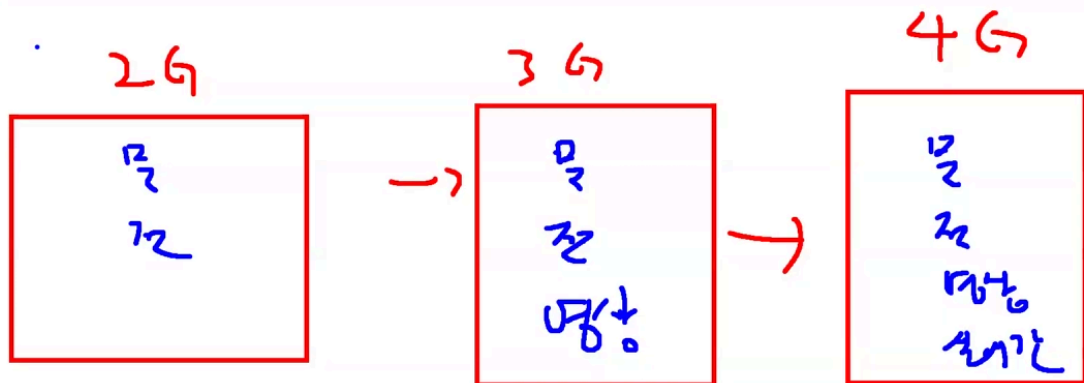
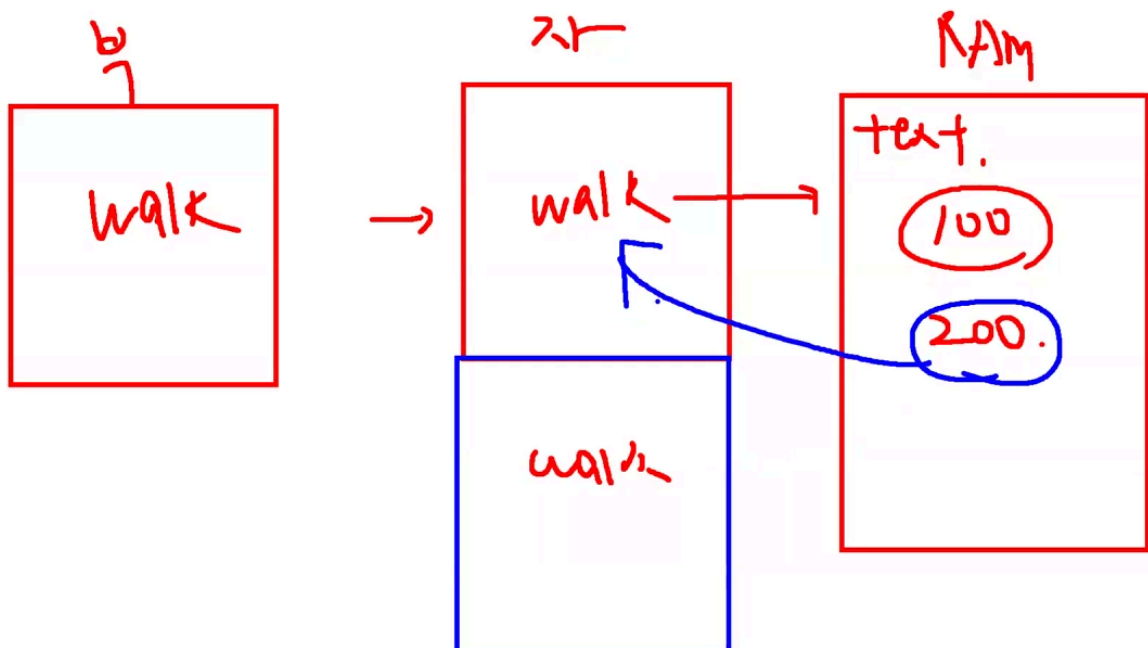


상속 (inheritance), Polymorphism, Overriding (간단정리)

상속



매번 같은 코드를 쓰는것보다 상속을 받아서 같은 기능에다가 더 추가를 하는게 효율적이다.



자식은 항상 메모리 상의 부모를 먼저 호출해야한다. 부모는 자식에게 모든 메서드를 물려준다. 여기서 Walk는 또 다시 생성되는게 아니라 부모가 가지고 있는거 그대로 덮어쓰기(overriding을 한다)

예외/오류 처리

- 1) 컴파일 오류(개발자가 가장 좋아하는 오류 - 눈에 이 부분이 오류가 나는것을 알수 있기때문)
 - a) 빨간줄이 막 뜨는 오류
- 2) 빌드 오류
 - a) 빨간줄은 없는데 실행과 동시에, 어떤 프로그램이 만들어질때 발생하는 오류
- 3) Runtime
 - a) 코드상에도 결함이 없고 실행상에서도 문제가 없는데 사용자가 쓰는 환경에서 생기는 오류들

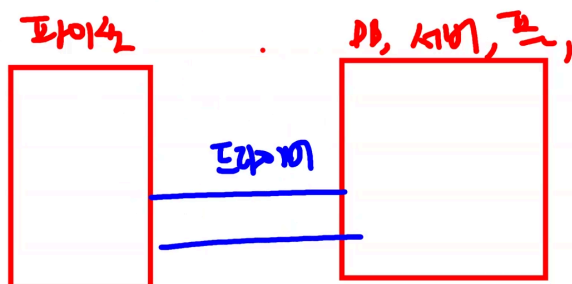
이 오류들을 핸들링하기 위해서 쓰는게 **Try, except** 이다:

```
1 try:
2     num = int(input('사용자에게 정수를 입력해주세요!'))
3     print(10 / num)
4 except ValueError:
5     print("정수만 입력할 수 있습니다.")
6 except ZeroDivisionError:
7     print("0으로 나눌 수 없습니다")
8 except Exception:
9     print("알 수 없는 오류 발생!")
10 else: #선택
11     print("예외 발생하지 않음! 😎")
12 finally:
13     print("무조건 한 번 실행! 😊")
14
```

****Exception** 오류는 모든 오류를 핸들링한다.

Finally 코드란?

