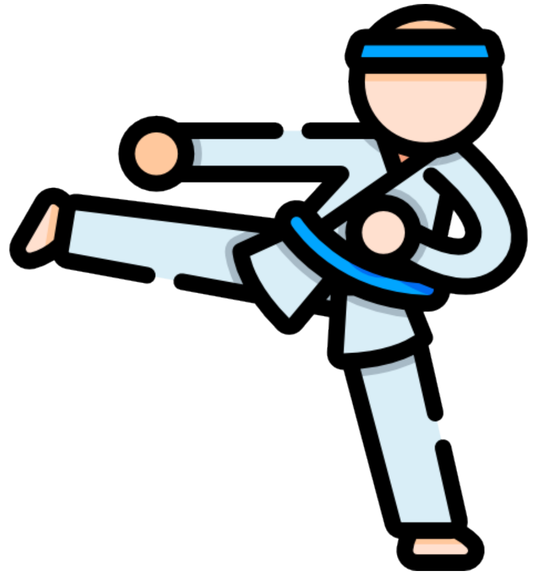
PyTER: Effective Program Repair for Python Type Errors

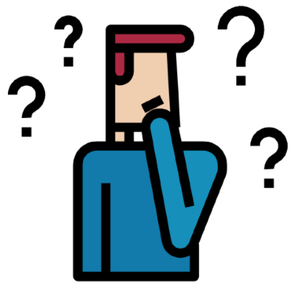
오원석, 오학주
고려대학교

09 Feb 2023
KCSE 23 @ 한화리조트, 평창



동기: 타입오류는 중요한 문제이다

1. 파이썬에서 가장 흔한 오류는 무엇일까?



파이썬 내장 오류 Top 5 조사 결과

	Type	Attribute	Value	Key	Import
StackOverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
GitHub	**29.2%**	19.4%	28.2%	12.9%	10.3%

동기: 타입오류는 중요한 문제이다

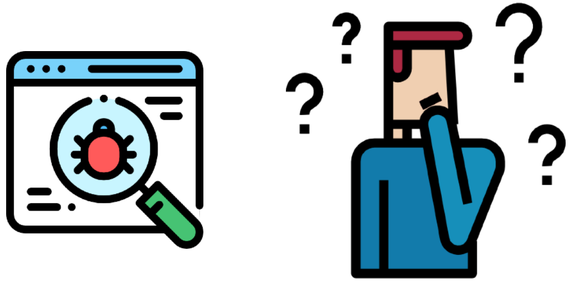
1. 파이썬에서 가장 흔한 오류는 무엇일까?

파이썬 내장 오류 Top 5 조사 결과

	Type	Attribute	Value	Key	Import
StackOverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
GitHub	**29.2%**	19.4%	28.2%	12.9%	10.3%

동기: 타입오류는 중요한 문제이다

1. 파이썬에서 가장 흔한 오류는 무엇일까?

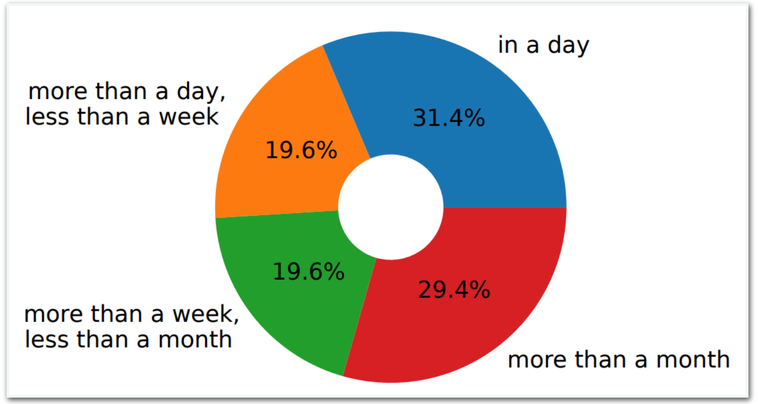


파이썬 내장 오류 Top 5 조사 결과

	Type	Attribute	Value	Key	Import
StackOverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
GitHub	**29.2%**	19.4%	28.2%	12.9%	10.3%

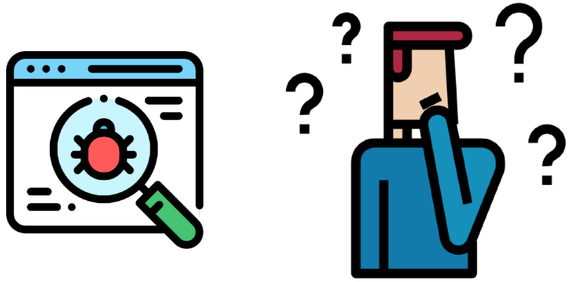
2. 개발자들이 타입오류를 고치는데 얼마나 시간이 소요될까?

