테크 지식 & 파이썬 프로그래밍 기초

1주차
컴퓨터와 네트워크, 그리고 프로그래밍

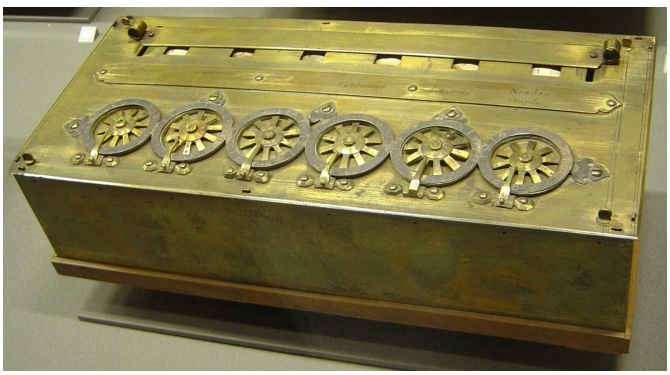
20 . .

지은이: 이은재

Ⅰ. 컴퓨터란 무엇일까요?

프로그래밍을 본격적으로 공부하고 싶은 사람이라면 그 전에 먼저 컴퓨터에 대해서 알아야 할 필요가 있습니다. 이후에도 다시 살펴보겠지만, 프로그램이란 우리가 컴퓨터에 내리는 명령의 집합이고 프로그래밍도 결국에는 그러한 프로그램을 만드는 과정이기 때문입니다. 요리사에게 아무리 귀하고 신선한 식재료가 있어도 전혀 처음 보는 것이라 어떻게 손질해야할지 모른다면 훌륭한 음식을 만들 수는 없겠지요? 마찬가지로 컴퓨터가 무엇이고 어떻게 구성되며, 어떤 일을 할 수 있고, 또 어떤 일은 할 수 없는지를 정확히 알지 못한다면 컴퓨터를 제대로 활용하는 일은 불가능할 것입니다.

그러면 컴퓨터가 무엇이고 어떻게 탄생하게 되었는지 한 번 알아보도록 할까요? 우리가 요즘 흔히 말하는 '컴퓨터'는 '개인용 디지털 컴퓨터'를 의미합니다. 말 그대로, 이진 논리를 기본 원리로 해서 누구나 사용할 수 있도록 만든 전자계산기입니다. 그런데 사실 컴퓨터 중에는 개인용만 있는 것도 아니고 디지털 컴퓨터만 있는 것도 아닙니다. 전기 없이 톱니바퀴와 나사로만 만들어진 것, 물론 동작하는 것, 심지어는 실제로 만들어진 적 없이 개념적으로만 존재하는 컴퓨터도 있습니다. 게다가 한때 '컴퓨터'는 전자계산기가 아니라 전문적으로 계산을 해주는 직업을 가진 사람(compute + -er)을 뜻하는 말로 쓰였습니다. 공학용 전자계산기가 보급되기 이전인 1960년대 까지만 해도, 대량의 복잡한 수학적 연산은 수많은 사람들이 한 방에 모여 기계식 계산기나 주판을 사용해 몇 시간씩을 분업해야만 가능했습니다.



▲ 파스칼 계산기, 1652년 제품

▲ 해석기관 모형

톱니바퀴와 나사, 온갖 금속 부품들로 이루어진 최초의 기계식 계산기는 1642년에 파스칼이 만들었습니다. 정확하기는 했지만 아주 간단한 덧셈과 뺄셈만 계산할 수 있었습니다. 영국의 찰스 배비지는 그보다 200년 정도가 지난 1833년, 증기기관에서 나오는 운동에너지를 동력으로 해서 일반적인 수학 계산을 자동으로 할 수 있는 '해석기관'이라는 기계를 고안합니다. 안타깝게도 제작비용 부족과 당시 공업기술 수준의 한계로 설계에 그쳐 실제로 만들어지지는 못했지만, 이 미완의 기계가 컴퓨터의 역사에 남은 이유는 따로 있습니다. 바로 처음으로 '프로그래밍 가능한(programmable)' 계산기로 설계되었기 때문입니다.

