파이썬 버그 찾기

오원석

26. 08. 26

SIGPL @ 한국문화연수원

파이썬 버그 찾기

- 파이썬 구현체의 버그 찾기

- 파이썬 라이브러리의 버그 찾기

파이썬 버그 찾기

- 파이썬 구현체의 버그 찾기
  - 최적화가 옳은지 **검증**하기

- 파이썬 라이브러리의 버그 찾기
  - **마구잡이 실행**(Fuzzing)으로 버그 찾기

파이썬 버그 찾기

- 파이썬 구현체의 버그 찾기
  - 최적화가 옳은지 **검증**하기

- 파이썬 라이브러리의 버그 찾기
  - **마구잡이 실행**(Fuzzing)으로 버그 찾기

Adaptive Specialization

적응형 특수화
Adaptive Specialization

~~적응형 특수화~~

Adaptive Specialization

익숙한 곳에서 지름길 타기


가장 무난해보이는 길
출발
도착
3 33

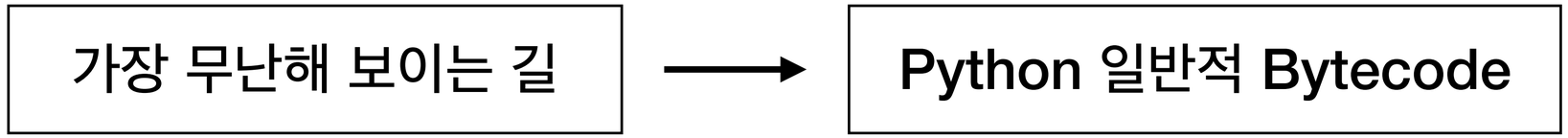
꽉 막히는 동대문역

시속 50km 제한

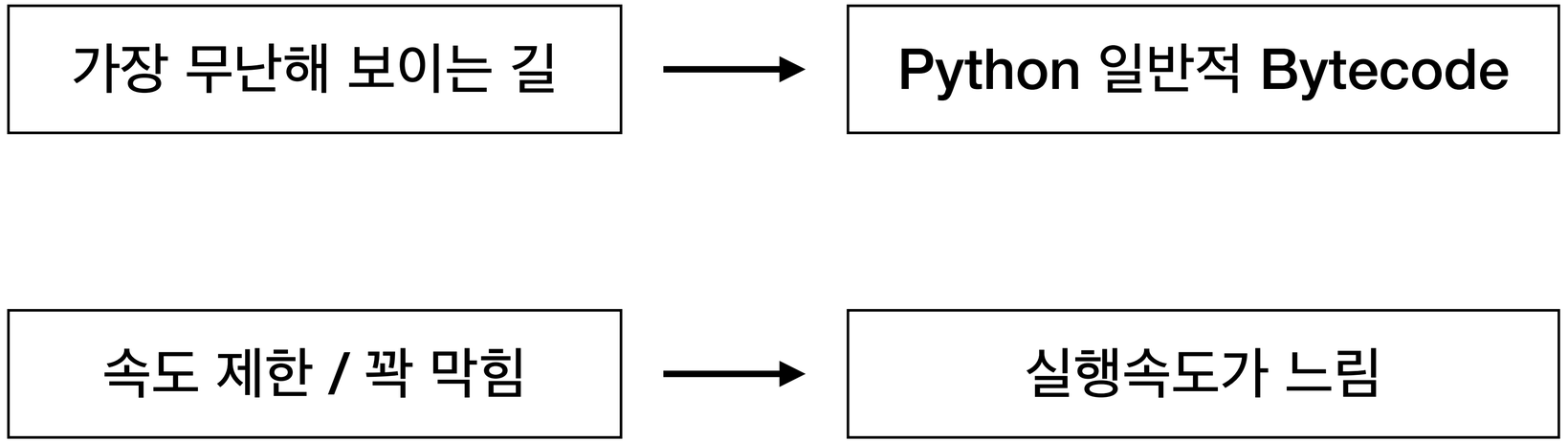



도착
출발
33분
서울 고수는 내부순환도로를 이용

익숙한 곳에서 지름길 타기



익숙한 곳에서 지름길 타기



익숙한 곳에서 지름길 타기

가장 무난해 보이는 길	→	Python 일반적 Bytecode
속도 제한 / 꽉 막힘	→	실행속도가 느림
서울 고수는 지름길 타기	→	반복되면 특수 Bytecode 사용