이슈가 처음 올라오고 고쳐질 때까지의 시간 조사 결과



동기: 타입오류는 중요한 문제이다

1. 파이썬에서 가장 흔한 오류는 무엇일까?

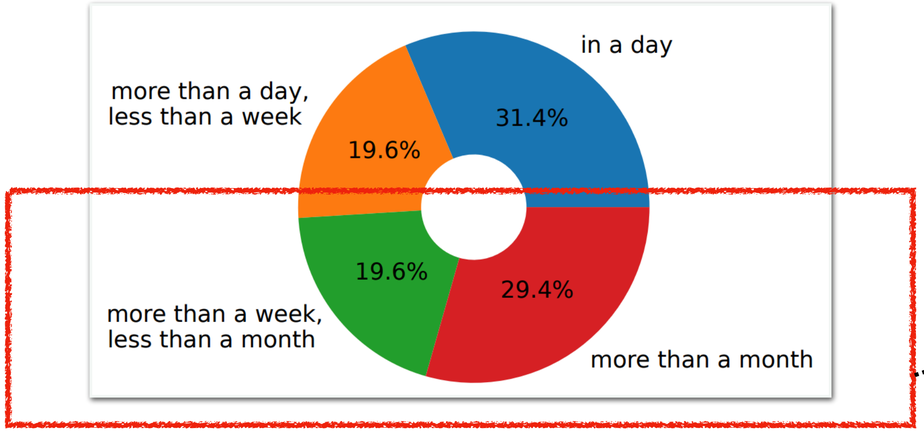


파이썬 내장 오류 Top 5 조사 결과

	Type	Attribute	Value	Key	Import
StackOverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
GitHub	**29.2%**	19.4%	28.2%	12.9%	10.3%

2. 개발자들이 타입오류를 고치는데 얼마나 시간이 소요될까?

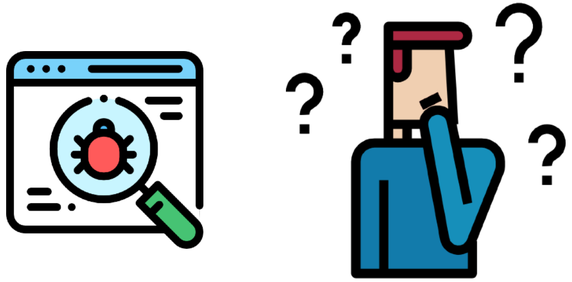
이슈가 처음 올라오고 고쳐질 때까지의 시간 조사 결과



타입오류 절반 이상이
일주일 이상 시간이 걸렸다

동기: 타입오류는 중요한 문제이다

1. 파이썬에서 가장 흔한 오류는 무엇일까?

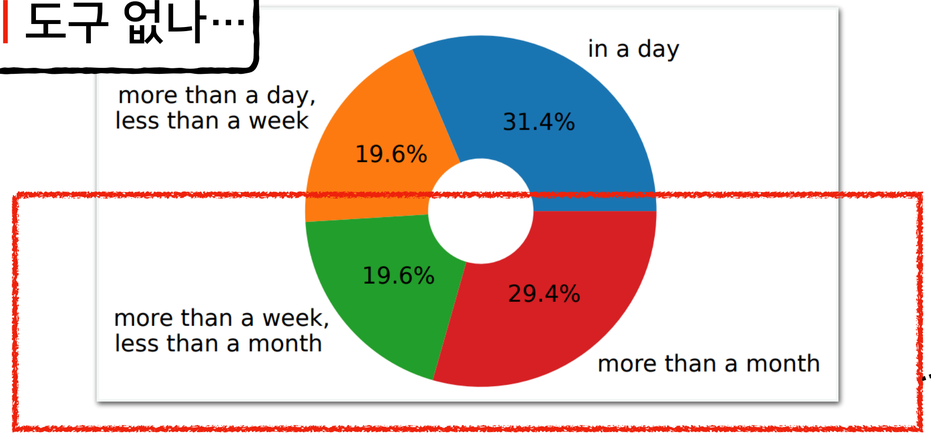


파이썬 내장 오류 Top 5 조사 결과

	Type	Attribute	Value	Key	Import
StackOverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
GitHub	**29.2%**	19.4%	28.2%	12.9%	10.3%

2. 개발자들이 타입오류를 고치는데 얼마나 시간이 소요될까?

