2017 2학기
인공지능 강의노트

2주차

강사: 양일호

School of Computer Science
University of Seoul

강의 계획

주차	강의	실습
1	과목 소개 인공지능 및 python 소개 Python 프로그래밍 환경 조성 및 기본 문법	Python 개발 환경 구축 간단한 expert system 구현
2	Search (Blind Search, Heuristic Search)	DFS, BFS, A* 구현
3	Learning / Simulated Annealing	Simulated Annealing 구현
4	Genetic Algorithm	Genetic Algorithm 구현
5	보강 주간 (개천절 / 추석 연휴)	
6	Neural Network	Perceptron 구현
7	Neural Network	Single-Layer Perceptron 구현
8	Neural Network	Multi-Layer Perceptron 구현
9	Deep Neural Network	DNN 구현
10	Deep Neural Network	DNN 구현
11	DNN 심화 (혹은 Particle Swarm Optimization)	관련 내용 구현
12	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
13	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
14	DNN 심화 (혹은 Bayesian Networks)	관련 내용 구현
15	DNN 심화 (혹은 Decision Networks)	관련 내용 구현
16	기말고사	

Python 숙달을 위한 준비 기간

예비 기간 (대체 가능)

전문가 시스템(expert system)

- 간단한 인공지능 예시
  - 물고기 종류 구분

어떤 가공식품 공장에서 식재료인 연어와 농어를 한꺼번에 납품 받는다.

어부들이 보내온 납품 박스 안에는 연어와 농어가 섞여있는데,
공정에 들어가기 전에 이것을 종류별로 분류해야 한다.

지금까지는 사람의 손으로 일일이 분류했지만 시간이 너무 많이 걸리므로
이제는 컴퓨터가 사람 대신 분류할 수 있도록 자동화하고 싶다.

공장에서는 인공지능 어종 분류기를 제작하기 위한 프로젝트에
어류 전문가인 여러분을 초빙하였다.

여러분은 어떻게 두 종류의 물고기를 분류할 것인가?

전문가 시스템(expert system)

- 간단한 인공지능 예시
  - 물고기 종류 구분
    - 또 다른 특징을 함께 고려
      - Ex〉 꼬리 길이

연어

몸길이 = 60	꼬리길이 = 11
몸길이 = 68	꼬리길이 = 18
몸길이 = 80	꼬리길이 = 15
몸길이 = 75	꼬리길이 = 13
몸길이 = 84	꼬리길이 = 20

농어

몸길이 = 99	꼬리길이 = 12
몸길이 = 102	꼬리길이 = 9
몸길이 = 110	꼬리길이 = 5
몸길이 = 85	꼬리길이 = 8
몸길이 = 82	꼬리길이 = 7

```
if 몸길이 < x and 꼬리길이 > y:
        return '연어'
else if 몸길이 >= x and 꼬리길이 <= y:
        return '농어'
```

Python 개발환경 준비

- cmd에서 "python -m pip install -U pip" 입력시 오류('python' 은 내부 또는 외부 명령, 실행할 수 있는 프로그램 또는 배치 파일이 아닙니다.)
  - python.exe 경로를 환경변수 PATH에 추가해야 함.
    (환경변수 수정 후 cmd 재실행)

- cmd에서 "pip install - upgrade setuptools" 입력시 usage 오류
  - 구 버전 강의노트에서 명령어가 자동완성때문에 잘못 기재돼 있음("...-upgrade..." -> "...--upgrade...")

- cmd에서 "pip install --upgrade setuptools" 입력시 오류('pip' 는 내부 또는 외부 명령, 실행할 수 있는 프로그램 또는 배치 파일이 아닙니다.)
  - pip.exe 경로를 환경변수 PATH에 추가해야 함.
    (환경변수 수정 후 cmd 재실행)

Python 개발환경 준비

- numpy / scipy whl 설치가 안 되는 경우
  - 다시 다운로드하여 재설치 시도
  - 정 안될 경우 실습 자리 변경(326호 37번 자리에서는 안 되는 것 확인)

- .keras 폴더를 못 찾는 경우
  - "import keras" 코드를 한 번 실행시켜야 폴더가 생성됨
  - win10 기준: c:₩Users₩[사용자 이름]₩.keras 경로에 생성됨
  - 간혹 [파이썬 설치 경로]₩settings 아래에 생성되는 경우도 있으니 확인

- keras import시 g++ 미설치 경고문
  - TDM-GCC 설치 및 환경변수 설정이 정상적으로 되었는지 확인

Python 개발환경 준비

- notepad++ 에서 플러그인 추가가 안 될 경우
  - notepad++를 관리자 권한으로 실행한 뒤 플러그인 가져오기

- pip install시 UnicodeDecodeError 발생
  - 컴퓨터 이름 혹은 사용자 이름이 한글인 경우 → 영문으로 변경

공지

- 1주차 실습 과제 제출기한 1주 연장(9/20까지로)

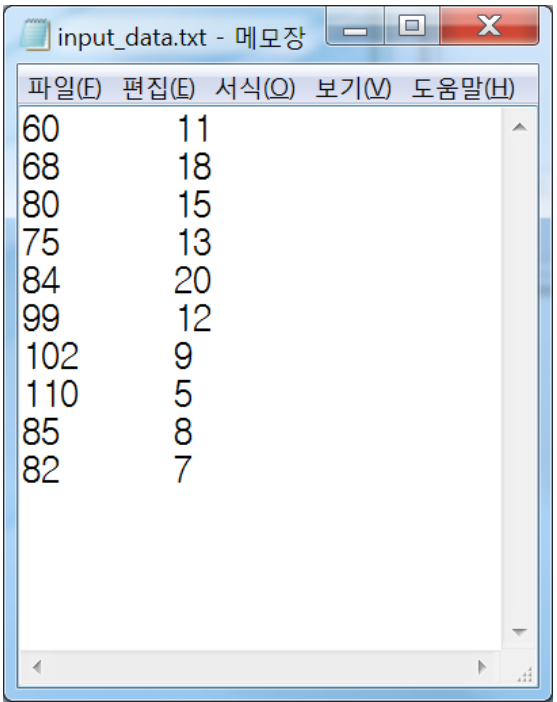
- 메일로 문의할 때 uos 메일 말고 다른 계정 이용 권장
  - Hanmail에서 uos 메일로 답장하면 반송됨

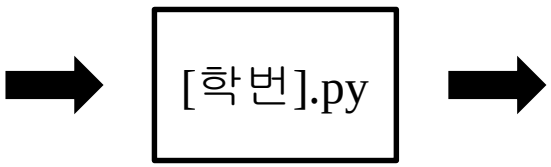
- 맥/리눅스 환경에서 개발해도 되지만 윈도우에서 동작해야 함
  - OS 종속적인 부분에 유의하여 코딩할 것
    - os.sep (윈도우: '₩', 리눅스: '/') 등…
  - 최소한 제출 전에 윈도우에서 한 번은 정상 동작 여부 확인해 볼 것
    - 과제 검사할 때 동작 안 하면 0점
    - 과제 검사 환경: win 7 64bit, python 2.7

실습과제

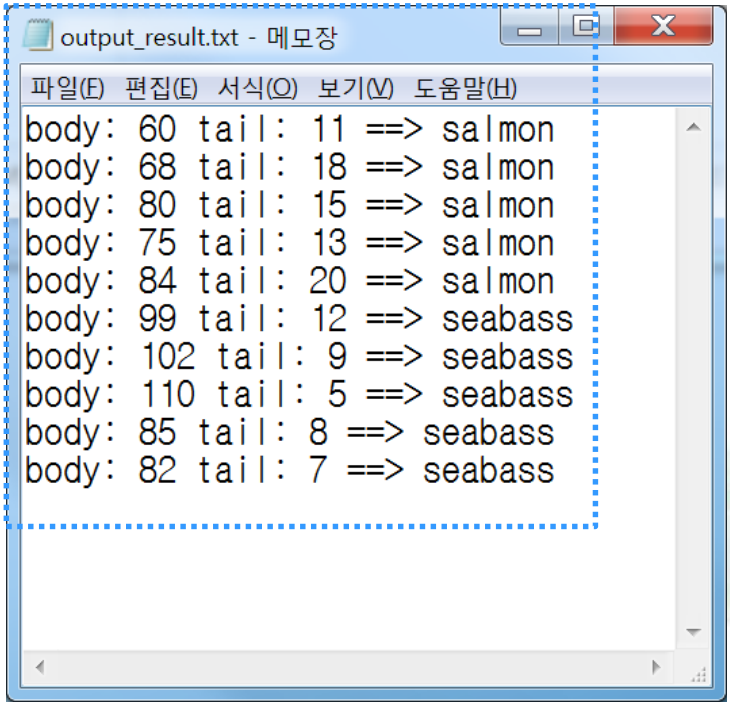
- "[학번].py" 스크립트 작성
  - 다음 기능을 구현(10점)
    - 같은 디렉토리의 "input_data.txt" 파일 읽기
    - 각 행의 데이터에 대해 순차적으로 어종 분류
    - 분류 결과를 같은 디렉토리의 "output_result.txt"에 출력
    - 위 세 파일을 압축하여 에듀클래스로 제출

