는 프로그램은 여러 종류의 이벤트를 기반으로 작동합니다. 이런 프로그램은 이벤트를 기다리고, 호출된 이벤트에 따라 일을 실행합니다.

다음과 같은 이벤트가 있을 수 있습니다.

- 키 입력
- 윈도우 창의 최소화/최대화/종료
- 마우스 버튼 클릭
- 마우스 커서가 창 안에 들어옴/나감

이벤트 기반 프로그래밍은 순차적으로 실행되지만 하는 프로그램을 개발하는 것이 아닌, 여러 이벤트들에 의한 함수 호출이 필요한 프로그램을 개발하는 것을 의미합니다.

정리 및 예습

본 강의 학습목표:

- 블랙잭 카드 게임에 필요한 자료구조들을 객체를 사용해서 만들 수 있다.
- 블랙잭 카드 게임에 필요한 사용자 인터페이스 프로그래밍을 할 수 있다.

다음 강의 학습 목표:

- 지금까지 강의에서 사용한 객체의 상태와 동작을 이해할 수 있다.
- 닭과 여러 병아리들이 움직이는 애니메이션을 객체를 사용해 만들 수 있다.

CS101 - 객체 (object)로 애니메이션 만들기

Lecture 23

School of Computing
KAIST

학습 목표:

- 지금까지 강의에서 사용한 객체의 상태와 동작을 이해할 수 있다.
- 닭과 여러 병아리들이 움직이는 애니메이션을 객체를 사용해 만들 수 있다.

객체: 상태와 동작

지금까지 우리는 여러 종류의 객체를 사용했습니다.

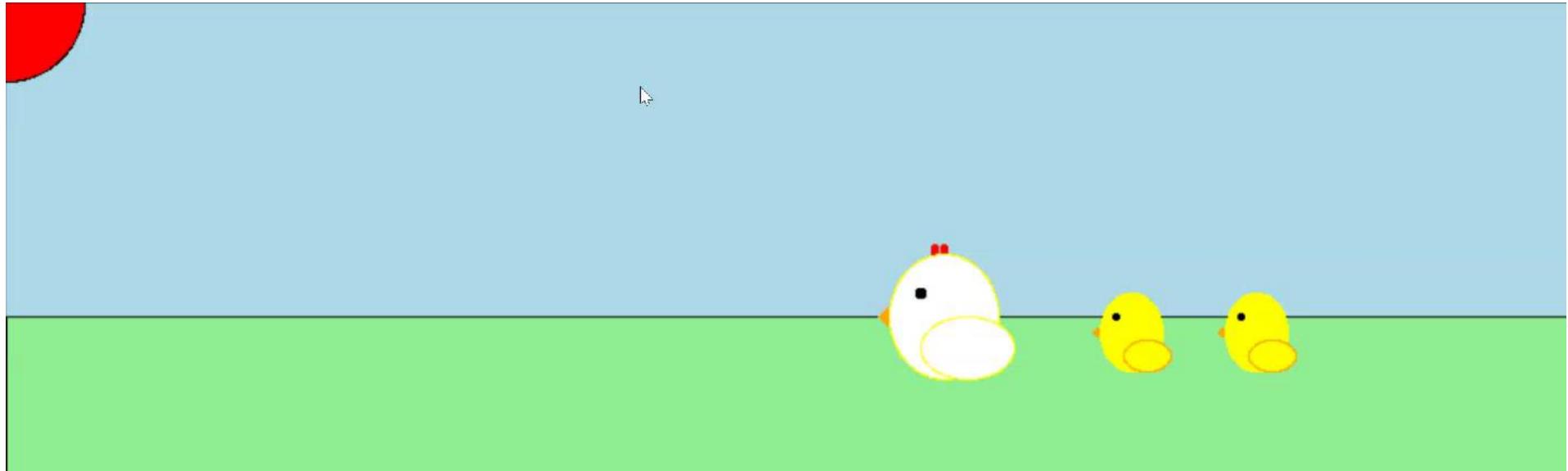
: 튜플, 문자열, 로봇, 이미지, 원이나 사각형 등의 그래픽 객체

객체는 **상태(State)**를 가지고 있고, **동작(Action)**을 수행할 수 있습니다.

Robot: 로봇의 상태는 로봇의 위치, 방향, 들고 있는 비퍼의 개수 정보를 가지고 있습니다. 로봇은 움직임, 회전, 비퍼 줍기/들어올리기, 그리고 로봇의 상태를 파악하는 동작이 가능합니다.

Circle: 반지름, 위치, 깊이, 외곽선 색, 채워진 색 정보를 상태로 가지고 있습니다. 색 변경, 크기 변경, 위치 변경 등의 동작이 가능합니다.

Picture: 사진의 가로 길이, 세로 길이, 각 픽셀의 색 정보를 상태로 가지고 있습니다. 각 픽셀의 색을 읽거나, 변경하는 동작이 가능합니다.



위 애니메이션은 2010년도 신입생인 유정은, 송금현 학생의 작품입니다.

이 애니메이션은 3개의 레이어 객체를 사용하고 있습니다 (닭, 병아리1, 병아리2).
닭과 병아리는 모두 몸통, 날개, 눈, 부리를 가지고 있습니다.
닭은 벼슬을 가지고 있지만, 병아리는 벼슬이 없습니다.

닭과 병아리의 생김새는 거의 비슷합니다.

닭과 병아리의 차이점은 닭이 더 크고, 하얀색이고, 벼슬을 가지고 있다는 점입니다.

비슷한 객체들을 간단하게 만드는 방법 중 하나는
코드를 작성하고 그 코드를 복사해서 여러 번 붙여 넣는 것입니다.

이 방법은 큰 단점을 가지고 있습니다

: 코드에 버그가 있어서 고쳐야 할 때, 복사했던 모든 코드 역시 고쳐야 합니다.
동일한 코드가 어디에 있는지 파악하기 위해서는 시간과 노력을 많이 기울여야 합니다.

닭을 하나의 클래스로 만들어봅시다.

```
class Chicken(object):
    """Graphic representation of a chicken."""
    pass
```

Chicken 객체는 레이어, 몸통, 날개, 눈, 부리를 속성으로 가지게 됩니다.

함수 make_chicken() 은 (0,0) 위치에 Chicken 객체를 만듭니다.

```
def make_chicken(hen = False):
    layer = Layer()
    if hen:
        body = Ellipse(70, 80)
        body.setFillColor("white")
    else:
        body = Ellipse(40, 50)
        body.setFillColor("yellow")
        body.move(0, 10)
        body.setBorderColor("yellow")
    body.setDepth(20)
    layer.add(body)
    # similar for wing, eye, beak, dots
```

함수 `make_chicken()`은 (0,0) 위치에 Chicken 객체를 만듭니다.
객체의 생성이 끝나면 해당 객체를 반환합니다.

```
def make_chicken(hen = False):
    # ... see previous page

    ch = Chicken()
    ch.layer = layer
    ch.body = body
    ch.wing = wing
    ch.eye = eye

    # return the Chicken object
    return ch
```

Chicken 객체는 객체의 속성을 (예. chick1.layer) 이용해서 움직일 수 있습니다.

```
hen = make_chicken(True)
chick1 = make_chicken()
chick1.layer.move(120, 0)
```

```
herd = Layer()
herd.add(hen.layer)
herd.add(chick1.layer)
herd.move(600, 200)
```

```
chick2 = make_chicken()
chick2.layer.move(800, 200)
```

정리 및 예습

본 강의 학습목표:

- 지금까지 강의에서 사용한 객체의 상태와 동작을 이해할 수 있다.
- 닭과 여러 병아리들이 움직이는 애니메이션을 객체를 사용해 만들 수 있다.

다음 강의 학습 목표:

- 프로그램 실행 속도에 영향을 주는 해석기와 (interpreter) 컴파일러의 차이를 이해할 수 있다.
- 프로그램 실행 속도에 보다 큰 영향을 미치는 좋은 알고리즘의 중요성을 이해할 수 있다.

CS101 - 프로그램 실행 속도 향상 방법

Lecture 24

School of Computing
KAIST

학습 목표:

- 프로그램 실행 속도에 영향을 주는 해석기와 (interpreter) 컴파일러의 차이를 이해할 수 있다.
- 프로그램 실행 속도에 보다 큰 영향을 미치는 좋은 알고리즘의 중요성을 이해할 수 있다.

왜 포토샵이 우리가 cs1media 모듈로 만든 그래픽 프로그램보다 빠를까요?

컴퓨터는 Python 언어를 직접적으로 이해할 수 없습니다.

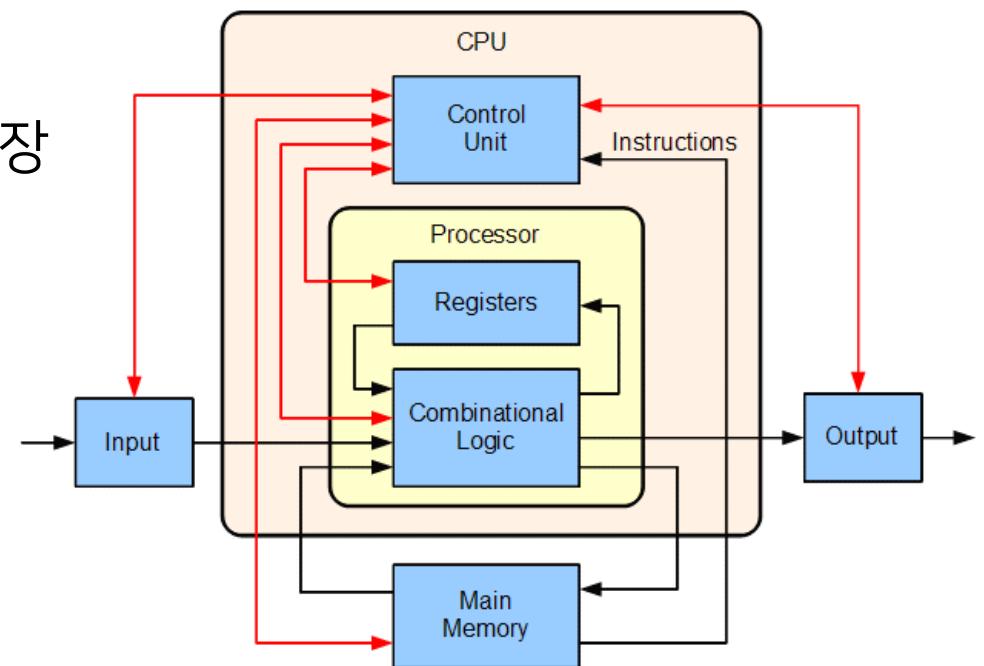
컴퓨터는 기계어라는 단 한 가지 언어만을 직접적으로 이해할 수 있습니다.
이 기계어는 CPU 종류마다 다르게 사용됩니다.

기계어는 숫자들의 나열로 이루어집니다.

21 37 158 228 255 10 49 26 88 250 12 ...

각 숫자는 의미를 가지고 있습니다.

- 저장소의 값을 읽어 CPU 레지스터에 저장
- 두 레지스터의 값을 더함
- 레지스터의 값을 저장소에 저장
- 두 숫자를 비교
- 다른 저장소 주소로 이동



기계어로 된 명령들은 굉장히 빠르게 실행됩니다.

2GHz 프로세서는 1초에 무려 2,000,000,000개의 명령을 실행할 수 있습니다.