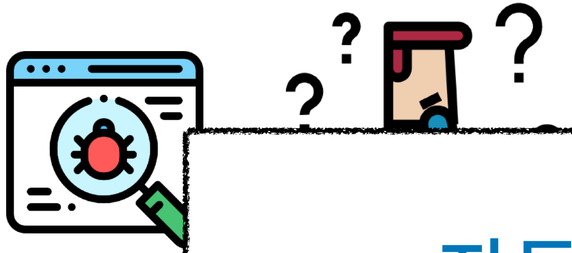
올라오고 고쳐질 때까지의 시간 조사 결과



타입오류 절반 이상이
일주일 이상 시간이 걸렸다

동기: 타입오류는 중요한 문제이다

1. 파이썬에서 가장 흔한 오류는 무엇일까?



파이썬 내장 오류 Top 5 조사 결과

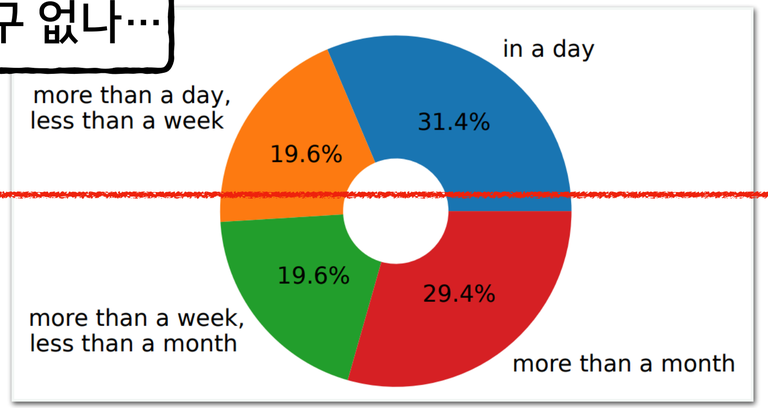
Type	Attribute	Value	Key	Import

자동으로 타입오류를 고쳐보자!

2. 개발자들

시간이 부족해…!
어디 자동패치 도구 없나…

라오고 고쳐질 때까지의 시간 조사 결과



타입오류 절반 이상이
일주일 이상 시간이 걸렸다

예제: 실제 타입오류 상황

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

https://github.com/saltstack/salt/pull/56381

예제: 실제 타입오류 상황

```
1  ret["comment"] = "  ".join(
2    [
3      "" if not ret["comment"] else ret[“comment"]
4    ]
5  )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

https://github.com/saltstack/salt/pull/56381

예제: 실제 타입오류 상황