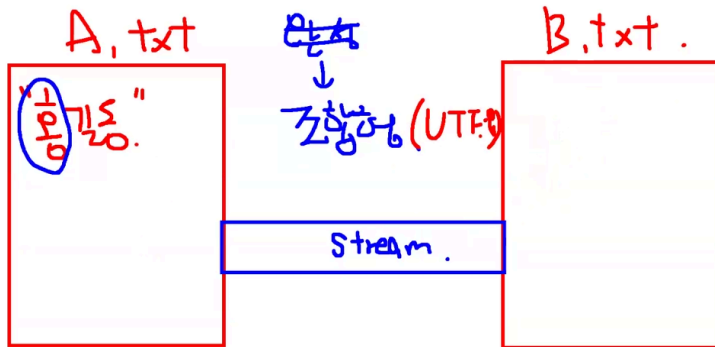
파이썬 프로그램과 서버가 연결되어있는 도중 예외가 발생했을때 양쪽 다 피해가 가지 않도록 연결되어있는 드라이버를 닫는 역할을 한다. (프로그램을 종료하는 역할)



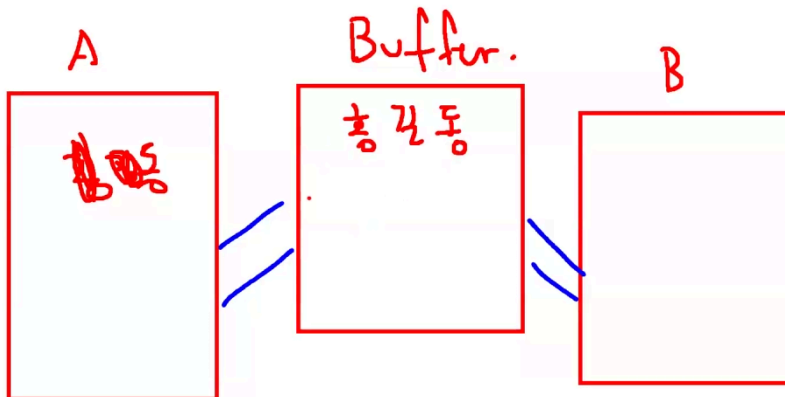
인코딩

- 안에 있는 내용을 0과 1로 즉, 바이트로 바꾸는 작업.
- ex) 줌 비디오를 인코딩해서 로컬로 저장

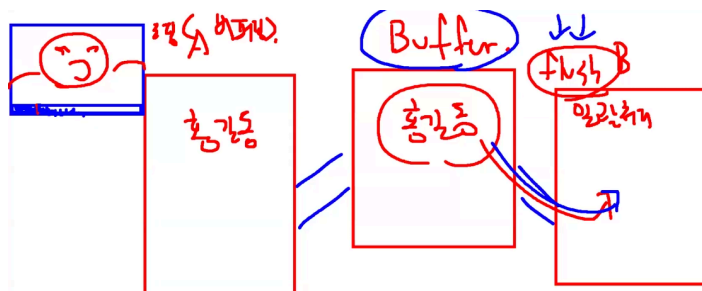
A.txt에서 B.txt로 데이터를 옮길때 데이터스트림을 거쳐서 인코딩을 통해 옮겨진다.
한국어는 주로 UTF-8 방식을 거친다 (조합형, 한 글자씩 옮기기). 완성형은 조합된 글자를 옮기기



파일을 옮기는 과정에서 오류가 발생했을시 피해를 최소화하기 위해서 버퍼를 쓴다.
옮기는도중에 오류가 났을경우 버퍼에 있었던 파일을 다시 원래대로 되돌린다.



버퍼에서 목적지로 보내는 과정을 **일괄처리** 라고 하고 버퍼를 비우는 과정을 **flush**라고 한다.
ex) 유튜브 로딩(버퍼링) - 버퍼에서 작업을 일괄로 보내는 중 (버퍼링이 짧을수록 좋음)



파일 입출력

open ~ close문

- 파일 열기

파일객체(f) = open(파일명, 모드, 인코딩 방식)

- 파일 내용 읽기

변수 = 파일객체.read()

- 파일 쓰기

파일객체.write(str)

파일객체.write("쓰고싶은 내용")

- 파일 종료

파일객체.close()

-finally에 작성이 되어야함

파일 입출력 모드(옵션)

r : 읽기 모드 (default)

w : 쓰기 모드, 파일이 없다면 만든다. 있으면 덮어 씌운다.

a : 쓰기 모드, 파일이 없으면 만든다. 있으면 맨 뒤에 이어 쓴다.

x : 쓰기 모드, 파일이 없으면 만든다. 있으면 오류를 발생시킨다.

t : 텍스트 모드, 우리 눈에 읽힌다.

b : 바이너리 모드, 음성, 이미지, 영상.

코드 예시:

<https://colab.research.google.com/drive/1wDW7I9R7hWjy0akimKP7DKYLZYJ2sdqs#scrollTo=6wFNhnN2N1Tk>

절대 경로와 상대 경로

- 절대 경로 : 처음부터 끝까지 모든 주소를 나열한 것
- 상대 경로 : 내 위치를 기준으로 주소를 나열한 것

ex)

```
C:\workspace>cd a
C:\workspace\Wa>cd f
C:\workspace\Wa\Wf>cd C:\workspace\Wc\Wd\We
C:\workspace\Wc\Wd\We>cd ../../../../a/f
C:\workspace\Wa\Wf>
```

Handwritten notes in yellow: "절대" (Absolute) above the first command, "내" (My) above the second, and "상대" (Relative) above the third. A yellow arrow points from the "상대" note to the relative path in the fourth command. A red box highlights the relative path in the fourth command.

