2020 POSCO 청년 AI Bigdata Academy 12th - B4 문인균

목차: 원하는 항목을 클릭하세요.

AI 맨하탄 프로젝트와 알파고

> 인공지능 개발을 위한 세기적 계획. 인공지능과 관련된 IT기업과 전문가들이 구글 속으로 빨려들고 있다.

2014년 1월 초, 스마트 온도조절기 업계 선두주자인 '네스트랩'을 32억달러에 매입했고, 2014년 상반기에만 10여개의 인공지능 관련 기업이 직간접적으로 구글의 프로젝트에 발을 담갔다.

2014년 1월 26일, 사원이 75명에 불과한 신생기업 딥마인드가 구글에 인수되었다. 인수 가격은 6억 5천만달러로, 당시 인수전에는 MS와 IBM도 나섰다고 알려져있다.

이로부터 약 2년 뒤, 구글은 대한민국의 프로 바둑기사 이세돌 9단과의 대국을 성사시켰고 결과는 4승 1패로 알파고의 완승이었고, 이 역사적인 대국은 우리의 미래에 대한 깊은 고민과 인공지능 분야에 대한 눈을 뜨게 만들었다.

머신러닝의 3가지 타입

지도 학습(Supervised learning)

> 컴퓨터가 입력값과 그에 따른 출력값이 있는 데이터를 활용하여 주어진 입력에 맞는 출력을 찾는 학습 방법.

예를 들어 개와 고양이 사진을 구분할 때 입력은 사진이고, 출력은 개 또는 고양이인지의 여부가 된다. 개인지 고양이인지의 여부가 기록된 사진(훈련 데이터)을 이용해 지도학습을 하며, 학습 결과는 훈련 데이터에 포함되지 않은 사진을 구분하는 데 적용된다.

비지도 학습(Unsupervised learning)

> 기계 학습 중 컴퓨터가 입력값만 있는 훈련 데이터를 이용하여 입력들의 규칙성을 찾는 학습 방법.
>
> 다시말해, 정답이 없는 데이터 속에서 스스로 규칙을 찾아내는 방법이다.

예를 들어 강아지, 고양이, 기린의 사진을 비지도 학습으로 분류한다고 가정할 때, 각각의 동물들이 어떤 동물인지 정답을 알려주지 않았기 때문에 이 동물을 '무엇'이라고 정의할 수는 없지만 동물의 특징(Feature)별로 분류를 하게 된다.

강화 학습(reinforcement learning)

> 강화학습은 지도/비지도 학습에 이용되는 훈련 데이터 대신, 주어진 상태에 맞춘 행동의 결과에 대한 보상을 준다. 컴퓨터는 보상을 이용하여 성능을 향상시킨다.

일단 해보면서, 경험을 통해 실력을 키워나가는 것. 그 행동의 결과가 자신에게 유리한 것이었다면 상을 받고, 불리한 것이었다면 벌을 받는것이다. 이 과정을 반복하여 더 많은 보상을 받을 수 있는 더 좋은 답을 찾아낼 수 있다는 것이 강화학습의 기본 개념이다.

Programming Introduction

운영체제와 프로그래밍 언어

운영체제

> 시스템 하드웨어를 관리할뿐 아니라 응용 소프트웨어를 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공하는 시스템 소프트웨어이다.

- 실행되는 응용 프로그램들이 메모리와 CPU, 입출력 장치 등의 자원들을 사용할 수 있게 함
- 이들을 추상화하여 파일 시스템 등의 서비스를 제공
- 멀티태스킹을 지원하는 경우, 모든 프로세스들을 스케줄링하여 동시에 수행되는 것 처럼 보이는 효과 제공

초기의 컴퓨터는 굉장히 크고 단가가 높았는데, 주로 공용으로 쓰였다. 컴퓨터와 물리적 거리가 있더라도 선으로 연결해서 공동작업이 가능했는데, 이와 같은 작업이 가능하기 위해서 하드웨어 위에서 작동하며 사용자와 소통할 수 있게 해주는 소프트웨어(OS)가 필요했다.

- 멀티 유저 지원
- 타임쉐어링 / 메모리 매니지먼트 기능

프로그래밍 언어

UNIX - 1969

- 미국 벨(Bell)연구소에서 개발된 소프트웨어 개발용의 운영체제
- 1969년에 원형이 완성되었지만 1973년에 대부분이 C 언어로 수정되었다.
- 다중 사용자/다중 태스크의 실행을 지원하는 **대화형 운영체제**
- 텍스트 조작 출, 문서 처리, 전자 메일 외에 취급이 쉬운 파일 시스템을 갖추고 있다.

C - 1972

> 멀티유저를 지원하는 OS에 대한 수요로 인해 만들어진 언어. 하드웨어를 얼마나 최적화 할 것인지를 중점으로 하며, 하드웨어를 컨트롤하는 모든 문법을 가지고 있다.

- 벨 연구소에서 리치(Dennis Ritchie)등에 의해 개발된 시스템 프로그래밍 언어
- UNIX의 OS시스템 기술에 사용할 것을 목적으로 설계
- 컴퓨터의 구조에 밀착한 기초 기술이 가능, **간결성**을 특징으로 한다.

