Type Annotation

2021.01.06

이지헌

Dynamic language

- 파이썬은 동적 프로그래밍 언어
- 변수의 타입은 고정되어 있지 않음
- 타입을 명시적으로 표시할 필요가 없음

[Python]

```
a = 1
type(a)    # <type ‘int’>
a = "Hi~"
type(a)    # <type ‘str’>
```

[C, Java]

```
int a = 1;
a = "Hi~";    // compile error
```

Dynamic language

- 파이썬은 다른 언어에 비해 유연하다 !
- 하지만 만약 타입 변경 코드가 1,000 줄 이상이 된다면 ?

예상하지 못한 타입이 변수에 할당 또는 치명적인 버그로 이어짐

Type Annotation (타입 주석)

- PEP 484 (Type hint 지원)
- Python 3.5 버전에는 **함수의 인자와 반환값**에 대한 타입 힌트가 처음으로 도입

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

- 이후 3.6 버전에는 인자와 반환값 만이 아니라 **변수**에도 타입 힌트 표기 가능

```
def greeting(name: str) -> str:
    s: str = 'Hello ' + name
    return s
```

참고 : PEP 484 (Python.org)
https://www.python.org/dev/peps/pep-0484/

Type Annotation (타입 주석)

- 기존 Type hint 는 타입 표시 코드가

  IDE나 Linter에서 오류라고 판단

- 따라서 문제점을 보완한 Type Annotation을 도입하여

  IDE나 Linter가 해석할 수 있도록 해결 !

도입 기대효과

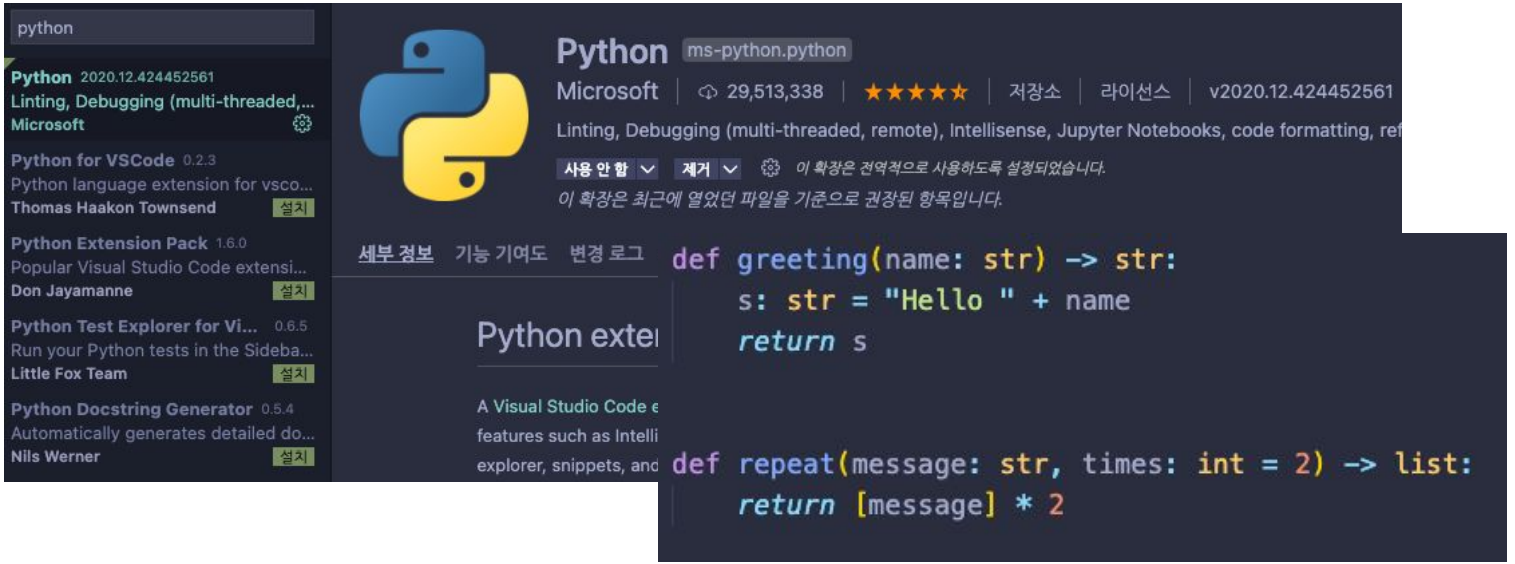
- 코드 내에 타입을 표시하려는 주석 불필요
- 다른 개발자가 코드를 읽기 수월해짐
- 예상하지 못한 타입을 방지하여 코드 품질을 향상

```
def repeat(message, times = 2):
    # type: (str, int) -> list
    return [message] * times
```

```
def repeat(message: str, times: int = 2) -> list:
    return [message] * times
```

타입 주석 활용 개발환경

- VSCode + Python (Extension)



참고 : PyCharm

타입 해석, 검사기 내장 및 편집 중 실시간 리포트 제공

변수 Type Annotation

- 변수 이름 뒤에 콜론 (:) 을 붙이고 타입을 명시

```
name: str = "Jiheon Lee"

age: int = 23

emails: list = ["test@abc.com", "test2@abc.com"]

subject: dict = {
    "frosh": "Python",
    "sophomore": "DataBase",
    "junior": "C"
}
```

함수 Type Annotation

- 함수 인자는 변수와 동일한 문법
- 반환값은 화살표 (->) 를 사용
- 콜론 (:) 은 뒤에만 한 칸, 화살표 (->) 는 앞뒤로 한 칸 띄어야 함

```
def add(a: int, b: int) -> str:
    result: int = a + b
    return str(result)
```

typing module

- 좀 더 복잡한 Type Annotation 을 추가해야 할 때는 typing module 을 사용

```
from typing import List, Set, Dict, Tuple

nums: List[int] = [1, 2, 3]

unique_nums: Set[int] = {5, 6}

vision: Dict[str, float] = {"left": 0.3, "right": 0.4}

jiheon: Tuple[int, str, List[float]] = (23, "Jiheon Lee", [0.3, 0.4])
```

Type Annotation 검사

- Type Annotaion 을 검사는 “__annotations__” 를 사용

```
# test.py
def add(a: int, b: int) -> str:
    result: int = a + b
    return str(result)

print(add.__annotations__)



$ python test.py
{‘a’: <class ‘int’>, ‘b’: <class ‘int’>, ‘return’: <class ‘str’>}
```

감사합니다