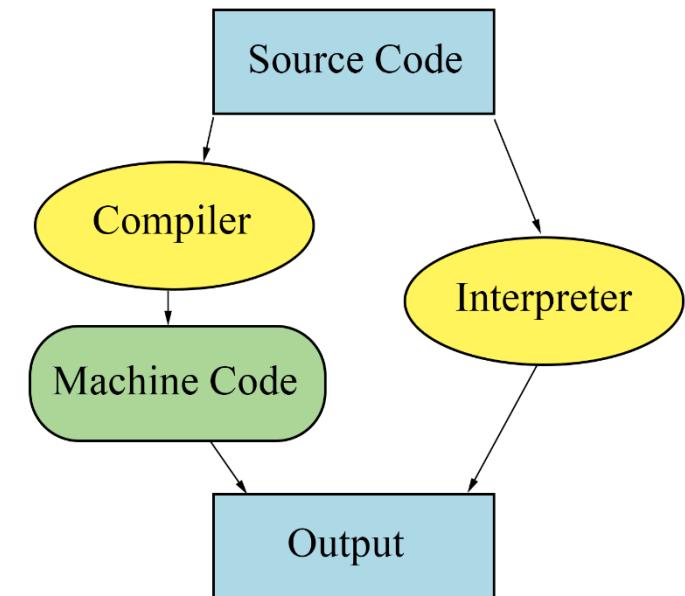
하지만, 기계어 명령들은 굉장히 알아보기 힘들게 되어 있어 기계어를 직접 다뤄서 프로그래밍하는 경우는 거의 없습니다.

Python은 **해석기(Interpreter)**를 사용합니다.

해석기는 Python 코드를 읽어서 명령을 하나씩 기계어로 바꿔 실행시켜주는 프로그램입니다.

해석기를 사용하는 프로그래밍 언어는 Scheme, Matlab, Flash 등이 있습니다.

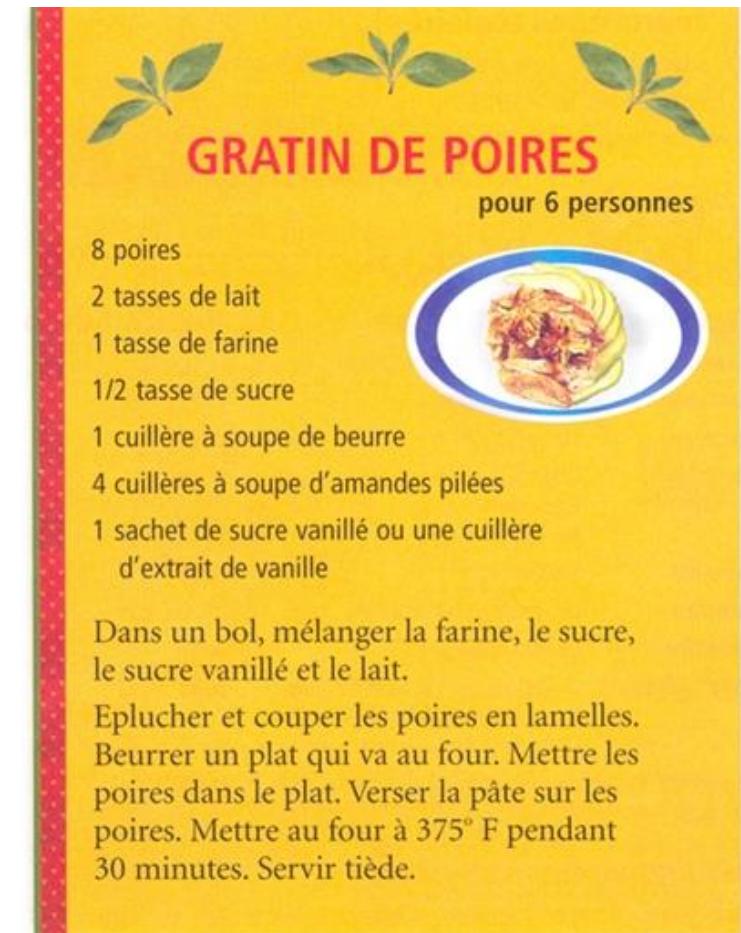
C, C++, Java, Fortran은 **컴파일러(Compiler)**를 사용합니다. 컴파일러는 프로그램 코드를 읽어서 기계어 명령들로 된 프로그램으로 바꿔주는 프로그램입니다.



해석기를 사용하는 것은, 외국어로 된 요리 책을 보면서 요리를 한 단계씩 만드는 것과 비슷합니다. 책을 조금씩 번역하며 요리를 만들기 때문에 요리를 천천히 만들게 됩니다.

비슷한 요리를 여러 차례 만들어야 할 때는,
요리 레시피를 모국어로 번역해서 사용하는 것이
더 효율적입니다. 컴파일러가 이런 일을 합니다.

레시피를 한 번에 모두 번역하는 데는 시간이
필요합니다. 한 번만 만들어야 하는 요리는
이렇게 하는게 더 비효율적일 수 있습니다. 하지만,
번역이 끝나면 같은 요리를 빠르게 여러 번
만들 수 있습니다.



Python 해석기는 사용자의 명령과 상호작용할 수 있습니다.

사용자는 명령을 하나씩 실행하며 쉽게 테스트를 해볼 수 있습니다.

사용자는 현재 사용되는 객체에 어떤 값이 들어있는지 쉽게 알 수 있습니다.

객체가 가진 값을 수학적으로 분석할 수도 있습니다.

(C++에서 비슷한 일을 하려면 더 어려운 방법을 써야 합니다)

해석기를 사용한 프로그램은 프로그램을 쉽게 바꾸고 실행할 수 있어 디버깅을 더 쉽게 할 수 있습니다.

해석기는 프로그램의 메모리도 자동으로 관리합니다.

좋은 알고리즘

왜 포토샵 프로그램이 우리가 cs1media 모듈로 만든 그래픽 프로그램보다 빠를까요?

- 포토샵은 C++로 작성되어 기계어로 컴파일된 프로그램입니다.
- 포토샵은 더 좋은 알고리즘을 사용합니다.

해석기를 사용하는 프로그램도 좋은 알고리즘을 사용하면 컴파일러를 사용하는 프로그램보다 속도가 더 빠를 수 있습니다.



예제: 정렬

다음은 리스트 a를 정렬하는 간단한 알고리즘입니다.

```
def simple_sort(a):
    for i in range(len(a) - 1):
        for j in range(len(a) - 1):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
```

리스트 a가 n개의 원소를 가지고 있으면 **if** 문은 총 $(n - 1)^2$ 번 호출됩니다.

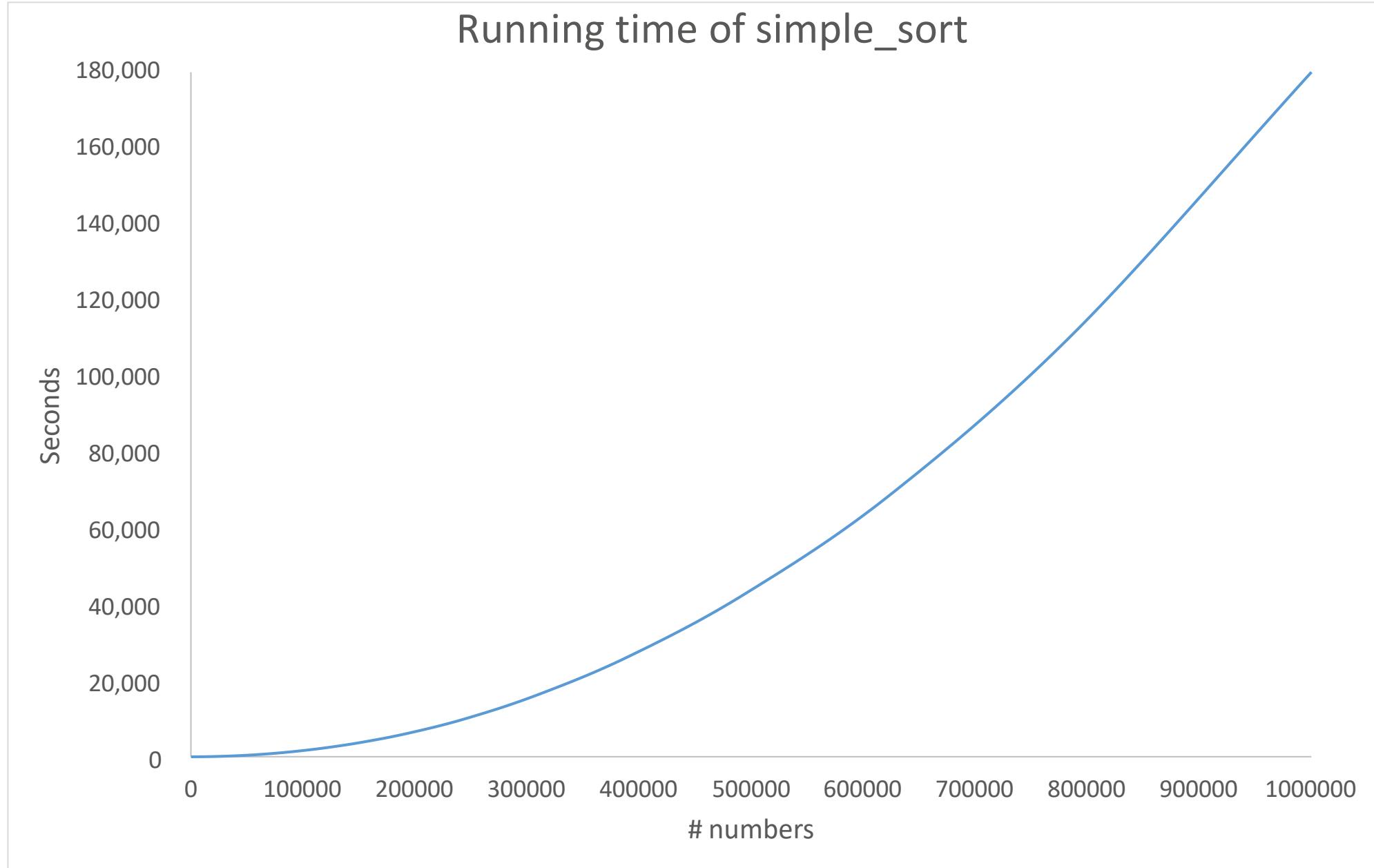
프로그램의 실행 시간을 측정해봅시다.

```
import random
import time

large_list = list(range(200000))
random.shuffle(large_list)
st = time.time()
simple_sort(large_list)
print("Running time: %f sec" % (time.time() - st))
```

예제: 정렬

아래 그래프는 위 프로그램을 Intel i7 3.60GHz 데스크탑에서 실행한 그래프입니다.
1백만 개의 원소를 가진 리스트를 정렬하는 데 50시간 (180,055초) 걸렸습니다.



하나의 리스트는 한 개의 원소를 가지는 조각들로 나눌 수 있습니다.

그 후, 리스트 전체가 정렬될 때까지 두 조각씩 합쳐서 정렬된 한 조각으로 만듭니다.

정렬된 두 리스트 a , b 를 합치는 것은 쉽습니다
(a , b 의 원소를 하나씩 순차적으로 비교하면 됩니다)

이 알고리즘은 두 원소의 값을 최대 $n \log_2 n$ 번 비교합니다.