DOS -1981

> 컴퓨터가 소형화되고 널리 보급되기 시작하던 시기에, 문자열 기반 사용자 인터페이스의 DOS가 탑재되어 널리 보급되었다. 그러나 명령어를 외워서 사용해야 하는 문제로 인해 사용이 어려웠다.

- DOS는 디스크에 운영체제를 저장, 디스크를 중심으로 시스템을 관리
- 만들어지던 당시에 여러 운영체제 중 하나였으나, IBM과 제휴를 맺은 뒤, IBM컴퓨터의 보급이 세계화되자 운영체제는 MS-DOS라는 공식이 생겨났다.
- 기본 메모리 640Kb에 최대 1Mb를 가지며, 16Bit 운영체제로 최대 12문자의 영문표기와 7문자의 한글 표기가 가능하다.
- 주로 하드디스크에 설치하여 사용하지만, 플로피디스크를 사용하기도 한다.

C++ - 1983

- 80년대까지 널리 사용된 C언어가 한계에 봉착, 고도로 복잡한 프로그램을 관리하기 위해 C++가 탄생한다.
- 처음 소개된 이후 3번에 걸쳐 개정되었다. 1994년에 최초 표준안이 발표되었다.
- ANSI C++ 위원회는 C개발자가 설정한 모든 사양을 그대로 수용하고, 약간의 사양을 덧붙였다. 따라서, C개발자가 C++를 쉽게 사용할 수 있다는 점에서 큰 장점이 있다.
- **C의 대부분의 특징을 포함**, 시스템 프로그래밍에 적합할 뿐만 아니라 클래스, 연산자 중복, 가상함수 등과 같은 특징을 갖추어 **객체 지향 프로그래밍에 적합**하다.

애플 매킨토시 - 1984

- 전문 지식이 없는 사람이라도 손쉽게 사용할 수 있도록 **인터페이스를 중시**한 설계가 특징
- 출시 초기, 사용하기 쉬운 PC로 인기가 높았으나 타 기종과 호환성 문제로 PC로서 IBM보다 점유율이 떨어져 전문 분야에서만 이용이되었다.
- 당시, 다른 컴퓨터들은 DOS명령어로 컴퓨터를 구동시켜야 했는데 아이콘, 메뉴, 마우스 등을 갖춘 매킨토시의 GUI시스템은 컴퓨터의 사용을 확실히 수월하게 만들었다.

Windows - 1985

> GUI로 사용이 쉬운 매킨토시에 대항하기 위해 DOS에 Windows라는 OS를 입혀 GUI로 재탄생했다.

- GUI 운영체제인 매킨토시에 대항하여 당시 널리 쓰이던 MS-DOS에 멀티태스킹과 GUI환경을 제공하기 위한 응용 프로그램으로 출시했다.
- 95년 32비트를 도입한 Windows 95부터 DOS가 아닌 독자 운영체제로 사용되고 있다.
- 현재 전 세계 PC 90%이상에서 사용되며 서버용 OS로도 영역을 확대하고 있다.

LINUX - 1991

> 당시 소프트웨어 개발을 하기 위해서는 로열티를 지불해야하는 등 진입장벽이 높았다. 당시 형성된 개발자 네트워크에서 소스코드를 업그레이드하고 공유하는 움직임이 사회 운동처럼 번지기 시작했다.

- 1989년 리누스 토르발스가 UNIX를 기반으로 개발한 OS로, 1991년에 일반에 공개되었다.
- 소스코드를 완전 무료로 공개하여 전세계적으로 5백만 명이 넘는 프로그램 개발자 그룹을 형성하게 되었으며, 이들에 의해 다수를 위한 공개라는 원칙하에 지속적 업그레이드가 이루어지고있다.
- 인터넷 프로토콜인 TCP/IP를 강력하게 지원하는 등 **네트워킹에 강점**을 지닌다.

분산 버전관리 시스템

- 리눅스 소스 코드를 완전 무료로 공개하여 전세계적으로 5백만 명이 넘는 프로그램 재발자 그룹을 형성하게 한 리누스 토르발스는 분산 버전관리 시스템 github을 개발했다.

Python - 1991

> 70년대 C와 C++를 알고있는 귀도 반 로섬. 언어를 직관적이고 쉽게 배울 수 있게 해주는 파이썬을 개발했다.

- C언어를 기반으로 한 오픈소스 고급 프로그래밍 언어로, 1991년 귀도 반 로섬에 의해 개발되었다.
- 플랫폼이 독립적이기 때문에 다양한 플랫폼에서 사용 가능하고, 기본 제공 라이브러리가 매우 많다.
- 인터프리터식 동적 타이핑 대화형 언어이며 인터프리터 형식이기 때문에 사용자가 컴파일을 하지 않고서도 작성한 프로그램을 바로 실행할 수 있다.