입력 텍스트 파일





출력 텍스트 파일

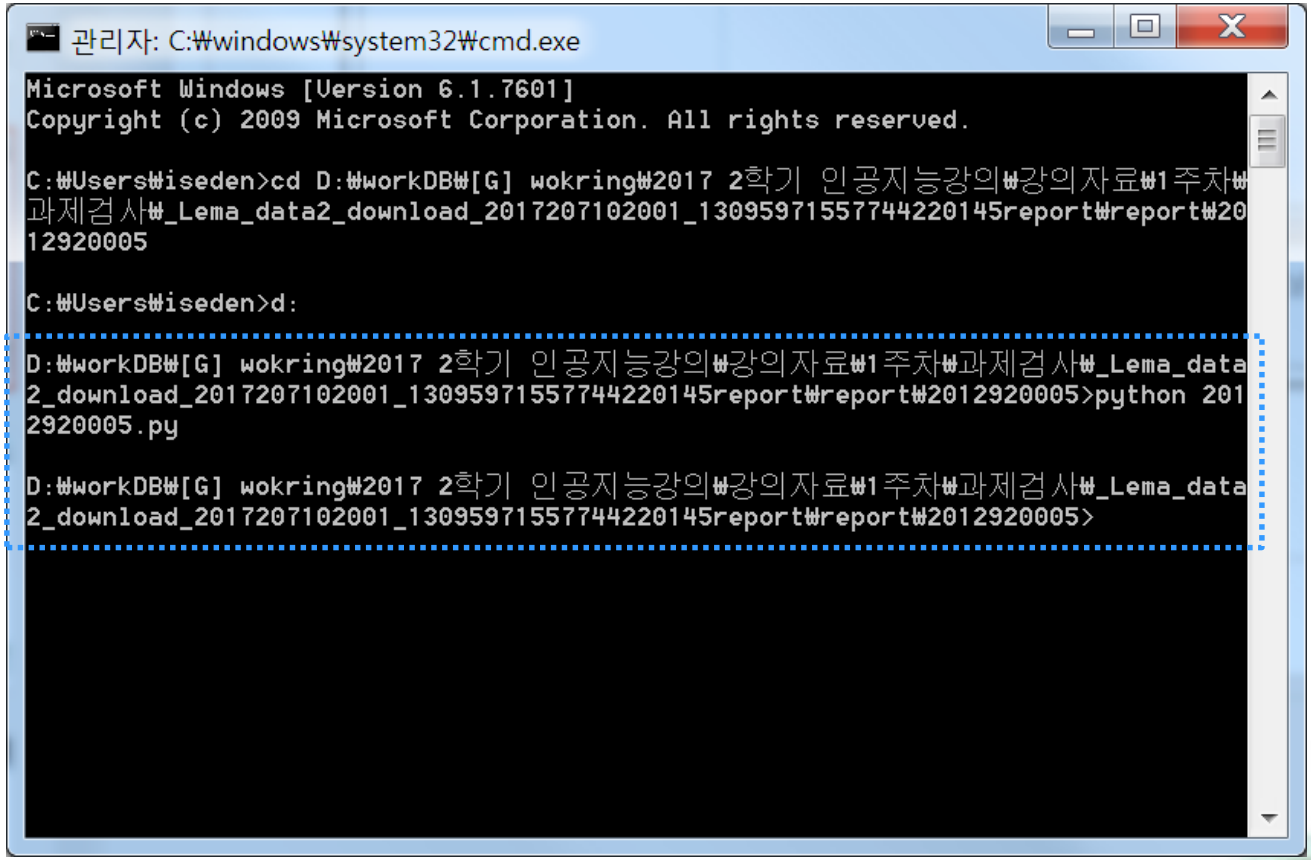


실습과제 평가기준

- 10점 만점
  - 기능 구현이 되었으면 10점, 동작하지 않으면 0점
    - 일부 부분 점수가 명시되어 있는 과제의 경우: 최소한 1개 이상의 기능 구현 필요
  - 감점요인: 최대 -5점
    - 제출 형식을 따르지 않음(-1점)
      - 과제 설명에서 제시한 제출 형식(ex〉 파일명 등)을 따르지 않은 경우
    - 제출 기한 초과(-1점씩):
      - 1주일 넘어서 제출할 때마다 -1점씩
    - 주석 미흡(-1점):
      - 어떤 동작을 하는 프로그램인지 주석을 보고 쉽게 알 수 있도록
    - 프로그램 출력 미흡(-1점):
      - 어떤 동작을 하는 프로그램인지 출력(콘솔 혹은 GUI)을 보고 쉽게 알 수 있도록
    - 예기치 못한 오류 발생 등(-1점씩):
      - Ex〉 대체로 잘 동작하나 특정 상황에서 예외 처리가 잘 되어 있지 않아 오류 발생

공지

- 모르는 사람이 실행했을 때에도 결과를 이해할 수 있도록 출력



강의 계획

주차	강의	실습
1	과목 소개 인공지능 및 python 소개 Python 프로그래밍 환경 조성 및 기본 문법	Python 개발 환경 구축 간단한 expert system 구현
2	Search (Blind Search, Heuristic Search)	DFS, BFS, A* 구현
3	Learning / Simulated Annealing	Simulated Annealing 구현
4	Genetic Algorithm	Genetic Algorithm 구현
5	보강 주간 (개천절 / 추석 연휴)	
6	Neural Network	Perceptron 구현
7	Neural Network	Single-Layer Perceptron 구현
8	Neural Network	Multi-Layer Perceptron 구현
9	Deep Neural Network	DNN 구현
10	Deep Neural Network	DNN 구현
11	DNN 심화 (혹은 Particle Swarm Optimization)	관련 내용 구현
12	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
13	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
14	DNN 심화 (혹은 Bayesian Networks)	관련 내용 구현
15	DNN 심화 (혹은 Decision Networks)	관련 내용 구현
16	기말고사	

Python 숙달을 위한 준비 기간

예비 기간 (대체 가능)

강의개요

- 강의
  - Tree-based search
    - Blind search : DFS, BFS
    - Heuristic search : A*
- 실습
  - Python 개발환경 준비(계속)
    - WinPython / notepad++(NppExec)
  - Python GUI 프로그래밍
    - Tkinter
  - Search 알고리즘 구현

강의

School of Computer Science
University of Seoul

Search

- 탐색
  - 찾고자 하는 어떤 것(goal)을 그것이 있음직한 곳에서 찾아보는 것

State: 어떤 시점에서의 상황 (ex〉 체스판의 말 배치)
Operator: state를 변경하는 동작 (ex〉 퀸을 4칸 전진)
Goal State: 탐색하고자 하는 목적 상황 (ex〉 아군의 말이 상대의 킹을 체크메이트)
Cost: 탐색의 질을 측정하는 척도 (ex〉 자신의 말을 128번 움직여서 상대방을 이김)

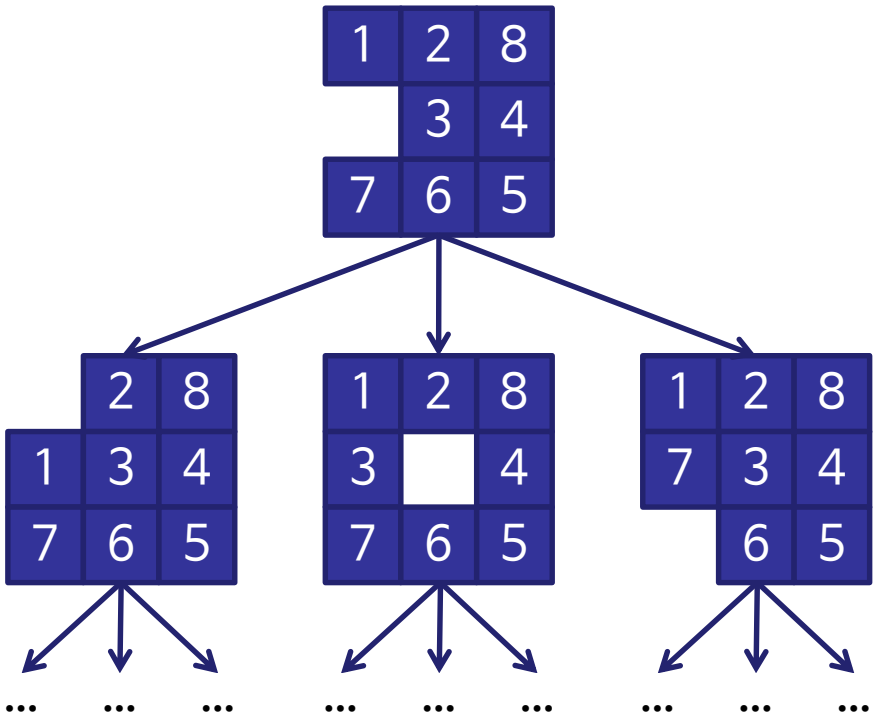
Some classical AI problems

- The eight-tile puzzle

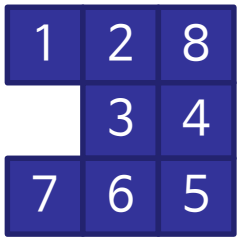
http://www.routerforums.com/tools-woodworking/33302-giant-sliding-puzzle.html

Some classical AI problems

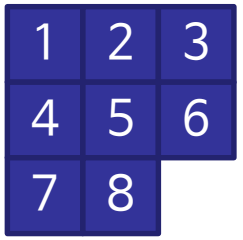
- The eight-tile puzzle



Start state:



Goal state:



Operator: 위쪽에서 옮김
왼쪽에서 옮김
오른쪽에서 옮김
아래쪽에서 옮김

Cost: 퍼즐을 움직인 횟수

Some classical AI problems

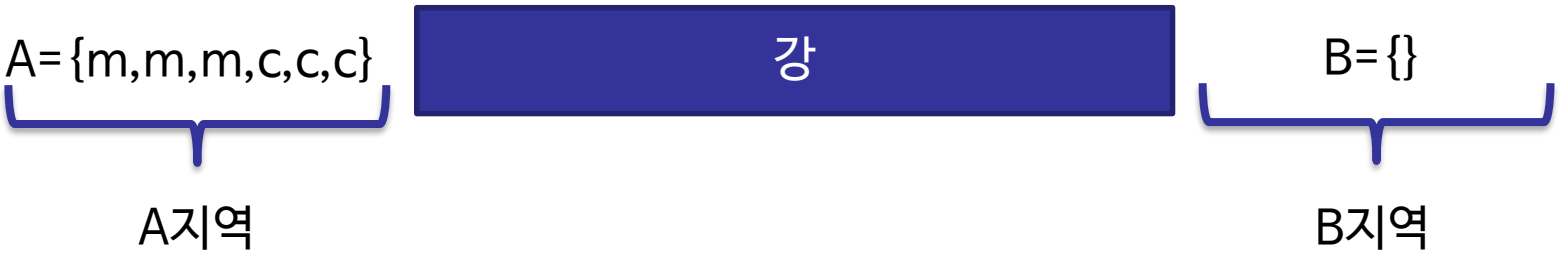
- Missionaries and cannibals

http://www.novelgames.com/en/missionaries/

Some classical AI problems

- Missionaries and cannibals

3명의 선교사(m)와 3명의 식인종(c)이 A지역에 있다
이들은 2인용 보트를 이용하여 B지역으로 이동해야 한다
단, 한 지역에서 식인종의 수가 선교사보다 많아지면 안 된다