/: 최상위 폴더

./: 현재폴더

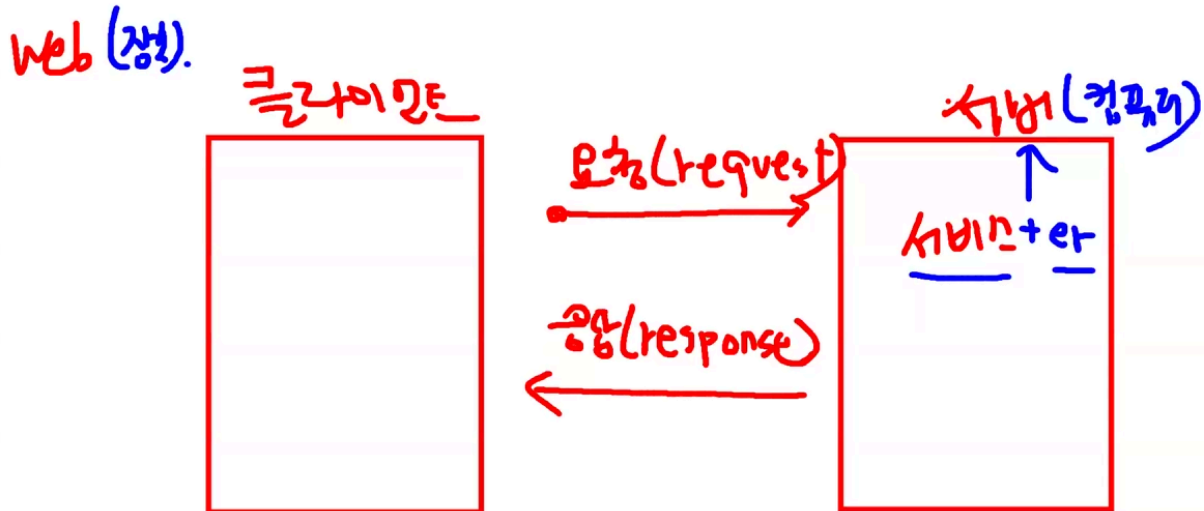
../: 상위폴더

Python2 Day-02

웹 크롤링(스크래핑)

서버와 클라이언트

- 서버는 컴퓨터이다.
- 클라이언트들의 요청(request)에 맞는 응답(resoponse) 처리를 해 준다.



웹(web)

- 요청과 응답이 이루어지는 장소 (서버와 클라이언트의 공간)
- 수억명 수만명들이 요청과 응답을 받기 때문에 마구잡이로 거미줄 모양으로 되어있기 때문에 웹 이라고 부른다.

웹 브라우저

- chrome, edge 등 인터넷에서 웹 서버의 모든 정보를 볼 수 있도록 하고, 문서 검색을 도와주는 응용 프로그램이다

URI(Uniform Resource Identifier)

- 프로토콜://도메인:포트번호/경로
- ex)
<https://sports.news.naver.com/basketball/index>
프로토콜(http)://도메인:포트번호(sports.news.naver.com)/경로(basketball/index)
- 프로토콜부터 포트번호까지를 URL (Uniform Resource Locator)이라고 부르며, 경로만 특정할 때 URI라고 한다.

도메인

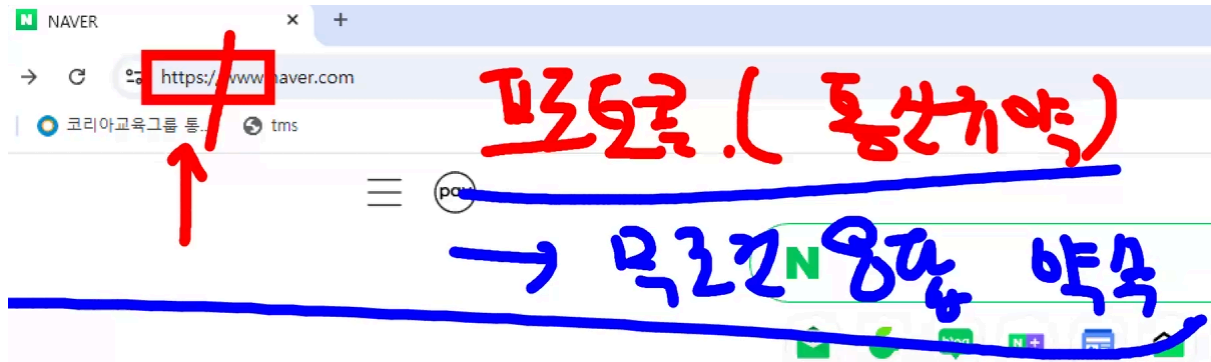
ex) naver.com
IP 대신에 사용하는 웹 상의 별칭

IP

PC의 고유한 주소값 ex) 198.234.321.32

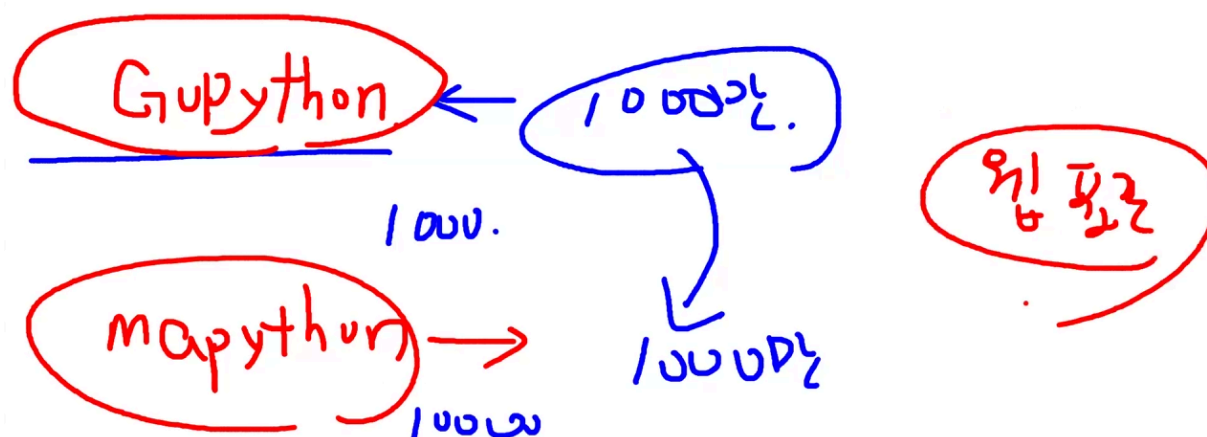
프로토콜

- 프로토콜이란 통신 규약이며, 무조건 응답 해주겠다는 약속이다.
- https라는 프로토콜이 붙어 있으므로 네이버는 어떤 값이든 응답을 해준다.
- https는 ID, PW 등 data를 암호화 시켜주는 프로토콜이다 (해커로부터 안전함).