JAVA - 1992

> 집집마다 있는 가전제품을 통신망을 이용해 업그레이드할 수 있는 언어가 필요했다. 이에 따라 네트워크 상에서 자유롭게 코딩할 수 있는 JAVA가 개발된다.
>
> - 자바스크립트: 가상머신에서 가볍게 돌아가게 만든 언어

- 썬 마이크로시스템즈 연구원들에 의해 인터랙티브 TV용 프로그램 작성을 위해 개발
- C/C++과 유사한 문법을 가지며, C++에 비해 단순하고 효율성을 높이기 위한 기능이 추가되었다.
- 네트워크 기능이 기본 탑재되었고 하드웨어 아키텍쳐에 중립적이고 안전하게 실행되는 언어를 목표로 개발되었으며, "write one, run anywhere"라는 슬로건으로 한번 구현한 코드를 플랫폼에 독립적으로 사용할 수 있도록 설계되었다.

Android, iOS, Scratch - 2007

기계의 선을 끊으며 모바일 기기가 등장하기 시작한다. 모바일 기기에 사용될 OS의 필요성이 대두되기 시작한다.

Android - 2007

- 리눅스 2.6커널을 기반으로 OS, 라이브러리 세트, 풍부한 멀티미디어 사용자 인터페이스, 폰 어플리케이션을 제공
- 애플의 iOS, MS의 윈도 모바일 등과 달리, 완전 개방형 플랫폼이다.
- 소스 코드를 모두 공개함으로써 **누구든 소프트웨어와 기기를 만들어 판매할 수 있도록 하였다.**

iOS - 2007

- 매킨토시 OS X를 터치 기반 모바일 기기에 최적화 된 형태로 재구성하였다.
- 아이폰, 아이팟 터치, 아이패드에 탑재되는 OS로, 주로 모바일 기기와 애플 TV등에 탑재된다.

Scratch - 2006

> 이전까지 똑똑한 몇명의 사람들이 이끌어갔다면, 이제는 후세들을 위해 교육의 필요성이 대두되기 시작한다. 블록 개념으로 즐겁게 코딩을 학습하기 위한 언어가 개발된다.

- 8~16살의 어린이가 쉽게 쓸 수 있도록 설계된 프로그래밍 도구, MIT Media Lab의 LKG이 만들어 무료로 공개했다.
- 저자와 독자가 양방향 소통하는 동화 게임 등을 만들 때 유용하며 이용자가 창의적으로 생각하고, 체계적으로 판단하며 협업하는 방법을 배울 수 있게 하는 것이 개발팀의 목표였다.
- 2003년 개발 프로젝트가 시작된 이해로 150여 국가에 40여개 언어로 공개되었다.

App Inventor - 2009

- 프로그래밍을 처음 접하는 사람들이 **안드로이드 운영체제용 소프트웨어를 만들 수 있도록 하는 오픈소스 웹 애플리케이션**
- 사용자들이 시각 객체들을 드래그 앤 드랍하여 안드로이드 장치에서 실행할 응용프로그램들을 만들 수 있다.

Entry - 2013

> 우리나라에서도 소프트웨어 교육의 중요성을 깨닫고 교육용 언어를 개발하기 시작한다.

- 대한민국의 교육용 프로그래밍 언어 플랫폼, 기존의 텍스트 코딩이 아닌 블록형 언어를 기반으로 제작된 그래픽 기반 프로그래밍 프로그램
- 한국형 스크래치라고 불리기도 하지만, 플래시기반인 스크래치와 달리 HTML과 자바스크립트, 파이썬 기반으로 웹에서 사용할 수 있으며 다양한 기기를 지원한다.

- 엔트리 파이선이라는 모드를 지원하기 때문에 블록코딩에서 텍스트코딩으로 전환하여 사용할 수 있다.

Tensorflow - 2015

> 파이썬으로 딥러닝 개발시 많은 경우의수와 데이터를 feeding하는 등의 기능을 구현하기가 힘들어 딥러닝을 위한 라이브러리 형태로 개발된다.

- 구글에서 공개한 딥러닝과 머신러닝 오픈소스 소프트웨어로서, 누구나 사용할 수 있다.
- 안드로이드, iOS와 같은 모바일 환경과 리눅스, MacOS등 데스크탑, 서버시스템의 다수 CPU와 GPU에서 구동될 수 있다.
- 텐서플로우의 연산은 상태를 가지는 데이터 흐름 유향그래프로 표현된다.

Keras - 2015

- 파이썬으로 작성된 오픈소스 신경망 라이브러리. 최소한의 모듈방식의 확장 가능성에 초점을 둔다.
- 2017년 구글의 텐서플로우 팀은 텐서플로우의 코어 라이브러리에 케라스를 지원하기로 결정했다.

CNTK - 2016

- 마이크로소프트 연구조직 내부에서 프로젝트로 진행하던 **딥러닝 프레임워크**로, 2016년에 명칭을 Microsoft Cognitive Toolkit으로 변경하며 C++, C# 인터페이스를 제공하기 시작했다.
- 현재 최신버전은 2.7로, 자바 프로그램에서 CNTK 모델 평가 기능을 사용할 수 있다.
- 64비트 윈도우 뿐만 아니라 리눅스 운영체제도 지원하는 등 오픈소스로서 자리매김하고자 지속적인 업데이트를 했다.