Start state: A={m,m,m,c,c,c}, B={}

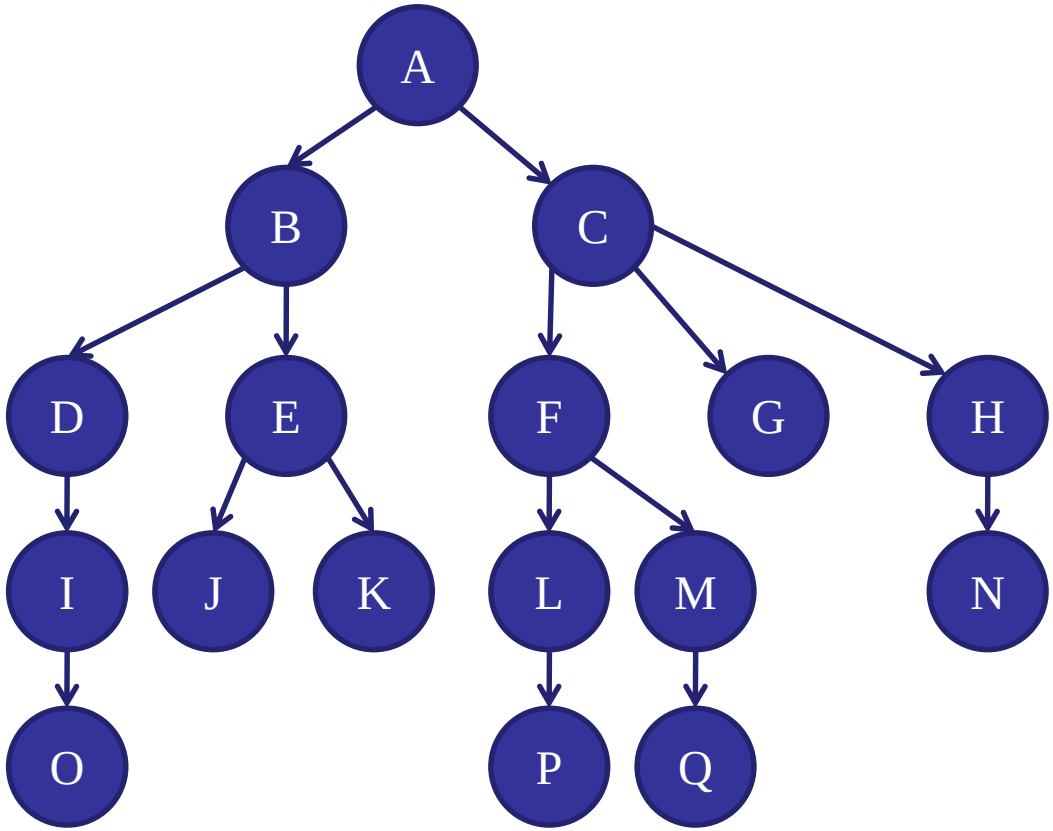
Goal state: A={}, B={m,m,m,c,c,c}

Operator: {m} / {m, m} / {c} / {c, c} / {m, c} 가 이동

Cost: 배가 몇 번 움직였는가?

Tree-based search

- 트리를 파이썬 코드로 표현하기




```
# dictionary로 tree 표현
treeDict = {}
treeDict['A'] = ['B', 'C']
treeDict['B'] = ['D', 'E']
treeDict['C'] = ['F', 'G', 'H']
treeDict['D'] = ['I']
treeDict['E'] = ['J', 'K']
treeDict['F'] = ['L', 'M']
treeDict['G'] = []
treeDict['H'] = ['N']
treeDict['I'] = ['O']
treeDict['J'] = []
treeDict['K'] = []
treeDict['L'] = ['P']
treeDict['M'] = ['Q']
treeDict['N'] = []
treeDict['O'] = []
treeDict['P'] = []
treeDict['Q'] = []
```

Blind search

- 탐색을 수행할 때 어떠한 정보도 주어지지 않음
  - Ex〉 DFS, BFS

당신은 눈을 감고 타일을 더듬어가며 eight-tile puzzle을 푼다
옆에서 친구가 지켜보며 퍼즐을 다 풀었을 때에만 알려준다

DFS (Depth-First Search)

- 시작점으로부터 한 방향으로 막다른 지점이 나올 때까지 탐색
  - → 안 가봤던 갈림길이 있는 곳까지 돌아 나와 반복
- Stack을 이용하여 구현 가능

```
Node current
Stack toVisit

add root to toVisit

while toVisit is not empty
        current <- first node on toVisit
        remove first node from toVisit
        if current = goal
                add current to alreadyVisited
                return true
        end if
        for each child node C of current
                add C to toVisit
        end for
        add current to alreadyVisited
end while
```

DFS (Depth-First Search)

- Python 코드

tree.py

```
# dictionary로 tree 표현
treeDict = {}
treeDict['A'] = ['B', 'C']
treeDict['B'] = ['D', 'E']
treeDict['C'] = ['F', 'G', 'H']
treeDict['D'] = ['I']
treeDict['E'] = ['J', 'K']
treeDict['F'] = ['L', 'M']
treeDict['G'] = []
treeDict['H'] = ['N']
treeDict['I'] = ['O']
treeDict['J'] = []
treeDict['K'] = []
treeDict['L'] = ['P']
treeDict['M'] = ['Q']
treeDict['N'] = []
treeDict['O'] = []
treeDict['P'] = []
treeDict['Q'] = []

# 이 소스코드를 메인으로 실행하였을 때만 수행됨
if __name__ == '__main__':
    print treeDict
```

dfs.py

```
# Depth-First Search 예제

# tree.py로부터 정의한 tree 불러오기
from tree import treeDict

# tree의 root는 node A
root = 'A'

alreadyVisited = [] # 이미 방문한 노드 리스트
toVisit = []        # 앞으로 방문해야 할 노드 리스트 (stack)

# root를 방문해야 할 노드로 추가
toVisit.append(root)
print toVisit, alreadyVisited

# 앞으로 방문해야 할 노드가 남아있으면 루프 반복
while len(toVisit) != 0:
    # 방문해야 할 노드 stack의 top으로 현재 노드 이동
    current = toVisit.pop(-1)

    # 현재 노드의 자식 노드를 stack에 추가 ([::-1]은 리스트 역순)
    for child in treeDict[current][::-1]:
        toVisit.append(child)

    # 이미 방문한 노드에 현재 노드 추가
    alreadyVisited.append(current)

    print toVisit, alreadyVisited
```

BFS (Breadth-First Search)

- 시작점으로부터 가까운 곳을 먼저 탐색
- Queue를 이용하여 구현 가능

```
Node current
Queue toVisit

add root to toVisit

while toVisit is not empty
        current <- first node on toVisit
        remove first node from toVisit
        if current = goal
                add current to alreadyVisited
                return true
        end if
        for each child node C of current
                add C to toVisit
        end for
        add current to alreadyVisited
end while
```

BFS (Breadth-First Search)

- Python 코드

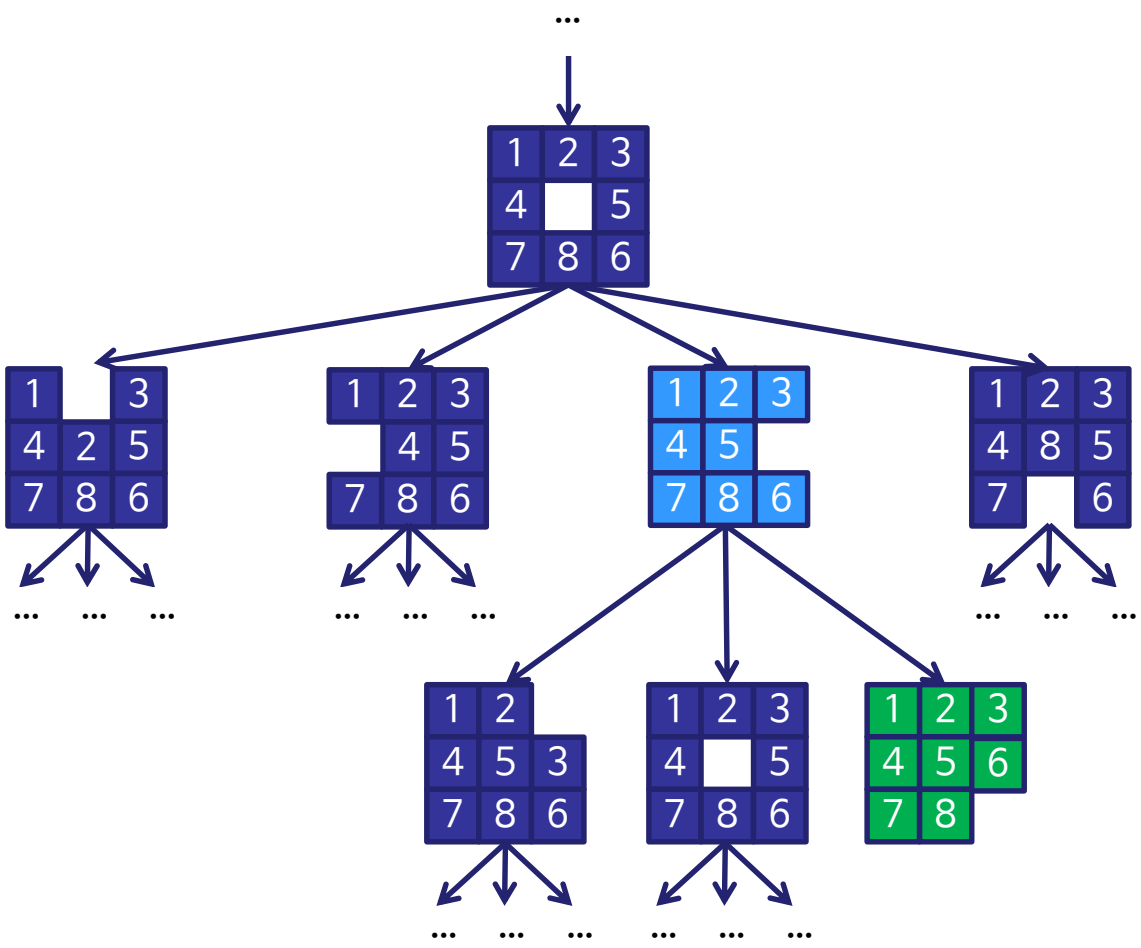
tree.py

```
# dictionary로 tree 표현
treeDict = {}
treeDict['A'] = ['B', 'C']
treeDict['B'] = ['D', 'E']
treeDict['C'] = ['F', 'G', 'H']
treeDict['D'] = ['I']
treeDict['E'] = ['J', 'K']
treeDict['F'] = ['L', 'M']
treeDict['G'] = []
treeDict['H'] = ['N']
treeDict['I'] = ['O']
treeDict['J'] = []
treeDict['K'] = []
treeDict['L'] = ['P']
treeDict['M'] = ['Q']
treeDict['N'] = []
treeDict['O'] = []
treeDict['P'] = []
treeDict['Q'] = []

# 이 소스코드를 메인으로 실행하였을 때만 수행됨
if __name__ == '__main__':
    print treeDict
```