Q. '프로그래밍 가능하다'는 것은 어떤 의미일까요?

▲ 에이다 러브레이스의 초상

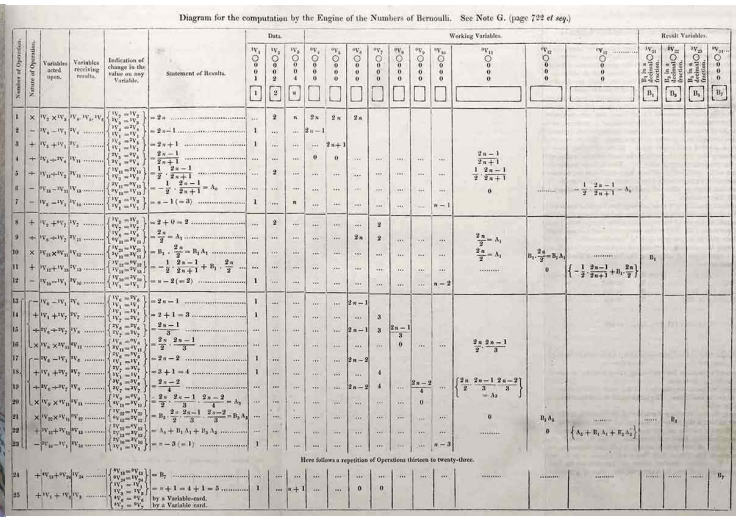
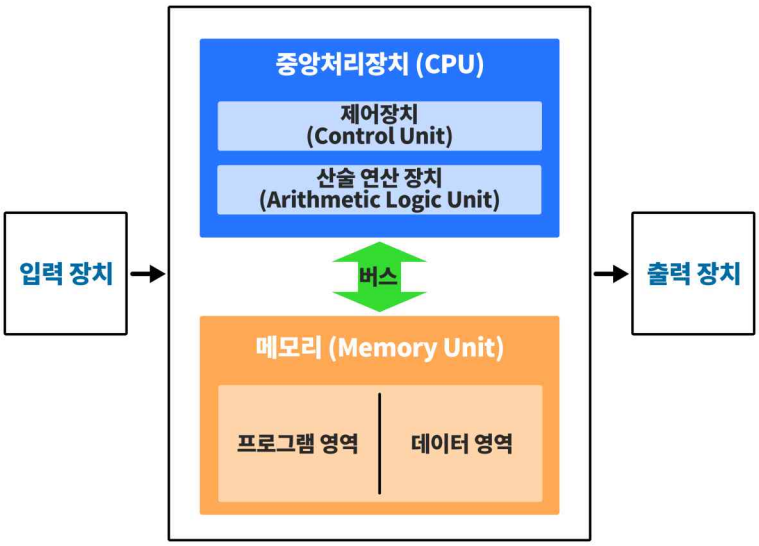


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

▲ 베르누이 수 구하기 프로그램

그런데 신기하게도 이 만들어지지도 않은 컴퓨터를 가지고 세계 최초의 프로그래머 타이틀을 거머쥔 사람이 있습니다. 영국의 시인인 조지 바이런의 딸로도 잘 알려진 '에이다 러브레이스' 입니다. 그의 아버지는 수백 명의 연인이 있었다고 할 정도로 워낙 유명한 바람둥이였던 터라 에이다의 어머니와도 금방 헤어지고 말았습니다. 바이런의 방탕한 삶의 원인이 문학이라고 여겼던 에이다의 어머니는 어려서부터 에이다에게 이름난 수학자나 과학자들을 가정교사로 붙여 주었습니다. 문학이나 시 대신 수학과 과학을 공부하도록 말이지요. 그중에 한 사람이 찰스 배비지였고, 자연스럽게 함께 연구를 하게 되면서 해석기관에 적용할 수 있는 최초의 프로그램을 만들게 된 것입니다. 자연수 거듭제곱의 합을 빠르게 계산할 때 필요한 베르누이 수를 구하는 이 알고리즘에는 우리가 앞으로 배우게 될 반복문, 조건문, 서브루틴 같은 컴퓨터 프로그래밍의 일반 개념들이 모두 포함되어 있다고 하니 놀라운 일이 아닐 수 없습니다.

그러면 이런 기계식 컴퓨터가 아닌 우리가 사용하는 것과 더 가까운 컴퓨터는 누가 언제 어떻게 만들게 되었을까요? 디지털 컴퓨터의 고안은 앨런 튜링과 폰 노이만, 두 사람의 업적이라고 할 수 있습니다. 튜링은 모든 수학적 문제를 순차적으로 해결할 수 있는 일반적인 절차를 만들려는 시도 끝에 지금은 우리가 '튜링머신'이라고 부르는 'a-기계'를 고안하게 됩니다. 튜링머신은 테이프, 헤드, 상태기록기, 행동표로 이루어져 있는데, 이론적으로는 우리가 상상할 수 있는 대부분의 연산을 자동화할 수 있는 기계의 일반화된 작동 원리를 함께 제시하고 있습니다. 이 원리는 나중에 '튜링 완전'이라는 특성으로 발전합니다. 어떤 기계는 반드시 튜링 완전해야만 우리가 사용하는 컴퓨터의 모든 기능들을 구현할 수 있다는 것이지요. 따라서 새로운 구조의 컴퓨터나 프로그래밍 언어를 만들고 싶다면, 그것들은 스스로가 튜링 완전함을 보장할 수 있어야 실제로 사용할 수 있는 것이 됩니다.