PyTorch - 2017

> 컴퓨터 비전쪽 머신러닝을 학습하기 위해 자주 사용되는 라이브러리

- Facebook의 인공지능 연구팀이 개발한 Torch를 기반으로 한 오픈소스 머신러닝 라이브러리
- 자연어 처리와 같은 어플리케이션을 위해 사용되며, GPU사용이 가능하기 때문에 속도가 상당히 빠르다.

Why Python?

파이썬의 특징

객체 지향 언어

해결해야 할 문제를 객체라는 관점으로 분석하고 정보와 처리할 기능들을 결합하여 프로그램을 작성하는 방식의 언어

객체 지향 프로그램이란?

> 레고와 같이 부품(객체)들을 조합해서 하나의 사물(소프트웨어)을 만들어 나가는 것

- 프로그램을 구성하는 로직을 상태와 행위로 구분, 서로 연관되어있는 것 끼리 그룹화 해놓은 것

  문법과 설계

  - 객체지향 프로그래밍 학습은 크게 문법과 설계로 나뉜다.
    - 문법: 객체지향을 편하게 할 수 있도록 언어가 제공하는 기능을 익히는 것

        - 설계: 복잡한 현실을 추상화하는 것
            - 추상화: 해결할 문제를 SW적으로 단순화 시켜 만드는 행위
            - 예) 현실을 단순화하여 만든 지하철 노선도

부품화

- 메소드는 부품화의 예.
- 연관된 로직들을 결합하여 메소드를 만들고, 이 메소드들을 부품으로 하나의 완제품(프로그램)을 만드는 것
- 메소드별로 기능이 분류되어있어 코드를 찾기 쉽고 문제 진단도 빨라진다.
    - 개선 필요시 메소드만 수정하면 그 메소드를 사용하는 곳에서 동시 다발적으로 개선이 이루어진다.

      (부품화의 장점)
    - 이와 같은 점들을 한 단계 발전시킨 것을 객체지향이라고 할 수 있다. 연관된 메소드와 그 메소드가 사용하는 변수들을 분류하고 그루핑하는 것. 이 그룹핑 대상이 객체이다.
    - 비유하자면 파일과 디렉토리가 있을 때, 메소드나 변수가 파일이라면 이 파일을 그룹핑하는 디렉토리가 객체라고 할 수 있다.

은닉화, 캡슐화

부품화라는 목표는 단순히 변수와 메소드를 그룹핑한다고 해서 달성되지 않는다. 제대로된 부품이라면, 그것이 어떻게 만들어졌는지 몰라도 그 부품을 사용하는 방법만 알면 쓸 수 있어야 한다.

- 모니터가 어떻게 동작하는지 몰라도 컴퓨터와 모니터를 연결하는 방법만 알면 화면을 표시할 수 있다.
- 예) Python에서 set()이 어떻게 동작하는지는 몰라도 사용하는 법을 알면 함수 기능을 활용할 수 있다.

동적 타이핑 언어

C언어는 OS를 만드는데 사용되므로, 미리 예측 가능하면서 효율적으로 메모리를 사용할 필요가 있다. 따라서, 변수를 정의할 때 데이터 타입과 크기를 정의하는데, 이것을 정적 메모리 할당(Static Memory Allocation)이라고 한다.

그러나 파이썬은 미리 타입과 크기를 정의하지 않고 X = 10과 같이 동적으로 메모리를 할당하는 방식(Dynamic Memory Allocation)으로 구현되었다.

```
x = 100
x = 'Hello!!!'
```

동적 타이핑은 이렇게 단순한 코드를 만들 수 있지만, 변수에 원하는 값이 제대로 할당되었는지 감시할 수 없다는 단점이 있다.

인터프리터 언어

컴파일 방법에는 두 가지 방식이 있다.

- 코드를 모두 적고 한번에 컴파일 하는 방법 - 컴파일러 언어
- 한줄씩 컴파일하는 방법 - 인터프리터 언어

자바와 같이 넷상에서 빠르게 해석되어야 하는 경우 인터프리터 기법을 사용한다.

플랫폼 독립적인 언어

운영체제에 상관없이 사용할 수 있다.

Computational Thinking

컴퓨팅 사고를 통해 일상생활의 문제를 해결할 수 있다. 그 단계는 추상화와 구조화로 나뉜다.

추상화

복잡한 현실의 문제를 컴퓨터가 이해할 수 있는 형태로 나타내는 것. 관리 가능한 작은 문제들로 나눈다.