WWW(World Wide Web), W3C

- www이란 웹표준을 지정을 한다.
- 웹 표준이라는것은 웹 안에서 약속을 지키자는 뜻이다.
- 웹 표준을 지정함으로써 웹 안에서는 모두가 공평하게 지정한 언어를 사용할 수 있다. (벤처사의 독식을 막을 수 있다)
 - ex) 구글, 마소 등에서 언어를 만들어 기업끼리 독식을 할 수 있다.
- W3C는 웹 표준을 정의하는 기관이다.



HTML(Hyper Text MarkUp Language)

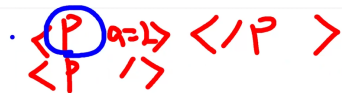
- 웹 페이지를 만드는 문법을 갖춘 언어, 태그(Tag)로 이루어져 있다.
- Markup: 웹 표준 (어디서든 똑같이 번역됨)
- Markdown: 웹 표준이 아님 (똑같이 번역되지 않음)

태그(Tag)

- 어떤 의미를 지니고 있다는 뜻
- 1) 여는 태그(Opening tag) : 요소의 이름과 열고 닫는 꺾쇠 괄호로 구성됨 <html>
- 2) 닫는 태그(Closing tag) : 요소의 이름 앞에 슬래시(/)를 써서 구성 </html>
- 3) 내용 (Content)요소의 내용이며, 단순한 텍스트 이다.
- 4) 요소(Element) : 여는 태그와 닫는 태그, 내용을 통틀어서 이르는 말

ex)

- 밑의 사진에서 태그의 이름은 P 이다.
- 여는 태그는 <P>, 닫는 태그는 </P>
- 이 전체를 통틀어서 element 라고 부른다

· 

태그의 속성

- HTML은 구조화된 언어이다. 쉽게 말하자면 각 태그는 고유한 기능을 가지고 있다. 하지만 추가적으로 기능이 필요할 때마다 태그를 만든다면 태그들의 종류가 다양해질 것이다. 그래서 재사용과 사용 목적에 따라 다르게 사용할 수 있도록 속성이 존재한다.

<태그명 key=value ></태그명>

key, value 한 쌍으로 이루어져 있으며, value에 따라 적용되는 값이 다르다.

ex)

```
▼<div class="image">
  ::before
  
```

· 

- id : 중복되지 않은 태그의 특징적인 속성 (고유 속성으로 하나만 존재한다)
- class : 태그들을 그룹화 하는 공통적인 속성 (공통 요소들을 묶기 위해서 사용한다)

ex) 여기서는 공통된 class = 'link_news_end' 를 사용한다.

```
<a href="https://m.sports.naver.com/wbaseball/article/382/0001184656" onclick="clickcr(this, 'pop.article', '', '', event);" class="link_news_end">아깝다 김혜성' 베츠, 도쿄시리즈 불기, '美' 소가 귀국</a> == $0
```

```
<a href="https://m.sports.naver.com/wbaseball/article/023/0003893900" onclick="clickcr(this, 'pop.article', '', '', event);" class="link_news_end">177kg 참치 해체쇼 선사한 오타니, 다저스 동료에 호화 일식 썼다</a> == $0
```

기본적인 HTML의 구조

```
<html>
  <head></head>
  <body></body>
</html>
```

크롤링이란?

- 여러 웹 페이지를 기계적으로 탐색하는 것

스크래핑이란?

- 특정한 하나의 웹 페이지를 탐색하고, 소스 코드 작성자가 원하는 정보를 얻어내는 작업

주의사항

- 크롤링을 무분별하게 사용하면 과부하를 일으킬 수 있다.
- 상업적인 용도나 불법적인 용도로 사용 시 법적 문제가 발생할 수 있다.

웹 크롤링 패키지

- requests : 파이썬에서 동작하는 작은 브라우저



```
1 import requests
2 url = 'https://www.naver.com'
3
4 # .get(): 페이지 요청
5 requests.get(url)
```



<Response [200]>

200: 서버 응답이 잘 넘어갔을때

400: 클라이언트쪽에서 응답이 안넘어갔을때

500: 서버쪽에서 문제가 생겼을때

requests의 메서드

- 1) .encoding() : 인코딩 설정
- 2) .status_code : 상태 코드
- 3) .text : 웹 페이지 소스
- 4) .context : 웹 페이지 소스(모든 문자)

BeautifulSoup

- html 태그들을 보기 쉬운 형태로 처리해주는 라이브러리
- 1) find('tagname') : 태그 중에서 태그명이 'tagname' 인 첫 번째 것
- 2) find('tagname').text : 위에 묶음 중 내용만 가져오기
- 3) find('tagname', class_='클래스속성명') : 'tagname' 인 것 중 클래스 속성명인 것의 첫 번째 것
- 4) find('tagname', id='아이디속성명') : 'tagname' 인 것중 아이디 속성명인 것
- 5) find_all('tagname') : 태그들 중 태그명이 'tagname' 인 모든 것을 리스트로 가져옴

User-Agent

- 사용자의 소프트웨어 식별 정보

user-Agent의 필요성

- 무분별한 크롤링으로 서버의 과부하를 막기 위해 프로그램을 통해서 접속하는 것을 차단하는 사이트 들이 있다. 그런 경우 원하는 정보를 추출할 수 없기 때문에 header에 user-agent 정보를 심어야한다.