▲ 폰 노이만 구조

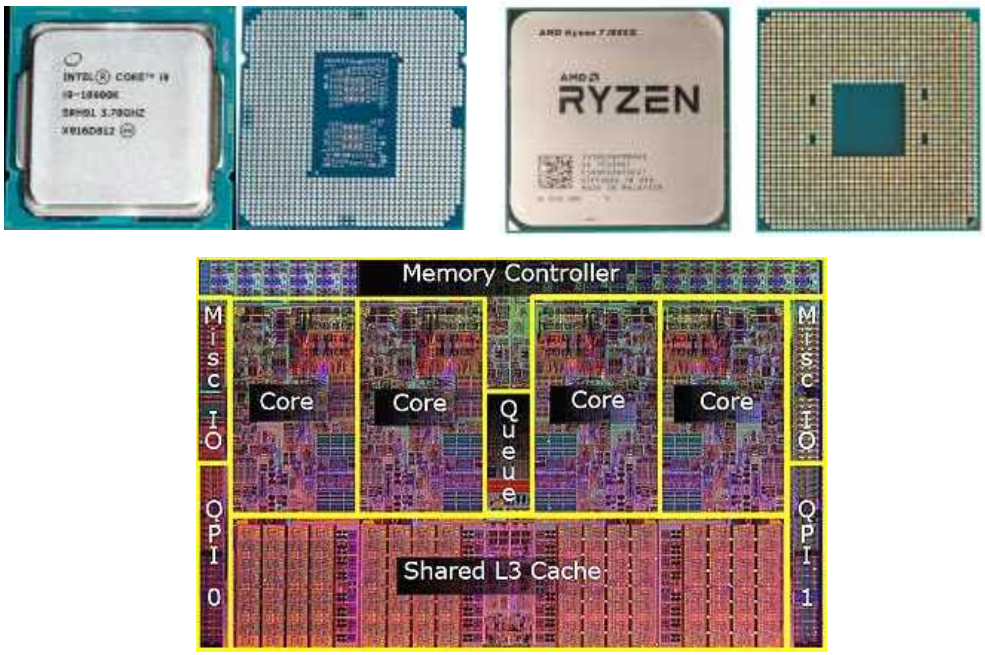
튜링이 컴퓨터의 논리적 기반을 마련한 사람이라면, 폰 노이만은 튜링의 이론을 바탕으로 범용적으로 사용할 수 있는 컴퓨터를 설계하고 제작한 사람입니다. 이전 까지는 프로그래밍 가능한 컴퓨터라고 할지라도, 새로운 프로그램을 만들기 위해서는 직접 스위치를 조작하고 전선을 재배열하는 등의 복잡한 작업을 거쳐야만 했습니다. 반면 '폰 노이만 구조'라고 불리는 그의 컴퓨터 구조는 최초로 프로그램의 저장을 가능하게 했기 때문에 진정한 의미의 '소프트웨어'라는 개념을 탄생시켰습니다. 지금까지도 사용되는 거의 모든 컴퓨터가 그의 설계를 바탕으로 하고 있으며 완전히 새로운 패러다임의 컴퓨터 구조가 만들어지지 않는 한 앞으로도 폰 노이만 구조는 더 오랜 시간 사용될 것으로 보입니다.

Ⅱ. 우리가 사용하는 컴퓨터는 어떻게 구성되어 있을까?

이제 실제로 컴퓨터가 어떻게 생겼는지 한 번 알아볼까요? 현대의 디지털 컴퓨터는 크게는 하드웨어와 소프트웨어로 구성된다고 할 수 있습니다. 우리의 목표는 물론 소프트웨어를 만들고 제대로 활용하기 위한 내용을 배우는 것이기는 하지만, 소프트웨어도 결국 하드웨어를 제어하기 위한 수단이기 때문에 하드웨어에 관해서도 기본적인 내용을 알아두는 것이 좋습니다.

1) 하드웨어

(1) CPU (Central Processing Unit)



▲ CPU의 생김새와 내부 구조

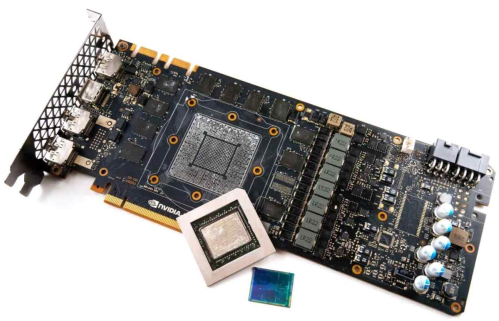
CPU는 반도체로 만드는 수십 억 개의 아주 작은 트랜지스터가 코어를 이루고, 다시 그 코어들이 하나, 또는 여러 개 모여 있는 판 형태로 되어 있습니다. 프로그램을 해석하고 데이터를 처리하고 다른 부품들을 제어하는 데에 필요한 모든 연산을 담당하는 부분이기 때문에 컴퓨터의 성능을 가장 크게 좌우하기도 합니다. CPU의 성능은 '클럭(clock)'의 주파수와 '코어(core)'의 개수로 결정됩니다. 클럭은 CPU가 1초에 처리할 수 있는 명령어의 개수를 의미하는데 당연히 많은 계산을 할 수 있는 것이 좋겠지요? 요즈음에는 일반적으로 2~5GHz 사이의 제품들이 개인용으로 사용되고 있습니다. 코어는 말 그대로 CPU가 몇 개의 중심으로 이루어졌는지를 의미합니다. 예전에는 CPU가 하나면 코어도 하나였지만 하나의 코어만으로는 전력소비나 발열 같은 물리적 문제들 때문에 성능 향상에 한계가 있어 현재 대부분의 CPU들은 코어를 여러 개 두고 각각의 코어가 동시에 작업을 처리하는 방식을 선택하고 있습니다.