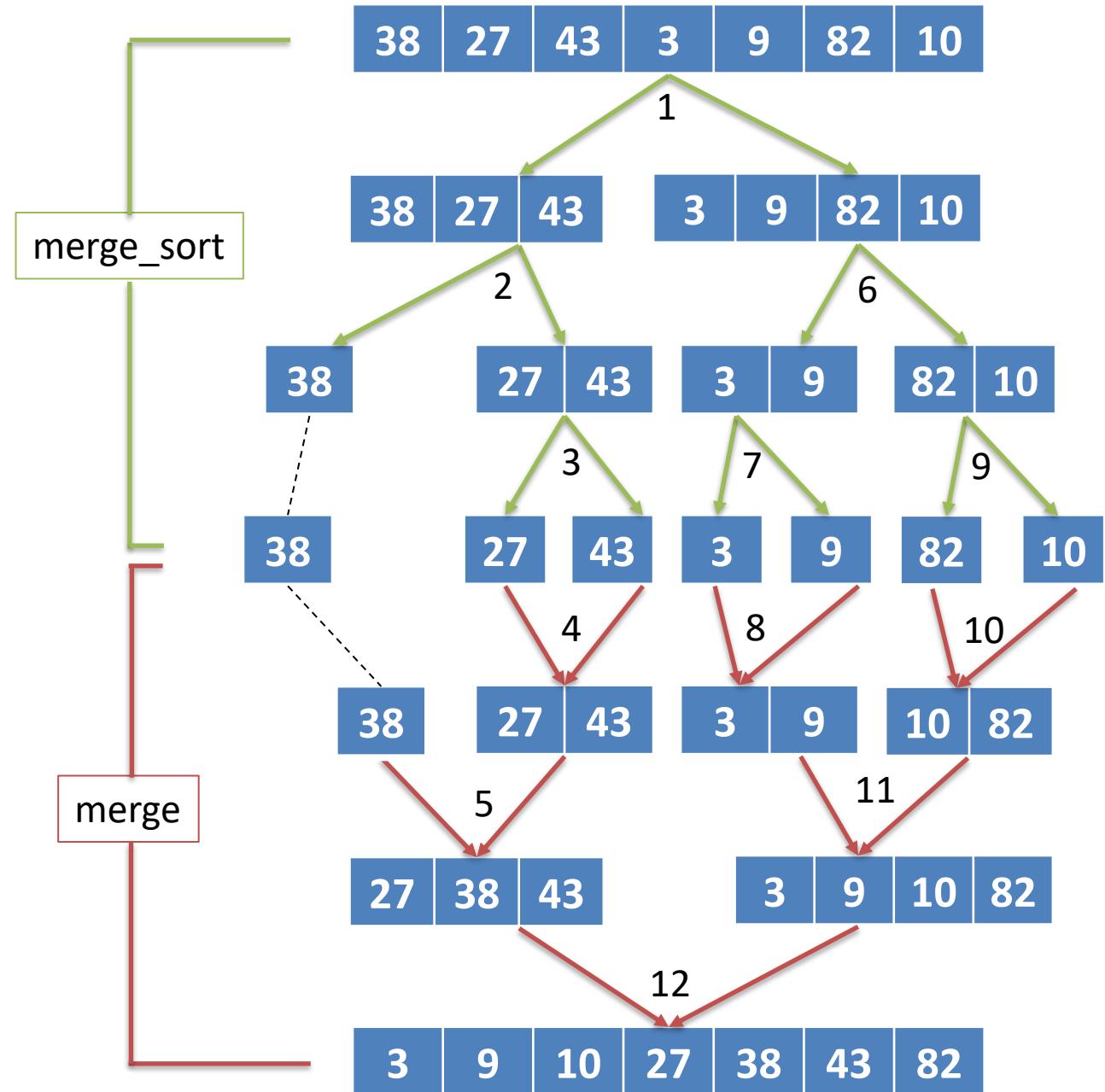
프로그램의 실행 시간을 측정해봅시다.

```
import random
import time

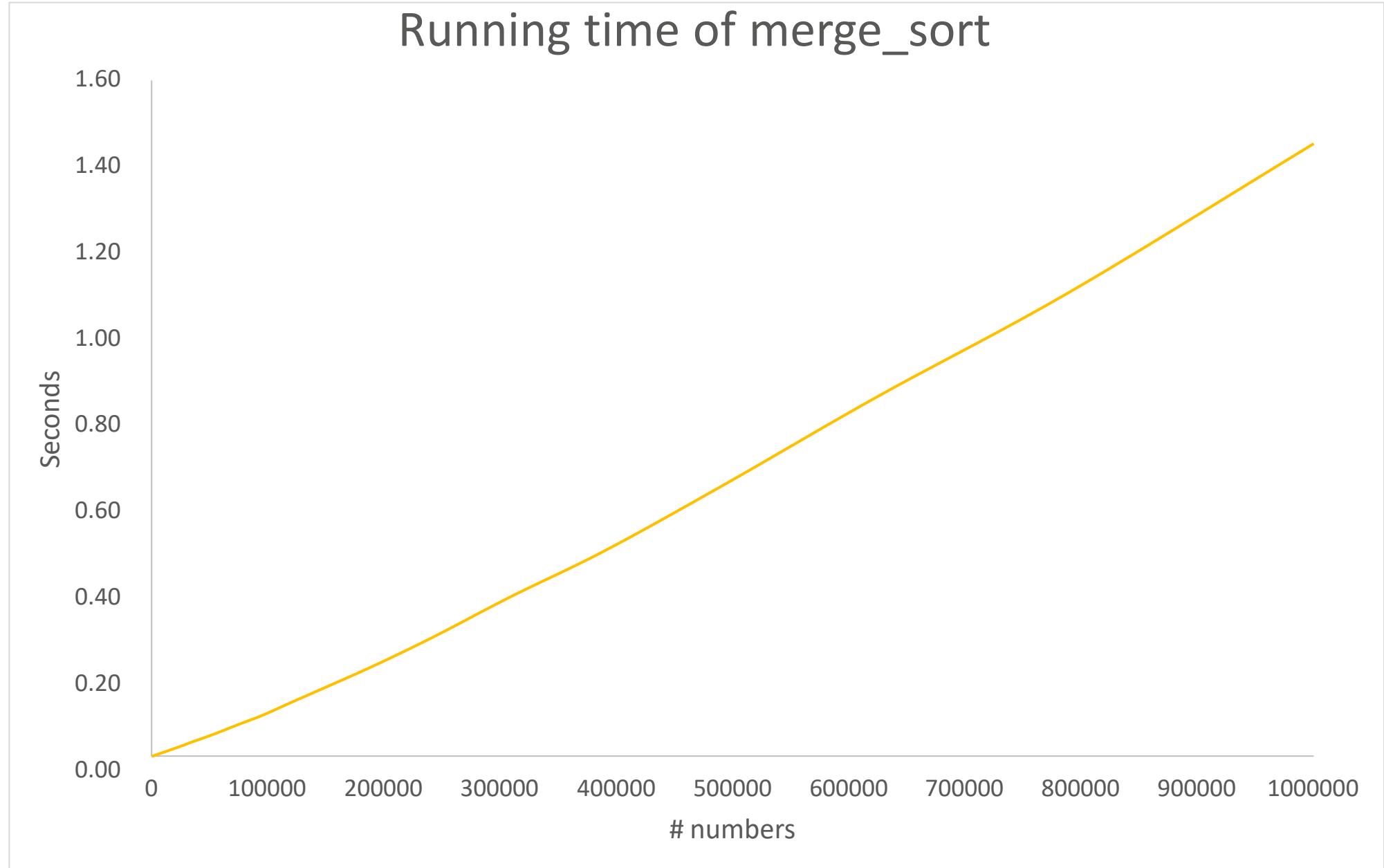
large_list = list(range(200000))
random.shuffle(large_list)
st = time.time()
merge_sort(large_list)
print("Running time: %f sec" % (time.time() - st))
```

