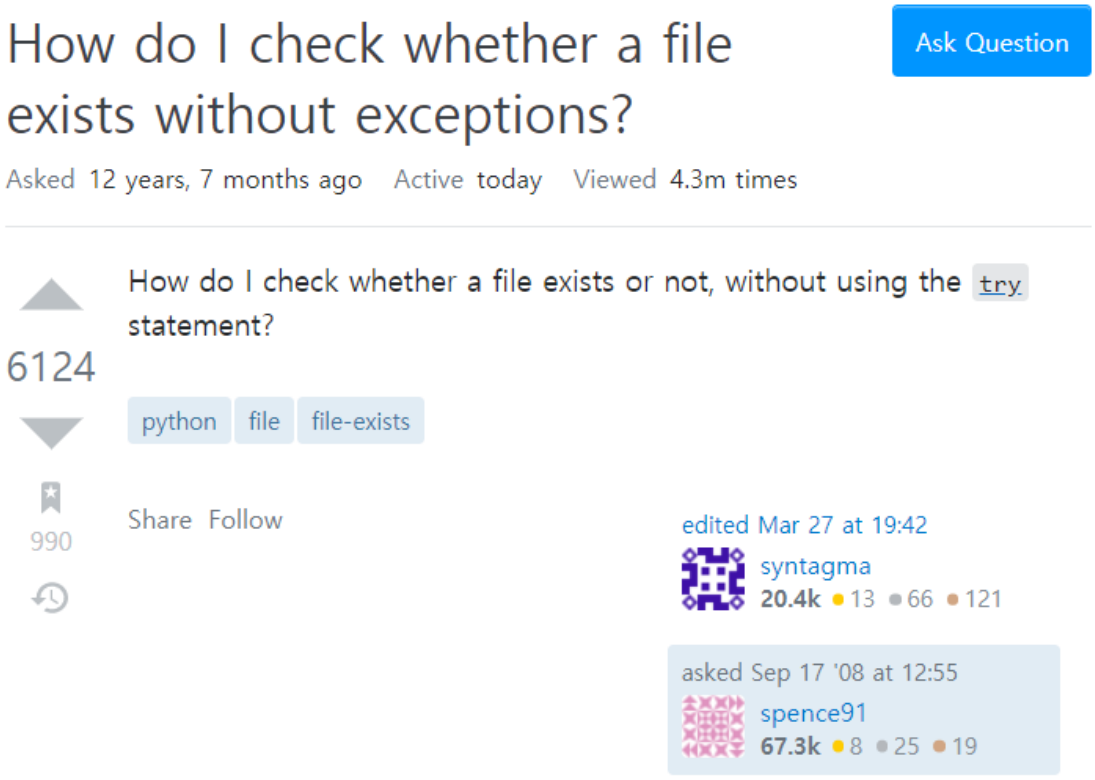
파트너

파(이썬)트(랜스래이트)너(를위한)

▼ Stackoverflow

▼ How do I check whether a file exists without exceptions?



Q. try 명령어 말고 어떤 특정 파일이 있는지 없는지 확인하려면 어떻게 해야하나요?

A. 뭐.. 그냥 확인만 해보려면 if file_exists: open_it()을 써서 열어보면 되긴하는데, 그런식으로 파일을 열면 파일이 의도치 않게 삭제되거나 강제로 이동 되어버릴 위험이 있으니 try 명령어를 쓰는 걸 권장함.

급하게 파일을 열거나 확인해야 하는 상황이 아니라면 os.path.isfile을 쓰는게 좋음.

경로(path)를 제대로 넣고 그곳에 파일이 잘 있다면 True를 리턴해줄거임. 이는 symbolic links에도 적용됨, islink()와 isfile()를 사용해 같은 경로를 넣어서 보면 True가 나올거임.(물론, symbolic links를 설정해주었다면..)

```
import os.path

path = './찾고자하는 파일명.확장자'
```

```
os.path.isfile(path)
    # exists
```

파일이 있다는 걸 더 확실하게 알고 싶다면 파이썬 3.4 버전에서 pathlib module을 써보셈.(파이썬 2.7에서는 pathlib2)

```
from pathlib import Path

my_file = Path("/파일경로")
if my_file.is_file():
    # file exists
    print('good')
else:
    # doesn't exists
    print('bad')
```

디렉터리를 확인해보려면 아래처럼 써보셈.

```
if my_file.is_dir():
    # directory exists
    print('good')
else:
    # doesn't exists
    print('bad')
```

지정한 Path에 해당하는 파일이나 디렉터리가 있는지 확인해보려면 아래처럼 써보셈.

```
if my_file.exists():
    # path exists
    print('good')
else:
    # doesn't exists
    print('bad')
```

try구문으로 쓰려면 아래처럼 쓰면됨.

```
try:
    my_abs_path = my_file.resolve(strict=True)
except FileNotFoundError:
    # doesn't exist
else:
    # exists
```

출처

How do I check whether a file exists without exceptions?