Q. 오버클럭과 쓰레드란?

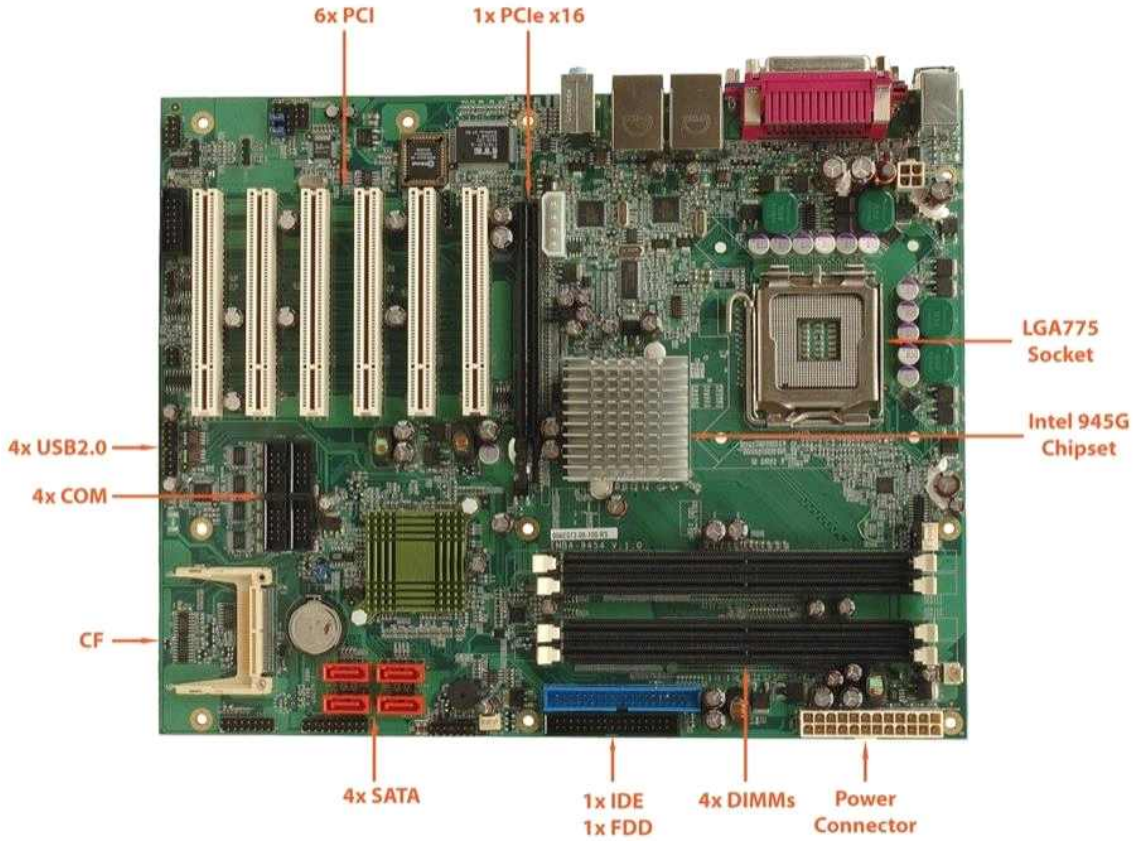
(2) RAM (Random Access Memory)

(3) HDD / SSD (Hard Disk Drive / Solid State Drive)

(4) GPU (Graphic Processing Unit)



(5) Main Board



(6) Power Supply / Battery

Q. 내 컴퓨터 사양 알아보기 (dxdiag) / 현명하게 컴퓨터 구매하는 방법

2) 소프트웨어

(1) OS (Operating System)