병합 정렬은 재귀적이다

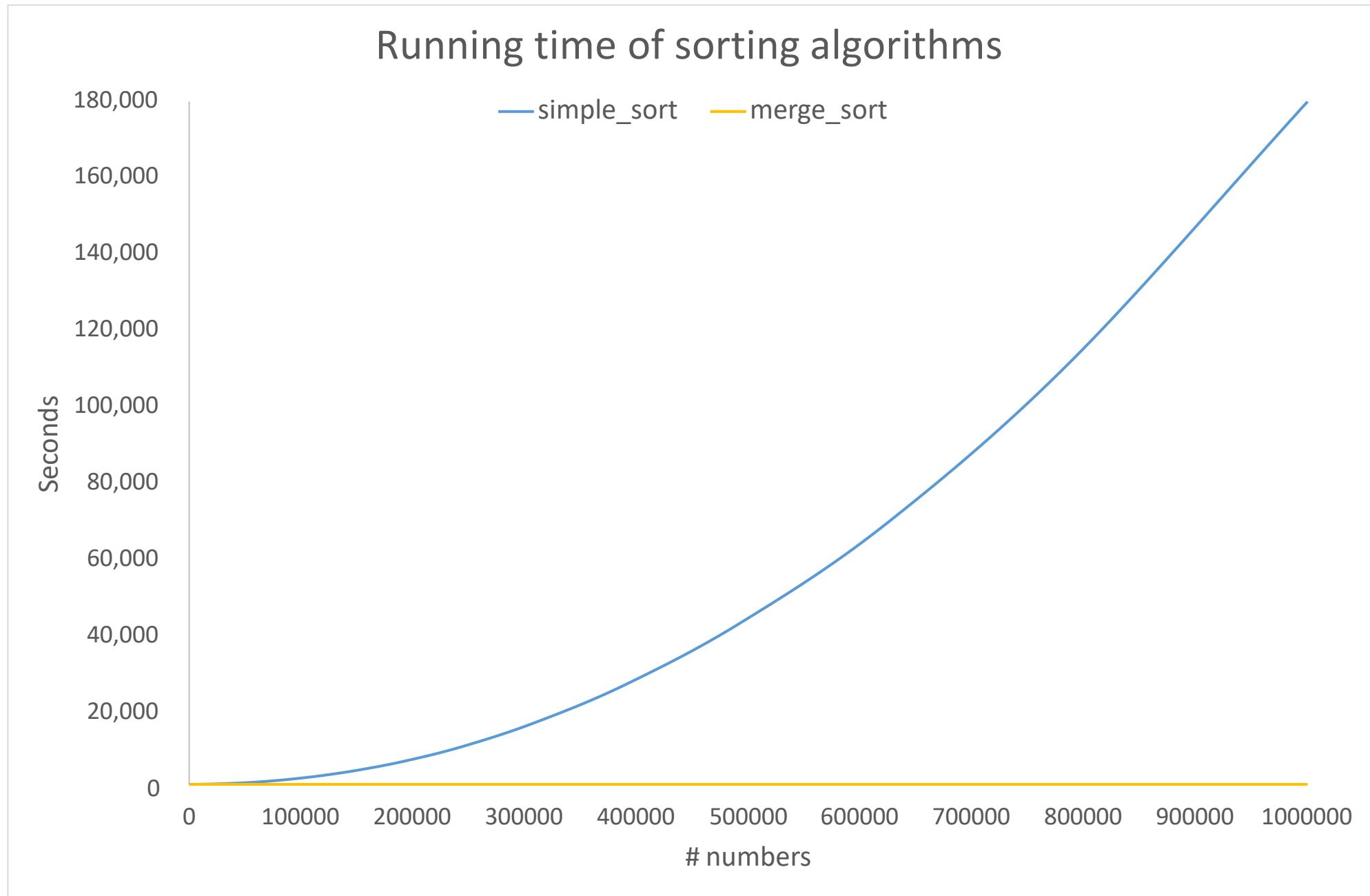
```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



아래 그래프는 위 프로그램을 Intel i7 3.60GHz 데스크탑에서 실행한 그래프입니다.
1백만 개의 원소를 가진 리스트를 정렬하는 데 1.45초가 걸렸습니다.



simple_sort와 merge_sort의 실행 시간을 비교해봅시다.



문제를 해결하기 위한 더 효율적인 알고리즘을 만드는 것은 전산학의 중요한 분야입니다.

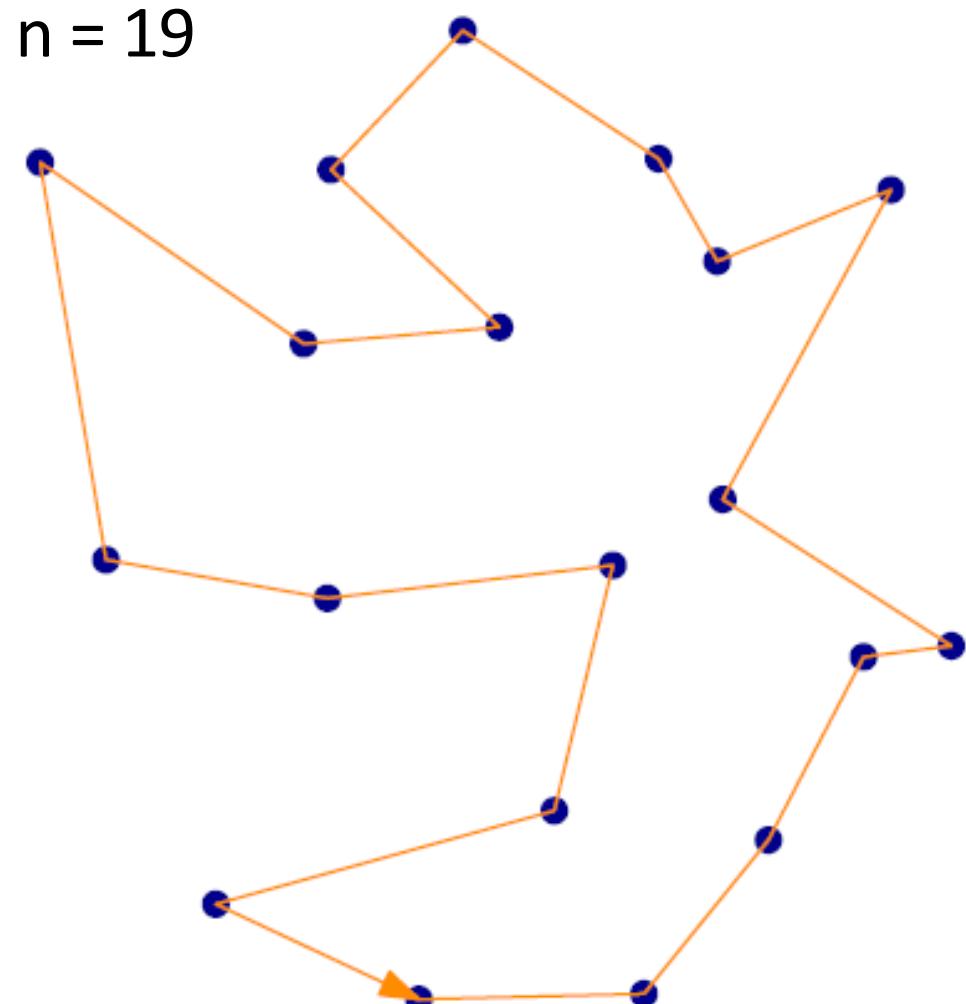
알고리즘이 사용하는 연산의 최대 횟수는 입력 값에 대한 사이즈 n 의 함수 형태로 나타낼 수 있습니다. 이 함수가 더 작을 때, 알고리즘이 더 **효율적(Efficient)**이라고 말합니다.

어떤 알고리즘이보다 더 적은 연산으로 문제를 해결할 수 없다는 것을 증명할 수 있으면, 그 알고리즘을 **최적(Optimal)**의 알고리즘이라 부릅니다.

n 개의 숫자는 $n \log_2 n$ 보다 적은 비교 횟수로 정렬할 수 없다는 것을 증명할 수 있습니다. 따라서 병합 정렬은 최적의 알고리즘입니다.

외판원 문제 :

평면상에 n 개의 점이 있을 때, 모든 점을 방문하는 가장 짧은 경로를 구하시오.



이 문제를 해결하는 지금까지 알려진 가장 효율적인 알고리즘은 2^n 개의 연산을 사용합니다.

하지만, 더 빠르게 n^2 개의 연산으로 문제를 해결할 수 없다는 것은 증명하지 못했습니다.

밀레니엄 문제:

$$P = NP?$$

문제를 해결하는 알고리즘이 없다는 것을 증명 할 수 없는 문제들도 있습니다.

정리 및 예습

본 강의 학습목표:

- 프로그램 실행 속도에 영향을 주는 해석기와 (interpreter) 컴파일러의 차이를 이해할 수 있다.
- 프로그램 실행 속도에 보다 큰 영향을 미치는 좋은 알고리즘의 중요성을 이해할 수 있다.

다음 강의 학습 목표:

- 반복문과 유사한 재귀 (recursive) 함수를 이해하고 활용할 수 있다.
- CS101 강의에서 배운 모든 내용을 정리할 수 있다.

CS101 – 재귀 (recursive) 함수 및 강의 마무리

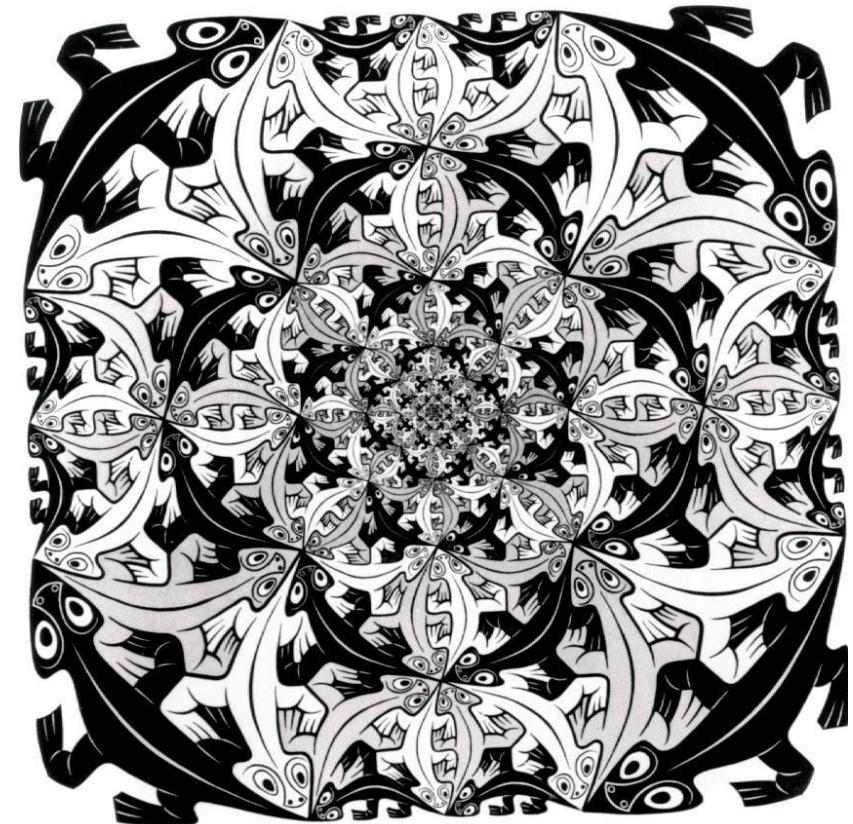
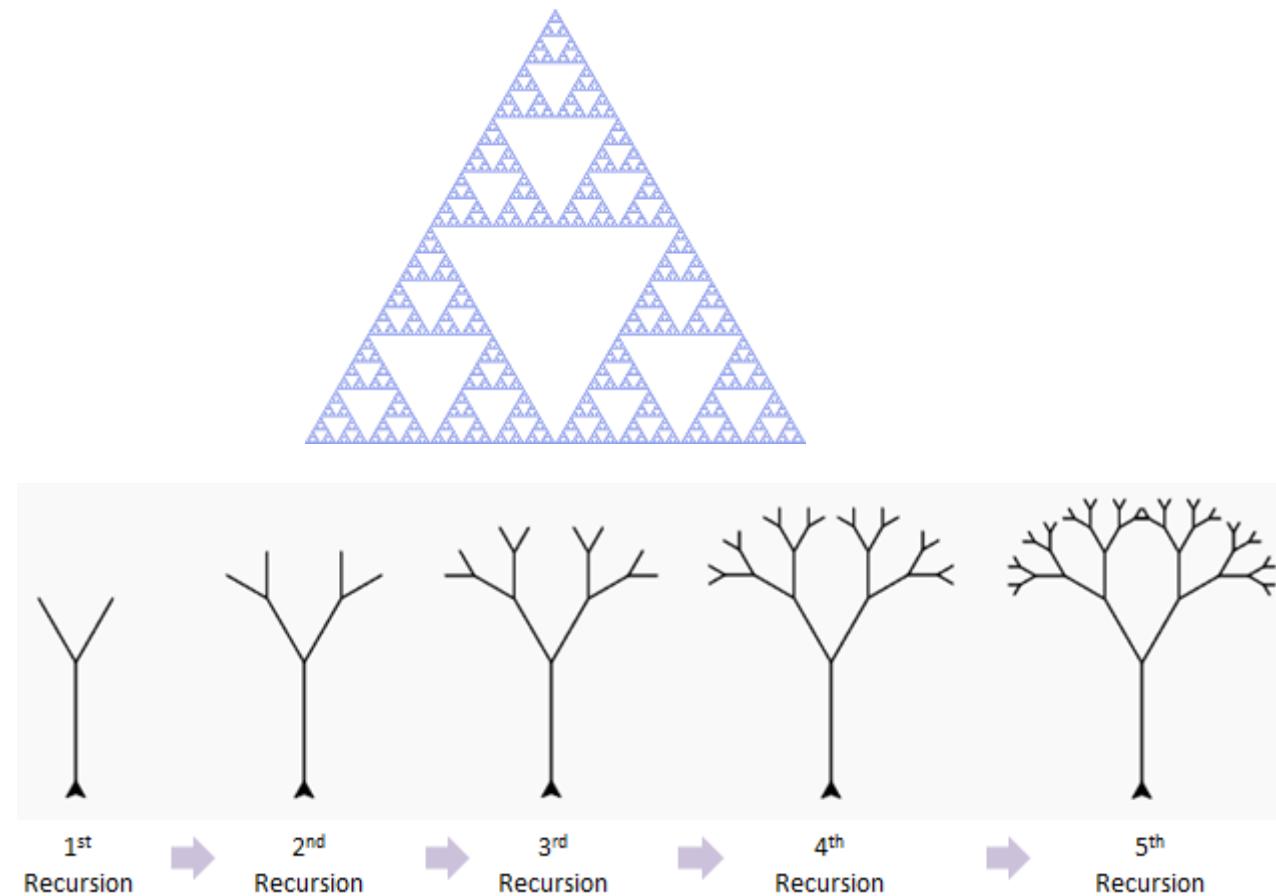
Lecture 25

School of Computing
KAIST

학습 목표:

- 반복문과 유사한 재귀 (recursive) 함수를 이해하고 활용할 수 있다.
- CS101 강의에서 배운 모든 내용을 정리할 수 있다.

재귀(Recursion)은 자기 자신을 이용해서 자신을 정의하는 것을 의미합니다.
(예. 폴더는 파일과 폴더들로 구성되어 있습니다)
(예. 사전의 단어들은 다른 단어들로 정의되어 있습니다)



팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

6!

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! \\ = (6 * 5!) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \\ &= (6 * (5 * (4 * (3 * 2)))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \\ &= (6 * (5 * (4 * (3 * 2)))) \\ &= (6 * (5 * (4 * 6))) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \\ &= (6 * (5 * (4 * (3 * 2)))) \\ &= (6 * (5 * (4 * 6))) \\ &= (6 * (5 * 24)) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \\ &= (6 * (5 * (4 * (3 * 2)))) \\ &= (6 * (5 * (4 * 6))) \\ &= (6 * (5 * 24)) \\ &= (6 * 120) \end{aligned}$$

팩토리얼

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} 6! &= (6 * 5!) \\ &= (6 * (5 * 4!)) \\ &= (6 * (5 * (4 * 3!))) \\ &= (6 * (5 * (4 * (3 * 2!)))) \\ &= (6 * (5 * (4 * (3 * (2 * 1!)))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 0!))))))) \\ &= (6 * (5 * (4 * (3 * (2 * (1 * 1))))))) \\ &= (6 * (5 * (4 * (3 * (2 * 1)))))) \\ &= (6 * (5 * (4 * (3 * 2)))) \\ &= (6 * (5 * (4 * 6))) \\ &= (6 * (5 * 24)) \\ &= (6 * 120) \\ &= 720 \end{aligned}$$

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

```
downup("Hello")
```

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

```
downup("Hello")
```

```
print("Hello")
```

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

downup("Hello")

print("Hello")

Hello

downup (w [:-1])

downup("Hell")

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

```
downup("Hello")
```

```
print("Hello")
```

Hello

```
downup("Hell")
```

```
print("Hell")
```

Hell

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

