**컴퓨터 개론(코드잇 필기)**

**[프로그래밍 시작하기 in Python]**

1. 파이썬 첫 걸음

Phython interpreter: 파이썬 코드를 머신 코드(0,1)로 변역

Pycharm: 올인원 솔루션, 통합 개발 환경(IDE)

2. 프로그래밍 기본 개념

(1) 코멘트

# 붙이고 내용을 쓰면, #이하의 내용은 코드로 인식하지 않음. = 주석을 다는 것

->복잡한 코드를 설명, 하다만 부분을 표시, 다른 개발자와의 소통 등을 위해 사용함.

(2) 자료형(Data type): 계산기가 숫자를 사용하듯, 프로그래밍에서 사용하는 것들.

숫자열(Number)

정수(Integer): -2, -1, 0, 1, 2

소수(Floating Point): 1.7, 2.0 (정수부분 + 소수부분)

문자열(String): "Hello world!", "2" (Ex. 따옴표 안에 키보드로 쓸 수 있는 문자들을 표현한 것)

☞ 2, 2.0, "2" 이 세가지는 다 다른 type임. 2+5=7이지만,"2"+"5"="25"임

불린(Boolean): True, False (참과 거짓, 주로 값을 비교할 때 사용)

(3) 추상화

추상화=복잡한 내용은 숨기고 주요 기능만 -> 변수(Variable), 함수(Function), 객체(Object)

변수(Variable): 값을 저장하는 것

Ex. X=25 -> 25를 x에 저장

☞ 프로그래밍에서 =는 등호가 아니라 오른쪽의 값을 왼쪽의 변수에 지정해주는 지정연산자

함수(Function): 명령을 저장하는 것

Ex. Print(x+y) -> x+y를 콘솔에 나타나게

내장함수(파이썬 개발자가 미리 만들어 둔 것)

사용자 지정 함수: 새롭게 사용자가 정의하는 것.

def hello():  
 print("Hello!")  
 print("Welcome to Codeit!") #hello 함수는 이 두줄을 저장하는 함수

☞ 이때 def hello():를 해더라고 함.

☞ 해더의 hello를 파라미터(함수에 넘겨주는 값)이라고 함.

한 함수에 여러 개의 파라미터가 올 수 있음.

def print\_sum(a, b): #파라미터 두개  
 print(a + b)  
print\_sum(7, 3)  
def print\_sum(a, b, c): #파라미터 세개  
 print(a + b + c)  
print\_sum(7, 3, 2)  
def print\_sum(num\_1, num\_2, num\_3, num\_4): #파라미터 네개  
 print(num\_1 + num\_2 + num\_3 + num\_4)  
print\_sum(7, 3, 2, 1)

리턴(Return): 함수에게 어떤 정보를 주면 다른 정보로 돌려준다.

**[프로그래밍 언어 이해하기]**

1. 프로그래밍 언어 살펴보기

많은 프로그래밍 언어가 있고, 진로와 적성에 따라 사용해야하는 게 다 다름.

하지만, 무엇이 내가 배워야하는 것인지 모르기 때문에 전반적인 틀을 배울 필요가 있음.

컴퓨터: 일을 대신해주는 기계

프로그램: 컴퓨터가 할 일을 알려주는 설명서

프로그래밍 언어: 설명서를 쓰기 위해 사용하는 언어

초창기 프로그래밍 언어?

컴퓨터는 계산을 위해 제작 파스칼 계산기(톱니바퀴가 달린 계산기)가 최초의 컴퓨터, 숫자 계산식이 프로그램, 계산을 얻어낼 수 있게 톱니바퀴를 돌리는 것이 프로그래밍 언어에 해당.

현대의 프로그래밍 언어

숫자 계산을 넘어 더 많은 일을 컴퓨터에게 시키려고 기능이 다양한 컴퓨터와 다양한 프로그래밍 언어를 만듦(영어, 숫자, 괄호, 따옴표, 점, 세미콜론 등의 다양한 기호로 구성)

용도, 상황, 프로젝트에 따라 사용하는 언어가 달라짐.