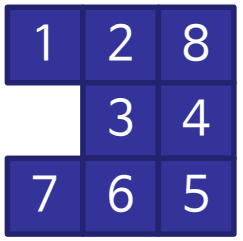
bfs.py

```
# Breadth-First Search 예제

# tree.py로부터 정의한 tree 불러오기
from tree import treeDict

# tree의 root는 node A
root = 'A'

alreadyVisited = [] # 이미 방문한 노드 리스트
toVisit = []        # 앞으로 방문해야 할 노드 리스트 (queue)

# root를 방문해야 할 노드로 추가
toVisit.append(root)
print toVisit, alreadyVisited

# 앞으로 방문해야 할 노드가 남아있으면 루프 반복
while len(toVisit) != 0:
    # 방문해야 할 노드 queue의 첫 번째로 현재 노드 이동
    current = toVisit.pop(0)

    # 현재 노드의 자식 노드를 queue에 추가
    for child in treeDict[current]:
        toVisit.append(child)

    # 이미 방문한 노드에 현재 노드 추가
    alreadyVisited.append(current)

    print toVisit, alreadyVisited
```

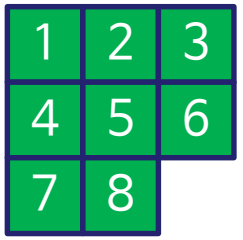
DFS? BFS?



Start state:



Goal state:



Operator:

1. 위쪽에서 옮김
2. 왼쪽에서 옮김
3. 오른쪽에서 옮김
4. 아래쪽에서 옮김

Cost: 퍼즐을 움직인 횟수

Heuristic search

- 탐색을 수행할 때 경험적으로 취득한 정보를 이용

당신은 눈을 뜨고 eight-tile puzzle을 푼다
당신은 eight-tile puzzle을 이전에도 여러 번 풀어 보았다
풀이와 거리가 먼 방향으로는 타일을 이동시키지 않을 것이므로
눈을 감고 풀 때보다 월등히 빠르게 문제를 해결할 수 있다

얼마나 답에 가까운 state인가? -〉 Heuristic function을 구성

Heuristic search

- Heuristic function의 예

당신은 어떤 음악 CD를 찾고 있다
도심에는 다양한 종류의 CD가게가 있는데 어딜 먼저 들어가보는 것이 좋을까?

상황 : 찾고 있는 CD가 희귀한 종류일 때

↓

경험 : 큰 가게에서는 다양한 종류의 CD를 판매한다

↓

Heuristic function : 가게의 크기를 반환

↓

탐색 전략 : 큰 가게를 우선적으로 탐색

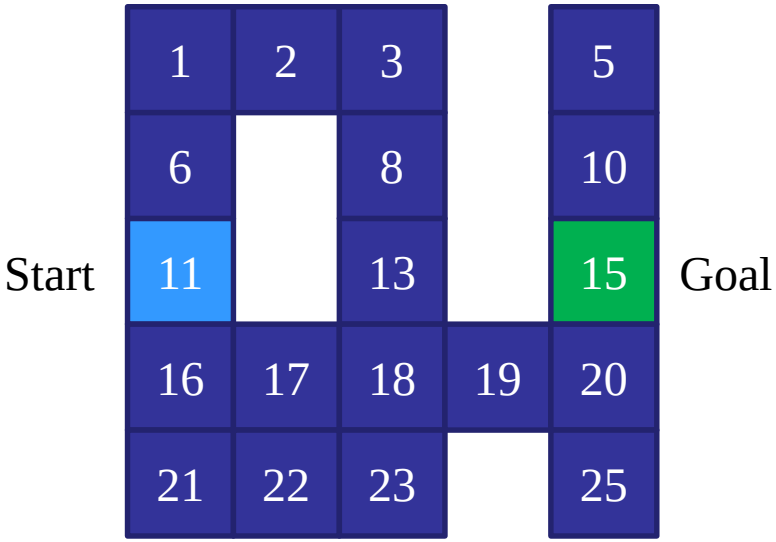
A* search

- Heuristic function
  - f(n) = g(n) + h(n)

g(n) = 현재 위치(n)까지 이동한 횟수

h(n) = 현재 위치(n)에서 목적지까지 적어도 최소한 이동해야 할 것 같은 횟수

실제 거리보다 h(n)이 크면 안됨



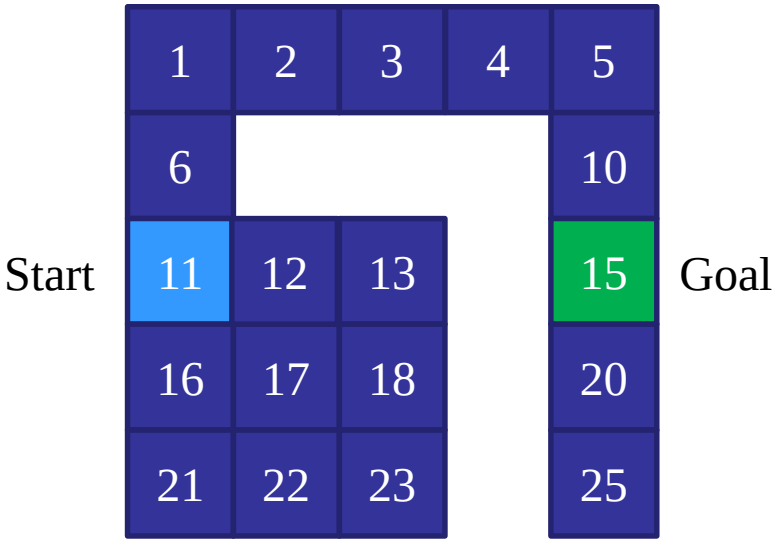
A* search

- Heuristic function
  - f(n) = g(n) + h(n)

g(n) = 현재 위치(n)까지 이동한 횟수

h(n) = 현재 위치(n)에서 목적지까지 적어도 최소한 이동해야 할 것 같은 횟수

실제 거리보다 h(n)이 크면 안됨

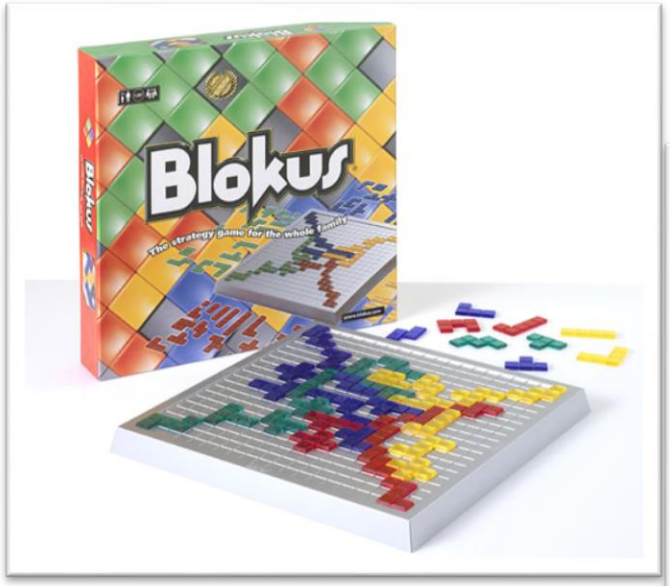


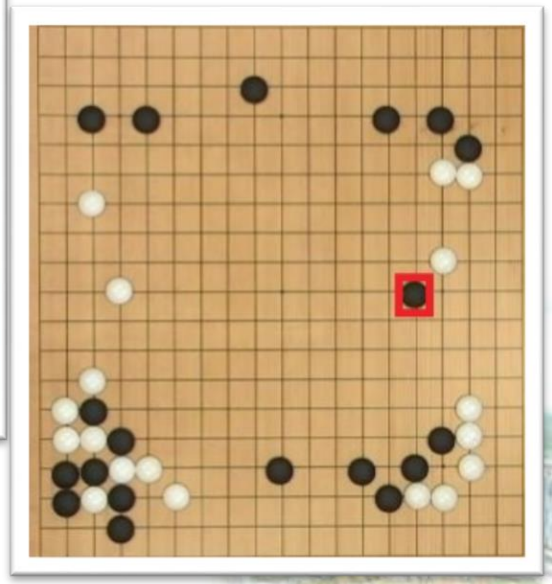
Heuristic technique

- Heuristic technique 설계의 어려움
  - (좋은 + 많은) 경험이 필요
    - 그 문제를 잘 알아야 잘 설계할 수 있음(해결해야 할 문제의 분야에 종속적)
      - Domain specific knowledge가 필요
  - 경험이 항상 맞는 것도 아님(최적의 해법이 아님)
    - 문제를 해결하는 최적의 해법을 모르는 경우라면 차선책으로 쓸 만함
    - 그럴 듯한 결과를 적은 노력(연산)을 들여 찾는 데에는 도움이 될 수는 있음
      - (ex〉 machine learning시 괜찮은 초기값을 지정할 때 쓸 만함)

Heuristic technique의 필요성

- 다음 게임들의 game tree를 그릴 경우 노드의 수는?
- Game tree를 저장하는 데 필요한 저장공간의 크기는?
- Tree 전체를 순회하는 데 걸리는 탐색 시간은?