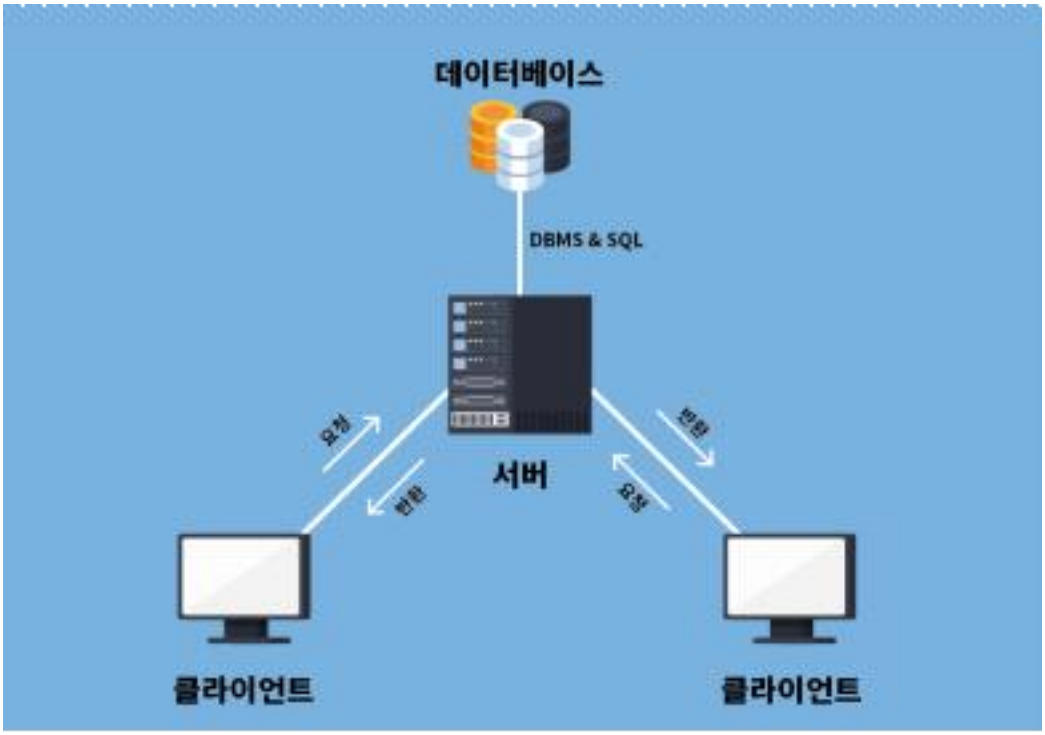
운영체제라고도 부르는 OS는 하드웨어와 소프트웨어 모두를 효율적으로 관리할 수 있도록 도와주는 소프트웨어입니다. 전원버튼을 누르면 컴퓨터가 켜지는 것부터 시작해서 모든 작업을 마치고 시스템을 종료하는 것까지 어느 하나도 OS 없이는 동작하지 않습니다. 이 때 하드웨어를 직접 제어하는 부분을 커널(kernel)이라고 하고, 다른 소프트웨어가 실행되도록 돕는 부분을 셸(shell)이라고 부릅니다. 가장 대중적으로 사용되는 OS는 Windows와 MacOS, 그리고 Linux이지만 하드웨어의 특성과 용도에 따른 수많은 OS가 사용되고 있습니다.

(2) 응용프로그램 (application software)

응용프로그램은 일반적으로 컴퓨터에서 실행되는 소프트웨어 중에서 OS를 제외한 나머지를 가리킵니다. 이름이 '응용'인 이유는 시스템 소프트웨어인 OS의 기능을 이용해서 사용자가 원하는 특정한 기능만을 수행하도록 만든 것이기 때문입니다. 컴퓨터 입장에서 보자면 응용프로그램은 OS의 보조적 기능이라고 할 수 있습니다. 하지만 사용자 입장에서는 컴퓨터를 사용하는 주 목적이 바로 응용프로그램을 사용하기 위한 것이기 때문에 '프로그램'이라고 하면 보통 응용프로그램을 의미하게 되었습니다.

Ⅲ. 네트워크

(1) 네트워크란?