How do I check whether a file exists, using Python, without using a try statement?

https://stackoverflow.com/questions/82831/how-do-i-check-whether-a-file-exists-without-exceptions

▼ Finding the index of an item in a list

Q. Given a list ["foo", "bar", "baz"] and an item in the list "bar", how do I get its index (1) in Python?

Q. 파이썬에서 ["foo", "bar", "baz"] 리스트 안에 'bar' 항목의 인덱스 값을 가져오고 싶은데 어떻게 함?

A.

```
# 리스트.인덱스('항목') 이런식으로 쓰셈
["foo", "bar", "baz"].index("bar")
> 1
```

A. Caveats follow

A. 주의사항

Note that while this is perhaps the cleanest way to answer the question *as asked*, `index` is a rather weak component of the `list` API, and I can't remember the last time I used it in anger. It's been pointed out to me in the comments that because this answer is heavily referenced, it should be made more complete. Some caveats about `list.index` follow. It is probably worth initially taking a look at the documentation for it:

이렇게 하면 질문에 답변은 되지만, 인덱스는 리스트 API의 다소 취약한 구성요소임. 언제 빡쳤는지 기억도 안나네.. 암튼 이 답변이 많이 참조되어서 눈에 띄었고 뭔가 좀 더 보충해야할 필요성을 느낌. 리스트 인덱스에 대한 몇 가지 주의사항을 알려드림. 아마 이에 대한 문서를 한번 살펴보는 것이 중요함.

```
list.index(x[, start[, end]])
```

Return zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item.
The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the start argument.

> x값과 같은 값의 인덱스중 가장 첫번째 인덱스를 반환해줌. 해당 항목이 없으면 ValueError가 발생함. start와 end는 슬라이스 표기법에서 나오듯이 특정 하위 시퀀스를 찾아 그 범위로 제한하는데 사용됨. 반환된 인덱스는 시작 값이 아닌 전체 시퀀스의 시작이 기준임.

Linear time-complexity in list length

리스트 길이의 선형 시간-복잡성

An index call checks every element of the list in order, until it finds a match. If your list is long, and you don't know roughly where in the list it occurs, this search could become a bottleneck. In that case, you should consider a different data structure. Note that if you know roughly where to find the match, you can give index a hint. For instance, in this snippet, l.index(999_999, 999_990, 1_000_000) is roughly five orders of magnitude faster than straight l.index(999_999), because the former only has to search 10 entries, while the latter searches a million:

인덱스 콜을 하면 일치하는 항목을 찾을때까지 리스트의 모든 요소를 순서대로 확인합니다. 리스트가 겁나 길고 그 위치를 대략적으로도 모른다면 병목 현상이 일어날 수 있는데, 그럴 경우 다른 데이터 구조를 고려해야함. 해당 위치를 대략적으로 알고 있다면 인덱스에 힌트를 줄 수 있음. 예를 들어 l.index(999_999, 999_990, 1_000_000)는 l.index(999_999)로 하는 것 보다 무려 5배나 더 빠름. 앞에 거는 10개 항목만 확인하면 되지만 뒤에 거는 100만개나 검색해야 하기 때문이쥐~

l.index(찾고 싶은 수, 예상 범위 시작, 예상 범위 종료)

```
import timeit
timeit.timeit('l.index(999_999)', setup='l = list(range(0, 1_000_000))', number=1000)
> 9.356267921015387

# 내가 직접 돌려봤는데 무려
# 15.4942919000000007초나 걸렸다.
```

```
timeit.timeit('l.index(999_999, 999_990, 1_000_000)', setup='l = list(range(0, 1_000_000))', number=1000)
> 0.0004404920036904514

# 체감상 훨씬 빠르다
# 0.0003159000025971676초 나옴
```

Only returns the index of the first match to its argument

찾는 수와 비교해서 처음 일치하는 인덱스만 반환해줌

A call to index searches through the list in order until it finds a match, and stops there. If you expect to need indices of more matches, you should use a list comprehension, or generator expression.

인덱스 콜을 하면 일치하는 항목을 찾을때까지 순서대로 검색하고 찾으면 거기서 멈춤. 일치하는 인덱스를 여러개 찾고 싶다면 list comprehension이나 generator expression을 사용해야함.

```
[1, 1].index(1)
> 0
# 1이 2개 있어도 처음 찾은 1만 검색됨

[i for i, e in enumerate([1, 2, 1]) if e == 1]
> [0, 2]
# for문을 써서 1을 모두 찾아보자

g = (i for i, e in enumerate([1, 2, 1]) if e == 1)
next(g)
> 0
# g 한번 돌리고

next(g)
> 2
# 또 한번 돌려서 다음 수 찾음
```

Most places where I once would have used index, I now use a list comprehension or generator expression because they're more generalizable. So if you're considering reaching for index, take a look at these excellent Python features.

인덱스 많이 써봤는데 이제는 일반화를 위해서 list comprehension나 generator expression을 사용함. 인덱스를 제대로 쓰려면 우수한 파이썬 특성들을 찾아보렴.

Throws if element not present in list

검색한 항목이 리스트에 없는 경우에 발생함

A call to index results in a ValueError if the item's not present.

해당 항목이 없다면 ValueError가 나오겠지.

```
[1, 1].index(2)
> Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  ValueError: 2 is not in list
# 애초에 리스트 안에 2가 없는데 2를 찾아달라고 하니 ValueError가 나올 수 밖에..
```

If the item might not be present in the list, you should either

만약 해당 항목이 리스트에 없다면 이렇게 해봐

1. Check for it first with `item in my_list` (clean, readable approach), or

   먼저 `item in my_list` 써서 깔끔하게 확인

2. Wrap the `index` call in a `try/except` block which catches `ValueError` (probably faster, at least when the list to search is long, and the item is usually present.)

   `try/except` 구문을 사용해서 인덱스 호출을 해보자. (검색할 리스트의 길이가 길고, 검색 항목이 존재하는 경우에)

   출처:

   Finding the index of an item in a list

   For a list ["foo", "bar", "baz"] and an item in the list "bar", what's the cleanest way to get its index (1) in Python?

   https://stackoverflow.com/questions/176918/finding-the-index-of-an-item-in-a-list