익숙한 곳에서 지름길 타기

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

- + 연산이 간단해 보이지만…
  - a와 b의 타입은 무엇인지…
  - 정의된 __add__ (__radd__)는 무엇인지…

익숙한 곳에서 지름길 타기

- 직관) **특정 라인의 코드**는 **"같은 타입"**이 실행될 때가 많다

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

- a + b여서 매번 타입검사를 해야할 것 같지만?
  - 실제 사용에서는 **int + int** 로만 쓰일 수 있다
- 이 점을 이용해 **특화 연산**을 활용하고자 한 것

~~적응형 특수화~~

Adaptive Specialization

익숙한 곳에서 지름길 타기

타입이 반복 되면 특수 Bytecode 사용

Adaptive Specialization

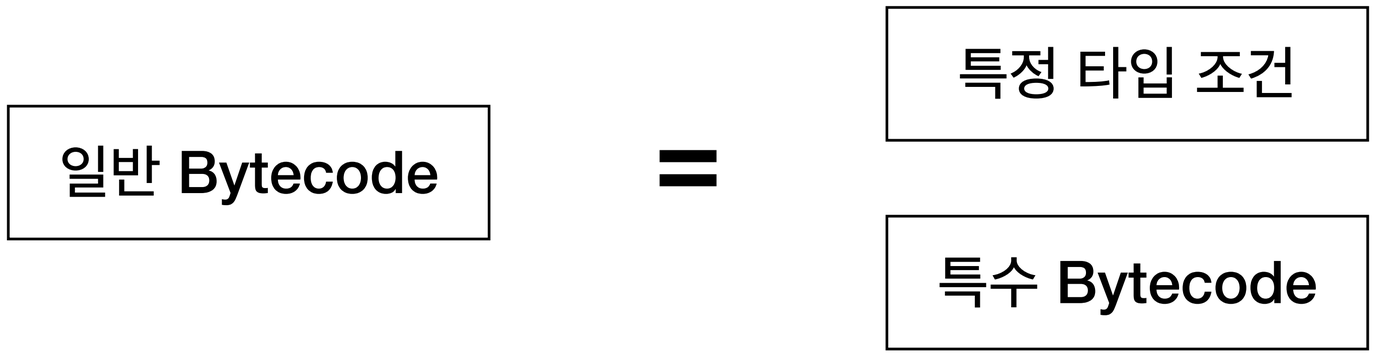
- Python 3.11 **속도 개선의 대부분을 담당**함
- 벤치마크들에 대해 약 **1.2~1.9배** 빨라졌다고 함

Adaptive Specialization

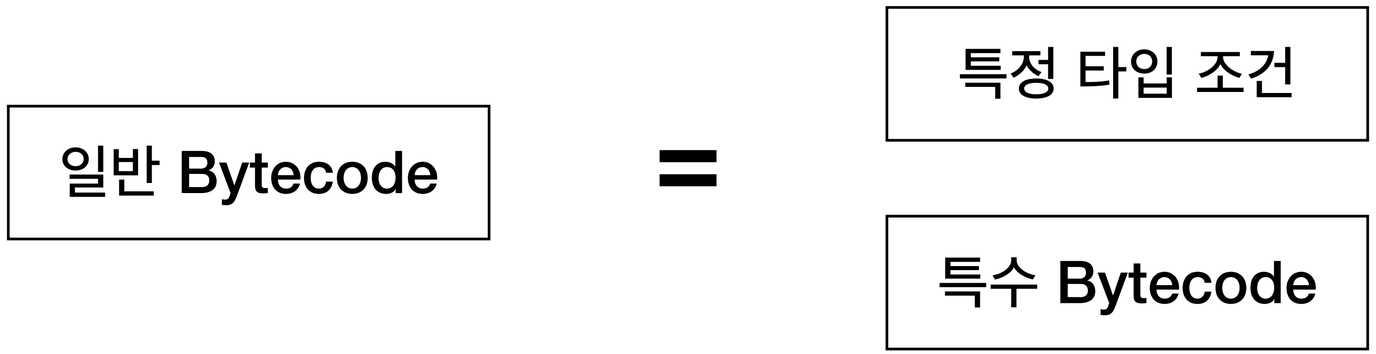
- Python 3.11 **속도 개선의 대부분을 담당**함
- 벤치마크들에 대해 약 **1.2~1.9배** 빨라졌다고 함