```
downup("Hello")
```

```
print("Hello")
```

Hello

```
downup("Hell")
```

```
print("Hell")
```

Hell

```
downup("Hel")
```

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

```
downup("Hello")
```

```
print("Hello")
```

Hello

```
downup("Hell")
```

```
print("Hell")
```

Hell

```
downup("Hel")
```

```
print("Hel")
```

Hel

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :
02:     print (w)
03:     if len (w) <= 1 :
04:         return
05:     # Recursive call
06:     downup (w [ :-1 ] )
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

>>> downup ("Hello")

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

다음 함수는 어떻게 만들 수 있을까요?

>>> downup ("Hello")

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

print("H")

H

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :  
02:     print (w)  
03:     if len (w) <= 1 :  
04:         return  
05:     # Recursive call  
06:     downup (w [ :-1 ] )  
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

print("H")

H

len <= 1



다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :
02:     print (w)
03:     if len (w) <= 1 :
04:         return
05:     # Recursive call
06:     downup (w [ :-1 ] )
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

print("H")

H

print("He")

He

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :
02:     print (w)
03:     if len (w) <= 1 :
04:         return
05:     # Recursive call
06:     downup (w [ :-1 ] )
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

print("H")

H

print("He")

print("Hel")

He

Hel

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :
02:     print (w)
03:     if len (w) <= 1 :
04:         return
05:     # Recursive call
06:     downup (w [ :-1 ] )
07:     print (w)
```

downup("Hello")

print("Hello")

Hello

downup("Hell")

print("Hell")

Hell

downup("Hel")

print("Hel")

Hel

downup("He")

print("He")

He

downup("H")

print("H")

H

print("He")

print("Hel")

Hel

print("Hell")

Hell

재귀 함수

다음 함수는 어떻게 만들 수 있을까요?

```
>>> downup ("Hello")
```

Hello

Hell

Hel

He

H

He

Hel

Hell

Hello

아래와 같은 방법이 있을 수 있습니다.

```
01: def downup (w) :
02:     print (w)
03:     if len (w) <= 1 :
04:         return
05:     # Recursive call
06:     downup (w [ :-1 ] )
07:     print (w)
```

downup("Hello")

└ print("Hello")

Hello

└ downup("Hell")

Hell

└ print("Hell")

Hel

└ downup("Hel")

He

└ print("Hel")

He

└ downup("He")

H

└ print("He")

He

└ print("Hel")

Hel

└ print("Hell")

Hello

└ print("Hello")

b진법으로 변환하기

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$

(즉, `to_radix(675, 8)`은

"1243"을 반환 합니다)

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	

b진법으로 변환하기

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n 을 b 진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
 (즉, `to_radix(675, 8)`은
 "1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	

b진법으로 변환하기

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	
3 rd radix	10 (=84 //8)	8	

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	
3 rd radix	10 (=84 //8)	8	
4 th radix	1 (=10//8)	8	

b진법으로 변환하기

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	
3 rd radix	10 (=84 //8)	8	
4 th radix	1 (=10//8)	8	"1"

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	
3 rd radix	10 (=84 //8)	8	"1" + "2" = "12"
4 th radix	1 (=10//8)	8	"1"

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	
2 nd radix	84 (=675//8)	8	"12" + "4" = "124"
3 rd radix	10 (=84 //8)	8	"1" + "2" = "12"
4 th radix	1 (=10//8)	8	"1"

10진법 숫자를 2진법으로 어떻게 변환하나요? 8진법으로는요?

n을 b진법으로 바꿀 때 마지막 자릿수는 다음처럼 얻을 수 있습니다: $n \% b$

나머지 숫자는 $n // b$ 로 얻을 수 있습니다.

```
def to_radix(n, b):
    if n < b:
        return str(n)
    s = to_radix(n // b, b)
    return s + str(n % b)
```

예제: $675 = 1243_8$
(즉, `to_radix(675, 8)`은
"1243"을 반환 합니다)

	Input		Return value
	n	b	
1 st radix	675	8	"124" + "3" = "1243"
2 nd radix	84 (=675//8)	8	"12" + "4" = "124"
3 rd radix	10 (=84 //8)	8	"1" + "2" = "12"
4 th radix	1 (=10//8)	8	"1"

병합 정렬은 재귀적이다

병합 정렬은 재귀 함수를 사용하는 알고리즘입니다.
이 알고리즘은 2개의 재귀 호출을 사용합니다.

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```

분할 정복(Divide & Conquer) : 하나의 문제를 여러 작은 문제들로 분할합니다. 작은 문제들을 풀고, 그 결과들을 합쳐서 전체 문제의 답을 찾습니다.

병합 정렬은 재귀적이다

```
def merge(left, right):
    result = []    # 합쳐진 정렬된 리스트
    i, j = 0, 0

    # left, right 두 리스트에서 가장 작은 숫자를 result에 넣는다.
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # left와 right 리스트에 남은 숫자를 모두 result에 넣는다.
    result += left[i:]
    result += right[j:]

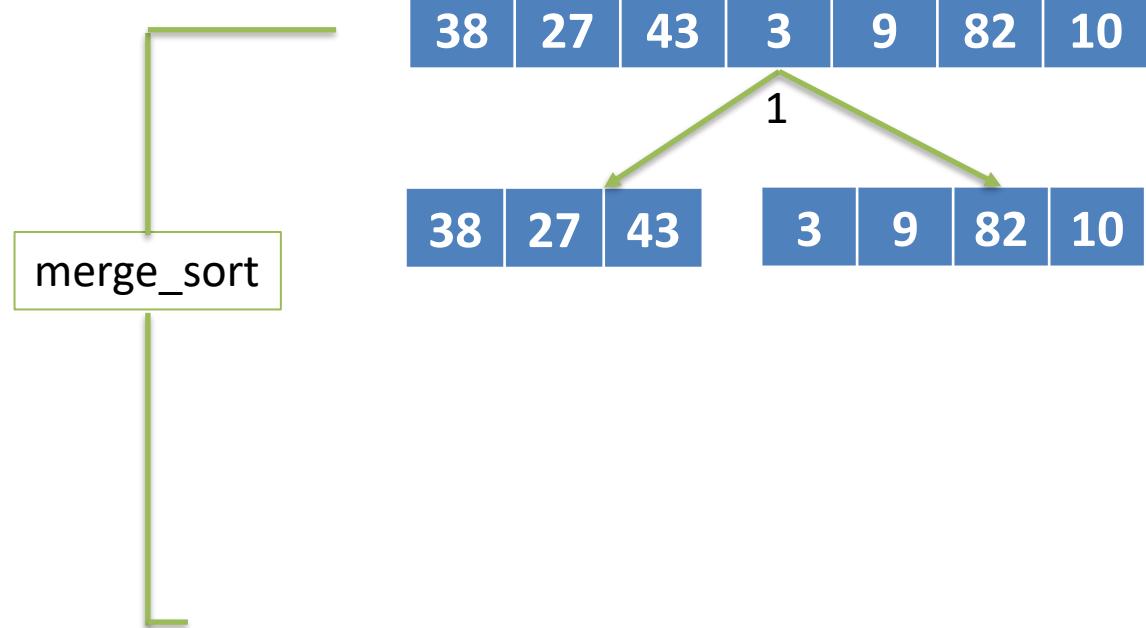
    return result
```

38	27	43	3	9	82	10
----	----	----	---	---	----	----

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```

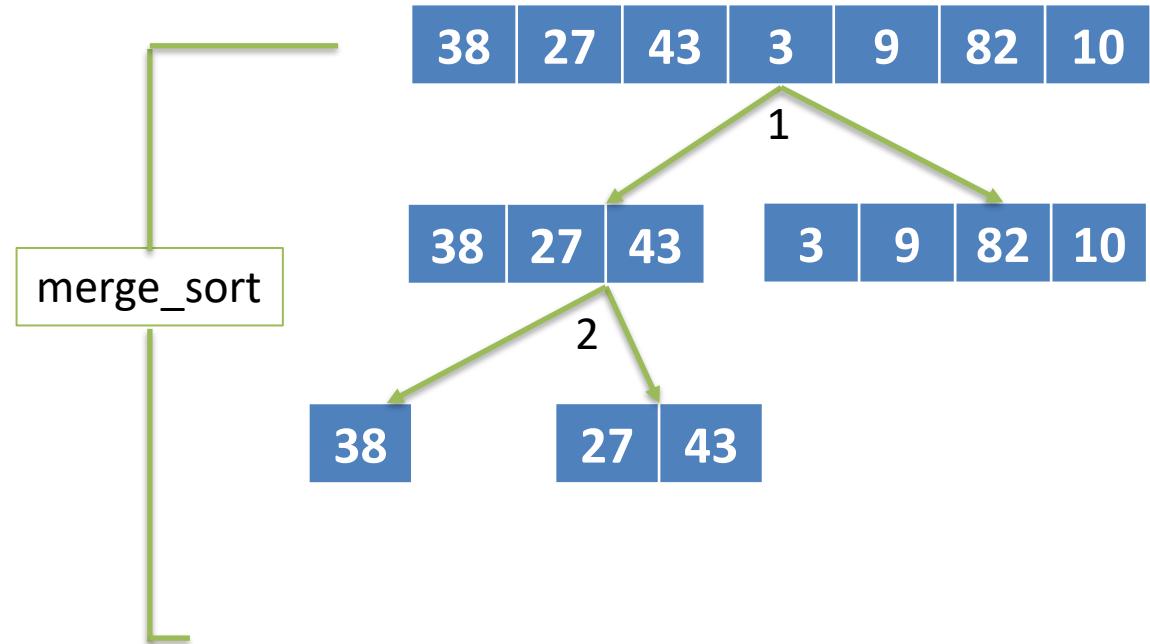
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



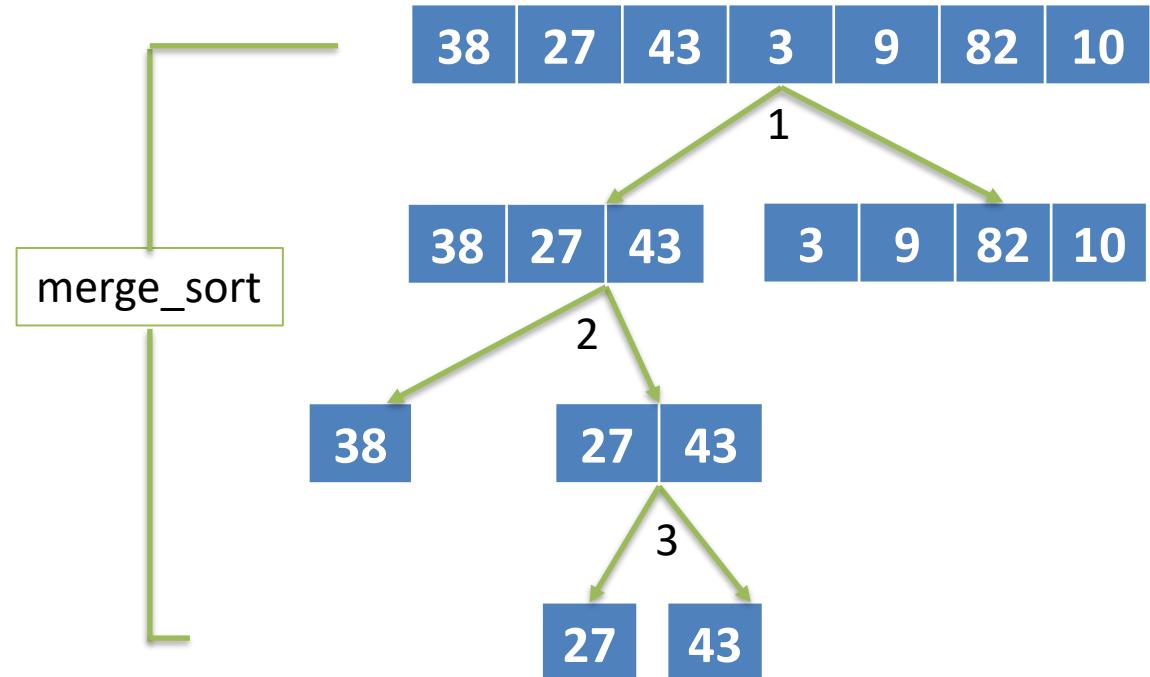
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