- 일반적으로 변수 하나하나에 넣는 것을 많이 쓴다.
- 같은 자료를 여러개 넣을 때는 배열
- 행 열로 되어있다면 2차원 배열
- 서로다른 자료형 여러개 있는 방법은 클래스

구조화

정보를 효율적으로 처리하고 관리하기 위해 자료를 정리하고 재배치하는 것. 그래프, 차트, 그림 등으로 구조화

- 체계적인 정보 관리 가능
- 원하는 정보 쉽게 파악
- 효율적인 정보 관리 가능

변수와 메모리, 그리고 함수(call by address & reference)

함수

> 일반적인 프로그래밍 언어에서 함수가 매개변수를 호출하는 방식에는 두가지가 있다.

- Call by value (값 호출)
- Call by reference (참조 호출)

정적 타이핑인 C에서는 변수를 정의할 때 어떤 것(값 혹은 참조)으로 정의할지 미리 정의한다. 참조호출의 경우

```
int *a
int b
```

첫번째 줄과 같이 포인터 변수 설정을 해서 정수a의 주소값에 접근가능하도록 설정하고, 값 호출의 경우 변수명만을 입력해준다.

하지만 파이썬은 이 두가지 방법을 혼합해서 사용한다.

변수

- 파이썬은 모두 객체이므로 변경여부 관리가 중요하다.

- 객체를 생성, 함수의 파라미터로 전달할 때에도 변경이 가능한(mutable) 객체와 불가능한(immutable)객체를 동일한 방식으로 관리한다.
    - Immutable: 변수에 저장된 것을 값으로 인식. 치환 가능, 변경 불가능

      Immutable 객체가 함수의 arguments로 전달되면 처음에는 call by reference로 받지만, 값이 변경되면 call by value로 동작한다.

        - 수치값
        - 문자열
        - 튜플
    - mutable: 변수에 저장된 것은 객체의 요소(값)들이 저장된 참조. 치환 가능, 변경 가능

      mutable 객체가 함수의 argument로 넘어가면 call by refernce로 동작한다. 즉, object reference가 전달되어 actual parameter의 값에 영향을 미칠 수 있다.

        - 리스트
        - 딕셔너리
        - 집합형

          > 이 객체들은 값을 모아놓은 것들이므로 함수 영역으로 넘길 때 주소를 넘기는 것이 효율적이라는 생각에서 call by reference로 동작한다.

파이썬은 호출된 객체(파라미터, 인수)에 대한 변경이 있을 경우. 즉, 값 추가(append)연산에 대해서만 Call by reference로 동작한다.

```
def spam(eggs):
    eggs.append(1) # call by reference로 동작 - caller scope까지 작용
    #**값을 추가할 때: 원본이 바뀐다.**
    eggs = [2, 3] # call by value로 동작, 새로운 객체를 가르킨다 - caller scope까지는 적용X
    #**값을 변경할 때: 원본 바뀌지 않음**
    # assignment operator가 선언과 동시에 배정의 역할을 하고 있다. append까지는 인자로 받은 eggs를 가리키고있다가, = 연산자를 만나면서 새로운 객체가 생성된 것

    print(eggs)

ham = [0] -> [0, 1]
spam(ham)
print(ham)
>>> [2, 3]
>>> [0, 1]
```

- 위의 예에서 list형인 eggs는 mutable이므로 함수 내부에서 연산시 call by reference로 동작하지만, 같은 이름의 객체를 새로 지정해주면 call by value로 동작한다. 변경을 원치 않을 경우 튜플을 쓰는 등의 안전장치를 사용해야한다.
- mutable value를 처리할 경우 처음 값이 변경되지 않으려면 참조만 복사하지 말고 전체 값을 복사해야 다른 값 객체로 인식한다.

  deepcopy를 이용하여 참조만 복사하지 않고 전체 값을 복사한다.

deep copy

```
# 1. 리스트 깊은 복사
a = [1,2,3]
b = a[:]
id(a), id(b)
>>> (2634897772232, 2634897772104) # 다른 객체로 인식
```

```
a[0] = 4
a,b
>>> ([4, 2, 3], [1, 2, 3])
```

```
# 2. copy모듈 사용
import copy
a = [1,2,3]
b = copy.deepcopy(a)
a[0] = 4
a, b
>>> ([4, 2, 3], [1, 2, 3])
```

```
list(range(2,2+1))
>>> [2]
```

shallow copy

```
a = [1,2,3]
b = a
id(a), id(b)
>>> (2146638945608, 2146638945608)
```

```
a[0] = 4
id(a), id(b)
>>> (2146638945608, 2146638945608)
```

```
print(a)
print(b)
>>> [4, 2, 3]
>>> [4, 2, 3]
```

파이썬2와 파이썬3의 차이

> 파이썬3: 모든 것이 객체로 바뀌었다.(변수, 함수 모두)
>
> 2버전과 3버전은 자동변환이 안되어 3버전이 개발된 초기에는 2버전을 쓰는 사람들이 많았다.