내 User-Agent를 확인하기

- 1) 브라우저
 - a) <https://www.useragentstring.com/>
- 2) 운영체제
 - a) https://www.whatismybrowser.com/detect/what-is-my-user-agent/#google_vignette

코드 예시:

https://colab.research.google.com/drive/1vA-UpFB_mw9s27QXG2PQkyz1ajkmNuLQ#scrollTo=DnDzek2WEKGP

Python2 Day-03

Numpy

Python2 Day-03

Numpy 마무리

Python2 Day-07

자료구조와 알고리즘 [Linked list]

컴퓨터

- 판정, 선택, 응답, 기억 네 가지 주요 회로로 구성된 기계

프로그래밍

- 문제해결을 위하여 수행할 작업을 단위 작업으로 나누고 어떻게 수행해야 문제를 해결할 수 있는지 연구하는 것

단위작업

- 개발자 입장에서 더이상 분해할 수 없는 기본 작업

코딩

- 프로그래밍된 것을 특정 언어로 번역하는 과정

알고리즘

- 문제를 해결해가는 절차 또는 순서

자료구조

- 컴퓨터가 다루어야하는 자료가 많은 경우에 이것을 다루는 방법
- 알고리즘으로 구현하기 위해 사용된다.

물리적으로 구현하는 방법 : 리스트, 연결리스트

- 리스트(List)
- 각 데이터를 연이어 저장하는 기술
- 연결리스트(Linked List)
- 각 데이터를 임의의 위치에 저장하고 서로를 연결하는 기술

자료구조의 분류: 단순구조, 선형구조, 비선형구조, 파일구조

자료구조(Data Structure)

- 컴퓨터에서 다루는 데이터 형

단순구조(Simple Structure)

- 배열 : 동일한 기본 자료형을 여러개 모인 것
- 구조체 : 기본 자료형이 여러개 모인 것
- 클래스 : 구조체에서 함수까지 모여있는 자료형으로 선언한 것
- 재정의 자료형 : def 등을 이용해 이름을 재정의하거나 넣을수 있는 종류를 제한한 것

선형구조(Linear Structure)

- 데이터들이 일렬로 쭉 저장되어있는 형태를 가짐
- 일렬로 저장하는 방식 : 리스트(배열 기반, 연속 방식)
- 각 데이터가 다음 데이터의 위치를 가지는 방식
- 연결리스트(포인터기반, 연결방식)
- 리스트와 연결리스트 외에 방법에 따라 : 스택, 큐, 데크

비선형구조(Non-Linear Structure)

- 데이터가 트리형태로 저장되어 있다고 생각하고 사용하는 자료구조 (자료가 순차적이지 않고, 연결관계를 갖는 형태)
- 방식: 트리, 그래피

파일구조(File Structure)

- 다양한 자료구조의 데이터를 파일에 저장하는 방식
- 방식: 순차파일, 색인파일, 직접파일
- 자료구조의 분류 이미지 제작

추상 데이터 자료형(ADT, Abstract Data Type)

- 각 자료구조의 구조와 기능이 어떻게 작동하는지 확인하기 위해 만들어놓은 추상 데이터 타입이다.

- 사용 설명서 같은 것이다.

- 가방 추상 데이터 자료형(ADT)

- 자료(data) : 핸드폰, 거울, 지갑, 동전, 손수건 넣을수 있는 저장소
- 연산 : 넣고 빼고 연산할 수 있는 insert(), remove(), 개수를 검사
- 데이터 : 중복된 항목을 허용하는 자료들의 저장소, 항목들을 특별한 순서없이 개발적으로 저장되지만 항목간의 비교는 가능해야한다.
- 연산
 - bag() : 비어있는 가방
 - insert(e) : 가방에 항목에 e를 넣는다.
 - remove(e) : 가방에 e가 있는지 검사하여 항목을 꺼낸다
 - contains(e) : e가 들어있는지 있으면 True, 없으면 False
 - count() : 가방에 들어있는 항목수를 반환한다.

알고리즘 성능 분석

1) 알고리즘의 효율성

- 계산속도와 메모리 사용량으로 효율성을 평가한다.

2) 시간 복잡도

- 특정 알고리즘이 어떤 문제를 해결하는데 걸리는 시간을 의미한다.
- 같은 결과를 나타내는 소스라면 최대한 시간 적게 걸리는 것이 좋은 소스 효율을 가지고 있으며,

알고리즘 구성을 위해 시간 복잡도 측면을 고려하고 중요하게 본다.

3) 점근적 표기법을 사용

- 오메가 표기법
 - 최상의 경우
 - 가장 빠르지만 이 상황을 기대할 수 없는 경우
- 세타 표기법
 - 평균의 경우
 - 일 년 동안 버스를 기다린 시간의 평균적인 경우
- 빅오 표기법(Big-O)
 - 최악의 경우
 - 커질수록 차수가 가장 큰 항의 영향이 절대적이다.
 - 알고리즘이 복잡할수록 평균 복잡도를 구하기 어려워서 최악의 경우로 알고리즘의 성능을 파악한다.

- 시간 복잡도 표기

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

$O(1)$ - 상수 시간 : 문제를 해결하는데 오직 한 단계만 처리함.

$O(\log n)$ - 로그 시간 : 문제를 해결하는데 필요한 단계들이 연산마다 특정 요인에 의해 줄어듦.

$O(n)$ - 직선적 시간 : 문제를 해결하기 위한 단계의 수와 입력값 n 이 1:1 관계를 가짐

$O(n \log n)$: 문제를 해결하기 위한 단계의 수가 $N \cdot (\log_2 N)$ 번 만큼의 수행시간을 가진다.

선형로그형

$O(n^2)$ - 2차 시간 : 문제를 해결하기 위한 단계의 수는 입력값 n 의 제곱.