```
1  ret["comment"] = "  ".join(
2    [
3      "" if not ret["comment"] else ret["comment"]
4    ]
5  )
```

False

List

Negative testcase

ret["comment"] = [1,2,3] (List of int)

not ret["comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함
=> Type Error

https://github.com/saltstack/salt/pull/56381

예제: 실제 타입오류 상황

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함
=> Type Error

Patch

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

예제: 실제 타입오류 상황

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Patch

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

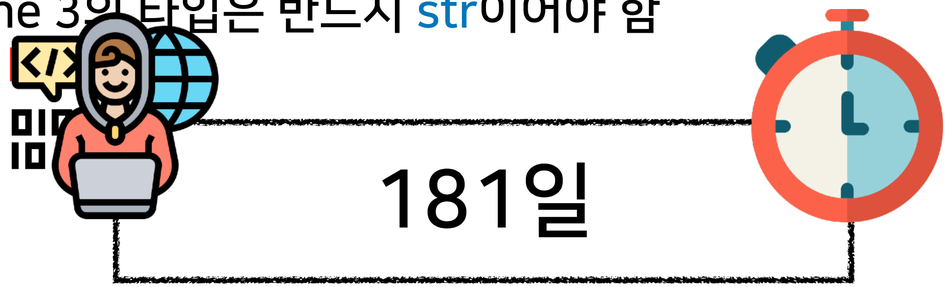
str

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함
=> Type E



이슈가 처음 발생하고 고쳐지기까지의 시간

https://github.com/saltstack/salt/pull/56381

예제: 실제 타입오류 상황

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Patch

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

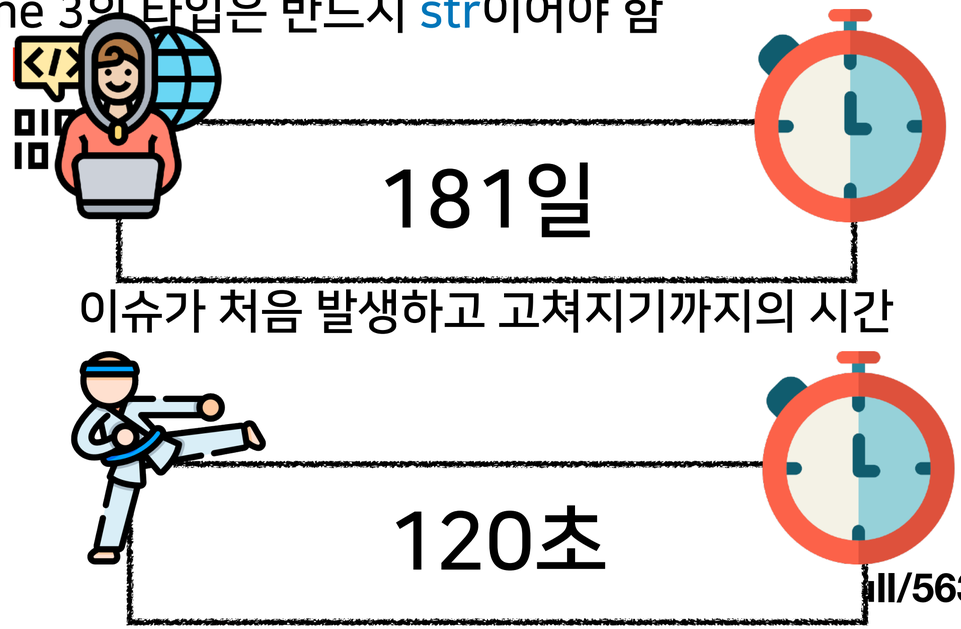
str

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

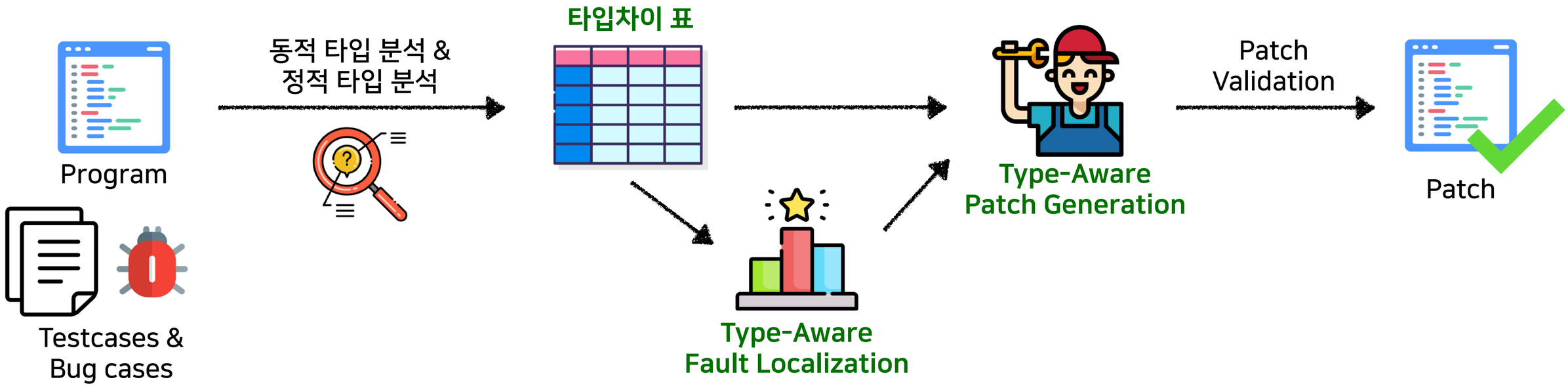
not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함
=> Type E



PyTER 개요

- 타입분석으로부터 타입차이 표 생성
- 타입차이 표를 이용하여 Fault Localization과 Patch Geneartion 진행



아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함

Patch

How?

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함

요약

변수이름	Negative type	Positive type
ret[“comment”]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

Patch

How?

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False — List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함

요약

변수이름	Negative type	Positive type
ret[“comment”]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

Patch

How?

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

ret[“comment”] 의 타입을
List 에서 str 으로 바꿔보자

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False — List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

not ret[“comment"] 표현식이 False이므로,
line 3의 전체 표현식의 타입은 List

그러나, *join* 함수의 파라미터 타입은…
List of str여야함. List of List면 안됨.
따라서, line 3의 타입은 반드시 str이어야 함

요약

변수이름	Negative type	Positive type
ret[“comment”]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

Patch

How?

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

ret[“comment”] 의 타입을
List 에서 str 으로 바꿔보자

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False　　List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

Patch

How?

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

타입차이 표

변수이름	Negative type	Positive type
ret[“comment”]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

ret[“comment”] 의 타입을
List 에서 str 으로 바꿔보자

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

타입차이 표

변수이름	Negative type	Positive type
ret[“comment”]		

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

False

List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

동적 타입 분석

타입차이 표

변수이름	Negative type	Positive type
ret[“comment”]	List	

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret["comment"]
4   ]
5 )
```

str　　False　　List

Negative testcase

ret["comment"] = [1,2,3] (List of int)