- 파이썬2 혹은 C++같은 경우, 변수 함수 객체 따로 지정하였다. 그러나 파이썬3은 보따리 하나에서 클래스로 생성하는것은 객체로 정의하여 단위를 맞춰주었다. 그로인해, 프로그래밍할 때 객체들을 조합하는 과정이 용이하다.

```
a = 10
type(a)
>>> <class 'int'> # python3에서는 변수, 함수 모두 클래스로부터 선언된 것 모두 '객체'로 처리된다
```

PageRank

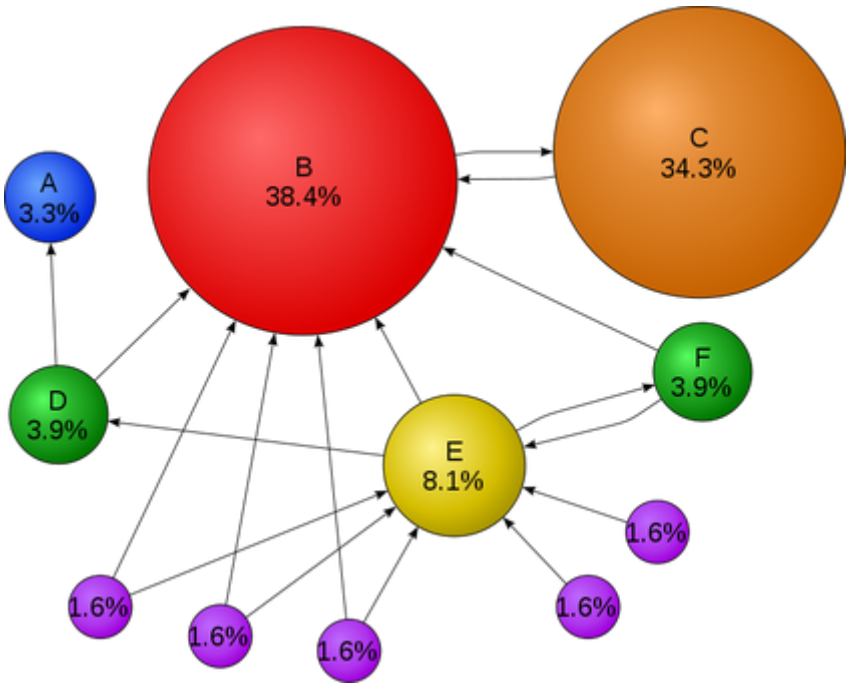
출처: [조성문님의 블로그(바로가기)]

> 하이퍼링크를 가지는 웹 문서에 상대적 중요도에 따라 가중치를 부여하는 방법이다. 서로간의 인용과 참조로 연결된 임의의 묶음에 적용할 수 있다.

검색으로 찾을 수 있는 문서의 수는 빠른 속도로 증가하였지만, 사람들이 문서를 읽는 속도는 같은속도로 성장하지 않았다. 따라서, 좋은 정보를 찾아내기 위한 방법이 필요했다.

특정 페이지의 중요도는 페이지 사이의 연결관계가 유용한 정보를 제공해줄 수 있다.

학술지 인용방식의 아이디어에서 연장, 단순히 인용된 수(다른 페이지의 링크)를 세는게 아니라 링크의 수를 정규화하는 방식.



- 어떤 페이지 A의 페이지 랭크는 그 페이지를 인용하고 있는 다른 페이지 T1, T2, T3, ...이 가진 페이지 랭크를 '정규화시킨 값의 합'이다.
- 즉, 페이지 A의 페이지 랭크는 A라는 페이지를 가리키고 있는 **다른 페이지의 랭크값이 높을수록 더 높아진다.**
- '정규화시킨 값의 합'이라는 의미는, 예를들어 T1페이지의 랭크가 높다고 하더라도 그 페이지에서 링크를 수천개 달아놓았다면(C(T1)의 값이 크다면) 그 페이지의 비중은 낮아진다는 것.
- 모든 웹페이지의 페이지 랭크 값을 합산하면 1이 된다.

$$PR(A) = \frac{(1-d)}{N} + d(\frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn))})$$

- RP : PageRank
- PR(A) : 'A'라는 웹페이지의 페이지 랭크
- Tn : 그 페이지를 가리키는 다른 **페이지들**

- PR(T1) : T1페이지의 랭크 값
- C(T1) : T1이라는 페이지가 가지고 있는 링크의 개수
- d(damping factor) : 웹 서핑하는 사람이 그 페이지에 만족하지 못하고 다른 페이지로 가는 링크를 클릭할 확률
    - 0과 1 사이의 값을 가지는 상수
    - d==1이면 위 수식에서 뒤의 합이 그대로 PR(A)가 된다.
    - d==0이면 PR(A)=1이 되므로 의미가 없어진다. 논문에서는 0.85로 설정했다고 한다.