$O(C^n)$ - 지수 시간 : 문제를 해결하기 위한 단계의 수는 주어진 상수값 C 의 n 제곱

단일 연결리스트(Singly Linked List)

- 배열의 공간 낭비를 피할 수 있는 자료구조
- 링크를 이용해서 리스트를 만든다는 뜻이다
- 노드(Node)들이 서로 연결된 형태로 구성된 선형 자료구조
 - *노드(Node)란? 데이터(값) + 다음 노드의 주소
 - 요소를 저장하는 data필드와 다음 노드를 가리키는 next필드로 구성되어 있다.
 - data 필드 : 정수, 실수, 숫자, 복잡한 객체
 - next 필드 : 다음 노드를 가리키는 링크(자바에서는 레퍼런스, C언어 포인터)

a) 연결리스트의 구성

- 연결리스트의 접근은 첫번째 노드부터 시작
- 첫번째 노드를 밟은 다음 링크(next)를 타고 그 다음 노드들에 차례대로 접근하는 방식이다.
- 연결리스트의 객체는 리스트의 노드들을 직접 저장할 필요가 없고 리스트의 첫번째 노드에 접근할 수 있는 레퍼런스만 갖고 있으면 된다.

head

- [데이터][주소값] → [데이터][주소값] → [데이터][주소값]

b) 연결리스트 종류

- 싱글(단일) 연결리스트
- 더블(이중) 연결리스트
- 원형 연결리스트

c) 노드 생성, 삭제, 삽입

- 단순연결리스트는 다음 데이터를 가리키는 링크가 필요
- 노드를 먼저 만들고 링크를 연결

연결리스트 필요 연산

- insert(i, x) : x를 연결리스트의 i번째 요소로 삽입(맨 앞은 0번)
- append(x) : x를 연결리스트의 맨 뒤에 삽입
- pop(i) : 연결리스트의 i번째 요소를 삭제하면서 알려줌
- remove(x) : 연결리스트에서 처음으로 나타내는 x를 삭제
- get(i) : 연결리스트의 i번째 요소를 알려줌
- index(x) : 요소 x가 연결리스트 몇 번째 요소인지 알려줌
- isEmpty() : 연결리스트가 빈 리스트인지 알려줌(True, False)
- size() : 연결리스트의 총 요소수를 알려줌
- clear() : 연결리스트를 깨끗이 비워줌
- count(x) : 연결리스트에서 요소 x가 몇 번 나타나는지 알려줌
- extend(a) : 연결리스트에서 나열할 수 있는 객체 a를 풀어서 추가함
- copy() : 연결리스트를 복사해서 새 연결리스트를 반환함
- reverse() : 연결리스트의 순서를 역으로 뒤집음
- sort() : 연결리스트 정렬함
- 추가, 수정, 삭제, 검색, 총요소개수, 총요소출력 확인

Python2 Day-08

자료구조와 알고리즘 [linked list]

단일 원형 리스트

- 단순 연결리스트의 마지막 노드가 다시 첫 번째 노드를 가리키도록 설정되어 리스트의 원형(circle)형태로 구성되는 자료 구조
- 계속 회전하면서 연속으로 찾아갈 수 있음
- 큐(queue) 또는 게임 개발 환경에서 많이 사용된다.

구성 환경

[데이터][다음 조회값] → [데이터][다음 조회값] → [데이터][첫 번째 주소값]

단일 연결리스트, 이중연결리스트, 원형연결리스트의 장단점

단일 연결리스트의 장점

- 메모리를 효율적으로 사용할 수 있음
- 노드의 삽입, 삭제 $O(1)$ 시간 복잡도
- 구현이 비교적 간단하고 적은 메모리를 사용한다

단일 연결리스트의 단점

- 특정 노드를 찾으려면 처음부터 찾아야되므로(순회) 탐색 $O(n)$ 시간복잡도
- 단방향으로만 이동이 가능하므로 역방향으로 이동은 어렵다

이중연결리스트의 장점

- 양방향으로 이동이 가능하므로 노드 검색이 용이하다
- 노드의 삽입, 삭제 $O(1)$ 시간복잡도
- 양방향으로 연결되어 있어서 역방향 탐색이 편리하다

이중연결리스트의 단점

- 단일연결리스트보다 메모리를 더 많이 사용한다
- 구현이 복잡하고 코드의 양이 많아질 수 있다

원형연결리스트의 장점

- 특정 노드에서 시작하여 전체 노드의 순회할 때 노드의 수 만큼 순회하면 원래 위치로 돌아올 수 있다
- 마지막 노드와 첫 번째 노드가 연결되어있어서 순회가 용이하다
- 원형적인 구조로 알고리즘에서 활용할 수 있는 이점을 제공한다

원형연결리스트의 단점

- 특정 노드를 탐색하려면 전체 순회해야하므로 탐색에 $O(n)$ 시간복잡도
- 삭제 연산이 복잡해진다.

Stack(스택) 자료구조

- 데이터가 입력되는 순서대로 쌓고, 나중에 들어온것부터 먼저 사용하는 자료구조
- 후입선출, 즉 LIFO(Last In First Out)
- ex) 스택에 A, B, C, D를 순서대로 입력했다면 꺼낼 때는 D, C, B, A 순으로만 꺼낼 수 있다.

스택 ADT

- 맨 윗부분에 요소를 추가한다 (스택에 포함)
- 맨 윗부분에 있는 요소를 알려준다 (스택에 포함)
- 맨 윗부분에 있는 요소를 삭제하면서 알려준다 (스택에 포함)
- 스택이 비어있는지 확인한다. (모든 자료구조에 포함)
- 스택이 깨끗하게 비운다 (모든 자료구조에 포함)

스택 주요 함수

- push() : bottom부터 차례대로 삽입된다.
- pop() : top부터 차례대로 삭제되며 top에 해당하는 값을 반환한다.
- peek() : top에 있는 요소를 알려준다(검색)
- clear() : 스택에 있는 모든 요소를 삭제한다(removeAll)

스택 필요연산

- create(size) : 스택을 생성(초기화)
- is_empty() 임피티 : 스택이 비어있다면 True, 아니라면 False
- is_full : 스택이 가득 차있다면 True, 아니라면 False
- push(item) : 만약 size를 초과한다면 False
- pop() : top에 위치한 item을 반환하면서 제거
- peek() : top에 있는 요소를 반환한다.

top이란?

