Programming Language

컴퓨터개론

(Introduction to Computer Systems)

GEN1030

High Level Language

High Level Language

- FORTRAN
- COBOL
- BASIC
- PASCAL
- C
- C++
- Java
- Python
- Go
- …

High Level Language

- FORTRAN
  - FORmual TRANSlating system
  - 가장 오래된 언어 중 하나 (1960년대)
  - 단순, 간결하며 수치 계산에 강함

```
program HelloWorldF90
   write(*,*) "Hello, Fortran!"
end program HelloWorldF90
```

- COBOL
  - Common Business Oriented Language
  - 1960년대 등장 및 사용
  - 사무처리 목적으로 설계, 파일 처리에 강점을 보임
    - 영어 문장과 비슷한 구조

```
IDENTIFICATION DIVISION.
PROGRAM-ID. YOUR-PROGRAM-NAME.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01 NUM1 PIC 9(4) VALUE 1458.
01 MESSAGE PIC X(11) VALUE 'HELLO WORLD'.
PROCEDURE DIVISION.
MAIN-PROCEDURE.

**********This is a comment in Cobol***************

DISPLAY NUM1.
DISPLAY MESSAGE.
STOP RUN.
END PROGRAM YOUR-PROGRAM-NAME.
```

High Level Language

- BASIC
  - Beginner's All-purpose Symbolic Instruction Code
  - 1980년대 PC의 발전과 함께 널리 사용
  - Microsoft의 Visual Basic 언어로 발전

```
READY
10 FOR X=1 TO 10
20 PRINT (11-X);" GREEN BOTTLES"
30 NEXT X
RUN
```

- PASCAL
  - 교육용으로 만들어진 프로그래밍 언어
  - 문법이 명확하고 배우기 쉬움

```
program HelloWorld;
uses crt;

(* Here the main program block starts *)
begin
   writeln('Hello, World!');
   readkey;
end.
```

C

- 1972년 데니스 리치(Dennis Ritchie)가 설계
  - 유닉스(Unix) OS 개발 위한 언어
  - 하드웨어 통제가 가능
    - High level과 Low level language 장점 모두 포함
  - 이식성(Portability)이 높음
    - 특정 컴퓨터 기종에 의존하지 않음
  - 풍부한 연산자와 데이터 타입으로 범용 프로그래밍 언어로서 널리 보급
  - 현재 사용되는 대부분의 운영체제, 컴파일러 등이 C 언어로 구현 됨

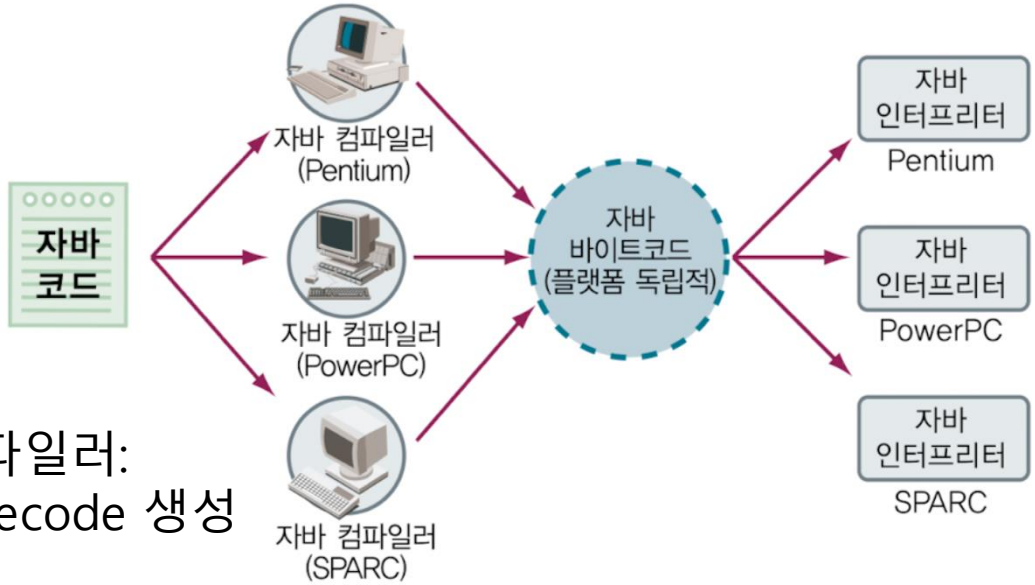
- C++
  - C언어 바탕으로 함 - C언어의 장점 계승
  - 객체지향 프로그래밍(Object-Oriented Programming, OOP) 지원