Object Oriented Programming - 2

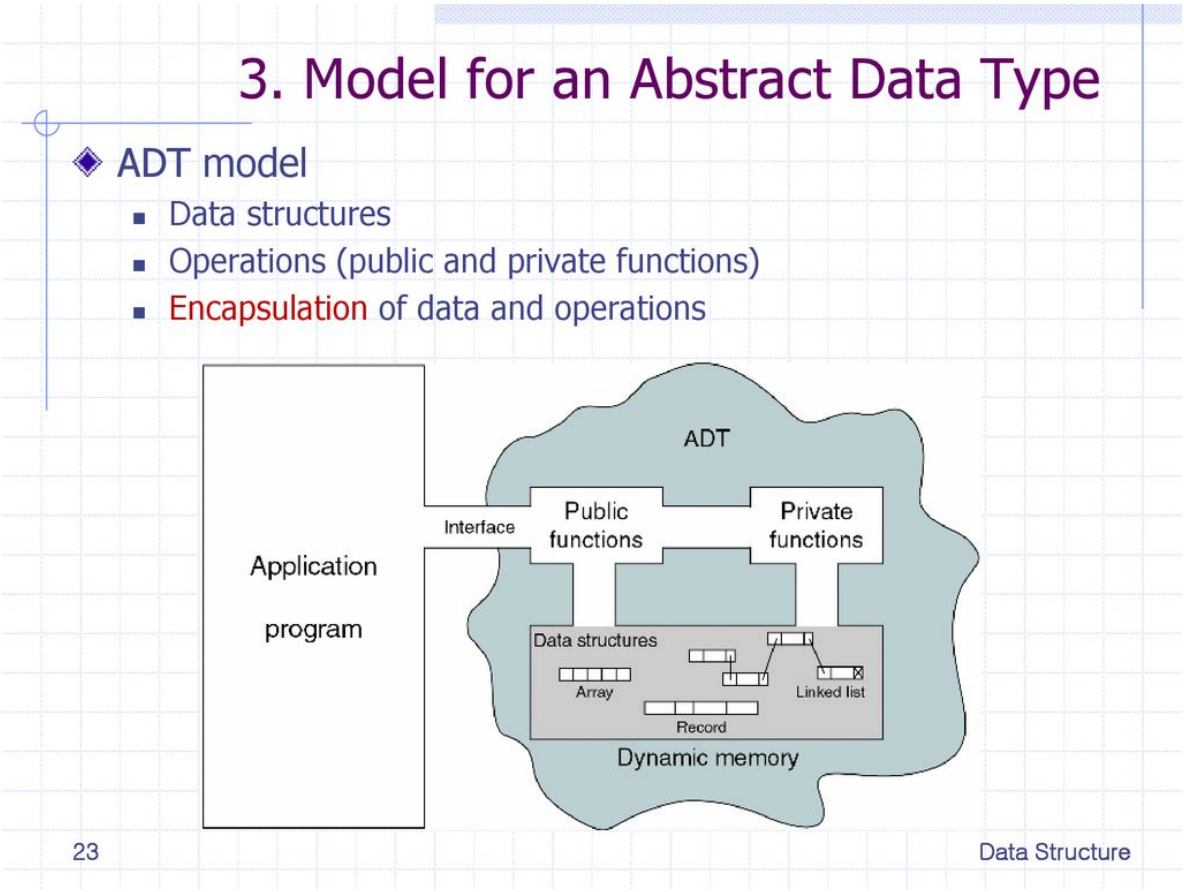
> 객체라는 개념을 지향하는 프로그래밍 기법
>
> 객체가 가지고 있는 데이터, 함수를 함께 묶어서 덩어리로보고 이것을 기반으로 프로그램을 설계하고 기반한다는 아이디어

ADT

> "사용자는 데이터나 연산이 무엇인지는 알 수 있으나 내부적 처리 방법은 몰라도 된다!"

추상 데이터 타입(Abstract Data Type: ADT) - 추상적이라는 말의 의미는 기능들만 기술해놓고, 구현은 내부적으로(안보이게) 구현해놓은 것.

사용자는 복잡한 부분을 알지 못해도 사용이 가능하다.



Application program에서는 Public function을 통해서 interface로 소통하고, 중요한 데이터와 함수 부분은 숨기겠다. 함수부분을 묶어서 감춤(Encapsulation)

- print()함수의 내부 구조를 몰라도 사용하는 방법만 알면 사용가능
- 이와 같은 ADT모델은 OOP 개념에 영향을 주었다.

객체지향 설계 - SOLID

집을 지으려면 설계도가 중요하듯이, 소프트웨어도 설계가 중요하다.

소프트웨어는 만들고 허무는 것이 자유로웠기 때문에, 전통적으로 설계에 큰 주의를 기울이지 않는 경향이 있었다. 그러다보니 여러 문제들이 발생했고, 그 문제들을 방지하기 위해 소프트웨어 설계에 주의를 기울이기 시작했다.

- S: Single responsibility principle - 한 클래스는 하나의 책임만 가져야 한다.
- O: Open/closed principle - 소프트웨어 요소는 확장에는 열려있으나, 변경에는 닫혀있어야 한다.
- L: Liskov substitution principle - 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.
- I: Interface segregation principle - 특정 클라이언트를 위한 인터페이스를 여러 개가 범용 인터페이스 하나보다 낫다.
- D: Dependency inversion principle - 프로그래머는 추상화에 의존해야지, 구체화에 의존하면 안된다.

OOP의 개념 중요 개념 3가지 + Class

1. Encapsulation(감싼다. 숨긴다.)

데이터타입 - 왜 필요할까?