2. 프로그래밍 언어를 분류하는 두 가지 기준

(1) 객체 지향 지원 여부

객체 지향=프로그래밍의 접근 방식 중 하나, 지배적으로 사용(Java, Python), Bottom up

Top-down(큰 걸 작은 것으로 쪼개기) 방식으로 프로프로그래밍을 하면, 코드 전체가 완성되지 않으면 동작을 제대로 확인할 수 없고, 하나의 기능 수정 시 모든 코드를 점검해야 하는 문제점을 해결하기 위해 등장.

Top down처럼 기능(함수)를 기준으로 나누는 게 아니라 함수랑 데이터를 기준으로 조금 더 의미 있는 단위로 나눔(객체, Object)

장점:

데이터나 기능의 변화가 생기더라도 관련 객체만 신경 쓰면 되니 프로그래밍하기 훨씬 수월함

조금 더 의미있는 단위로 나누니 코드를 이해하거나 다른 프로그램에서 재사용하기 수월함.

전체가 완성되지 않더라도 일부 기능만 사용할 수 있어 테스트에 유리

에러나 수정 사항이 발생해도 전체가 아닌 객체만 확인하면 됨.

☞ 많은 데이터를 다루는 프로그램, 규모가 큰 프로젝트에는 지향.

단점:

코드를 잘 나누고 설계하기 어려움 (코드를 작성하기 전에 설계에 많은 노력 필요)

사람이 작성하고 이해하기 편한 방식으로 코드를 나누다 보니, 컴퓨터가 이해하는데 시간이 걸려 실행 속도가 느려지거나, 저장공간을 많이 차지하기도 함.

☞ 극도로 빠른 실행 속도가 필요한 분야, 저장 장치나 성능이 제한적인 기기(스마트워치, 스마트 냉장고 등)에 들어가는 프로그램 만들 때, 임베디드와 같은 특정분야에서는 지양

Ex. sns앱을 만들떄

Top down:

로그인 기능, 팔로우 기능, 차단 기능 등, 기능을 기준으로 나눠 각각의 개발자가 만들고, 후에 하나로 합쳐 실행.

Bottom up:

사용자라는 객체를 만들고, 이와 관련있는 아이디, 비밀번호, 자기 소개 데이터 등을 객체가 갖게 함. 사용자 기능으로는 로그인 기능, 친구추가 기능, 게시글 작성 기능 등을 줌.

게시글이라는 객체를 만들고, 텍스트 내용, 업로드 시간, 좋아요 수 등의 데이터를 갖게 하고 게시글 작성 기능을 줌.

(2) 변수의 데이터 타입 실행 전 지정 여부

Ex. 변수 지정 시

Python: 변수명, =, 값 -> 세 가지 요소로만 이루어짐

Java: 자료형(변수의 데이터 타입 지정), 변수명, =, 값 -> 네 가지 요소로 이루어짐

Dynamic typing: 변수에 자료형을 지정해주지 않은 언어 (Python, Ruby)

간결하고 편리함을 추구해 컴퓨터적 구조가 생략되는 경우가 많아 코딩 속도가 빠르지만 실행속도가 느림. 코드의 내용과 로직이 눈에 잘 들어옴. 배우기 쉬움.

☞ 혼자서 빠르게 하는 프로젝트에 적합

Static typing: 변수에 자료형을 실행 전에 미리 지정하는 언어 (Java, C)

명확하고 정교한 코드를 추구하기 때문에 코드가 길어져 코딩 속도가 느리지만, 많은 컴퓨터적 구조들을 사람이 적어주어 실행속도가 빠름. 코드의 구조가 눈에 잘 들어움. 사람의 실수를 방지하고 더 체계적으로 코드를 작성하기 위해 제약이나 규칙이 많음. 배우기 어려움.

☞ 많은 사람이 협업해야 하는 규모가 크고 복잡한 프로젝트에 적합(임베디드, 실시간 투자, 인공지능)

(3) 컴퓨터와 가까운가? 인간과 가까운가?

상대적인 개념(스펙트럼)

인간과 가까운 언어=고수준 언어, High-level programing lanuage

컴퓨터와 가까운 언어=저수준 언어, Low-level programing lanuage

☞ 변수의 자료형과 연관이 있음

컴퓨터: 1과 "1"을 다르게 인식(문자열과 숫자열로 각각 인식)

사람: 1과 "1"을 똑같이 인식

Static typing은 자료형의 type을 지정해주기 때문에 컴퓨터가 더 빨리 인식

Dynamic tying은 컴퓨터가 자료형을 스스로 생각해야 해서 인식이 느림.