- JIT / Free-Threading을 도입 중인 Python
  - 반드시 Adaptive Specialization을 고려해야함

Adaptive Specialization 검증하기



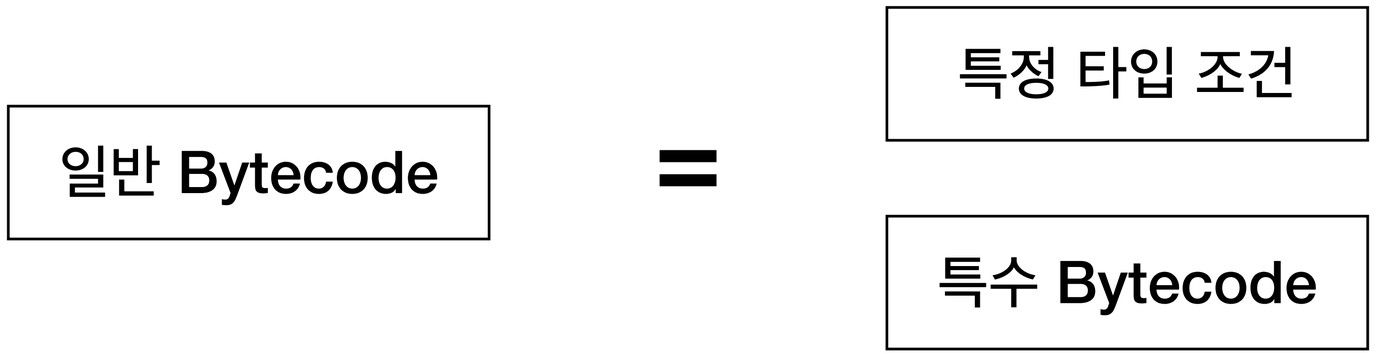
Adaptive Specialization 검증하기



(같은 스택 입력이 들어왔을 때, 같은 스택 출력을 내는가?)

(둘 다 예외처리를 적절하게 내는가?)

Adaptive Specialization 검증하기



(같은 스택 입력이 들어왔을 때, 같은 스택 출력을 내는가?)

(둘 다 예외처리를 적절하게 내는가?)

같은 Python 실행 상황에서 둘을 실행하면 값 / 스택 / 예외 / 분기 등의 결과가 같은가?

Adaptive Specialization 검증하기

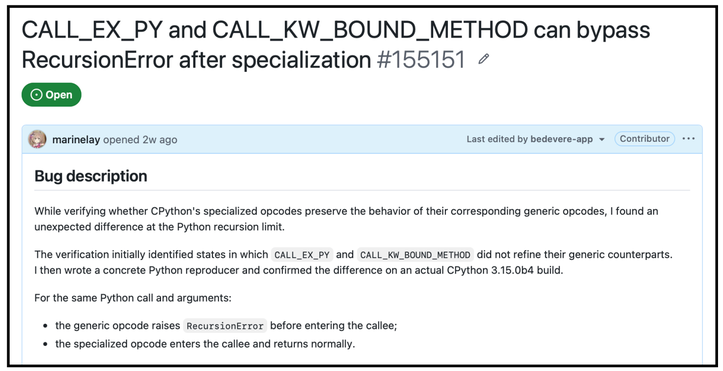
- 올바름(Soundness)을 증명
  - 증명이 되면, 이 **지름길 타기**는 **무난해 보이는 길**과 같다
  - 증명 실패 시, (가능하면) 반례를 찾아 수정 코드 생성

Adaptive Specialization 검증하기

- 올바름(Soundness)을 증명
  - 증명이 되면, 이 **지름길 타기**는 **무난해 보이는 길**과 같다
  - 증명 실패 시, (가능하면) 반례를 찾아 수정 코드 생성
- 빠트림없음(Completeness)은 부분적으로 보여줌
  - 조건을 완화해도 검증이 되는가? (무작위 변환 기반)
  - CEGIS를 이용해 최적화 함

결과

- 올바름 버그 2건 검출


CALL_EX_PY and CALL_KW_BOUND_METHOD can bypass RecursionError after specialization #155151