개발자는 타입을 유지하려고 한다

정적 타입 분석

타입차이 표

변수이름	Negative type	Positive type
ret["comment"]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret["comment"])
4   ]
5 )
```

str

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

str　　False　　List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

Patch

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str

타입차이 표

변수이름	Negative type	Positive type
ret[“comment”]	List	str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

ret[“comment”] 의 타입을
List 에서 str 으로 바꿔보자

https://github.com/saltstack/salt/pull/56381

아이디어 1: 타입차이 찾기

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else ret[“comment"]
4   ]
5 )
```

"" : str　　not ret["comment"] : False　　ret[“comment"] : List

Negative testcase

ret[“comment”] = [1,2,3] (List of int)

아이디어 1
적절한 패치를 만들기 위해
타입분석을 이용하여 타입차이 표를 만들고
타입차이를 찾아낸다

Positive type
str

Negative type : 타입오류 발생시의 타입
Positive type : 프로그램이 올바르게 동작할 때의 타입

```
1 ret["comment"] = "  ".join(
2   [
3     "" if not ret["comment"] else str(ret[“comment”])
4   ]
5 )
```

str(ret[“comment”]) : str

ret[“comment”] 의 타입을
List 에서 str 으로 바꿔보자

https://github.com/saltstack/salt/pull/56381

아이디어 2: Type-Aware Fault Localization

```
 1 def _serialize_value(self, value):
 2     if isinstance(value, BaseItem):
 3         ...
 4     if isinstance(value, dict):
 5         ...
 6
 7
 8     if self.binary:
 9         return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, str):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```


https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6
7
8      if self.binary:
9          return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, str):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
*value*를 return 해라

라인 11, 16, 17을 따라 타입오류 발생

https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6
7
8      if self.binary:
9          return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, (str, int)):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
*value*를 return 해라

라인 11, 16, 17을 따라 타입오류 발생

만약 *value*가 int이면, value를 return
올바른 패치일까?

https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6
7
8      if self.binary:
9          return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, (str, int)):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
*value*를 return 해라

라인 11, 16, 17을 따라 타입오류 발생

self.binary 가 True이면,
라인 9, 23, 24를 따라 타입오류 발생

만약 *value*가 int이면, value를 return
올바른 패치일까?

Overfitting Patch

https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6
7
8      if self.binary:
9          return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, str):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
*value*를 return 해라

라인 11, 16, 17을 따라 타입오류 발생

함수이름	변수이름	Negative type	Positive type
_serialize_value	value	int	BaseItem, dict, str
to_unicode	value	int	str

_serialize_value 함수에서 타입차이가 더 많이 관찰됨

더 의심스럽다!

https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6      if isinstance(value, int):
7          return value
8      if self.binary:
9          return to_bytes(value)
10     else:
11         return to_unicode(value)
12
13 def to_unicode(value):
14     if isinstance(value, str):
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ...
19
20 def to_bytes(value):
21     if isinstance(value, bytes):
22         return value
23     if not isinstance(value, str):
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
valu eturn 해라

라인 11, 16, 17을 따라 타입 생

함수이름	수이름	Negative type	Positive type
_serialize_value	value	int	BaseItem, dict, str
to_unicode	value	int	str

_serialize_value 함수에서 타입차이가 더 많이 관찰됨

더 의심스럽다!

https://github.com/scrapy/scrapy/pull/1737

아이디어 2: Type-Aware Fault Localization

```
1  def _serialize_value(self, value):
2      if isinstance(value, BaseItem):
3          ...
4      if isinstance(value, dict):
5          ...
6      if isinstance(value, int):
7          return value
8      if self.bi
9          return
10     else:
11         return
12
13 def to_unicode
14     if isinsta
15         return value
16     if not isinstance(value, bytes):
17         raise TypeError
18     ..
19
20 def to_
21     if
22
23     if n
24         raise TypeError
25     ...
```

Negative testcase

value = 1
self.binary = False

Assertion

*value*가 int이면,
valu eturn 해라

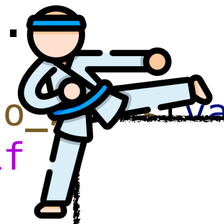
라인 11, 16, 17을 따라 타입 오류 발생

	Positive type
	BaseItem, dict, str
	str

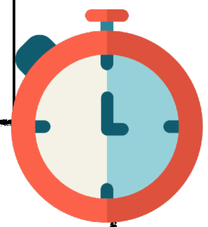
찰됨

아이디어 2
타입차이가 더 많이 관찰된 곳일수록,
더 의심스럽다!

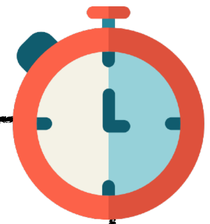
더 의심스럽다!



15초



81일



https://github.com/scrapy/scrapy/pull/1737

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5
6
7
8      j = self.index_col[i]
9      ...
```