☞ 일종의 추상화

추상화= 불필요한 것은 지우고 필요한 것만 들어내는 것

이에 따라 문법, 코드를 작성하는 방식, 코드를 바라보는 관점이 달라짐 (초점을 어디에 둘 지가 달라짐)

고수준 언어: 프로그래머가 필요한 기능과 논리에 집중할 수 있지만, 컴퓨터에게 너무 많은 걸 맡겨 고사양을 요구하고, 실행시간이 느림.

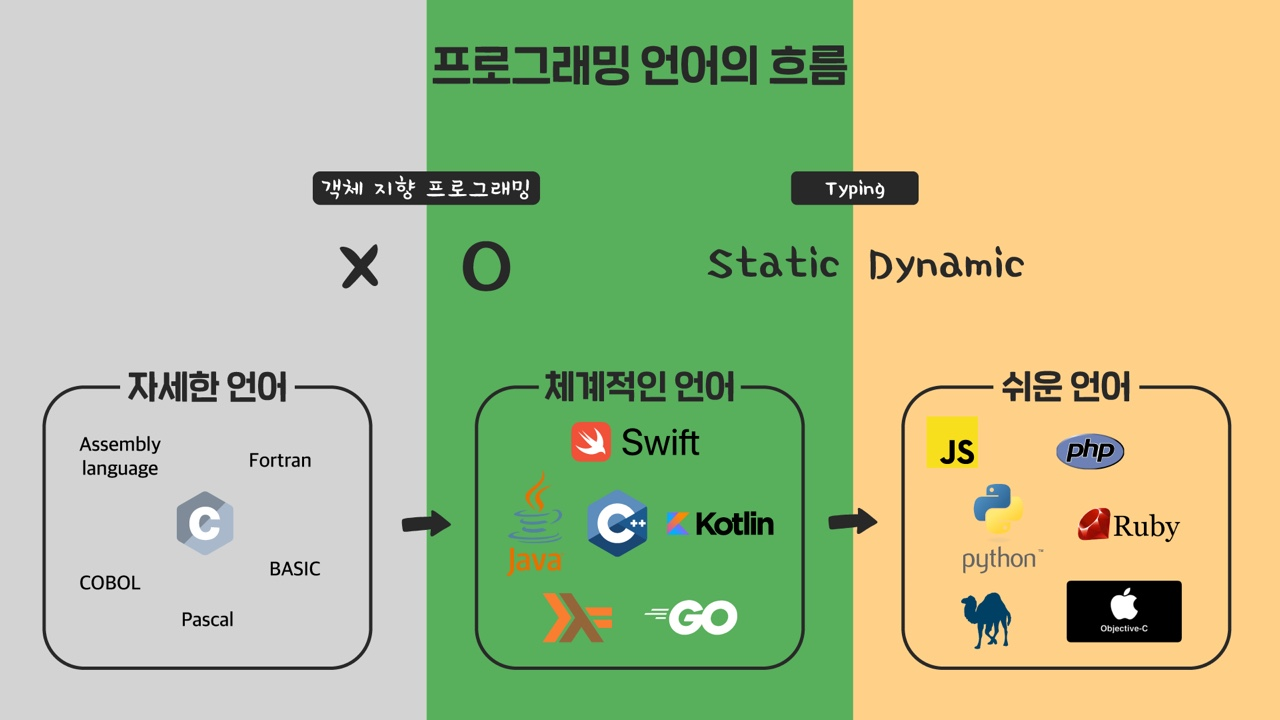
저수준 언어: 프로그래머가 복잡하게 코드를 작성해야 하지만, 컴퓨터는 기계적으로 작동하기만 하면되니 제한적인 사양에서도 효율적으로 동작함.

3. 프로그래밍 언어 Overview

프로그래밍 언어는 크게 3가지 종류 (자세한 언어, 체계적인 언어, 쉬운 언어) + 특별한 목적을 위한 프로그래밍 언어(Domain-specific programming language)가 있음.