Metaheuristic technique

- higher-level heuristic
  - 해결해야 할 문제의 분야에 덜 종속적(혹은 독립적)
    - Domain specific knowledge가 필요 없음(?)
      - Framework: domain이 달라져도 같은 방법을 적용 가능
      - Implementation: 구현한 코드를 그대로 가져다 쓰기는 어려움
        - (→ hyper-heuristic technique)

- Ex〉
  - SA(simulated annealing)
  - GA(genetic algorithm)
  - PSO(particle swarm optimization)
  - …

강의 계획

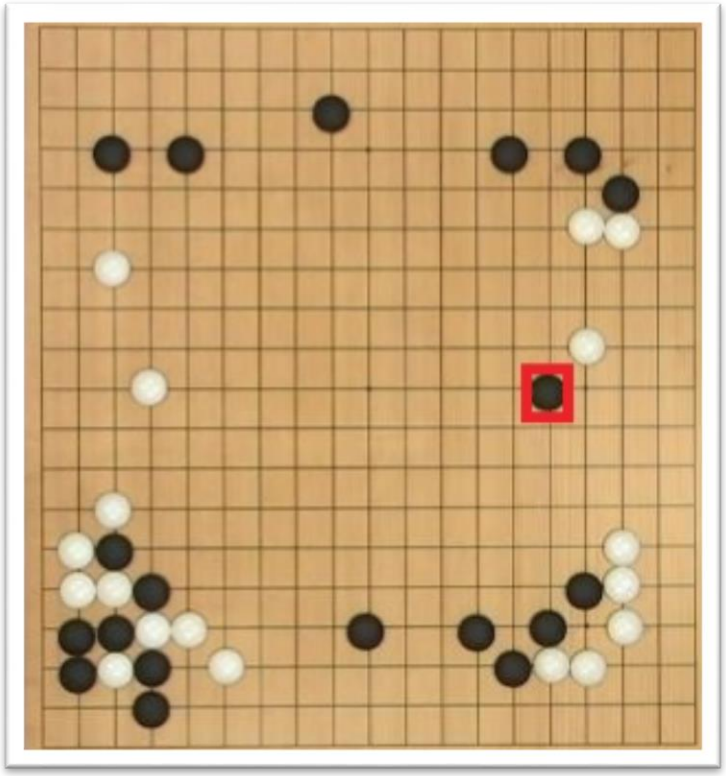
주차	강의	실습
1	과목 소개 인공지능 및 python 소개 Python 프로그래밍 환경 조성 및 기본 문법	Python 개발 환경 구축 간단한 expert system 구현
2	Search (Blind Search, Heuristic Search)	DFS, BFS, A* 구현
3	Learning / Simulated Annealing	Simulated Annealing 구현
4	Genetic Algorithm	Genetic Algorithm 구현
5	보강 주간 (개천절 / 추석 연휴)	
6	Neural Network	Perceptron 구현
7	Neural Network	Single-Layer Perceptron 구현
8	Neural Network	Multi-Layer Perceptron 구현
9	Deep Neural Network	DNN 구현
10	Deep Neural Network	DNN 구현
11	DNN 심화 (혹은 Particle Swarm Optimization)	관련 내용 구현
12	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
13	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
14	DNN 심화 (혹은 Bayesian Networks)	관련 내용 구현
15	DNN 심화 (혹은 Decision Networks)	관련 내용 구현
16	기말고사	

Python 숙달을 위한 준비 기간

예비 기간 (대체 가능)

Heuristic vs machine learning

- 주어진 시간 안에 game tree를 다 탐색할 수 없는 상황?



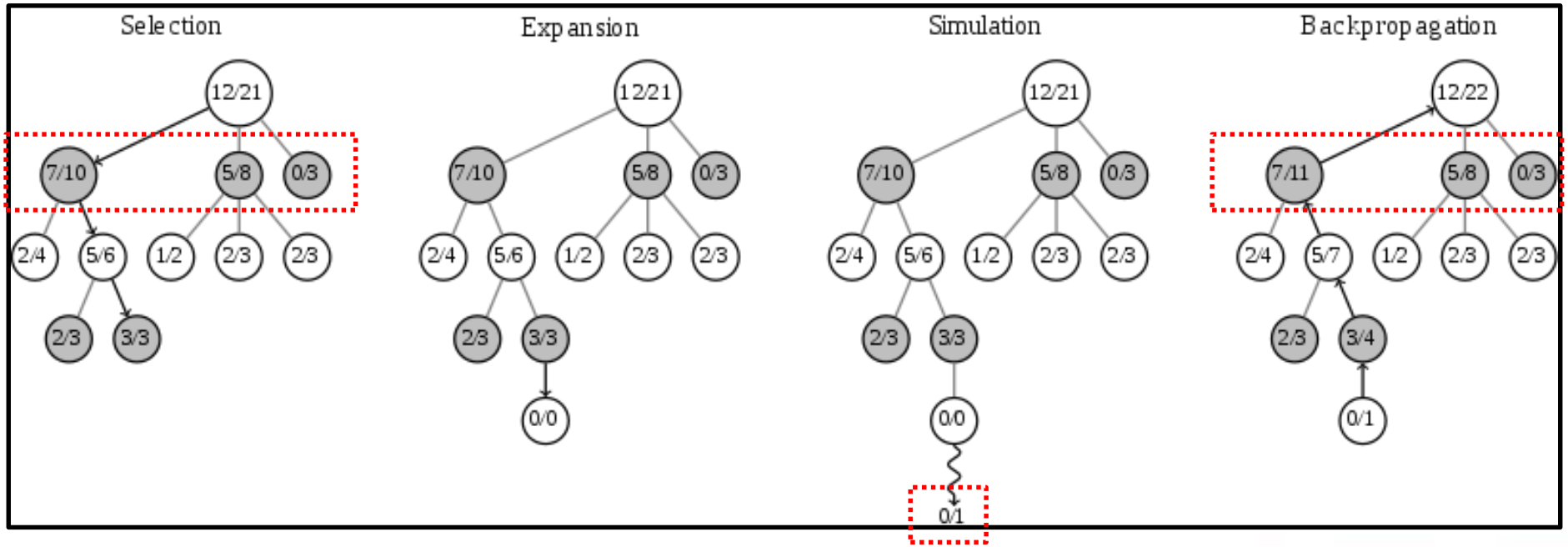
다음에 어디다 둬야 좋을까?

Heuristic vs machine learning

- Game tree를 일부만 탐색하는 방법
  - MCTS (Monte Carlo tree search)

다음 state들의 가치

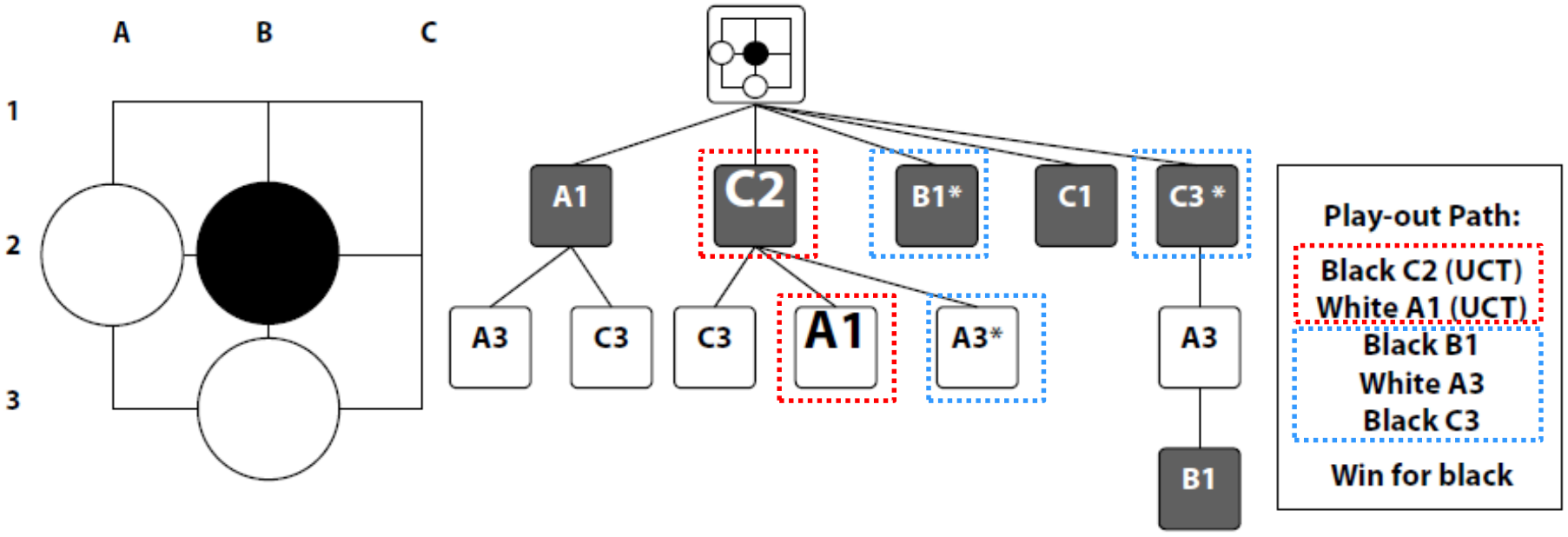
가치 갱신



제한 시간 동안 끝까지 플레이해보기

Heuristic vs machine learning

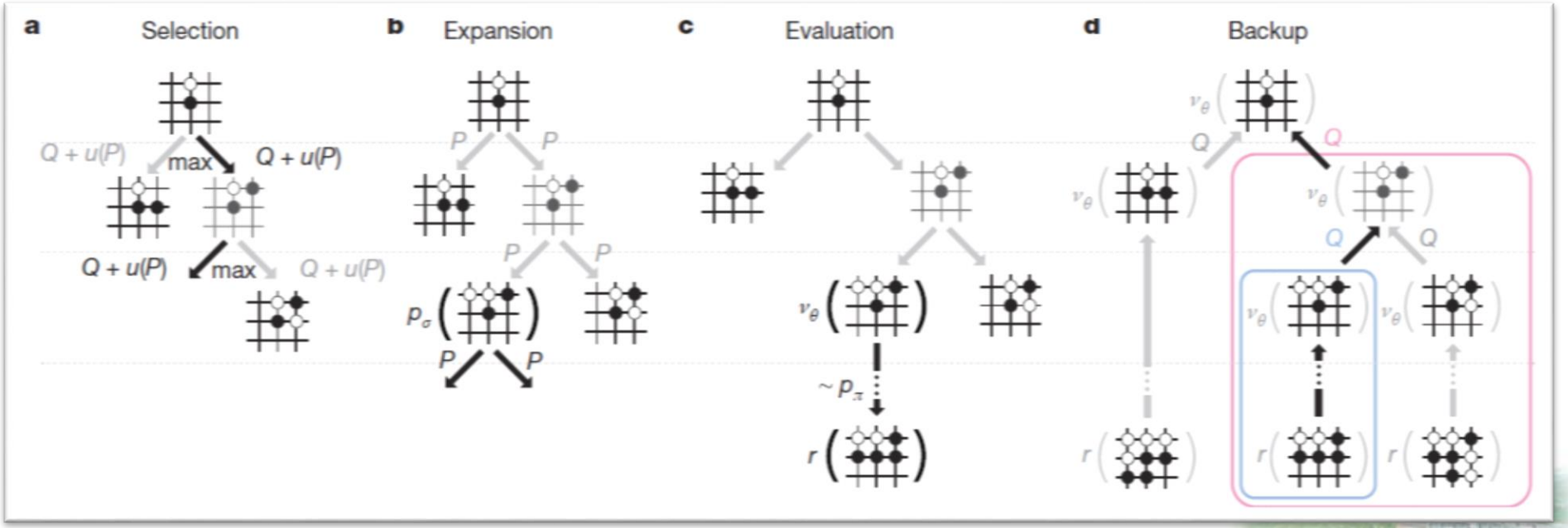
- Game tree를 일부만 탐색하는 방법
  - MCTS (Monte Carlo tree search)
    - AMAF (all-moves-as-first) heuristics
      - Helmbold, D. P., & Parker-Wood, A. (2009). All-Moves-As-First Heuristics in Monte-Carlo Go. In IC-AI (pp. 605-610).