- 스택의 윗부분(push, pop)이 이루어지는 부분

bottom이란?

- 스택의 아랫부분 (즉, 인덱스가 0인 부분)

capacity

- 스택의 최대크기(int형)
- 스택 포인터 ptr : 스택에 쌓여있는 데이터의 개수를 나타내는 정수값을 의미한다.

알고리즘 대회, 코테

- 괄호 검사 문제 (), [], {}
- 수식 계산 (중위표기법, 후위표기법)
- 미로 탐색
- 회문 탐색

큐(Queue)

- 가장 먼저 넣은 데이터를 가장 먼저 꺼내는 구조
- 선입선출형, FIFO(First In First Out)형
- 큐의 구현 → 배열을 이용(순환 큐)
- 연결리스트 이용(링크드 큐)

큐 자료 구조의 용도

- 데이터의 흐름 제어하는데 사용(프로세스 간 통신에서 메시지를 전송, 네트워크 트래픽 조절 등)
- BFS(Breadth First Search) : 그래프 자료에서 사용되는 대표적인 탐색알고리즘

큐 자료 구조의 활용 예시

- 프린터의 출력처리(스폴링)
- 프로세스 관리 등 작업 스케줄링

큐의 ADT

- 맨 끝에 요소를 추가한다 (큐)
- 맨 앞의 요소를 삭제하면서 알려준다. (큐)
- 맨 앞의 요소를 알려준다 (큐)
- 큐가 비어있는지 확인한다
- 큐를 깨끗하게 비워준다

큐의 주요 함수

- enqueue : 큐 뒤쪽에서 항목을 삽입한다.
- dequeue : 큐 앞쪽의 항목을 반환하고 제거한다
- peek/front : 큐가 비어있는지 확인한다.
- empty : 큐가 비어있는지 확인한다
- size : 큐의 크기를 확인한다

큐의 필요연산

- create(size) : 큐를 생성 초기화
- is_empty() : 큐가 비어있다면 True, 아니라면 False
- is_full() : 큐가 가득차있다면 True, 아니라면 False
- enqueue() : 큐에 데이터를 추가
- dequeue() : 큐에 데이터를 삭제
- peek() : 맨 처음 데이터를 반환 (디큐에서 꺼내질 데이터)

원형큐 (링버퍼)

- 선형 큐의 문제점인 (디큐할 때 배열 안의 요소를 옮겨야 하는 문제점)을 해결할 수 있어 사용한다
- 배열 맨 끝의 요소와 맨 앞에 요소가 연결되는 자료구조이다.
- 디큐할 때 배열 안에 요소를 옮기지 않는 큐를 구현할 때 사용한다.
- 원형큐는 크기가 항상 고정이다.
- rear : 가장 최근에 삽입된 항목의 위치를 저장
- front : 가장 최근에 삭제된 항목의 위치를 저장
- 맨 처음에는 $front = rear = 0$

삽입연산 : rear 하나 증가시킨 후 그 위치에 항목을 넣는것

삭제연산 : front 하나 증가시킨 후 그 위치의 항목을 반환

Python2 Day-08

자료구조와 알고리즘 [hash table]

Hash Table

- Dictionary (key: value) 형태로 구성된 자료구조
- 키를 해시함수에 넣으면 나오는 고유함수를 인덱스로 사용하여 테이블에 값을 지정하거나 검색한다.

해시테이블의 장점

- 자료의 검색, 읽기, 저장속도가 빠르다.
- 자료가 중복되는지 확인하기 쉽다 (set 자료 구조의 특징을 갖는다)

해시테이블의 단점

- 보통 저장공간이 더 필요하다
- 충돌을 해결할 방법이 필요하다

해시테이블 충돌 해결 방법

- 서로 다른 키가 같은 해시코드로 매핑되는 상황
 - 1) 체이닝
 - a) 각 해시함수를 연결리스트로 구현하는 방법
 - 2) 어드레싱
 - a) 충돌이 발생했을 때

Algorithm -sort

정렬 알고리즘

- 정렬 : 자료들을 일정한 순서대로 나열하는 것을 의미한다.
- 기본 정렬
 - 선택정렬, 삽입정렬, 버블정렬
- 고급 정렬
 - 병합정렬, 퀵정렬, 힙정렬, 셸정렬
- 특수 정렬
 - 계수정렬, 기수정렬, 버킷정렬

안정정렬과 불안정정렬

- 안정 정렬 : 중복된 값을 입력된 순서와 동일하게 정렬한다.
- 불안정 정렬 : 중복된 값을 입력된 순서와 상관없이 무작위로 뒤섞인 상태에서 정렬한다.

[12, 6, 3, 5, 8, 10, 3]

1. 선택정렬

- 최솟값을 뽑아 첫 번째 요소와 교환하고 그 다음 작은 값을 찾아 두 번째 요소와 교환하며 정렬하는 방식이다.
- 불안정 정렬에 해당된다.

2. 삽입정렬 (=서들정렬, insert sort)

- 기존 데이터 중에서 자신의 위치를 찾아서 데이터를 삽입하는 정렬방식이다.
- 미리 정렬된 리스트에 새 항목을 추가할 때 좋은 정렬 알고리즘이다.
- 불안정 정렬, 최선의 경우 $O(n)$, 최악의 경우 $O(n^2)$ 의 시간 복잡도를 가진다.