> 변수에 저장공간을 잡아주는 역할을 수행하기 위해

- 메모리에 변수 저장할 것인데, 그때마다 데이터의 크기가 다르면 저장, 관리가 어려우므로 공통 특징인 데이터타입을 만들어놓고 저장할때 규정하여 사용한다.

C에서 서로다른 자료형을 묶어서 저장을 하고 싶은 경우, Structure를 사용했다. -> User-defined data type

과정(C언어 에서의)

1) 데이터 타입 만들기 - 구조체 정의, '설계도 만들기' - 규격을 만든다.

2) 만든 데이터타입을 가지고 변수 선언 - 구조체 변수 선언

위와 같은 과정을 통해 만들던 것을 객체지향에서는 Class라는 이름으로 하나로 Encapsulation 함.

```
class Car:      # C에서는 초기값을 선언할 수 없었다!
    color = '' # 그러나 python은 데이터타입을 지정하지 않고 assignment operator로 선언과 함께 배                정까지 하는 언어이기 때문에, speed = 0처럼 선언이 가능하다.
    speed = 0

    def upSpeed(self, value):
        self.speed += value
    def downSpeed(self, value):
        self.speed -= value

    # Encapsulation - upSpeed & downSpeed를 모두 들여쓰기로 한데 묶었다.

mycar1 = Car() # 변수를 선언하는 작업을 선언을 통해서 수행
               # 변수와 메서드에 관한 정보를 한번에 전달
```

self란?

- C에서는 함수단위로 움직였다. main 함수, add 함수가 따로 있어서 기능이 필요할때 왔다갔다 하며 사용.

- 호출할때 생성했다가, 필요할때 쓰고 없애는 식으로 사용.

list.append() 와 같이 사용했던 것 처럼, 객체로 넘어오면서 .연산자를 가지고 그 안에있는 함수도 호출한다.

구현하는 입장에서 함수를 계속 만들어주는 것 보다, 공통적인 함수를 하나 만들어주는게 효율적이다. 기본적으로 어떤 객체가 이 호출했는지 정보만 알면 해당 함수를 어느 호출 객체에서든 사용할 수 있다. (호출한 객체(self)의 레퍼런스를 자동으로 넘겨주는 개념)

이와 같은 원리로 객체지향의 클래스를 만들었다.

```
myCar1 = Car()
```

클래스로부터 객체를 생성할 때 메모리를 할당해야 한다. 이 작업은 자동으로 이루어진다.

잡아주는 기능은 메소드로 표현되는데, 그로 인해 클래스를 생성할 때 내부적으로 무조건 호출되는 메서드가 있다. 이것은 Constructor(생성자, __init__)라고 부른다.

- __init__도 메서드이기 때문에 self가 인수로 들어간다.

인스턴스 & 클래스 변수

- 프로그래밍 중, 클래스로부터 생성된 객체들이 공유해야하는 데이터가 있는 경우 - 클래스 변수
    - 예) 차가 몇대 생산되었는가?

```
class Car:
    color = '' # 인스턴스 변수
    speed = 0 # 인스턴스 변수
    count = 0 # 클래스 변수

    def __init__(self):
        self.speed = 0
        Car.count += 1 # -> 클래스 배리어블로 생성
# 클래스가 생성될 때 무조건 count가 하나 증가하게 된다.

myCar2.count # 위에서는 Car.count로 접근했는데, 여기서는 객체 이름으로 접근.
# 위에서 클래스 변수로 접근하면 객체 이름으로도 접근 가능하다. 그러나 이와 같은 경우 혼동이 발생할 수 있으므로 Car.count로 사용하도록 하자.
```

2. Inheritance(상속)

위의 예에서 승용차와 트럭의 클래스를 만들고 싶은데, 공통되는 클래스를 부모클래스로 만들고 그로부터 상속된 클래스로 하위에 필요한 기능이 추가된 형태로 만들 수 있다.

```
class Car:
    speed = 0

    def upSpeed(self, value):
        self.speed = self.speed + value
    def downSpeed(self, value):
        self.speed = self.speed - value

class Sedan(Car): # Car클래스를 상속받은 Sedan
    seatNum = 0

    def getSeatNum(self): # 상속받은 클래스(Sedan)에는 추가하는 변수와 추가하는 메소드만 적어주면 된다.
        return self.seatNum
```

그렇다면 Sedan이라는 클래스(Car를 상속받은 자식 클래스)로부터 객체를 생성했을 때, 생성자는 어떻게 호출이 되어야 할까? 하위 클래스를 호출할 때, 생성자는 어떤 것이 호출될까?

- 실행은 부모클래스 생성자 실행되고, 다음으로 자식 클래스의 생성자가 실행된다.

Method Overriding

> 부모클래스의 메소드를 하위 클래스에서 (같은 이름으로)재정의하는 것

- 일단 상속받고 **파라미터의 개수까지 똑같이 재정의**한다.

3. Polymorphism(동질 이상)

- 동종 집단 가운데에서 2개 이상의 대립 형질이 뚜렷이 구별되는 것