Open

marinelay opened 2w ago — Last edited by bedevere-app — Contributor

Bug description

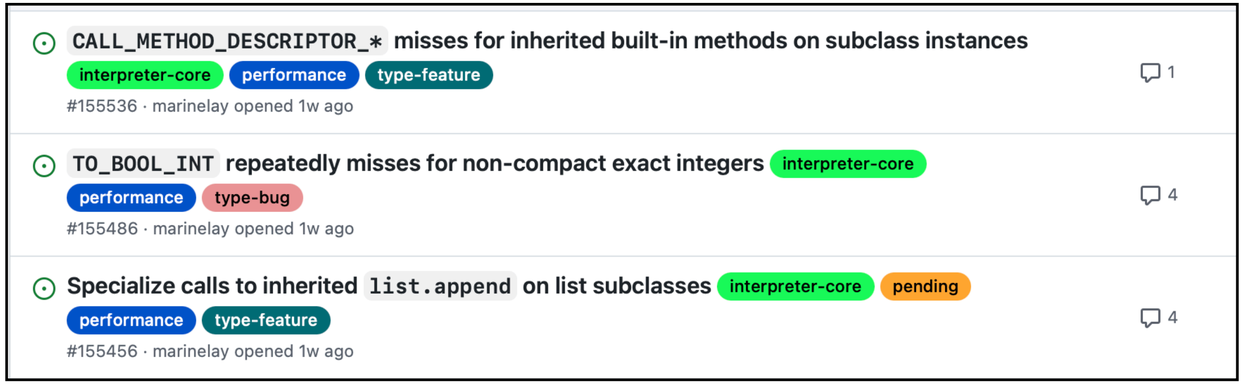
While verifying whether CPython's specialized opcodes preserve the behavior of their corresponding generic opcodes, I found an unexpected difference at the Python recursion limit.

The verification initially identified states in which `CALL_EX_PY` and `CALL_KW_BOUND_METHOD` did not refine their generic counterparts. I then wrote a concrete Python reproducer and confirmed the difference on an actual CPython 3.15.0b4 build.

For the same Python call and arguments:

- the generic opcode raises `RecursionError` before entering the callee;
- the specialized opcode enters the callee and returns normally.

- 빠트림없음 버그 21건 검출 (실제 보고는 6건)


`CALL_METHOD_DESCRIPTOR_*` misses for inherited built-in methods on subclass instances

interpreter-core performance type-feature

#155536 · marinelay opened 1w ago — 1

`TO_BOOL_INT` repeatedly misses for non-compact exact integers interpreter-core

performance type-bug

#155486 · marinelay opened 1w ago — 4

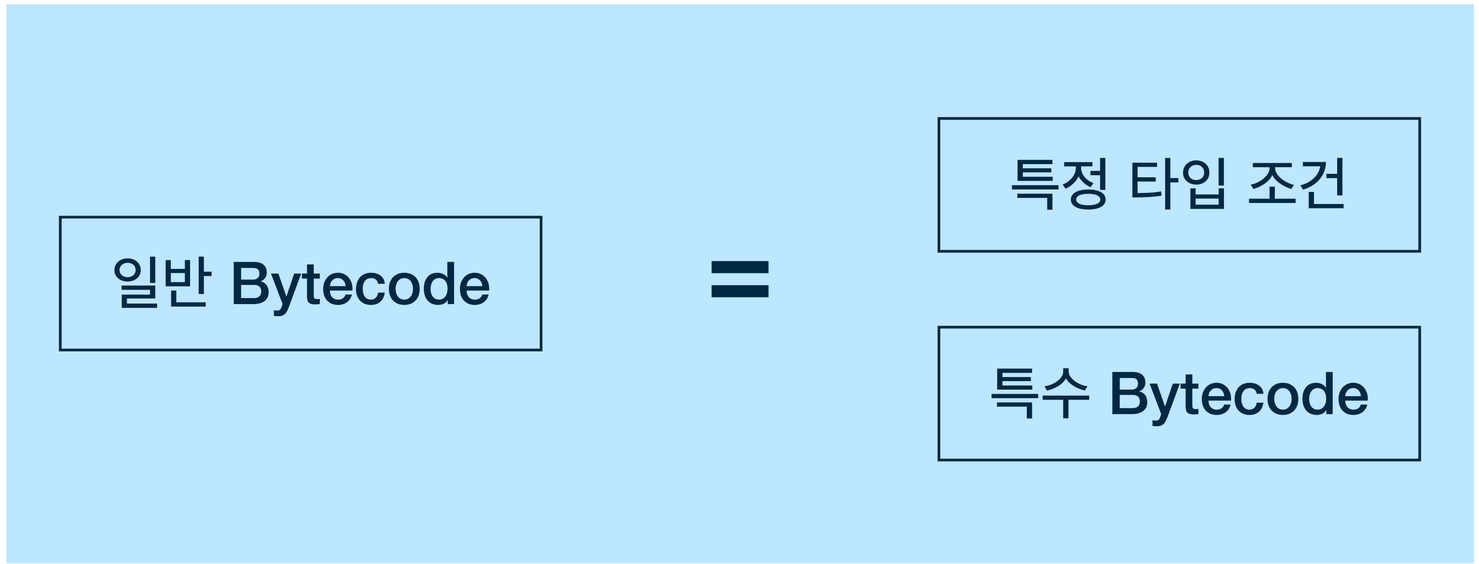
Specialize calls to inherited `list.append` on list subclasses interpreter-core pending