비슷한 형제 노드들을 함께 갱신

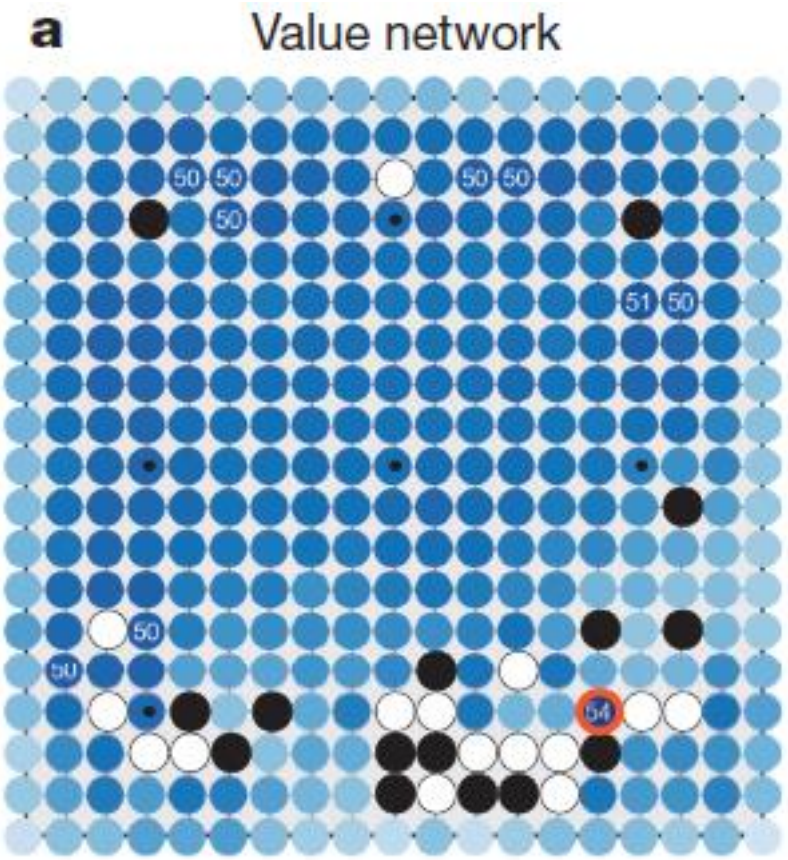
Heuristic vs machine learning

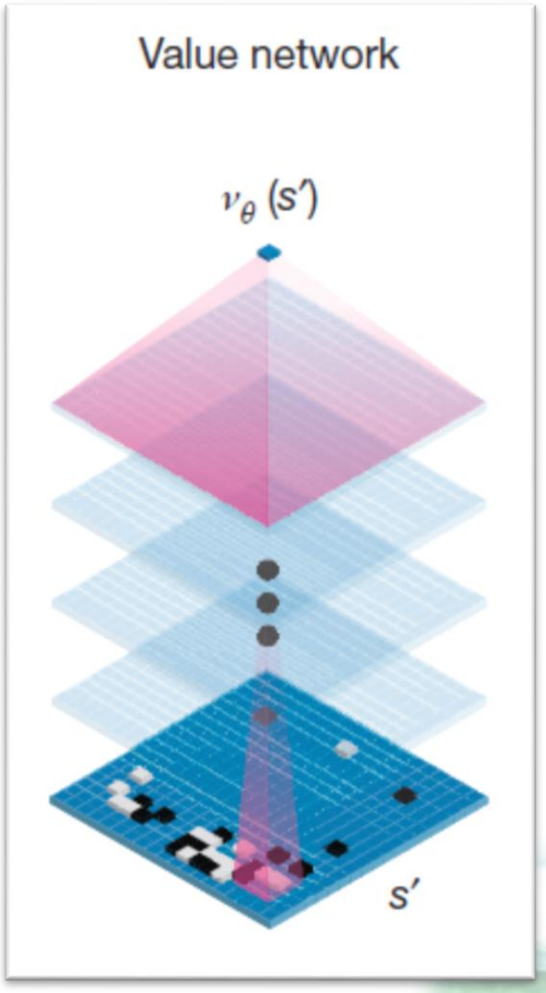
- AlphaGo
  - Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489.