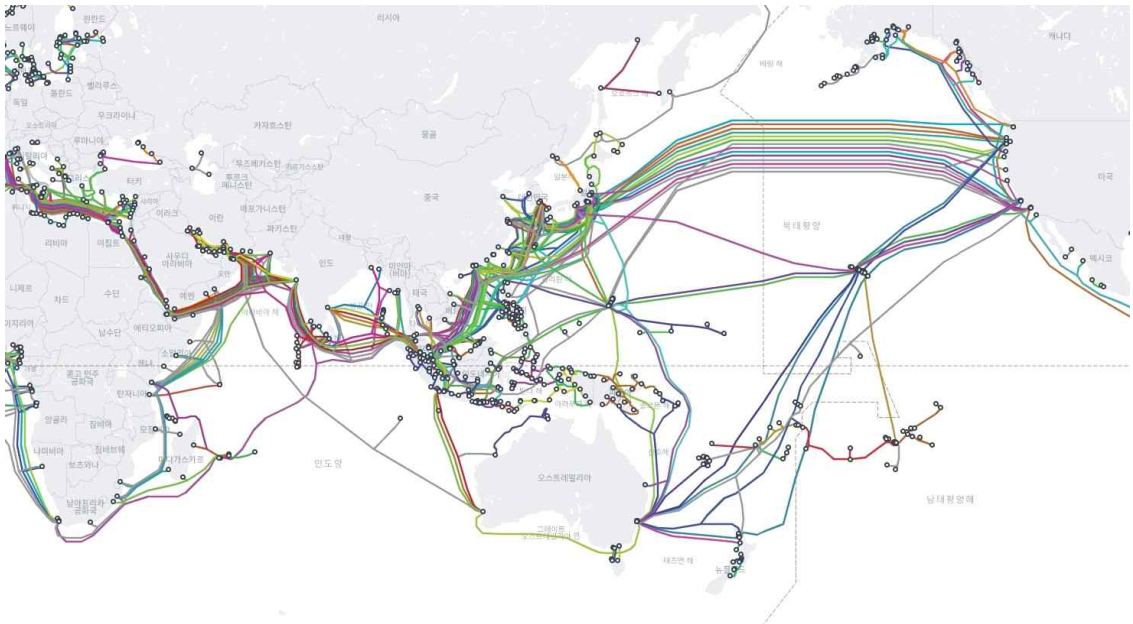
▲ 네트워크 모식도

네트워크는 여러 군데에 분산되어있는 컴퓨터들이 서로 데이터를 공유할 수 있도록 하나의 통신망으로 연결한 것을 말합니다. 일상에서 가장 많이 사용하는 네트워크는 물론 인터넷입니다. 하지만 네트워크에는 인터넷만 있는 것은 아닙니다. 사용하려는 목적과 범위, 용도 등에 따라서 네트워크 구축에 사용하는 통신 방법과 그 규모가 달라집니다. 각각의 컴퓨터를 직접 케이블로 연결하거나 간단한 라우터만을 사용하는 소규모 네트워크도 있지만 서버용 컴퓨터를 사용하거나 전용 데이터 센터를 새로 짓기도 하는 방대한 네트워크도 있습니다.

우리가 인터넷을 사용할 때 컴퓨터와 네트워크에서는 어떤 일이 일어나고 있는 것일까요? 지금부터 우리가 어떤 홈페이지에 접속해서 정보를 조회한다고 해봅시다. 이때 내가 사용하는 컴퓨터를 '클라이언트'라고 부르고 접속하려는 홈페이지의 정보를 담고 있는 컴퓨터를 '서버'라고 부릅니다. 각각의 컴퓨터에는 고유한 주소가 있습니다. 이 때, 서로 약속한 통신 형식인 '프로토콜'에 따라서 접속하고자 하는 서버의 주소로 원하는 정보를 전송해달라는 요청을 보냅니다. 서버는 받은 요청에 문제가 없는지 확인하고 데이터베이스의 정보를 가공한 뒤 그 결과를 클라이언트에게 전송합니다. 이 과정을 쉽게 바꾸어주는 소프트웨어가 우리가 매일 사용하는 인터넷 브라우저입니다.

(2) 인터넷

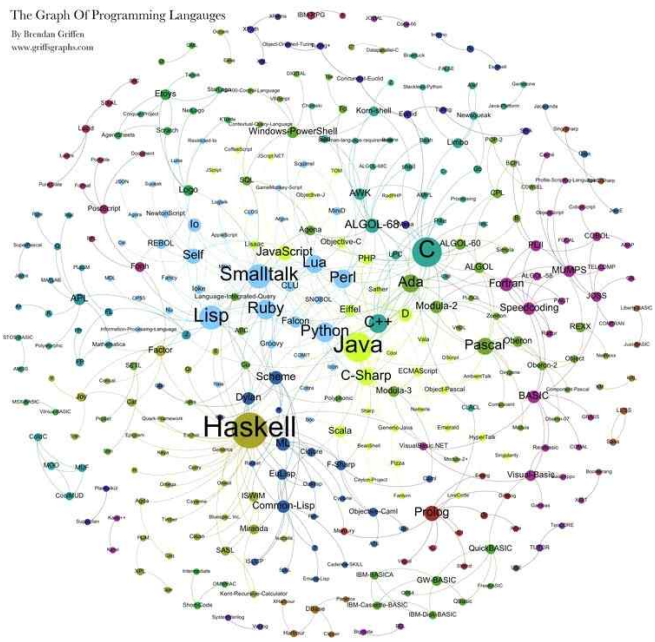
블루투스나 핫스팟 같은 기능부터 시작해서 국가 간 해저케이블을 사용하는 월드와이드웹에 이르기까지, 네트워크는 그 범위에 따라서 PAN, LAN, MAN, WAN 등으로 구분합니다. 냉전 시대 미국의 고등연구계획국에서 군사적인 목적으로 만들어진 최초의 인터넷인 ARPA넷이 본래 의도와는 다르게 학술 정보 교류에서 각광을 받으면서 민간용 인터넷의 필요성이 점점 높아졌습니다. 이후 CERN의 팀 버너스리가 월드와이드웹, HTTP, URL을 고안하고 무료로 공개하면서 본격적으로 인터넷 보급이 시작되었습니다.