performance type-feature

#155456 · marinelay opened 1w ago — 4

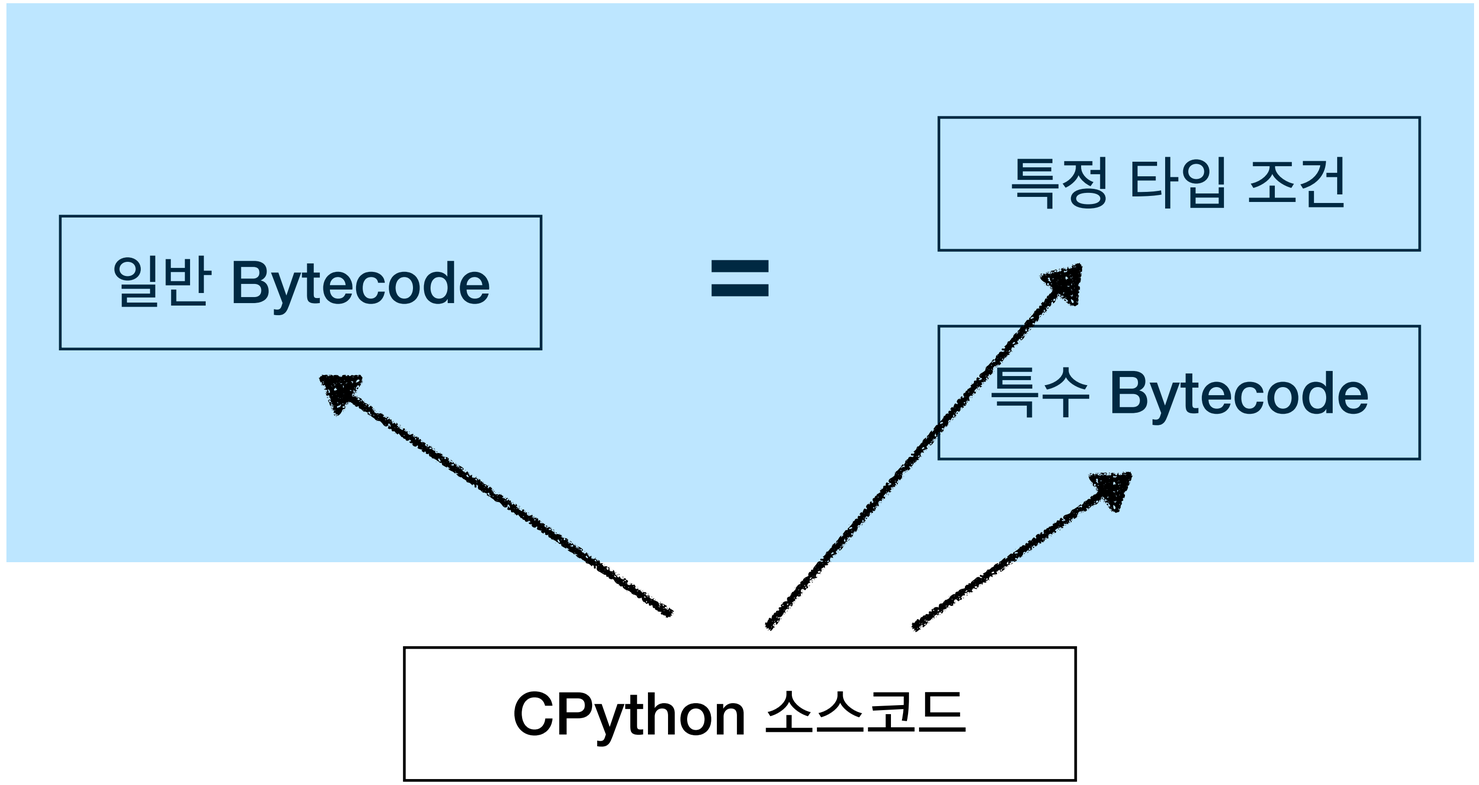
방법론

템플릿으로 변환