자세한 언어>체계적인 언어>쉬운 언어 순으로 발전

☞ 컴퓨터와 칩의 성능이 좋지 않던 시기에는 자세한 언어가 등장, 협업과 생산성 향상에 좋은 객체 지향 프로그래밍이 대세가 되면서 체계적인 언어가 인기, 누구나 프로그래밍을 배우고 할 수 있는 지금은 쉬운 언어가 인기



(1) 자세한 언어

☞Low-level에 가깝고 객체지향 개념이 적용 안되는 static typing인 어셈블리 언어, C 언어

장점: 컴퓨터의 논리적 사고방식에 맞춰진 언어이기 때문에 실행 속도가 빨라 성능이 안 좋은 컴퓨터에 적용하기 용이함.

단점: 객체 지향 개념이 적용되지 않고, 코드를 작성할 때 자료형을 써주는 등 컴퓨터적 구조를 다 고려해야 해서 코드 작성과 수정이 어려움.

Ex. 임베디드 시스템

빠르고 효율적인 프로그램을 만드는데 사용되어 저장공간이나 성능이 열악한 기계에 핵심적으로 사용됨 (스마트 워치, TV, 전광판 등에 사용)

Ex. 빅테이터 처리 속도와 알고리즘이 중요한 분야

사람이 코딩하는 시간을 길어지지만, 컴퓨터의 실행 시간을 조금이라도 빠르게 하기 위해 수많은 문자를 실시간으로 전송해야 하는 채팅 어플리케이션, 트위치 같은 동영상 스트리밍 서비스, 검색 엔진, 개발자들이 코드 관리를 위해 쓰는 github, 머신 러닝 등과 같은 분야에서 사용된다.

(2) 체계적인 언어

☞ Low-level에 가깝지만, 객체지향 개념이 적용되는 static typing인 Java, C++

장점: 짜임새와 협업이 중요한 규모가 큰 프로젝트에 적합하다. 객체 지향 개념이 적용되어서, 자세한 언어보다 유지보수하기 용이하다.

단점: 자세한 언어보다는 아니지만, 문법이 어렵다. 또한 코드를 작성하는 시간보다 체계적으로 설계하는 데에 시간이 오래 걸릴 수 있다.

Ex. 규모가 큰 기업들 (구글, 에어비엔비, 우버, 이베이, 핀터레스트, 스포티파이, 아마존, 등)

(3) 쉬운 언어

☞객체 지향 개념이 적용된, dynamic typing 언어인 Phython과 Ruby

장점: 배우기 쉽고, 코드를 읽고 수정하기에 가장 간편하며 코드를 빨리 작성할 수 있다.

단점: 컴퓨터적 구조가 많이 생략되어, 컴퓨터의 코드 실행 속도가 느리다. 성능 제약이 심한 곳에서 빠른 속도를 내야하는 경우, 큰 규모의 프로젝트에서 실수를 방지해야 하는 경우, 많은 사람들과의 협업을 해야 하는 경우에는 부적합하다.

Ex.

(4) 특별한 목적을 위한 프로그래밍 언어(Domain-specific programming language)

R: 통계 소프트웨어 개발, 데이터 분석 및 시각화에 활용됨. 데이터 전처리, 분석, 수학적 계산, 그래프 그리는 기능에 특화되어 있음.

SQL: 데이터 베이스 관리하기 위한 목적. 데이터를 모아둔 공간인 데이터 베이스를 설계하고 운영 및 관리할 때 이용함.

HTML, CSS: 웹페이지를 만들 때 사용함. 웹페이지의 구조와 레이아웃은 HTML, 웹페이지의 시각적인 표현은 CSS, 사용자와의 상호작용할 수 있게 동작하는 것은 JavaScript를 사용함.