Negative testcase	Assertion
self.index_col = None	타입오류 없음

https://github.com/pandas-dev/pandas/pull/38431

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5      if isinstance(self.index_col, None):
6          return self.index_col
7
8      j = self.index_col[i]
9      ...
```

Negative testcase	Assertion
self.index_col = None	타입오류 없음

https://github.com/pandas-dev/pandas/pull/38431

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5      if isinstance(self.index_col, None):
6          return self.index_col
7
8      j = self.index_col[i]
9      ...
```

Negative testcase	Assertion
self.index_col = None	타입오류 없음

Overfitting Patch

https://github.com/pandas-dev/pandas/pull/38431

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5      if isinstance(self.index_col, None):
6          return self.index_col
7
8      j = self.index_col[i]
9      ...
```

Negative testcase	Assertion
self.index_col = None	타입오류 없음

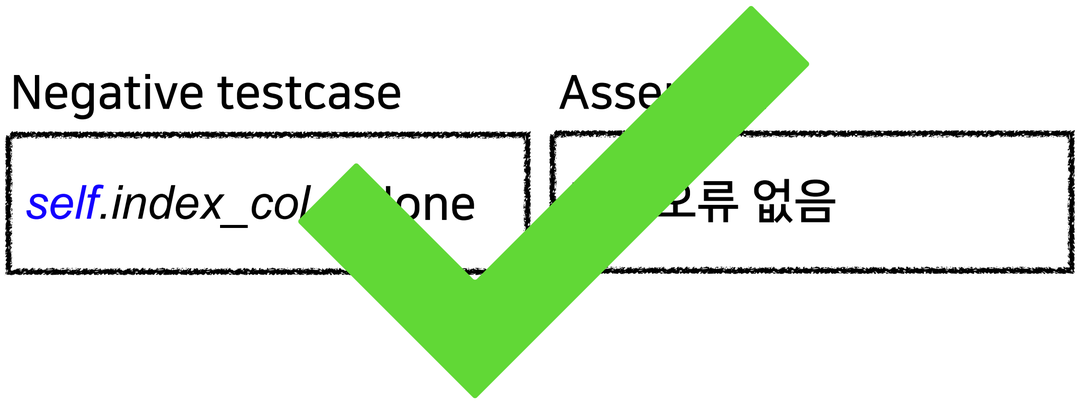
Overfitting Patch

이 함수의 return 타입은 아마 bool 타입일 것이다…
이것으로부터 이 패치의 순서를 뒤로 미룰 수 있음

https://github.com/pandas-dev/pandas/pull/38431

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5      if isinstance(self.index_col, None):
6          j = i
7      else:
8          j = self.index_col[i]
9      ...
```



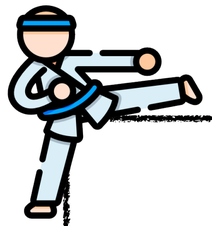
https://github.com/pandas-dev/pandas/pull/38431

아이디어 3: Type-Aware Patch Generation

```
1  def _should_parse_dates(self, i):
2      ...
3      if isinstance(self.parse_dates, bool):
4          return self.parse_dates
5      if isins
6          j =
7      else:
8          j =
9      ...
```

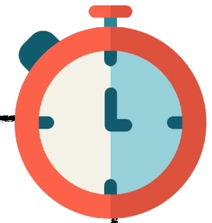
Negative testcase	Asse
self.index_col one	오류 없음