3. 버블정렬

- a. 인접한 두 수를 비교하여 큰 수를 오른쪽으로, 작은 수를 왼쪽으로 교환하며 정렬하는 방식
- b. ex) [1, 3, 5, 2, 9]
 - 1와 3비교
 - 3과 5를 비교
 - 5와 2를 비교
 - 5와 9를 비교
 - 다시 반복 ..
 - 안정정렬, 시간 복잡도 $O(n^2)$

4. 병합정렬(Merge Sort)

- a. 분할정복과 재귀를 이용한 알고리즘이다.
- b. 데이터를 최소 단위로 쪼갬 후 병합하면서 정렬하는 방식이다.
- c. 추가적인 메모리 공간이 필요하다
- d. 안정정렬, 시간복잡도 $O(n \log n)$

5. 퀵정렬(Quick Sort)

- a. 분할정복알고리즘을 이용한 방식으로 평균적으로 빠른 속도를 수행한다
- b. 데이터를 기준점(pivot)을 중심으로 좌우로 분할하며 정렬하는 방식이다
- c. 불안정정렬, 평균 시간 복잡도 $O(n \log n)$, 최악 시간 복잡도 $O(n^2)$
- d. 재귀함수로 구현한다

Data Structure - Tree

트리

- 계층적으로 자료를 표현할 때 적합한 자료구조
- 데이터를 순차적으로 저장하지 않기 때문에 비선형 자료구조
- 트리 내에 또 다른 트리가 있는 재귀적 자료구조
- 하나의 루트 노드와 0개 이상의 하위 트리로 작성됨

트리의 사용 예시

- 계층적 데이터 저장: 파일 및 폴더
- 효율적인 검색속도: 효율적인 삽입, 삭제, 검색을 위해 사용
- 데이터 베이스 인덱싱 구현
- 힙(Heap)

트리 용어 정리

- 루트(Root)노드 : 트리의 가장 높은 곳에 있는 노드
- 모든 노드는 루트노드의 서브트리에 속한다
- 리프(Leaf)노드 : 트리의 가장 하단에 있는 노드
- 자식노드가 없는 노드
- 자식(Child)노드 : 동일한 부모를 가진 노드들을 의미함
- 리프노드가 아닌 자식이 있는 노드 = 내부(Internal)노드, 비 단말(Non-Terminal)노드
- 형제(Sibling)노드 : 동일한 부모를 가진 노드들을 의미한다
- 조상(ancestro)노드 : 루트까지 경로상에 있는 모든 노드들의 집합
- 후손(Descendant)노드 : 노드 아래로 매달린 모든 노드들의 집합
- 서브트리(Subtree)노드 : 노드 자신과 후손 노드로 구성된 트리
- 차수(Degree) : 어떤 노드가 가지고 있는 자식의 수
- 레벨(Level) : 루트에서 얼마나 멀리 떨어져 있는지를 나타내는 것
- 키(Key) : 탐색에 사용되는 노드에 저장된 정보

이진트리(Binary Tree)

- 각 노드의 자식 수가 2이하인 트리(모든 노드가 2개의 서브트리를 갖는 트리)
- 컴퓨터 분야에서 활용되는 가장 기본적인 자료구조
- 데이터의 구조적인 관계를 잘 반영한다
- 효율적인 삽입과 탐색을 가능하게 한다
- empty이거나 empty가 아니면 루트와 2개의 이진트리인 왼쪽 서브트리와 오른쪽 서브트리로 구성된다.

이진트리의 종류

- 포화 이진트리
 - 모든 리프의 깊이가 같고 각 내부노드가 2개의 자식노드를 가지는 트리
 - 즉 노드 개수가 $2^{(높이 + 1)} - 1$ 공식으로 계산
 - 이진트리의 번호를 루트노드부터 왼쪽에서 오른쪽으로 번호를 부여한다
- 완전 이진트리
 - 마지막 레벨을 제외한 각 레벨이 노드들로 꽉 차있고 마지막 레벨에는 노드들이 왼쪽부터 빠짐없이 채워진 트리
- 편향 이진트리
 - 왼쪽이나 오른쪽 서브트리만 가진 트리를 의미한다

이진트리 순회

- 이진트리의 노드 전체를 한 번씩 방문하는 것을 의미한다
- 자료구조를 효율적으로 활용하는 방법은 데이터를 효과적으로 검색하는 것이다
- 노드 데이터를 처리하는 순서에 따라 전위순회, 중위순회, 후위순회를 사용할 수 있다

- 순서 트리 탐색

[0]

[1][2]

[3][4] [5][6]

깊이 우선 탐색(DFS, Depth-First Search)

- 전위 순회(preorder traversal)

- 노드의 데이터를 먼저 처리한다
- 현재 노드 데이터를 처리 → 왼쪽 서브트리로 이동 → 오른쪽 서브트리로 이동
- root → left → right
- 0 → 1 → 3 → 4 → 2 → 5 → 6

- 중위 순회(Inorder traversal)

- 노드 데이터를 중간에 처리한다
- 왼쪽 서브트리로 이동 → 현재 노드데이터 → 오른쪽 서브트리
- left → root → right
- 3 → 1 → 4 → 0 → 5 → 2 → 6

[0]

[1][2]

[3][4] [5][6]

- 후위 순회(postorder traversal)

- 노드의 데이터를 마지막에 처리함
- 왼쪽 서브트리로 이동 → 오른쪽 서브트리로 이동 → 현재 노드데이터를 처리
- left → right → root
- 3 → 4 → 1 → 5 → 6 → 2 → 0

[0]

[1][2]

[3][4] [5][6]

너비 우선 탐색(BFS, Breadth-First Search)

- 레벨 단위로 왼쪽부터 오른쪽까지 순차적으로 탐색하는 방법
- 수평검색, 가로검색

이진 탐색 트리(BST, Binary Search Tree)

- 탐색을 위한 이진트리 기반의 자료구조
- 이진 탐색은 데이터를 항상 정렬된 상태로 유지해야되기 때문에 삽입과 삭제가 많은 경우 효율적이지 않다.
- 트리의 연산들을 단순하게 설계하기 위해서 중복을 허용하지 않는다.

이진 탐색 트리의 정의

- 모든 노드는 유일한 키를 갖는다
- 왼쪽 서브트리의 키들은 루트의 키보다 작다
- 오른쪽 서브트리의 키들은 루트의 키보다 크다
- 왼쪽과 오른쪽 서브트리도 이진 탐색트리이다