MATLAB: 공학 시뮬레이션용

Csound: 소리와 음악 합성에 사용

Scratch: 블록을 조립하는 형태의 교육용

4. 프로그래밍 언어 제대로 사용하기

(1) 가독성 높은 코드

프로그래밍 언어: 컴퓨터, 다른 개발자와 소통하기 위해 사용

좋은 코드= 사람이 읽기 좋은 코드= 가독성이 좋은 코드

\* 라이브러리, 개발 문서, 커뮤니티, 구글에 Best practice 검색 등에서 남이 짠 좋은 코드들을 익히는 것이 중요

1) comment

컴퓨터가 아닌 사람에게만 전하는 메시지, 언어마다 쓰는 방식이 다름 (Phython은 #, C는 //)

2) 이름을 잘 짓기

변수, 함수 등의 이름이 너무 일반적이거나 구체적인 명칭이면 안됨. 전체적인 코드의 흐름 안에서 충분히 구체적이고 간결한 이름을 지어야 함. 변수의 이름은 코드 내에서 일관성이 있어야함.

\* 코멘트, 변수 등은 모두 주관적인 성격이 있어, 일정한 규칙(Style guide)을 지켜야 가독성이 높아지고 여려 명이 작업할 때 혼선을 방지할 수 있음.

ex. 변수

변수 이름은 보통 띄어쓰기 불가라 다양하게 표현됨

math-score(snake 표기법), MathScore(pascal 표기법), mathScore(camel 표기법) 등이 있음.

(2) 구조화가 잘 된 코드

의미 있는 코드들을 묶어서 코드 분리를 해야함(파일분리: 관련된 코드를 필요한 만큼 한곳에 모아두는 것). 코드 분리를 하면 내가 프로그램의 기능을 바꾸고 싶을 때, 오류가 생겼을 때 원하는 부분만 변경할 수 있음.

(3) Design Pattern

프로그래밍에서 고질적으로 발생하는 문제들을 해결하기 위한 테크닉을 모아둔 책

(4) 라이브러리

자주 쓰일 법한 비슷한 코드들을 모아둔 것,

언어마다 맞는 라이브러리가 있음(ex. Pandas, seaborn은 Python으로만 라이브러리가 있음),

(5) 프레임워크

프레임워크는 프로그래밍의 뼈대가 이미 만들어져 있고, 내가 코딩해야 하는 부분이 비워져 있기 때문에 적은 양의 코드만으로도 프로그램을 만들 수 있음. 다만, 이미 짜여진 틀 안에서 프로그래밍을 하기 때문에 프레임워크의 구조를 바꾸는 것은 쉽지 않음

☞ 라이브러리 vs 프레임워크

라이브러리: 틀은 내가 프로그래밍 해야 하지만, 이미 누가 만들어 준 도구를 가져와서 그대로 사용, 코드를 가져오는데 돈이 안 듦.

프레임워크: 틀이 다 짜여 있어서 최소한의 노력으로 프로그램을 완성할 수 있음. 내가 바꾸고 싶은 곳이나 채우고 싶은 곳만 코드를 채워넣으면 됨.

**[프로그래머의 세계 이해하기]**

1. 프로그래밍의 다양한 분야

(1) 컴퓨터 사이언스의 기본

어느 프로그래밍 언어든 기본기가 충실하면 다른 언어, 환경에서도 쉽게 코딩할 수 있음,

☞ 객체 지향 프로그래밍, 알고리즘, 자료구조

+ 컴퓨터 구조, 운영 체제, 컴파일러, 웹 개발, 데이터베이스, 네트워크 등이 있음.

(2) 소프트웨어 공학

소프트웨어를 활용하는 공학, 소프트웨어어로 제품, 서비스를 만드는 방법에 대한 학문

어떤 제품이나 서비스를 기획(어떤 것을 만들지), 개발(기획한 것 만들기), 테스트(개발이 기획대로 잘 되었는지 확인), 출시/배포(개발한 제품, 서비스를 사용자가 사용), 유지/보수(출시된 서비스를 변화시킴) 등의 모든 프로세스를 포함함.

(3) 프로세스 관리

소프트웨어 출시 과정: 기획>개발>테스트>배포>유지/보수

관리자, 기획자, 디자이너, 개발자, 테스터 등의 여러 사람이 협업함 -> 각기 다른 분야 종사자들이라 소통에 어려움을 겪음 -> 다양한 협업방식 고안(제품의 성격, 팀 문화 등에 따라 달라짐.)