아이디어 3
타입 정보를 활용하여 패치 우선순위를 정할 수 있다



20 seconds

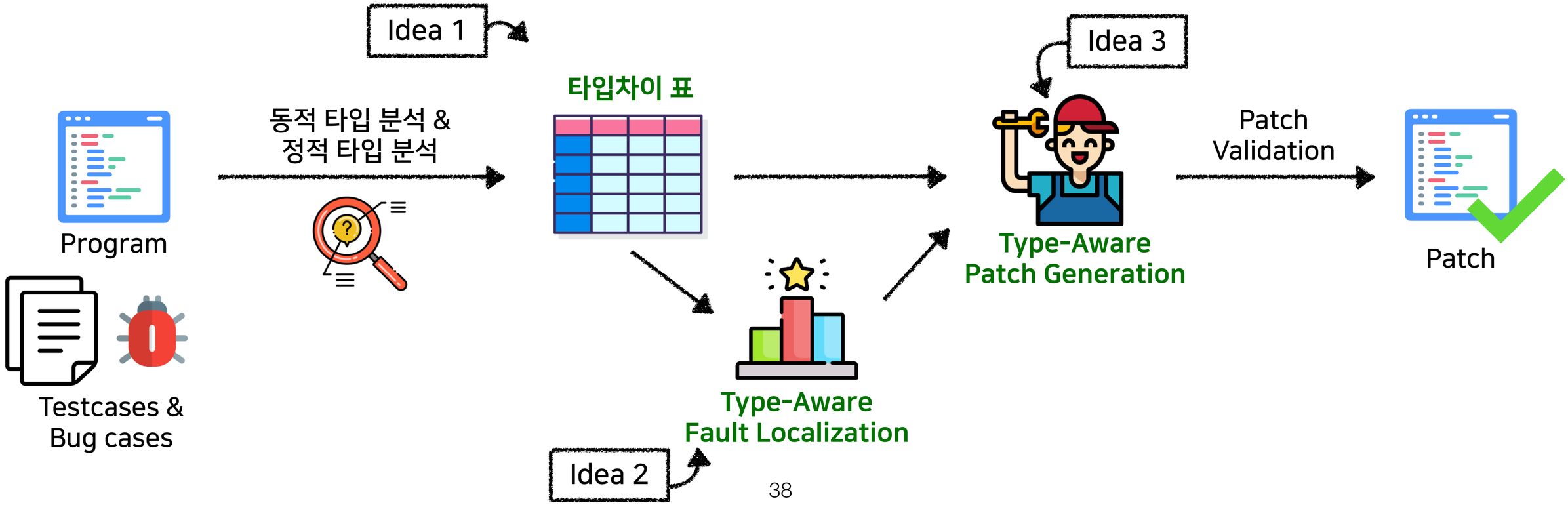
236 days



https://github.com/pandas-dev/pandas/pull/38431

PyTER 개요

- 타입분석으로부터 타입차이 표 생성
- 타입차이 표를 이용하여 Fault Localization과 Patch Geneartion 진행



성능 평가

- RQ1: PyTER가 얼마나 효율적으로 실제 타입오류를 고치는가?
- RQ2: PyTER에 사용된 핵심 기술이 얼마나 중요한가?
- **실험 Setting**
  - 벤치마크: TypeBugs (93), BugsInPy (57)
    - TypeBugs : Github 오픈소스 프로젝트를 대상으로 수집
    - BugsInPy (Ratnadira Widyasari et al.) : 타입오류 케이스만 이용
  - 베이스라인
    - Type-unaware 자동패치 기술 (= without PyTER 기술)
  - 시간 제한: 1h