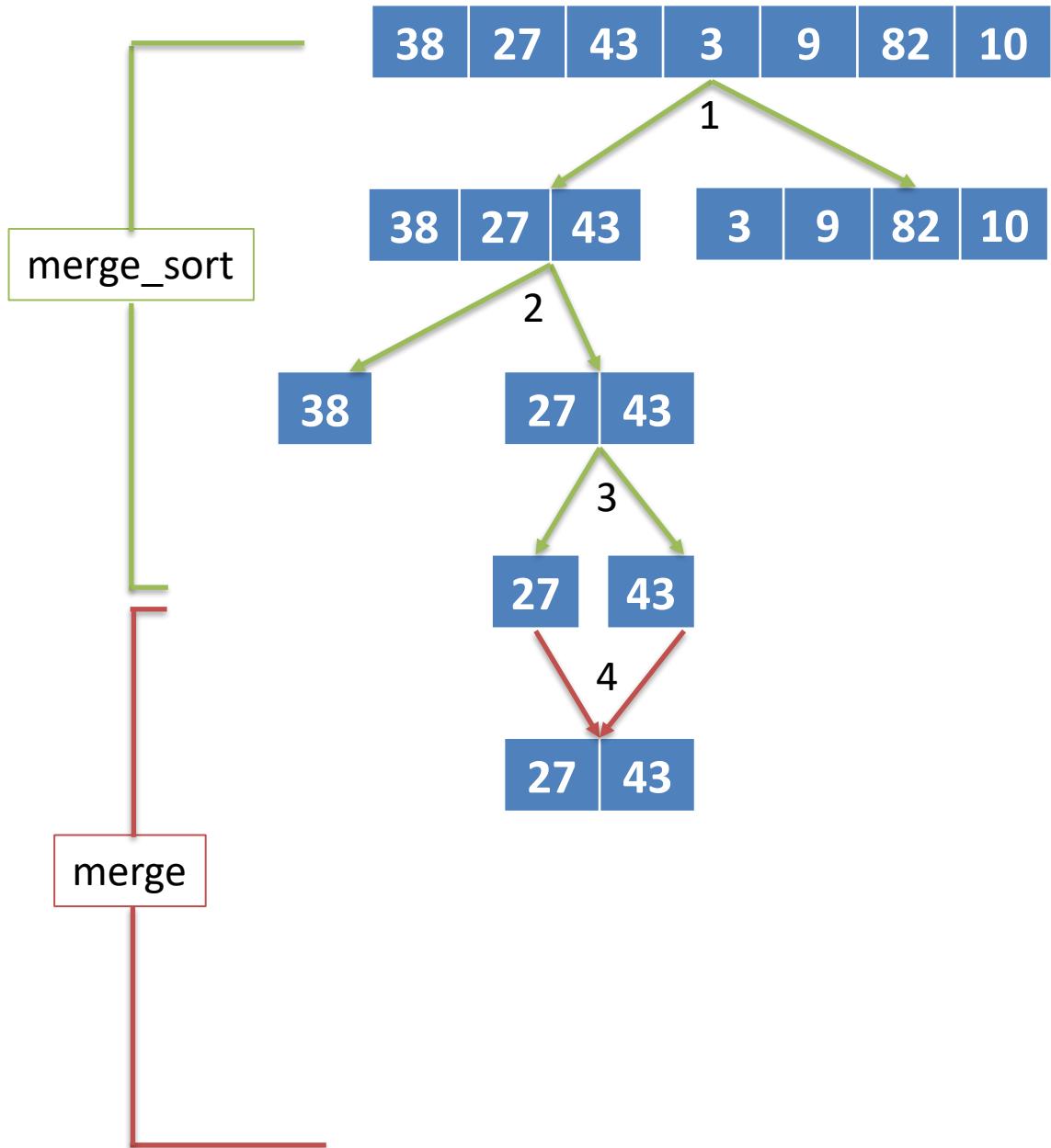
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



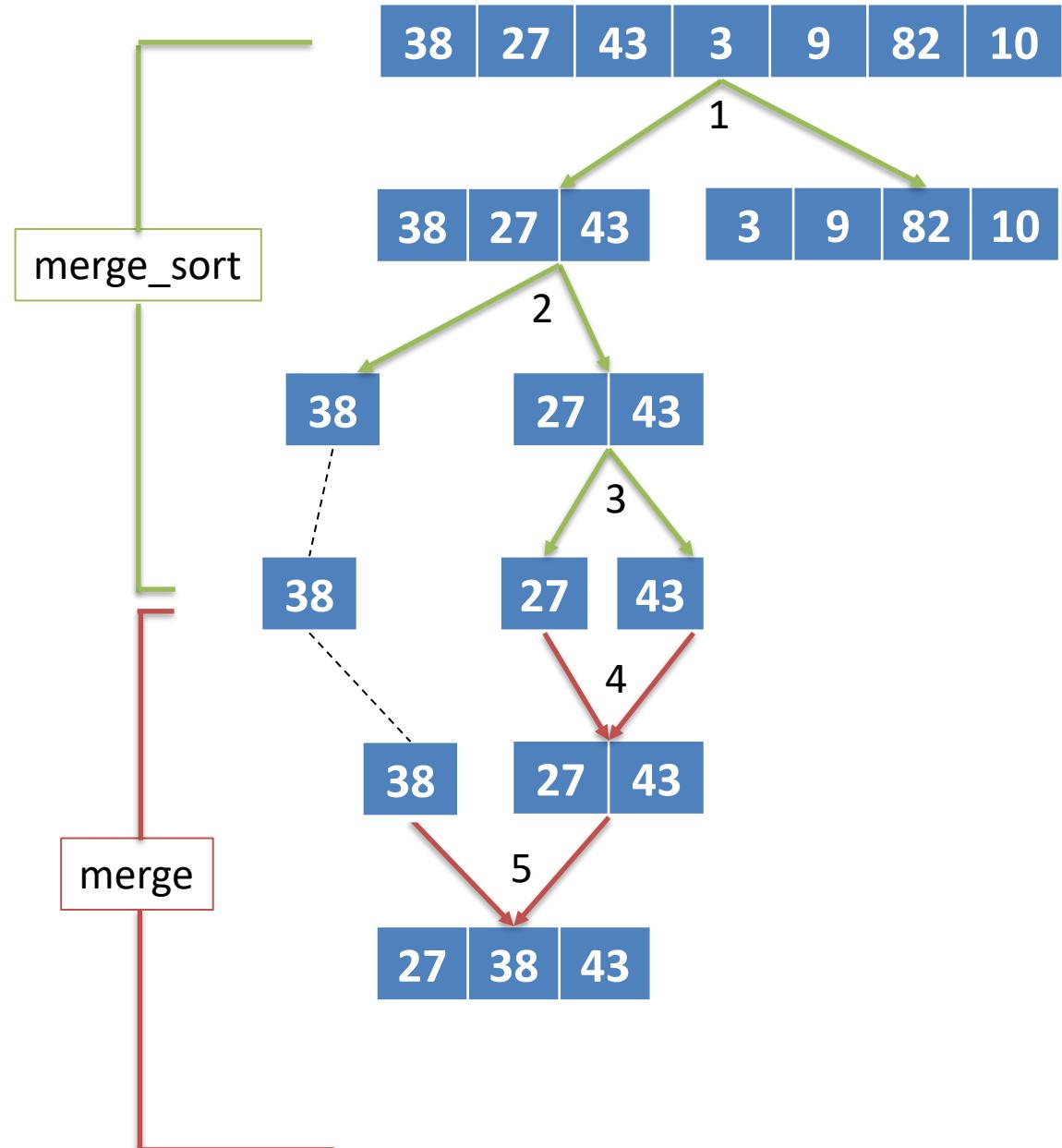
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



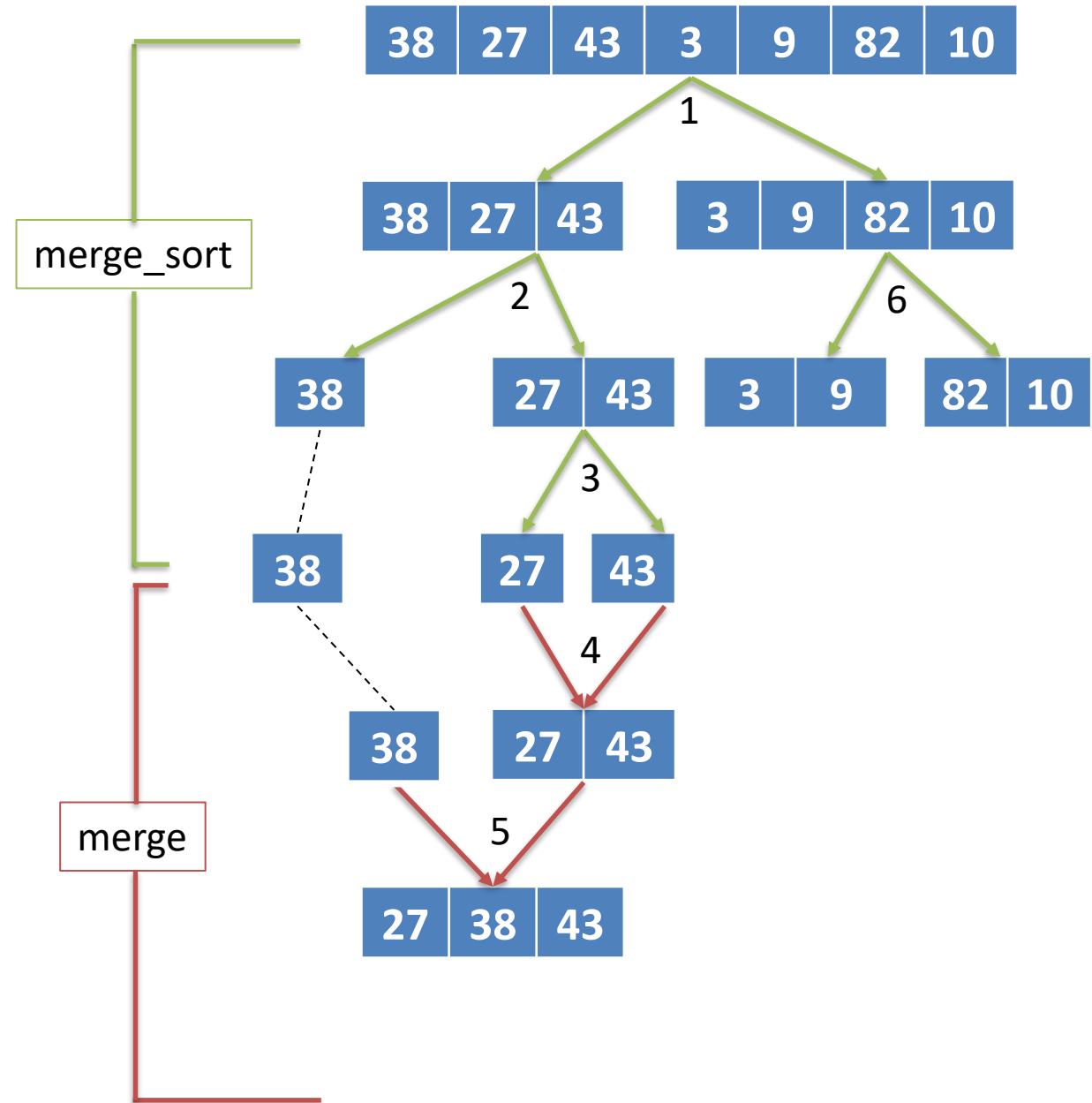
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



