인터프리터 언어

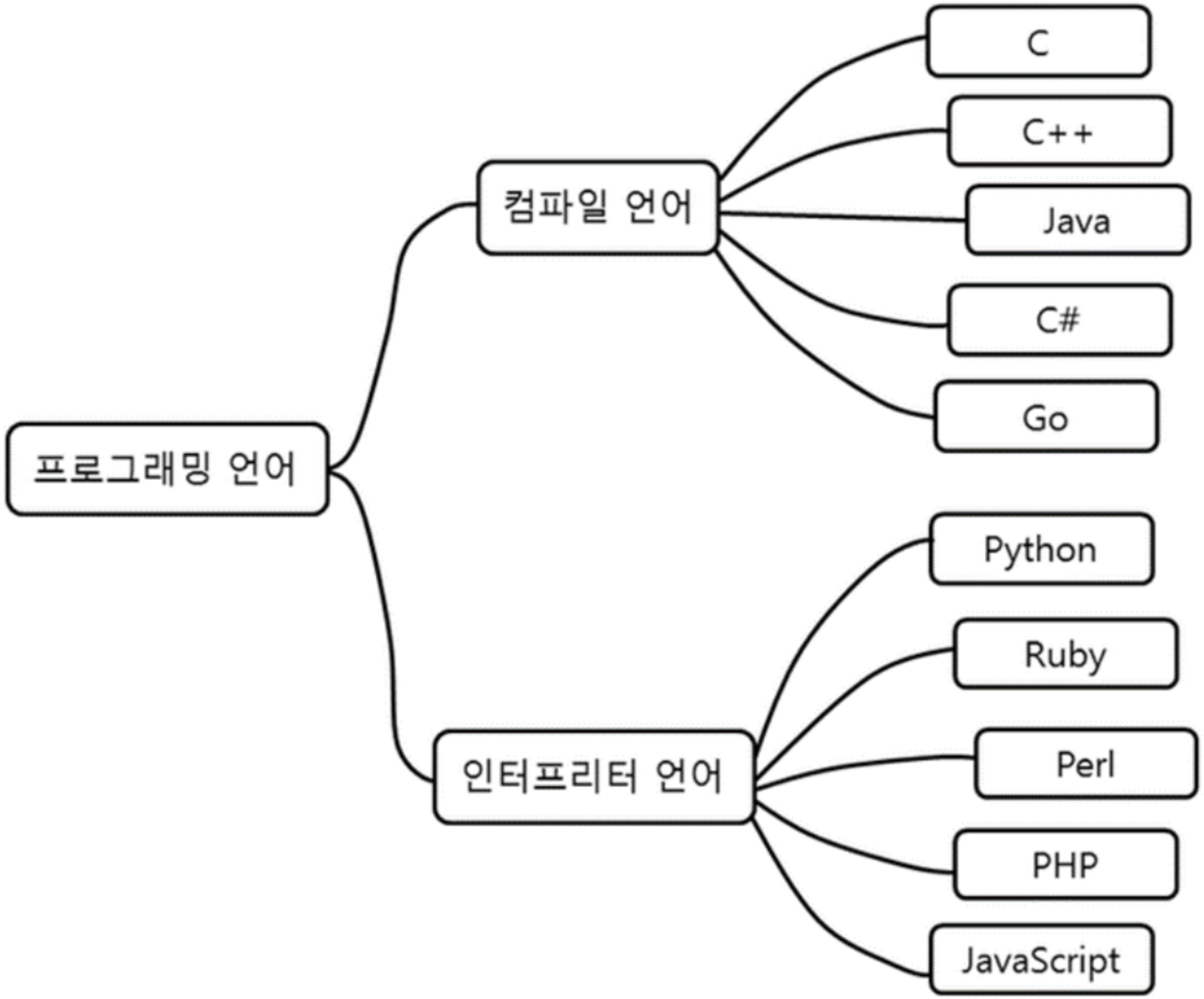
: 컴파일러 vs 인터프리터

92313350 박태민

프로그래밍 언어 만들기 part.1

목차

컴파일러 vs 인터프리터

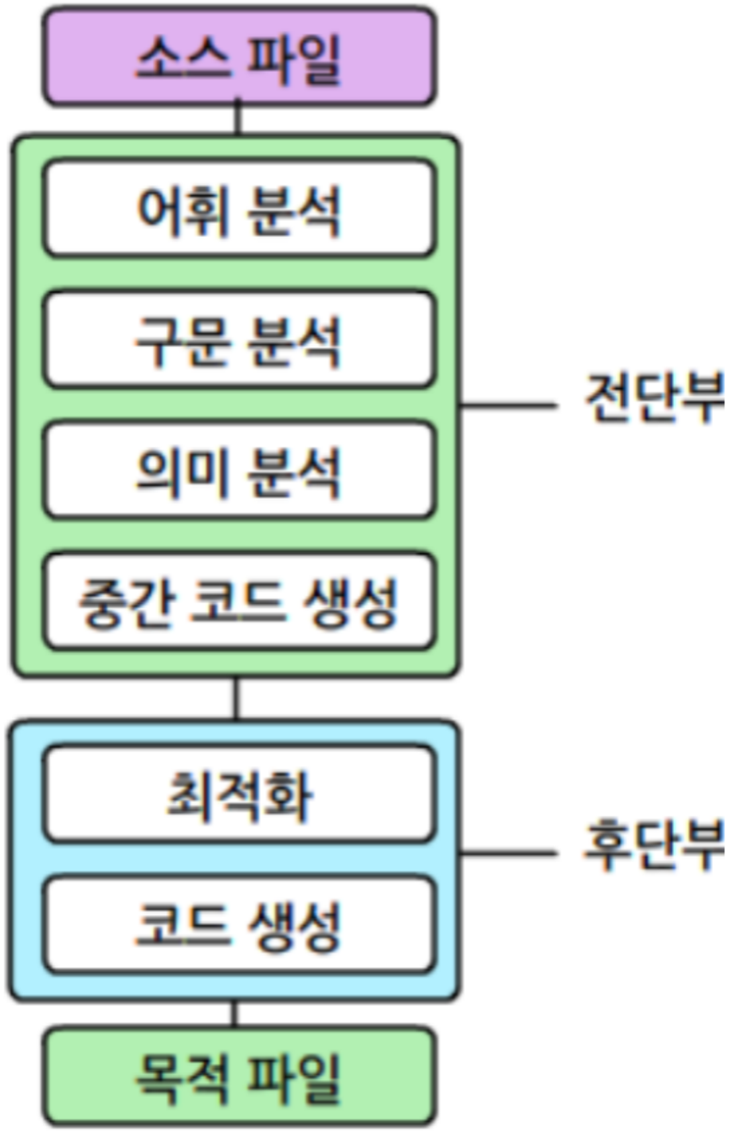


01

컴파일러

컴파일러를 간단 히 말하면 특정 프로그램 소스 코드를 기계어로 변환하는 것

장단점: 개발 편의성은 떨어지지만, 실행 속도는 빠르다. 한꺼번에 컴파일을 하기 때문에 컴파일 시간은 오래 걸리지만 실행 단계에서 이미 기계어로 변환된 목적 파일을 실행만 하면 되므로 속도가 월등히 빠르다.
전체 코드를 컴파일 후에 에러를 알려주므로 수정이 용이하지 않다



02

인터프리터

인터프리터는 실행 전에 컴파일을 하지 않고 바로 실행하는 일종의 가상머신이나 실행 환경이다.

장단점: 개발 편의성이 높지만, 실행 속도는 느리다.
변환과 실행을 동시에 진행해야 하므로 프로그램 자체 속도는 느리다. 소스 코드 한 줄을 변환해서 바로 실행하기 때문에 실행 시작 시간은 빠르지만 전체 실행 속도는 컴파일러가 훨씬 빠르다.
소스 코드를 한줄씩 실행하므로 에러를 바로 알려줘서 실시간 코드 수정이 가능하다.