Q. 최초의 웹사이트에 접속해보기 (info.cern.ch) / 내 IP 주소 확인하기 (ipconfig)

Ⅳ. 프로그래밍과 언어

(1) 프로그래밍 언어 / 왜 파이썬인가?



▲프로그래밍 언어들

로우레벨언어 하이레벨언어

기계어 < 어셈블리어 < C < C++ < JAVA < PYTHON < SCRATCH

사람이 한국어나 영어 같은 자연어를 사용하는 것처럼 컴퓨터도 컴퓨터가 알아들을 수 있는 언어인 기계어를 사용합니다. 기계어는 추상화된 정도에 따라서 그 계층이 달라집니다. 계층이 가장 낮은 언어 즉, 실제로 기계가 정보를 기록하는 데에 사용하는 언어는 0과 1로 이루어진 '바이너리 코드'입니다. 흔히 '컴퓨터는 0과 1밖에는 모른다'는 이야기가 알려진 것도 이 때문입니다. 따라서 우리가 컴퓨터에게 어떤 프로그램을 실행하라고 지시하려면 컴퓨터가 이해할 수 있는 기계어로 입력해 주어야 할 것입니다.

문제는 우리가 바이너리 코드를 이해하지 못한다는 데에 있습니다. 초창기의 프로그래머들은 정말로 바이너리 코드로 프로그램을 만들었다는 전설적인 이야기들이 전해지기는 합니다만, 다행히도 현대에는 더 이상 바이너리 코드를 사용하지 않아도 프로그램을 만들 수 있도록 도와주는 다양한 고급 언어들이 개발되어 있습니다. 그 종류는 수 천 가지가 된다고 하지만 가장 대표적인 예가 바로 우리가 배우려고 하는 '파이썬'입니다.

코드 0-1 'Hello World!'를 출력하는 C 코드

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

↳ 실행화면

```
Hello World!
```

코드 0-2 'Hello World!'를 출력하는 자바 코드

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

↳ 실행화면

```
Hello World!
```

코드 0-3 'Hello World!'를 출력하는 파이썬 코드

```python
print('Hello World!')
```

↳ 실행화면

```
Hello World!
```

Q. 왜 파이썬인가?

- ▶ **파이썬의 슬로건 - “Life is too short, You need Python!”**
- ▶ **자연어스럽고 직관적인 코드작성 방식**
- ▶ **높은 범용성**
- ▶ **커뮤니티가 크고 많다**
- ▶ **개발환경 구성이 간단한 편이다**

(2) 프로그래밍 과정

▶ **순차 / 선택 / 반복**
▶ **자동화와 멀티태스킹**
▶ **문법 / 자료구조 / 알고리즘**
▶ **백엔드 / 프론트엔드 / 풀스택**
▶ **IDE / GitHub**

문제 인식

: 해결해야 할 문제가 명확하지 않으면 일단 코드를 작성할 이유가 불분명해지기 때문에 코딩 자체가 길고 지루한 작업이 됩니다. 그 결과로 복잡하고 비효율적인 문장들을 사용하게 되는데, 흥미 면에서도 성능 면에서도 결코 좋은 일이 아닙니다. 자신이 어떤 문제를 해결하고 싶은지 확실히 정해두도록 합시다.

의사코드(pseudo code)작성

: 프로그래밍 언어로 코딩을 하기 전에 미리 프로그램을 설계해보는 단계입니다. 이때는 프로그래밍 언어를 사용할 필요 없이 자연어나, 기타 각자에게 편한 방식을 고르면 됩니다. 굳이 할 필요가 있을까 싶지만 의사코드 작성을 해보느냐 마느냐에 따라서 실제 코딩 시간이 길어지기도, 줄어들기도 합니다.

코딩

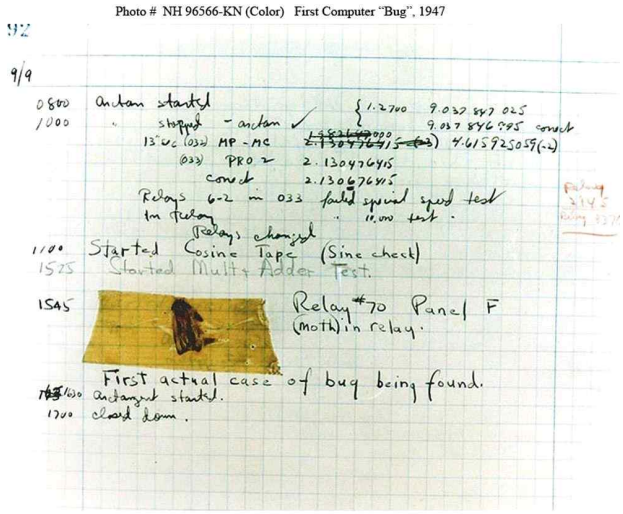
: 실제 프로그램을 작성하는 과정입니다. 영화에서 보던 멋진 해커의 화려한 모습은 사실과는 거리가 아주 멉니다. 사실은 멍하게 모니터를 보고 생각하는 시간이 더 많습니다.