성능 평가: RQ1

#Correct : 올바른 패치 수
#Plausible : 테스트 통과 패치 수
#Total : 총 버그 개수

	Program	#B	Avg. KLoC	NegTest Avg. Num	NegTest Avg. Time	PosTest Avg. Num	PosTest Avg. Time	Baseline #G	Baseline #C	Baseline Fix Rate	Baseline Prec	Baseline Avg. Time	PyTER #G	PyTER #C	PyTER Fix Rate	PyTER Prec	PyTER Avg. Time
TypeBugs	airflow	7	73.9	1.0	53.4	13.3	43.9	3	0	0.0%	0.0%	232.0	4	3	42.9%	75.0%	50.5
	beets	1	21.4	1.0	4.0	9.0	7.0	1	0	0.0%	0.0%	8.1	1	0	0.0%	0.0%	3.6
	core	9	230.5	1.4	11.6	23.7	14.7	6	4	44.4%	66.7%	303.7	6	5	55.6%	83.3%	88.7
	kivy	1	44.5	1.0	6.0	4.0	26.0	1	0	0.0%	0.0%	400.57	1	0	0.0%	0.0%	54.3
	luigi	1	13.2	4.0	4.0	1.0	2.0	1	0	0.0%	0.0%	20.6	1	0	0.0%	0.0%	1.5
	numpy	3	53.4	2.0	6.0	72.3	4.7	1	0	0.0%	0.0%	1356.8	2	0	0.0%	0.0%	225.5
	pandas	45	86.9	7.0	42.6	141.1	125.8	21	11	24.4%	52.4%	1478.2	24	19	42.2%	79.2%	1063.4
	rasa	1	45.8	1.0	9.0	7.0	6.0	1	1	100.0%	100.0%	7.3	1	1	100.0%	100.0%	7.3
	requests	4	9.4	1.5	8.0	184.8	33.3	3	1	25.0%	33.3%	942.4	4	4	100.0%	100.0%	167.7
	rich	1	16.1	1.0	9.0	7.0	6.0	0	0	0.0%	n/a	3600.0	0	0	0.0%	n/a	3600.0
	salt	9	364.8	1.0	31.0	12.7	165.6	3	2	22.2%	66.7%	1265.5	6	6	66.7%	100.0%	93.8
	sanic	3	5.6	3.3	5.7	51.0	9.3	1	1	33.3%	50.0%	1406.6	3	3	100.0%	100.0%	43.5
	scikit-learn	5	63.5	1.0	8.0	75.8	13.2	3	2	40.0%	66.7%	1561.6	3	2	40.0%	66.7%	408.7
	tornado	1	12.6	1.0	8.0	171.0	20.0	1	1	100.0%	100.0%	1307.9	1	1	100.0%	100.0%	199.2
	Zappa	2	4.1	1.0	7.0	46.5	748.5	1	0	0.0%	0.0%	1876.3	1	1	50.0%	100.0%	1663.9
	Total	93	112.7	4.1	30.7	91.9	101.2	47	23	24.7%	48.9%	1196.0	58	45	**48.4%**	**77.6%**	651.2
BugsInPy	ansible	1	126.5	1.0	21.0	30.0	13.0	0	0	0.0%	n/a	3600.0	0	0	0.0%	n/a	2549.7
	fastapi	2	7.4	1.0	5.5	7.0	3.0	0	0	0.0%	n/a	10.8	0	0	0.0%	n/a	1.5
	keras	5	26.5	1.0	23.8	14.8	50.4	1	0	0.0%	0.0%	2162.9	1	1	20.0%	100.0%	1769.8
	luigi	7	12.3	1.1	9.3	54.9	23.0	4	1	14.3%	25.0%	207.1	5	3	42.9%	60.0%	15.1
	matplotlib	1	94.9	1.0	8.0	214.0	41.0	0	0	0.0%	n/a	0.0	0	0	0.0%	n/a	0.0
	pandas	25	87.9	9.2	67.4	250.8	130.5	11	4	16.0%	36.4%	2040.3	13	7	28.0%	53.8%	1636.7
	scrapy	10	12.7	1.6	8.7	21.0	6.5	6	2	20.0%	33.3%	62.2	6	5	50.0%	83.3%	12.0
	spacy	1	78.6	1.0	7.0	6.0	7.0	1	0	0.0%	0.0%	9.7	1	0	0.0%	0.0%	79.8
	tornado	2	13.3	1.0	3.5	44.0	3.0	1	0	0.0%	0.0%	1.2	1	1	50.0%	100.0%	0.8
	tqdm	1	1.9	1.0	6.0	54.0	4.0	0	0	0.0%	n/a	0.0	0	0	0.0%	n/a	0.0
	youtube-dl	2	112.0	1.0	9.0	81.5	8.5	1	1	50.0%	100.0%	1815.2	1	1	50.0%	100.0%	1802.9
	Total	57	54.6	4.7	35.7	131.7	67.3	25	8	14.0%	32.0%	1248.4	28	18	**31.6%**	**64.3%**	968.5

패치율 #Correct / #Total
정확도 #Correct / #Plausible
평균 시간

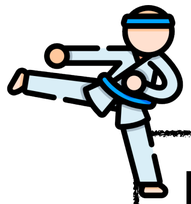
TypeBugs

PyTER — **48.4%** — **77.6%** — **651.2s**

Baseline — 24.7% — 48.9% — 1196.0s

x 93

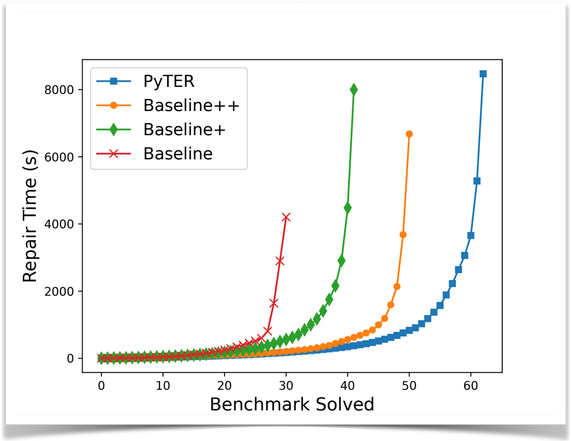
BugsInPy



PyTER — **31.6%** — **64.3%** — **968.5s**

Baseline — 14.0% — 32.0% — 1248.4s

x 57



성능 평가: RQ2

	Baseline	+ patch generation → Baseline+	+ fault localization → Baseline++	+ 정적 타입 분석 → PyTER
핵심 기술		type-aware patch generation	type-aware patch generation type-aware fault localization	type-aware patch generation type-aware fault localization 정적 타입 분석
#Correct	31	42	51	63

x 150

정리

- 실제 파이썬 프로그램의 타입오류를 고치는 첫번째 기술인 PyTER 제안

- 타입오류를 고치기 위해 타입차이를 활용하는 기술들을 제안

감사합니다!