Java

- 미국 Sun Microsystems 사에서 가전제품 제어 위해 고안한 언어에서 시작
- 컴파일러, 인터프리터 두 방식의 장점을 혼합한 언어
- 시스템 독립적(System independence)
  - OS, CPU와는 독립적으로 실행 가능
  - 하나의 플랫폼에서 만든 자바 프로그램은 다른 플랫폼에서 별도의 작업 없이 실행 가능 (ex. Microsoft → Linux)
  - "Write Once, Run Anywhere"
- 자바 가상 기계, 바이트코드

Java Virtual Machine, Bytecode

- 자바 가상 머신(Java Virtual Machine, JVM)
  - 자바 프로그램 실행해주는 가상의 컴퓨터(Virtual machine)
  - 컴퓨터(OS) 위에서 동작
- 바이트코드(Bytecode)
  - Java 코드 컴파일 시 생성되는 중간코드
  - 구조 중립적: 한 시스템에서 컴파일 된 바이트코드(bytecode)는 다른 시스템에서도 컴파일 하지 않고 바로 실행 가능



컴파일러:
Bytecode 생성

인터프리터(JVM에 포함): bytecode 한 줄씩 읽어 기계어로 해석 및 실행

C#

- Microsoft에서 만든 프로그래밍 언어
- 객체 지향 언어
  - C++, Java 등의 언어로부터 영향
- 시스템에 독립적인 프로그램 개발 가능
  - 코드 컴파일 하면 중간 코드인 MSIL (MicroSoft Intermediate Language) 생성
  - 인터프리터인 CLR (Common Language Runtime)에 의해 실행
- 게임 개발에 많이 사용
- .NET
  - Microsoft의 오픈 소스 개발 플랫폼
  - C# 같은 언어 지원

Python

- 인터프리터(interpreter) 언어
  - 컴파일 과정 없이 작성된 코드 한 줄 씩 읽어가며 실행
- 객체 지향

- 동적 타이핑(Dynamic Typing)
  - 변수의 타입을 미리 선언하지 않아도 됨
  - 값이 할당 될 때 타입이 결정

```
x = 10
x = “hello”
```

cf) 정적 타이핑(Static Typing) – ex) C, Java 등

```
int x = 10;
x = “string”      → 오류 발생
```

Python

- 문법이 비교적 쉬워 빠르게 개발 가능

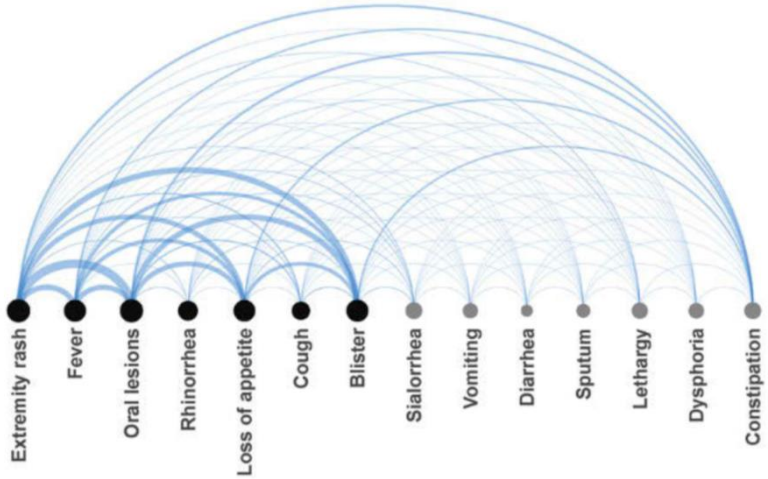
- 인공지능, 머신러닝, 빅데이터 처리 분야에서 각광
  - Github 저장소에 가장 많이 올라온 언어
  - 다양한 라이브러리(library) 제공
    - 복잡한 통계, 수학 공식을 직접 구현할 필요 없음
    - NumPy, Pandas, TensorFlow, PyTorch 등