컴파일 (compile)

: 우리가 프로그래밍 언어로 작성한 코드는 사실 우리가 알아보기 위한 것이지, 컴퓨터가 바로 알아들을 수 있는 형태가 아니라는 것은 앞에서도 설명했습니다. 그래서 몇 번의 번역을 거쳐야만 하는데, 이 과정을 컴파일 이라고 합니다. 다행히 여러분이 직접 할 필요는 없고 컴파일러라는 프로그램이 자동으로 해줍니다!

빌드 (build)

: 컴파일까지 끝난 프로그램은 '빌더'라는 프로그램에 의해서 다시 바로 실행 가능한 형태로 저장됩니다. 치명적인 오류가 있다면 아예 빌드가 되지 않는 때도 있지만, 자잘한 오류만 있는 경우에는 빌드가 되기는 합니다. 여기까지 성공적으로 되었다면 일단은 프로그램을 만들었다고 할 수 있겠습니다. 물론 좋은 프로그램인지 아닌지는 전혀 다른 문제이기는 하지만요.



▲ 최초의 컴퓨터 버그

디버깅

: 1차로 완성된 프로그램이 오류 때문에 아예 실행되지 않거나 실행은 되더라도 원하는 결과를 출력하지 못할 경우, 프로그램에 '버그'가 있다고 표현합니다. 그리고 버그를 찾아 고치는 것을 '디버깅'이라고 부릅니다. 짧은 코드일 때는 오류의 원인을 금방 찾을 수 있지만 10줄 단위만 넘어가도 오류를 찾기가 대단히 어려워집니다. 그래서 디버거라는 프로그램의 도움을 받는데 디버거는 우리가 작성한 프로그램을 한 줄 한 줄 실행하여 문제가 있는 부분을 표시해줍니다. 그렇게 표시된 버그는 다시 적절한 코드로 수정하고 새로 빌드하면 문제가 대부분 해결됩니다. 실제로 최초의 컴퓨터 버그는 기계부품 사이에 끼어들어간 '버그' 때문이었습니다.

테스트 (프리알파, 알파, 베타)

: 디버깅을 통해서 버그를 완전히 잡아냈더라도 완전한 프로그램이 되는 것은 아닙니다. 버그가 없다는 것은 논리적 오류가 없다는 의미이지 성능이 더 좋다거나, 디자인적인 측면에서 훌륭하다는 것 등을 의미하지는 않기 때문입니다. 따라서 출시를 목적으로 하는 대부분의 프로그램은 다양한 사용자들에게 미리 테스트를 받는 과정을 거칩니다. 그리고 그 결과를 바탕으로 제작자와 사용자 모두가 만족할 때까지 수정이 이루어집니다. 프로그램이 어느 정도 완성되었는가, 그리고 얼마나 개방적인 테스트인가에 따라 프리알파, 알파, 베타 테스트 등으로 나뉘어 진행됩니다.

릴리즈

: 프로그램의 시장 출시를 의미합니다. 릴리즈 된 프로그램은 실제로 수많은 사람들의 컴퓨터에 설치되어 사용될 수 있기 때문에 여러 버전의 릴리즈 후보를 개발해두고 어떤 것으로 릴리즈할지 신중히 결정합니다.

버전관리

: 프로그램이 한 번 출시되었다고 해서 개발자의 역할이 끝나는 것은 아닙니다. 테스트에서는

발견하지 못한 문제들이 생길 수도 있고 프로그램에 기능을 추가하거나 빼야할 수도 있습니다. 많은 프로그램들이 업데이트를 하는 이유이기도 한데, 업데이트를 할 때마다 어떤 부분이 어떻게 바뀌었는지 정확히 기록해두지 않으면 점점 관리하기가 어려워집니다.

유지보수

: 프로그램이 출시된 후로 시간이 지나면, 호환성이나 성능, 보안상의 다양한 문제가 발생할 수 있습니다. 따라서 지속적인 유지보수가 필요한데, 내가 개발한 것을 내가 고치는 것조차도 굉장히 힘든 작업입니다. 하물며 다른 사람이 작성한 코드를 아무런 설명도 없이 스스로 이해해서 고치는 것은 거의 불가능에 가깝습니다. 개인 프로젝트라면 상관이 없지만 협업을 해야 하는 상황이라면 주석과 별도의 문서로 프로그램 개발과 관련된 사항들을 정리해두어야 합니다.