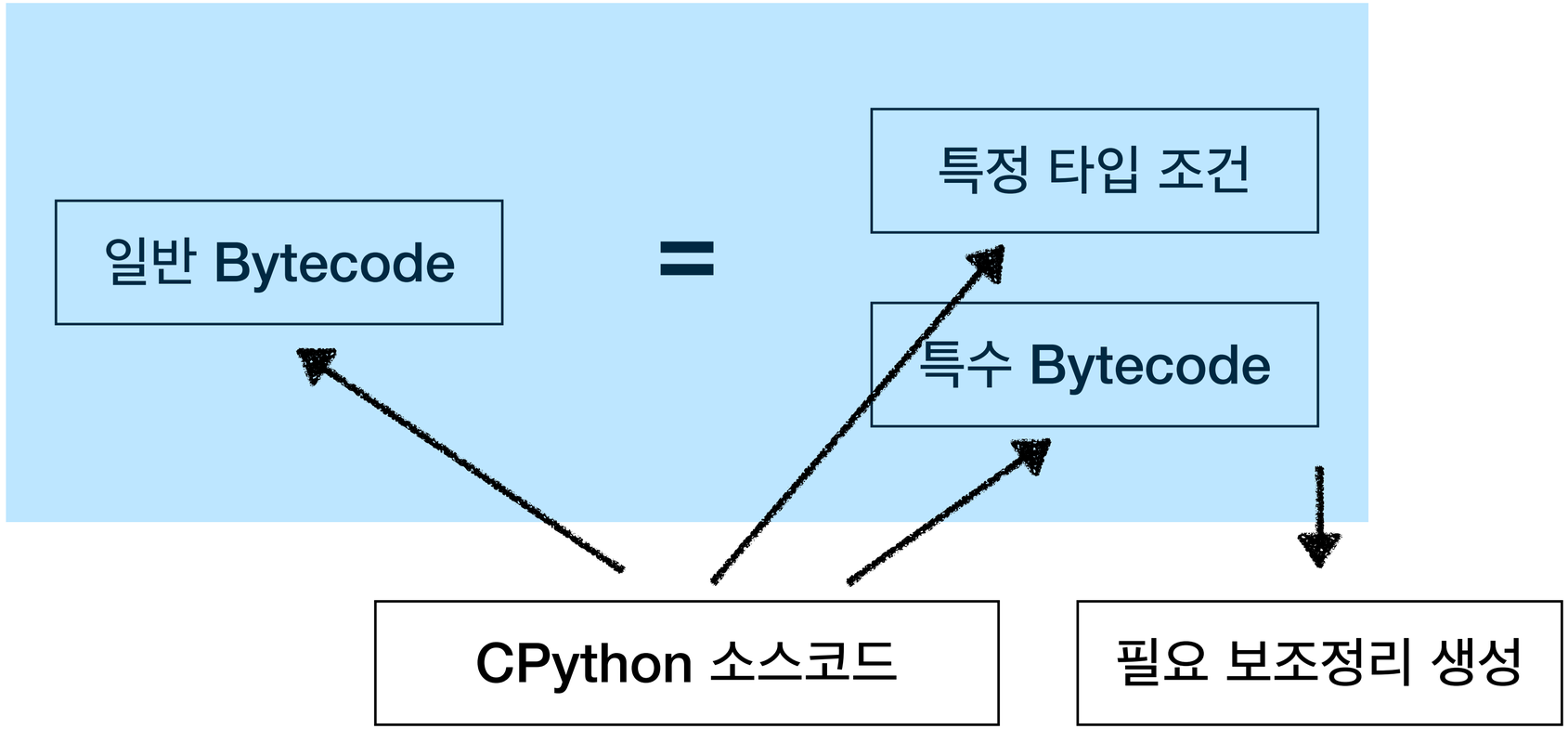
방법론

템플릿으로 변환



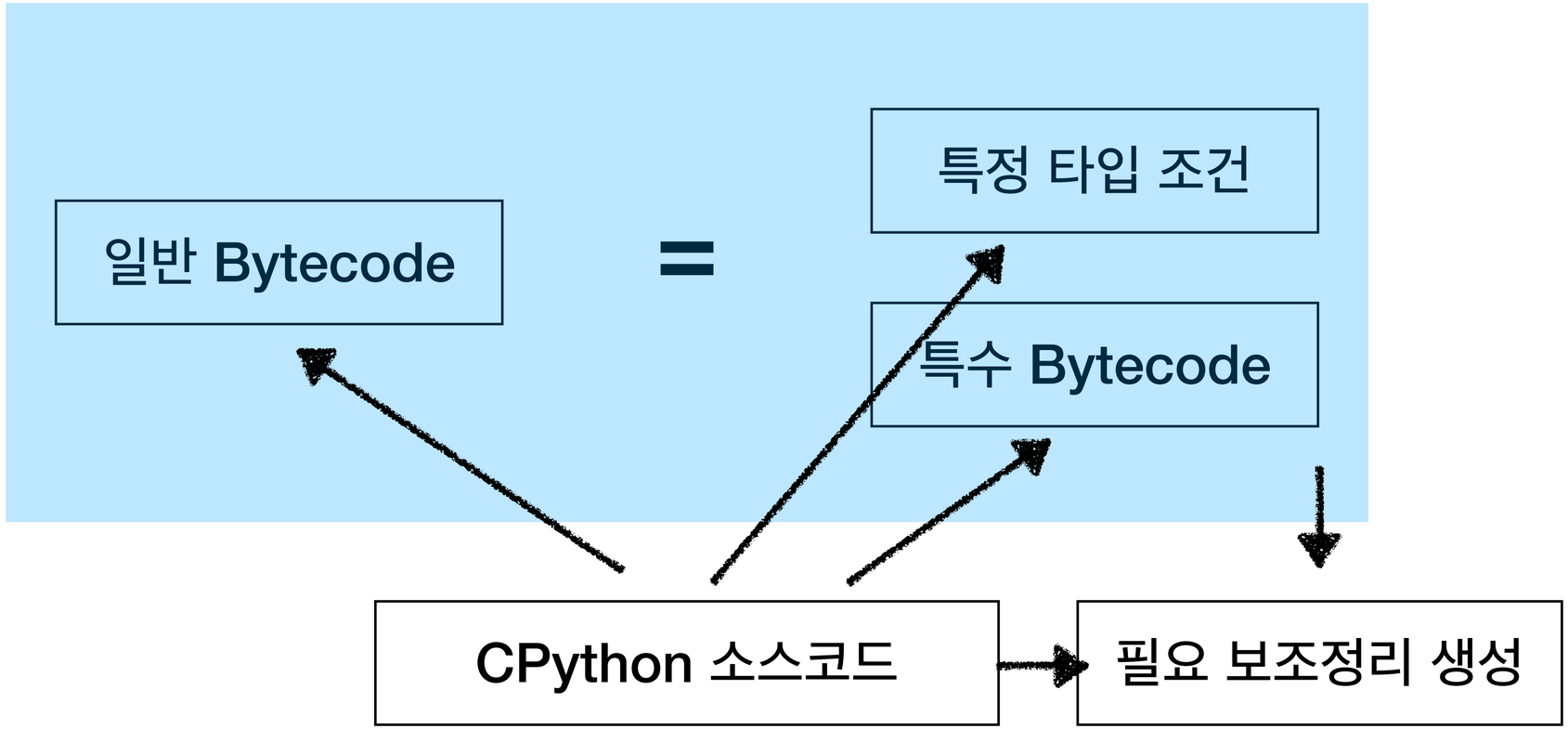
방법론

템플릿으로 변환



방법론

템플릿으로 변환



해야할 것

- 아직 통밥(Heuristic)으로 디자인된 부분이 많음
  - 이 부분을 최대한 없애야 함

- 목표) 사람이 작성해야할 부분이 최소화 된 증명 도구 만들기

감사합니다