- 거대한 커뮤니티와 생태계
  - 많은 연구와 새로운 기술들이 Python 코드와 함께 공개 됨

R

- 데이터 분석, 빅데이터 처리 등의 통계 분석에 특화 된 프로그래밍 언어
  - 통계학자들이 계산과 그래프 작성을 위해 개발
  - cf) Python: 범용 언어
  - Open source

- 그래프, 시각화 기능이 강함



- 풍부한 통계분석 패키지 제공

- 인터프리터 방식

Go

- 구글에서 만든 프로그래밍 언어 (2009년)
  - Open source

- 비교적 배우기 쉬움
  - 심플함을 추구 - 문법이 간단함
- 컴파일 언어
  - C, C++에 근접할 정도로 실행 속도가 빠름
- 강력한 동시성(Concurrency) 지원
  - 병렬 컴퓨팅에 적합: 수천, 수만 개의 작업을 동시에 처리하는데 특화

Kotlin

- JetBrains이 개발 (2016년)

- Google 선정 Android 앱 개발 공식 언어
- 객체 지향
- Java와 비슷한 구조
  - 코드가 간결함(Java보다 짧게 작성 가능)
  - 자바 가상 머신(JVM)에서 실행 가능 → Java와 100% 호환

Swift

- Apple이 발표한 프로그래밍 언어 (2014년)
  - Open source

- Apple의 모든 생태계에서 돌아가는 앱을 만들기 위한 언어
  - iPhone, iPad, Mac 등
- 파이썬 만큼 문법이 깔끔하고 읽기 쉬움
- 실행속도가 C++ 등에 비견될 정도로 빠름

웹 기반 언어

- HTML
  - HyperText Markup Language
  - 웹 페이지(webpage)의 구조 표현하는 언어
  - 제목, 문단, 이미지, 링크 등을 태그(tag)로 작성
  - 웹 브라우저(browser)가 해석하여 화면에 표시

```
<html>
 <body>
  <p>This is a web page.</p>
 </body>
</html>
```

Python

Python

- 최근 가장 활발히 사용되는 언어 중 하나 

- 인터프리터(interpreter) 언어
  - 컴파일 과정 없이 작성된 코드 한 줄 씩 읽어가며 실행

- 크게 두 가지 주요 version
  - Python 2: 2020년 공식 지원 종료
  - Python 3: Python 2의 문제점 개선, 현재 표준

Python – 기본 문법

- 변수 선언

	자료형
x = 10	(integer)
name = “Alice”	(string)
lst = [1, 2, 3, 4]	(list)
flag = True	(boolean)

  - 타입 선언 없음
  - 실행 중 타입 결정 - **동적 타이핑(Dynamic Typing)**

x = 10	(integer)
x = “Bob”	(string)

Python – 기본 문법

- 자료형(Data Type)
  - 변수에 저장되는 데이터의 종류

- 정수형(int)
  - 소수점이 없는 숫자

x = 10

- 실수형(float)
  - 소수점이 있는 숫자

x = 3.14

- 불리언형(bool)
  - 참과 거짓(True or False) 나타냄

flag = True

Python – 기본 문법

- 자료형(Data Type)
  - 변수에 저장되는 데이터의 종류

- 문자열(str)
  - 문자(글자)들의 집합
  - 따옴표로 감싸면 문자로 인식 ('…' 혹은 "…")

```
name = "Kim"
name = 'Lee'
```

- 리스트(list)
  - 순서가 있는 여러 데이터의 목록
  - 대괄호 사용
  - 인덱스(index)로 데이터 접근

```
x = [1, 2, 3]
y = ["a", "b", "c"]

x[2] = 10
y[1] = "x"
```