☞ 폭포수(Waterfall): 각 단계를 완료하고 다음 단계로 넘어가는 방식, 이해하고 관리하기 쉬움. 다만, 각 단계를

완벽하게 끝내기 힘들기에 수정사항이 많이 생길 가능성이 높고 개발자들이 수정사항을 반영하기 힘듦. 프로그램의 기획, 계발 등의 단계가 복잡하지 않을 경우 용이. 만들고자 하는 서비스가 복잡하다면 요구사항 변경이 잦아 설계를 다시 해야하는 경우, 처음으로 돌아가는 비효율이 발생할 수도 있음.

애자일(Agile): 결과물을 만들어 미리 확인하고 수정하며 진행. 프로그램을 적당한 크기의 기능으로 나누고,

기능에 대해 실제 동작하는 소프트웨어를 확인하며 소통하는 방식임. 중간 중간 사용하면서 발전시키기 때문에 기능 변경이 유연함. 기능별로 나누어 프로그램을 실행하면서 확인하다보니 관리가 복잡하고 어려워 프로젝트의 크기가 커질 수 있다는 단점이 있음. 스크럼(소규모 팀에서 단시간 동안 작은 것을 만들도록 하되, 전체적인 모습을 볼 수 잇게 정기적으로 통합)과 칸반(일을 작은 조각으로 나누고, 이를 카드에 써서 벽에 붙이고 작업별 현황을 카드에 시각화)을 통해 애자일 방식을 쉽게 도입하기도 함.

(4) 테스트 프로세스

버그 발생시 얼마나 시급한지, 담당자는 누구인지, 빼먹은 버그가 있는지, 버그가 잘 고쳐졌는지 확인해야함.

버그 관리 툴: Jira, asana, Trello

Issue: 어떤 상황에서 버그가 발생하는 지, 얼마나 시급한 버그 인지인지 등의 정보를 버그 관리 툴에 입력한 것

Issue tracking tool: issue들을 모아 둔 정보 관리 시스템

Issue의 상태: open(이슈가 생김)>In-progress(개발자가 개선 작업 진행)>Resolver(개발자가 작업 완료해 문제가 해결됨)>Closed(테스트 결과 버그가 잘 해결됨)>Reopen(추가적인 문제 발생시 다시 Open으로 돌아감)

Assign: 상태의 각 과정에 담당자를 지점

이슈가 너무 많아지면, 담당자에게 할당이 되지 않아 오랫동안 해결되지 않는다거나, 한 사람에게 많은 이슈가 몰리거나, 급한 이슈가 아니라 덜 급한 이슈가 진행되는 등 이슈가 많고 프로젝트가 큰 경우 원활한 이슈 진행을 위해 관리자, 프로젝트 매니저가 있기도 함.

(6) 버전 관리

프로그래밍 하다 보면 수정이 잦아, 수정 이력이나 버전 관리를 하는 것이 어려움

Git(버전 관리를 위한 소프트웨어), git hub(git을 이용하는 온라인 코드 저장 공간)

Git에서는 저장 용량을 줄이고 각 상황에서의 변화를 코멘트로 기록하기 위해 수정을 하게 되면 수정 사항, 수정한 사람, 변경된 부분, 왜 그런 수정을 했는지 기록함. 특정 버전을 확인하고 싶다면, 처음부터 그 시점까지의 수정사항을 합치면 확인할 수 있음.

라이트-스탠다드-프리미엄(비슷한 여러 버전의 프로그램)처럼 하나의 프로그램에서 나뭇가지처럼 뻗어 나가는 프로그램, 다양한 버전의 프로그램 등을 만들 때 유리함.

(7) IDE

통합 개발 환경(Integrated Development Environment): 개발자들이 코딩할 때 도움이 되는 기능을 모아둔 프로그램

2. 트렌드를 읽는 능력

프로그래밍은 선택의 연속

내가 만들고 싶은 프로그램에게 가장 적합한 라이브러리와 프레임워크, 프로그래밍 언어를 트렌드에 맞춰 잘 선택해야함.

(1) 활발한 기술

좋은 언어/프레임워크/기술은 사람들이 활발하게 사용>커뮤니티, 좋은 자료, 문서들이 잘 만들어짐>더 많은 사람들이 사용

활발한 기술이 좋은 기술->많은 사용자가 있는지? 기술이 활발한 지

커뮤니티(stackoverflow, Quora)에 그 언어 관련 자료, 오류에 대한 답변 등이 많음.