프로그래밍 언어 만들기 실습

LEXER

TOKENS

```
# TOKENS
#######################################

TT_INT = '정수'
TT_FLOAT = '실수'
TT_PLUS = '더하기'
TT_MINUS = '빼기'
TT_MUL = '곱하기'
TT_DIV = '나누기'
TT_LPAREN = '왼쪽소괄호'
TT_RPAREN = '오른쪽소괄호'


class Token:
    def __init__(self, type_, value=None):
        self.type = type_
        self.value = value

    def __repr__(self):
        if self.value: return f'{self.type}:{self.value}'
        return f'{self.type}'
```

TOKENS,LEXER

```
tokens = []

while self.current_char != None:
    if self.current_char in ' \t':
        self.advance()
    elif self.current_char in DIGITS:
        tokens.append(self.make_number())
    elif self.current_char == '+':
        tokens.append(Token(TT_PLUS))
        self.advance()
    elif self.current_char == '-':
        tokens.append(Token(TT_MINUS))
        self.advance()
    elif self.current_char == '*':
        tokens.append(Token(TT_MUL))
        self.advance()
    elif self.current_char == '/':
        tokens.append(Token(TT_DIV))
        self.advance()
    elif self.current_char == '(':
        tokens.append(Token(TT_LPAREN))
        self.advance()
    elif self.current_char == ')':
        tokens.append(Token(TT_RPAREN))
        self.advance()
    else:
        pos_start = self.pos.copy()
        char = self.current_char
        self.advance()
        return [], IllegalCharError(pos_start, self.pos, "'" + char + "'")
```

LEXER

```
ef make_number(self):
    num_str = ''
    dot_count = 0

    while self.current_char != None and self.current_char in DIGITS + '.':
        if self.current_char == '.':
            if dot_count == 1: break
            dot_count += 1
            num_str += '.'
        else:
            num_str += self.current_char
        self.advance()

    if dot_count == 0:
        return Token(TT_INT, int(num_str))
    else:
        return Token(TT_FLOAT, float(num_str))
```

LEXER

```
#######################################
# LEXER
#######################################

class Lexer:
    def __init__(self, fn, text):
        self.fn = fn
        self.text = text
        self.pos = Position(-1, 0, -1, fn, text)
        self.current_char = None
        self.advance()

    def advance(self):
        self.pos.advance(self.current_char)
        self.current_char = self.text[self.pos.idx] if self.pos.idx < len(self.text) else None
```

POSITION

```
def __init__(self, idx, ln, col, fn, ftxt):
    self.idx = idx
    self.ln = ln
    self.col = col
    self.fn = fn
    self.ftxt = ftxt

def advance(self, current_char):
    self.idx += 1
    self.col += 1

    if current_char == '\n':
        self.ln += 1
        self.col = 0

    return self

def copy(self):
    return Position(self.idx, self.ln, self.col, self.fn, self.ftxt)
```

ERROR

```
    def __init__(self, pos_start, pos_end, error_name, details):
        self.pos_start = pos_start
        self.pos_end = pos_end
        self.error_name = error_name
        self.details = details

    def as_string(self):
        result = f'{self.error_name}: {self.details}\n'
        result += f'File {self.pos_start.fn}, line {self.pos_start.ln + 1}'
        return result


class IllegalCharError(Error):
    def __init__(self, pos_start, pos_end, details):
        super().__init__(pos_start, pos_end, 'Illegal Character', details)
```

MAIN

```python
import basic

while True:
    text = input('실행문 > ')
    result, error = basic.run('<stdin>', text)

    if error: print(error.as_string())
    else: print(result)
```

PROGRESS

```
C:\Users\SongM\anaconda3\python.exe C:/Users/SongM/PycharmProjects/web_gripper/main.py
실행문 > 1 + 3
[정수:1, 더하기, 정수:3]
실행문 > (2 * 5) - 3 + 2 / 3
[왼쪽소괄호, 정수:2, 곱하기, 정수:5, 오른쪽소괄호, 빼기, 정수:3, 더하기, 정수:2, 나누기, 정수:3]
실행문 > 5 + 안녕
Illegal Character: '안'
File <stdin>, line 1
실행문 > 2.31 + 4.2
[실수:2.31, 더하기, 실수:4.2]
실행문 > 💚 + ★
Illegal Character: '💚'
File <stdin>, line 1
실행문 >
```


THANK YOU!