Python – 기본 문법

- 자료형(Data Type)
  - 변수에 저장되는 데이터의 종류

- 딕셔너리(dict)
  - 키(Key)와 값(Value)의 쌍으로 데이터를 저장
  - 중괄호를 사용

    student_name_id = {"alice": 20260001, "bob": 20260003}

  - 키 사용해서 값에 접근

    student_name_id["alice"]

    student_name_id["bob"] = 20260002

    → {"alice": 20260001, "bob": **20260002**}

Python – 기본 문법

- 입출력
  - 프로그램은 사용자와 입력/출력으로 상호작용

- 입력(input)
  - 사용자로부터 값을 입력 받음
  - 데이터를 받아 변수에 저장

```
name = input()
```

- 출력(output)
  - 결과를 보여줌
  - 변수에 담긴 값이나 결과를 화면에 보여줌

```
print("Hello")
```

```
name = input()
print(name)
```

Python – 기본 문법

- 조건문(Conditional Statement)
  - 주어진 조건(Condition)이 참(True)인지 거짓(False)인지에 따라 프로그램 실행 흐름을 바꾸는 문법
  - if (elif) else 구조
  - Python에서는 **들여쓰기(indentation)** 강제

```
if x == 0:
    print("zero")
elif x > 0:
    print("positive")
else:
    print("negative")
```

x = 0 → "zero"

x = 100 → "positive"

x = -23.5 → "negative"

Python – 기본 문법

- 반복문(Loop Statement)
  - 동일하거나 유사한 작업을 정해진 횟수만큼, 혹은 특정 조건이 만족할 때까지 되풀이하는 문법
  - for, while

```
for i in range(3):
    print(i)
```

range(n): 0이상 n 미만 정수 범위 생성

→ 출력 0
1
2

```
x = 0
while x < 3:
    print("Hi")
    x = x +1
```

→ 출력 "Hi"
"Hi"
"Hi"

Python – 기본 문법

- 함수(function)
  - 특정 작업을 수행하는 코드의 묶음
  - 필요할 때마다 재사용 가능
    - 같은 코드 반복하지 않을 수 있음
  - 요소
    - 입력(input, parameter): 함수가 작업 위해 외부에서 전달받는 데이터
    - 처리(process, body): 실제 실행 코드
    - 출력(output, return): 작업 끝내고 최종적으로 내놓는 결과물
  - "**def**" 키워드 사용

def 함수이름(*입력*):
    처리
    return *출력*

```
def add(a, b):
    result = a + b
    return result
```

Python – 기본 문법

- 함수(function)
  - **호출(call)**되어야 실제 실행 됨

```
def add(a, b):
    result = a + b
    print(result)
    return result

x = add(2, 5)
print(x)
print(result)
```

→ ?

  - 일반적으로 함수 내에서 선언한 변수는 함수가 종료되면 사라짐

Python – 기본 문법

- 라이브러리(library)
  - 특정 기능을 수행하기 위해 미리 작성 된 코드 묶음/집합체
  - 복잡한 기능을 직접 구현하지 않아도 됨
    - 재사용
  - Python은 다양한 라이브러리가 잘 구축되어 있음
  - "import" 키워드 사용해서 외부의 기능을 현재 코드 안으로 가져옴

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

```
import random
x = random.randint(1, 45)
print(x)
```

Python – Examples

- 입력

```
x = input("Enter a number: ")

print(x)
```

```
x = input("Enter a number1: ")
y = input("Enter a number2: ")

z = (x+y)/2        ???
```

- 입력 + 조건문

```
x = int(input("Enter a number: "))

if x % 2 == 0:
        print("Even")
else:
        print("Odd")
```

input(): 입력으로 받은 데이터를 문자열(string)을 반환

Python – Examples

- 반복문

```
n = 5
total = 0

for i in range(1, n+1):
    total += i

print(total)
```

i	total
1	1
2	1 + 2 =3
3	3 + 3 = 6
4	6 + 4 = 10
5	10 + 5 =15

Python – Examples

- 리스트 + 반복문

```
nums = [1, 2, 3, 4]
total = 0

for i in range(4):
    total += nums[i]

print(total)
```

index 기반 접근

```
nums = [1, 2, 3, 4]
total = 0

for x in nums:
    total += x

print(total)
```

item 직접 접근

Summary

- Programming languages
  - C
  - Java
  - Python
  - …
- Python basic
  - Data type
  - Conditional statement
  - Loop statement
  - Function
  - Library
  - …