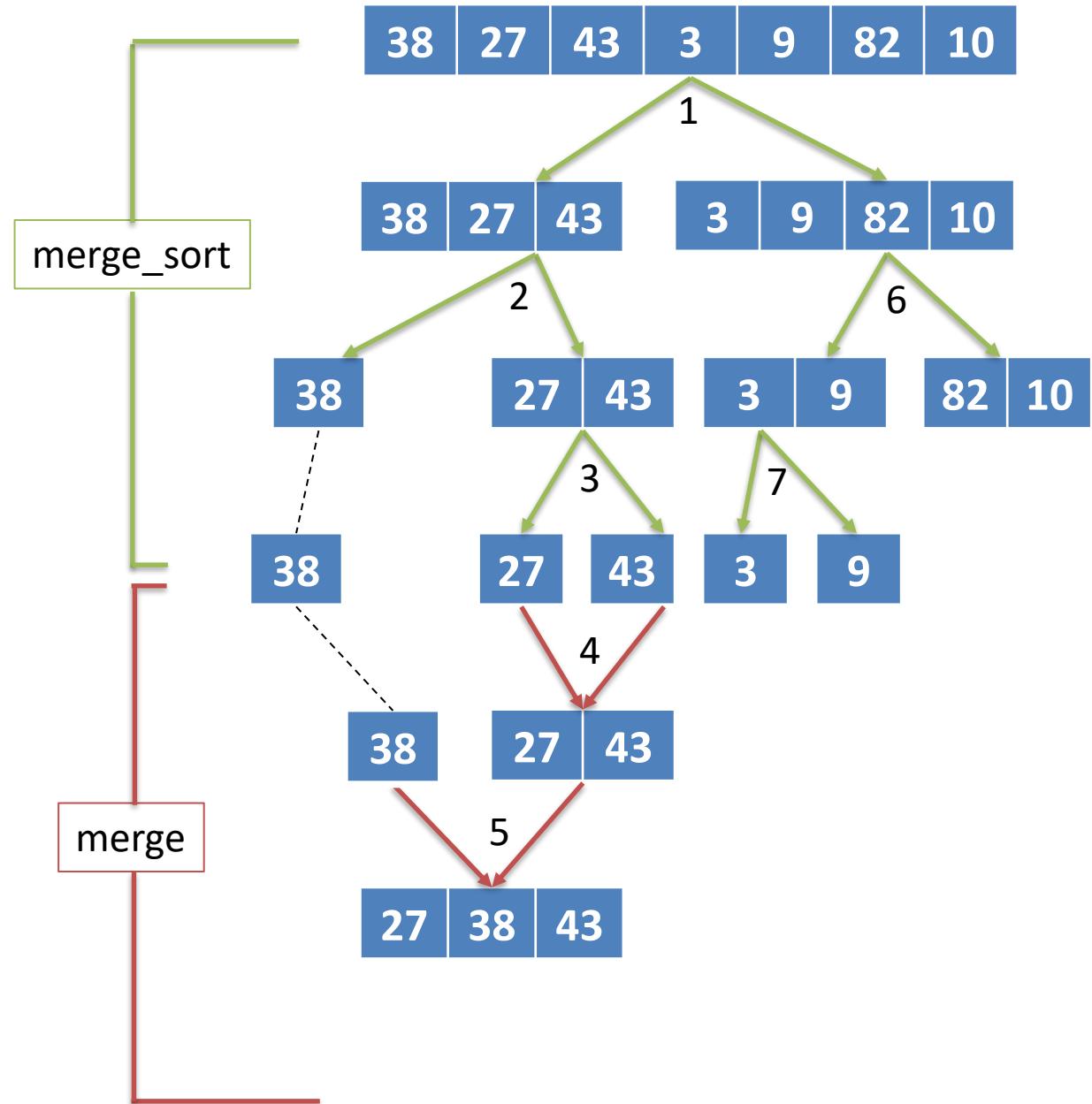
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



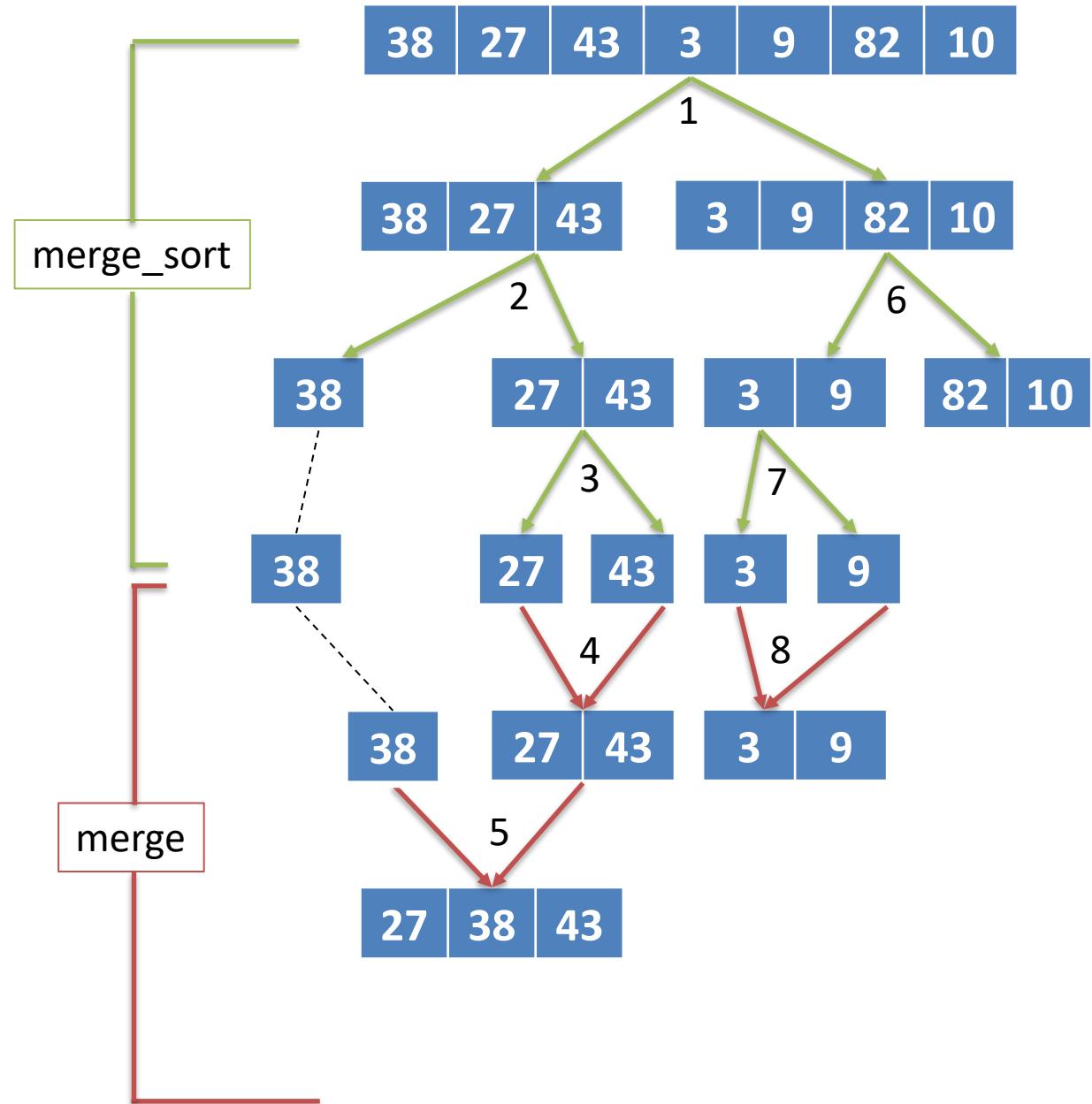
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



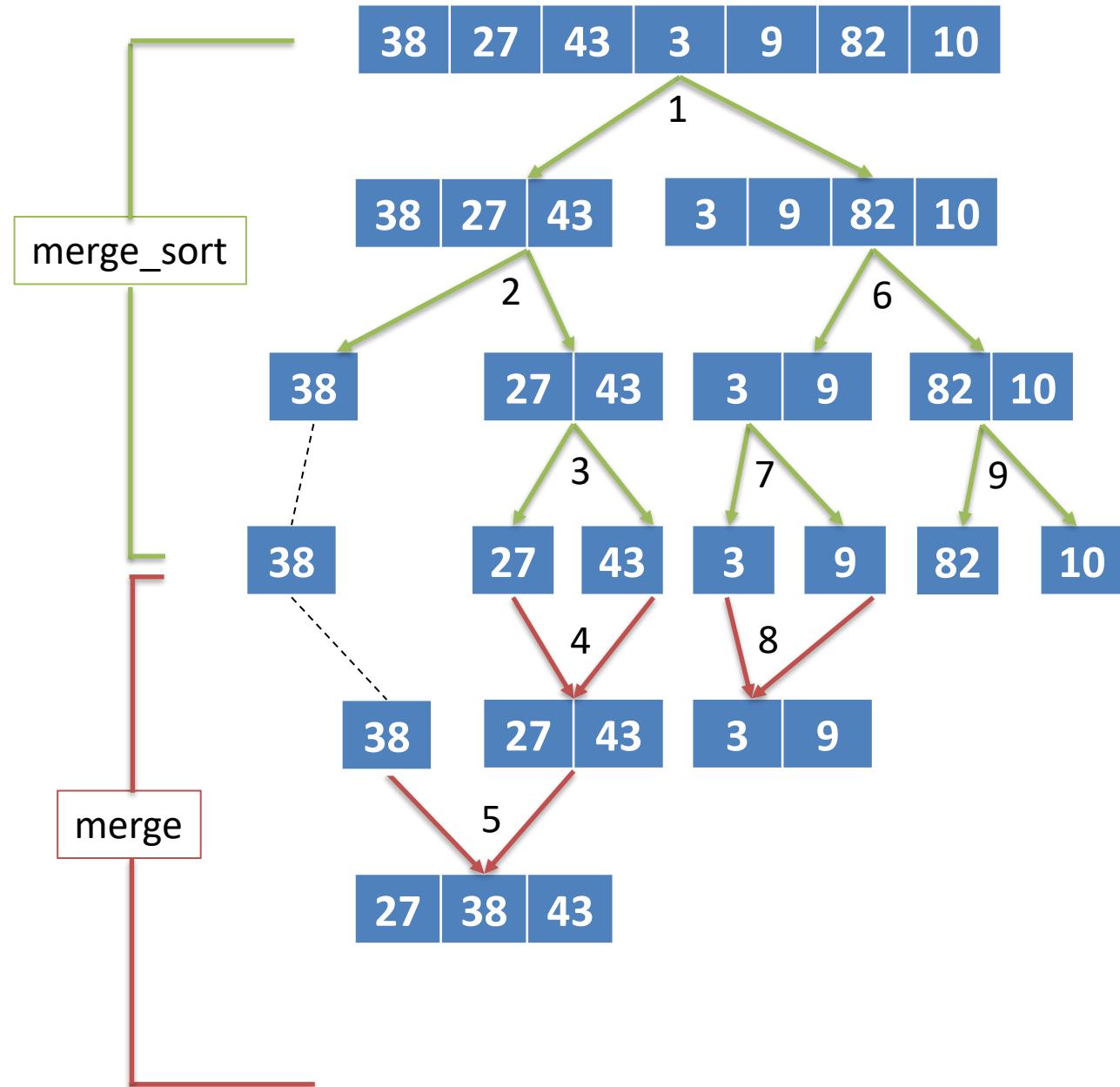
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