Heuristic vs machine learning

- AlphaGo

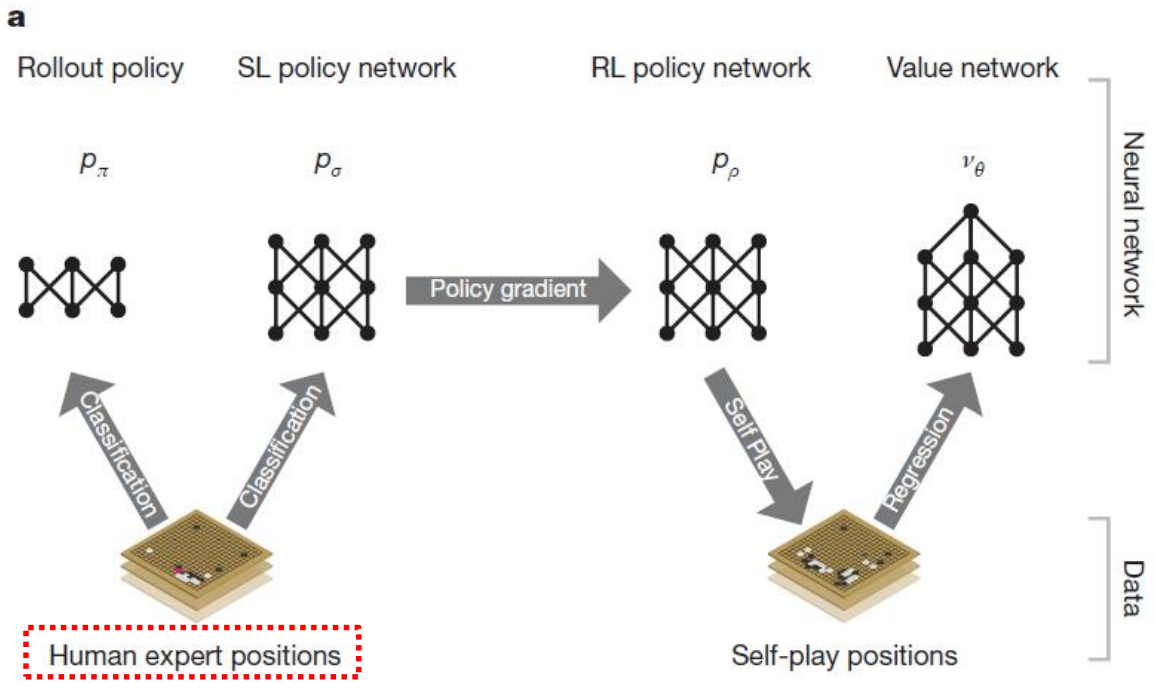


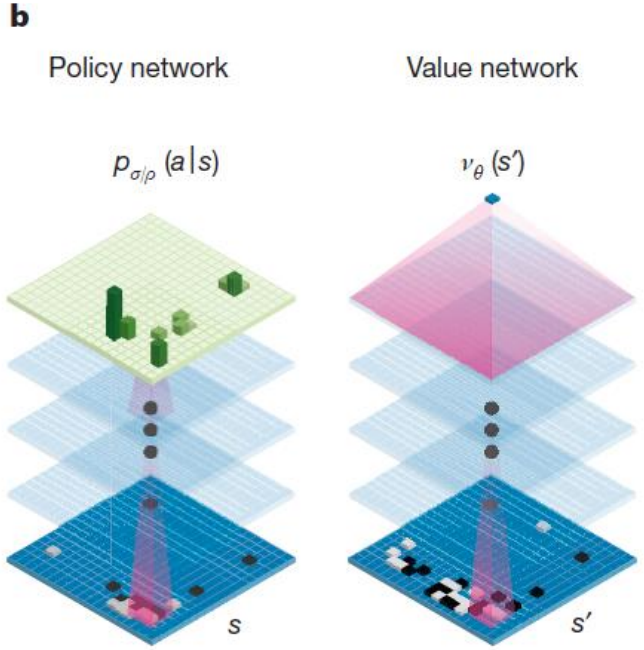


다음에 둘 수 있는 위치별로 얼마나 괜찮은 state인지 신경망으로 계산

Heuristic + machine learning

- AlphaGo





Heuristic knowledge는 사용

QnA

실습

School of Computer Science
University of Seoul

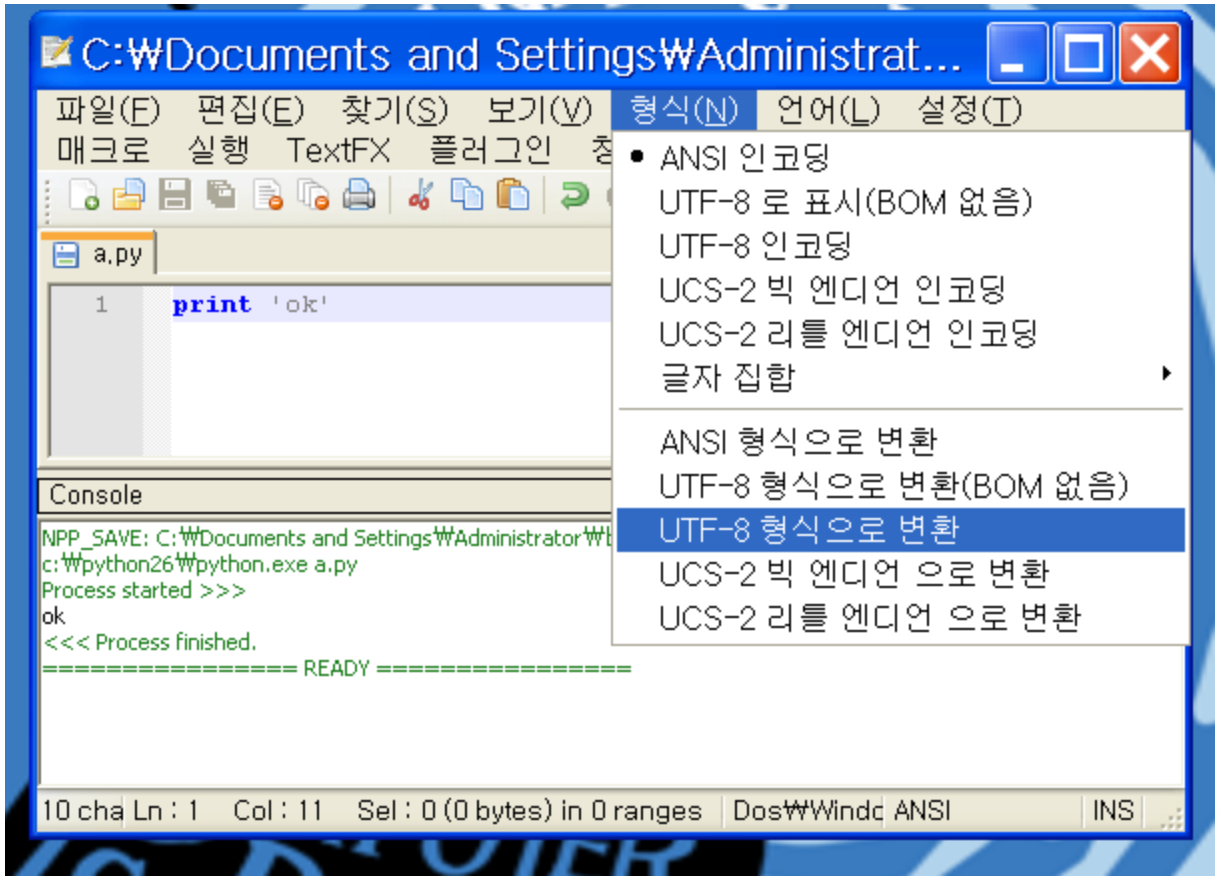
Python 기본문법

- 라이브러리 로드
  - “import [라이브러리명]”
    - 라이브러리를 통째로 로드
    - ‘[라이브러리명].[함수명]’으로 호출
  - “import [라이브러리명] as [축약할 라이브러리명]”
    - 라이브러리명을 다 쓰기 귀찮을 때 유용(권장)
    - ‘[축약한 라이브러리명].[함수명]’으로 호출
  - “from [라이브러리명] import [함수명]”
    - 라이브러리에서 특정 함수만 로드(함수명 대신 ‘*’를 붙이면 모든 함수 로드)
    - ‘[함수명]’으로만 호출
    - (권장하지 않음)
      - 서로 다른 라이브러리에 동일한 이름의 함수가 있는 경우
      - 코드 리뷰할 때 어떤 라이브러리의 함수인지 헷갈림