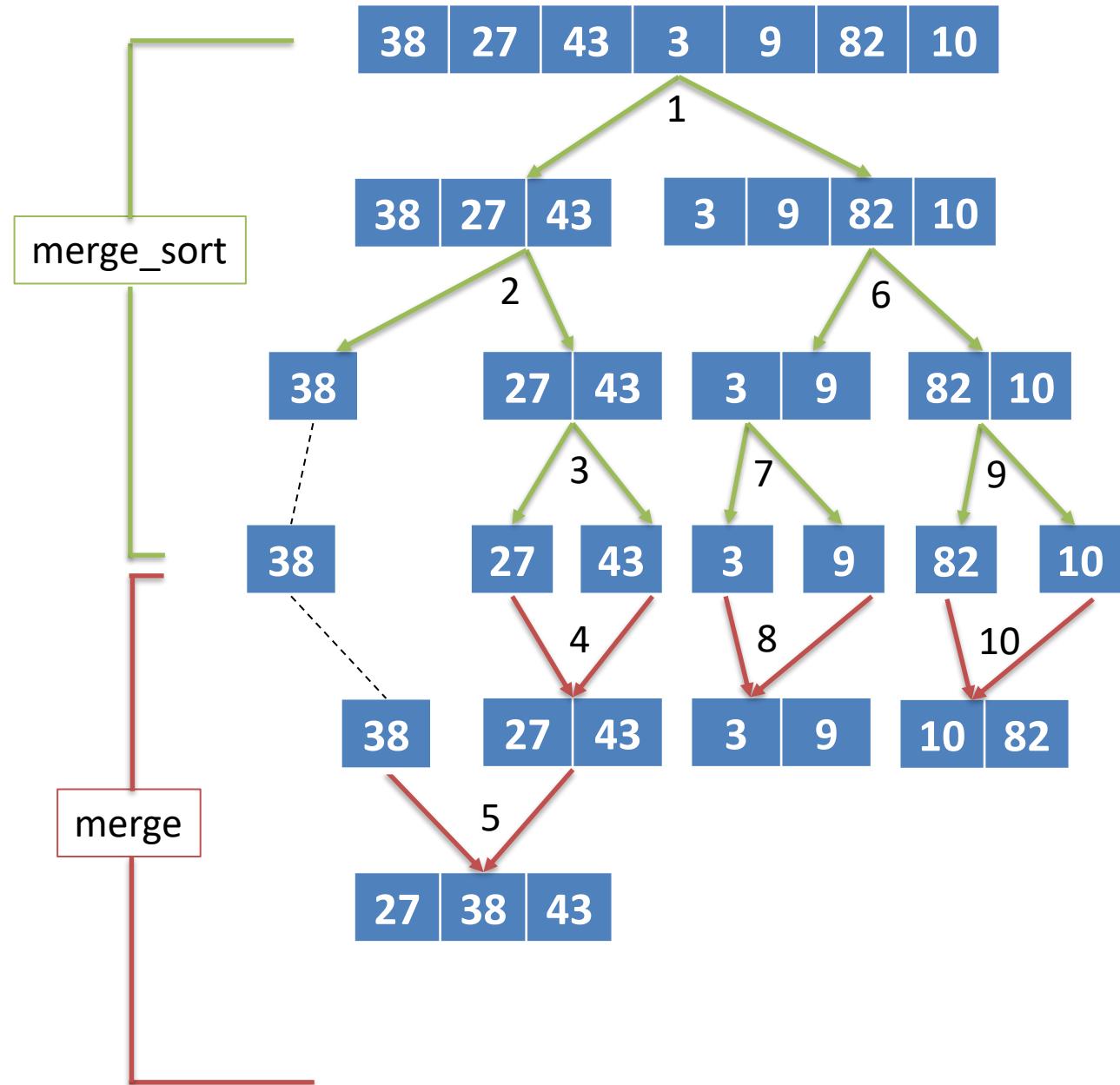
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



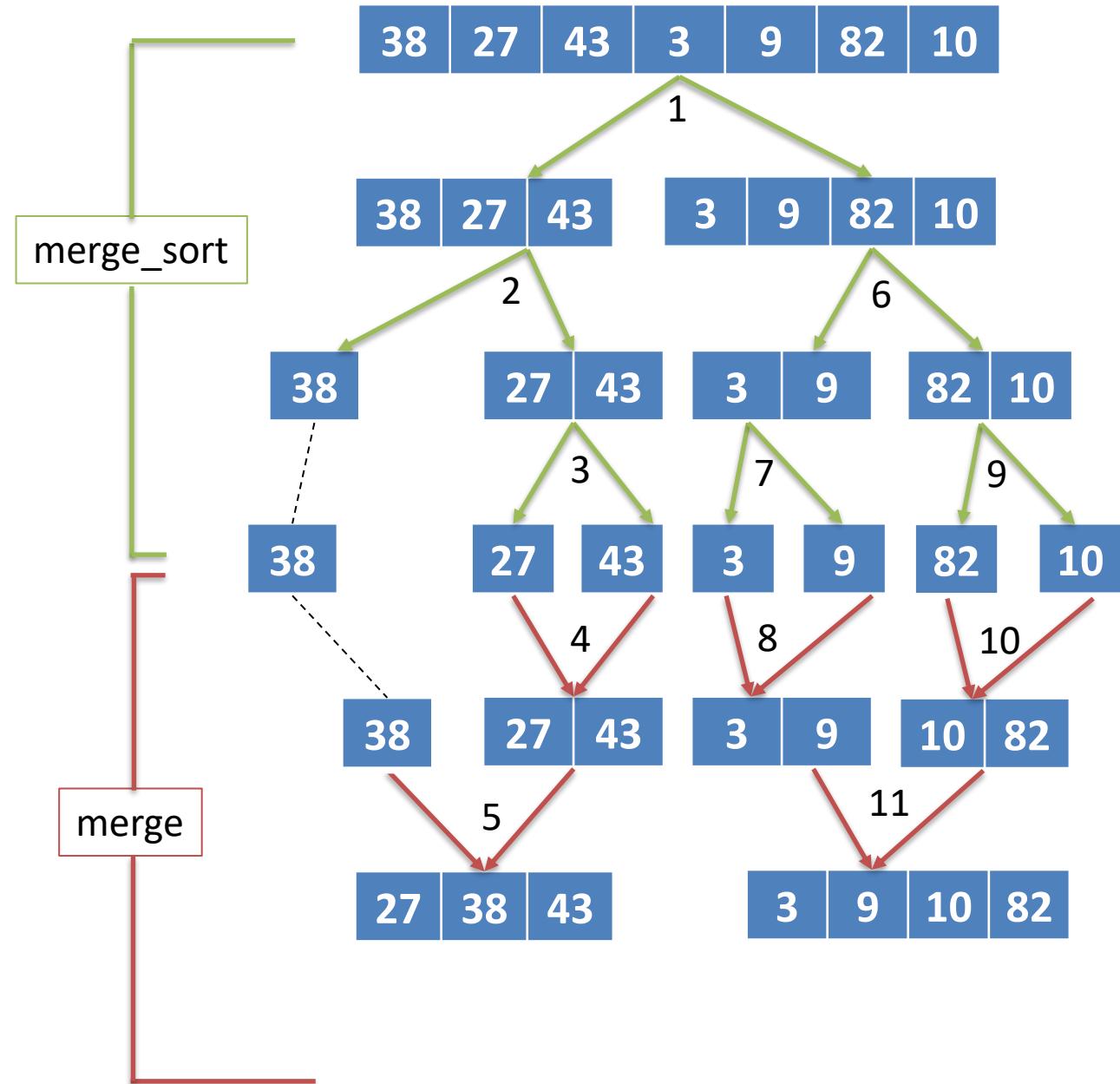
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



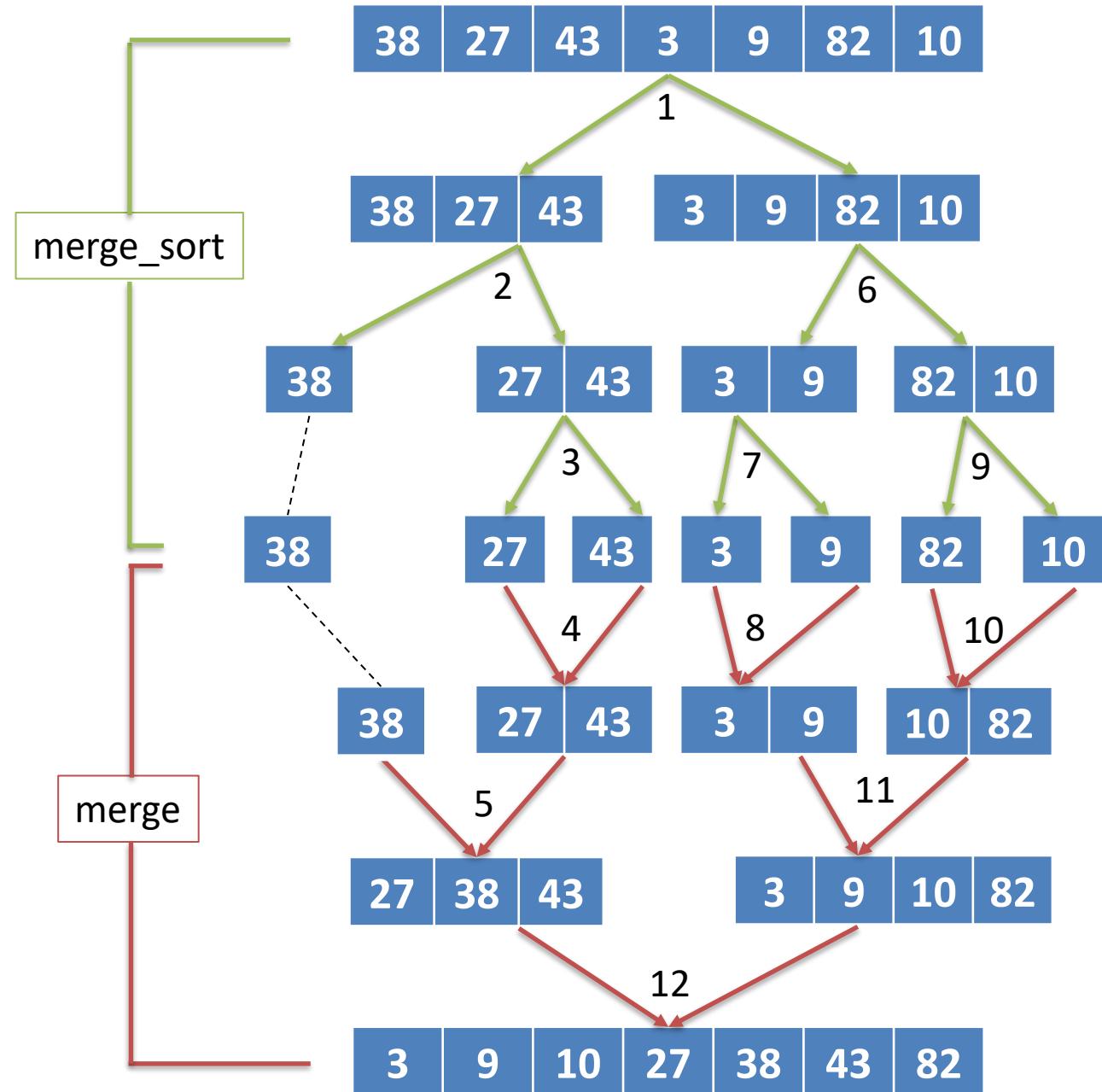
병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



병합 정렬은 재귀적이다

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    m = len(a) // 2
    a1 = a[:m]
    a2 = a[m:]
    l = merge_sort(a1)
    r = merge_sort(a2)
    lst = merge(l, r)
    return lst
```



여러 계산식들을 표현하기 위한 프로그래밍 언어를 배웠습니다. (Python)

프로그램을 작성하고, 디버깅하는 과정에 대해서 배웠습니다.

추상화에 대해서 배웠습니다. (자료와 함수)

문제를 작은 여러 문제로 나눠서, 각 문제를 하나씩 프로그램으로 해결하는 방법을 배웠습니다.

프로그래밍을 통해 문제의 답을 찾을 수 있다는 것을 기억하세요.

수업의 앞쪽에는 다음 내용들을 배웠습니다.

- 프로그래밍을 배워야 하는 이유
- 논리값, 조건문, 반복문
- 객체, 형태, 변수, 멤버 함수, 연산자, 식, 튜플
- 매개 변수와 반환값을 가진 함수
- 지역 변수와 전역 변수, 모듈, 그래픽 객체
- 시퀀스: 리스트, 문자열, 튜플

수업의 뒤쪽에는 다음 내용들을 배우고 있습니다.

- 문자열 멤버 함수, 집합, 사전, 이미지 프로세싱
- 파일 입출력
- 객체 생성과 객체의 속성
- 객체 생성자, 사용자 인터페이스 프로그래밍
- 해석기 vs 컴파일러, 알고리즘