        ≫ eclipse pydev 환경에서 자동 분석이 잘 안될 수 있음

한글 주석 사용 방법

- Notepad++

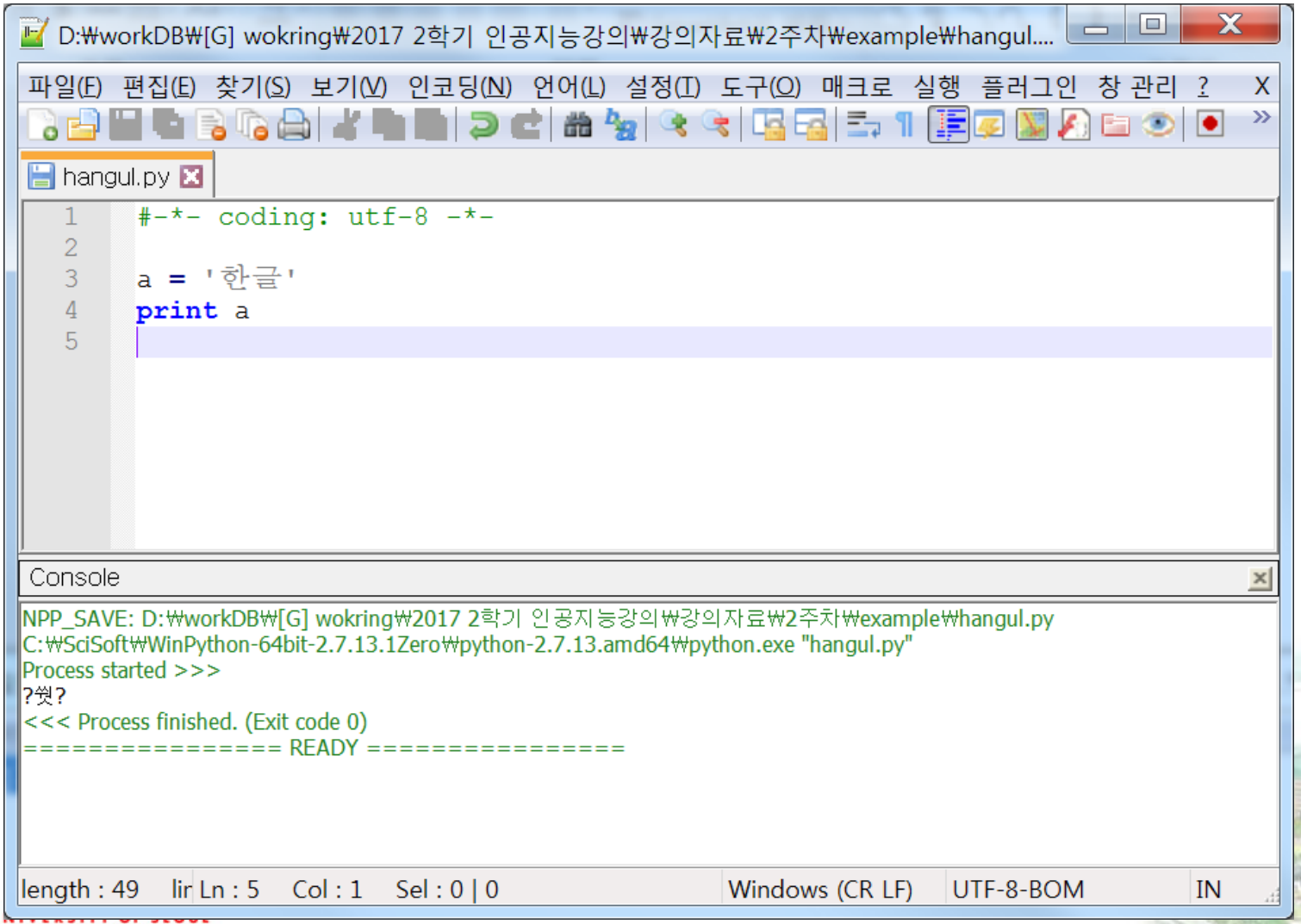


- 저렇게 해도 안 되거나 다른 편집기 사용시 코드 첫 줄에 주석 추가

```
#-*- coding: utf-8 -*-
```

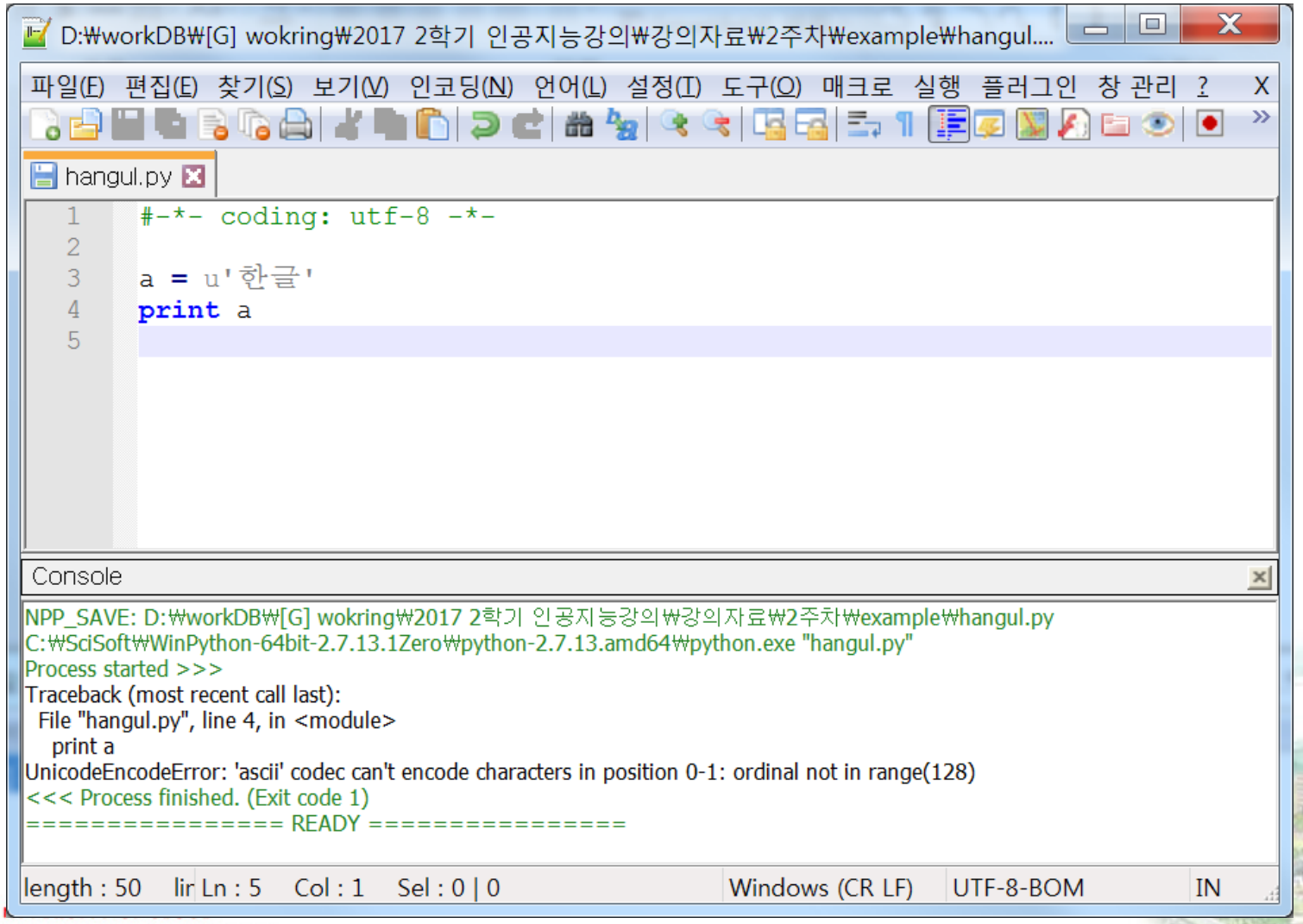
한글 출력 방법

- 문자열을 그대로 출력시



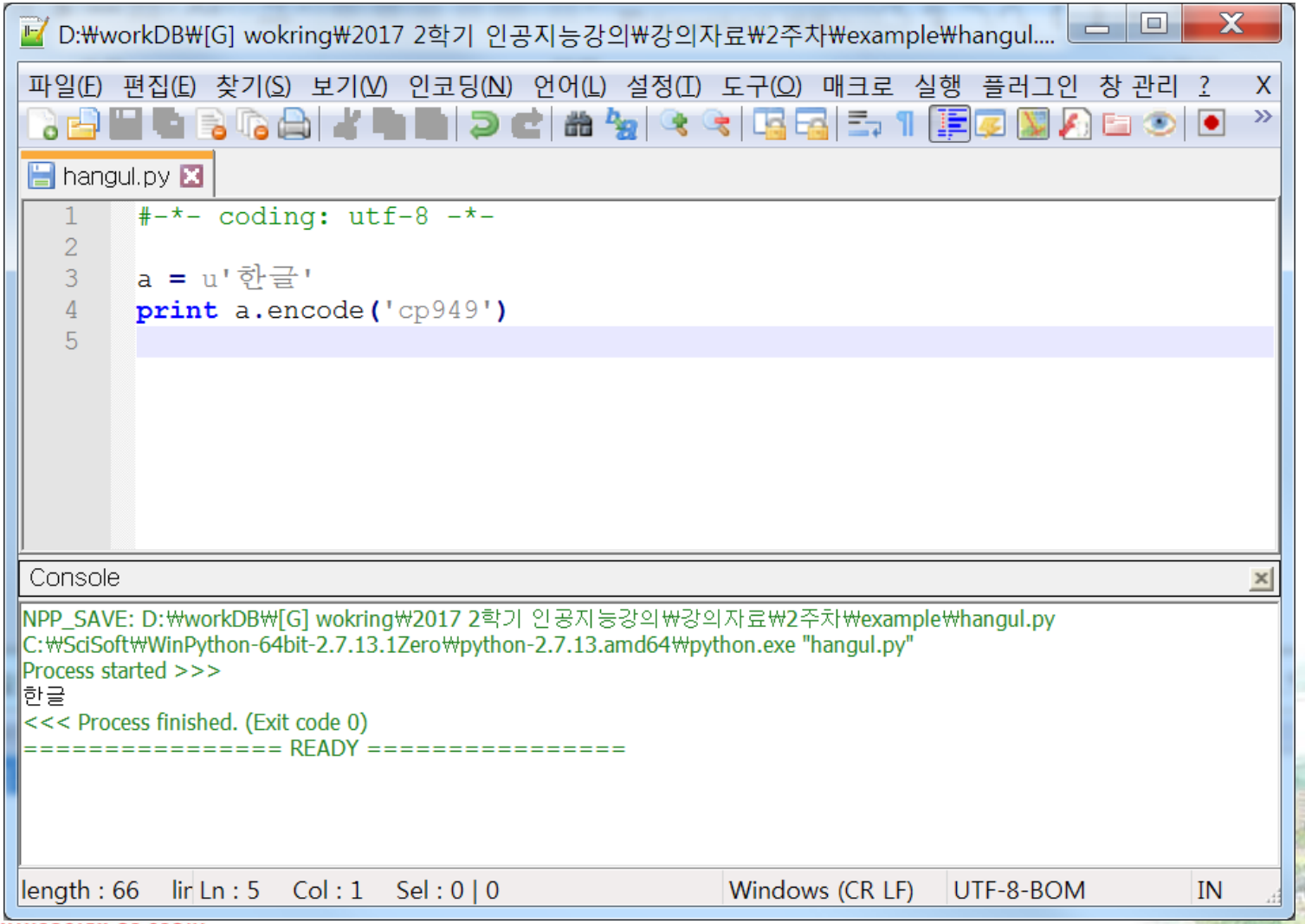
한글 출력 방법

- 유니코드 문자열로 정의하여 출력시



한글 출력 방법

- 유니코드 문자열로 정의 / 출력할 때 인코딩 지정



한글 출력 방법

- Console 출력
  - IDE / 실행 환경에 따라 인코딩을 다르게 해야 할 수 있음
    - 어떤 경우는 유니코드로 되고, 어떤 경우는 utf-8 / cp949
  - Console 출력은 영문만 하는 것을 권장

- GUI 출력(Tkinter 사용시)
  - 유니코드 문자열로 정의하면 큰 문제 없이 출력됨

Python 실습 – 리스트 다루기

```
a = [1, 2, 3, 4, 5]

a.append(6)
a.append(7)
a.append(8)

print a
```

리스트 맨 뒤에 추가하기

```
a = [1, 2, 3, 4, 5]
b = [6, 7, 8]

a.extend(b)

print a
```

리스트끼리 연결하기

Python 실습 – 리스트 다루기

```
a = [1, 2, 3, 4, 5]
b = [6, 7, 8]
a = a[2:5]

print a

b = b[1:-1]

print b
```

리스트의 일부만 선택하기

```
a = [5, 2, 3, 4, 1]

print sum(a)

a.sort()

print a
```

리스트 내용 계산하기

Python 실습 – 문자열 다루기

```
a = 'apple'
a = a[1:]

print a
```

```
a = 'apple'

if a == 'apple':
    print 'ok'
```

문자열도 일종의 리스트

문자열 비교

Python 실습 – 형 변환

```
a = '1'
b = 2

print int(a) + b
```

문자 -〉 정수형 숫자

```
a = 'apple'
b = 1
c = 6.55

d = '%s %d %.2f' % (a, b, c)

print d
```

문자열로 치환하기

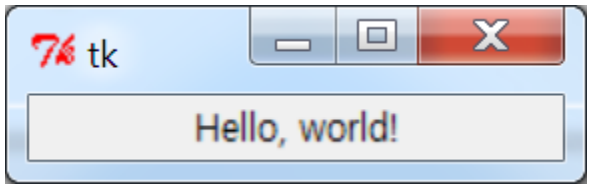
Python Tkinter 모듈

- Python을 이용한 GUI 프로그래밍 모듈
- 참고 페이지
  - http://effbot.org/tkinterbook/tkinter-index.htm

Python Tkinter 모듈

- Hello World 예제

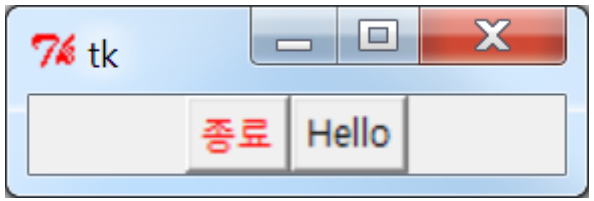
```
# Tkinter 모듈 import
from Tkinter import *

# Tk() 인스턴스(root widget, 기본 윈도우) 생성
root = Tk()

# root widget에 레이블 widget 생성
w = Label(root, text="Hello, world!")

# 생성한 레이블 widget 등록 (실제 윈도우에 붙여서 표시)
w.pack()

# root widget(기본 윈도우)가 끝나지 않도록 반복
root.mainloop()
```



Python Tkinter 모듈

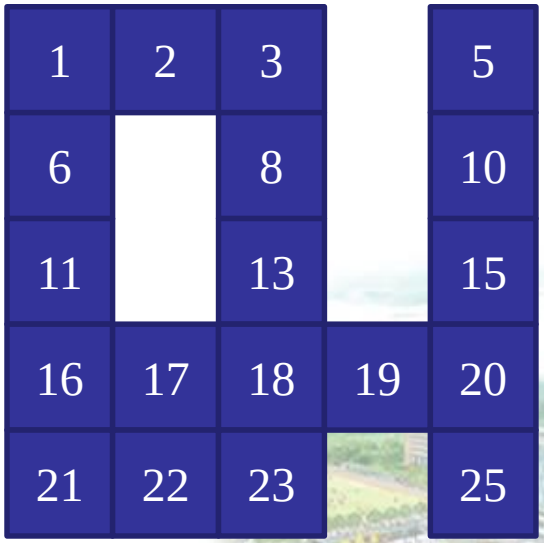
- Hello World 예제
  - Class를 정의하는 경우

```
from Tkinter import *

class App:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.button = Button(frame, text=u'종료', fg="red", command=frame.quit)
        self.button.pack(side=LEFT)
        self.hi_there = Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)

    def say_hi(self):
        print "hi there, everyone!"

root = Tk()
app = App(root)
root.mainloop()
```



실습과제 (A* search 구현)

- 다음 기능을 구현(총 10점)
  1. 아래와 같은 미로 구조를 python 코드로 표현
  2. 11을 시작점으로 하여 15까지 A* search로 이동(5점)
     - 좌우상하 4방향으로만 이동 가능
     - 탐색하는 중간 과정 출력(현재 위치, f(n), g(n), h(n)등)
     - 최종적으로 탐색을 몇 번 수행한 끝에 도착했는지 탐색 횟수 출력
  3. python Tkinter 모듈을 이용하여 GUI 구현(5점)
     - 미로 출력
     - 최종적으로 찾아낸 하나의 이동 경로 출력
       (중간 과정은 console에서만 출력해도 됨)



QnA