Is it a good practice to use try-except-else in python?

From time to time on python, I see the block

```
try:
   try_this(whatever)
except SomeException, exception:
   #Handle exception
else:
   return something
```

What is the reason for the try-except-else to exist?

I do not know if it is out of ignorance, but i do not like that kind of programming, as it is using exceptions to perform flow control.

However, if it is included in the language itself, there must be a good reason for it, isn't it?

It my understanding that exceptions are not errors, they should only be used for exceptional conditions (ie: I try to write a file into disk and there is no more space, or maybe I do not have permission), not to perform flow control.

Normally I handle exceptions as

```
something = some_default_value
try:
    something = try_this(whatever)
except SomeException, exception:
    #Handle exception
finally:
    return something
```

of, if I really do not want to return anything if an exception happens, then

```
try:
    something = try_this(whatever)
    return something
except SomeException, exception:
    #Handle exception
```

Am i missing something?

python exception exception-handling try-catch

edited Apr 22 '13 at 7:39

asked Apr 22 '13 at 1:44
Juan Antonio Gomez Moı
1,177 ●11 ●28

2 It's better to ask forgiveness than permission – jamylak Apr 22 '13 at 1:46

2 Just to be clear, this question revolves around the usefulness of the `else` clause in exception handling, not the usefulness of exception handling itself. Is that correct? – mgilson Apr 22 '13 at 1:48

Yes. Otherwise you may catch something that you shouldn't – JBernardo Apr 22 '13 at 1:48

7 Note that `except SomeException, exception:` is better written as `except SomeException as exception:`. This is much more clear IMHO – mgilson Apr 22 '13 at 1:52

1 Never **ever** use `try: ... except: ...`! **Always** specify which kind of exception you want to catch! – Bakuriu Apr 22 '13 at 7:35

show **9** more comments

4 Answers

> "I do not know if it is out of ignorance, but i do not like that kind of programming, as it is using exceptions to perform flow control."

In the Python world, using exceptions for flow control is common and normal.

Even the Python core developers use exceptions for flow-control and that style is heavily baked into the language (i.e. the iterator protocol uses *StopIteration* to signal loop termination).

In addition, the try-except-style is used to prevent the race-conditions inherent in some of the "look-before-you-leap" constructs. For example, testing *os.path.exists* results in information that may be out-of-date by the time you use it. Likewise, *Queue.full* returns information that may be stale. The try-except-else style will produce more reliable code in these cases.

> "It my understanding that exceptions are not errors, they should only be used for exceptional conditions"

In some other languages, that rule reflects their cultural norms as reflected in their libraries. The "rule" is also based in-part on performance considerations for those languages.

The Python cultural norm is somewhat different. In many cases, you *must* use exceptions for control-flow. Also, the use of exceptions in Python does not slow the surrounding code and calling code as it does in some compiled languages (i.e. CPython already implements code for exception checking at every step, regardless of whether you actually use exceptions or not).

In other words, your understanding that "exceptions are for the exceptional" is a rule that makes sense in some other languages, but not for Python.

> "However, if it is included in the language itself, there must be a good reason for it, isn't it?"

Besides helping to avoid race-conditions, exceptions are also very useful for pulling error-handling outside loops. This is a necessary optimization in interpreted languages which do not tend to have automatic loop invariant code motion.

Also, exceptions can simplify code quite a bit in common situations where the ability to handle an issue is far removed from where the issue arose. For example, it is common to have top level user-interface code calling code for business logic which in turn calls low-level routines. Situations arising in the low-level routines (such as duplicate records for unique keys in database accesses) can only be handled in top-level code (such as asking the user for a new key that doesn't conflict with existing keys). The use of exceptions for this kind of control-flow allows the mid-level routines to completely ignore the issue and be nicely decoupled from that aspect of flow-control.

There is a nice blog post on the indispensibility of exceptions here.

Also, see this StackOverFlow answer: Are exceptions really for exceptional errors?

> "What is the reason for the try-except-else to exist?"

The else-clause itself is interesting. It runs when there is no exception but before the finally-clause. That is its primary purpose.

Without the else-clause, the only option to run additional code before finalization would be the clumsy practice of adding the code to the try-clause. That is clumsy because it risks raising exceptions in code that wasn't intended to be protected by the try-block.

The use-case of running additional unprotected code prior to finalization doesn't arise very often. So, don't expect to see many examples in published code. It is somewhat rare.

Another use-case for the else-clause it to perform actions that must occur when no exception occurs and that do not occur when exceptions are handled. For example:

```
    recip = float('Inf')
    try:
        recip = 1 / f(x)
    except ZeroDivisionError:
        logging.info('Infinite result')
    else:
        logging.info('Finite result')
```

Lastly, the most common use of an else-clause in a try-block is for a bit of beautification (aligning the exceptional outcomes and non-exceptional outcomes at the same level of indentation). This use is always optional and isn't strictly necessary.

edited Apr 22 '13 at 5:59

community wiki
11 revs
Raymond Hettinger

2 The very last example clarifies everythin, although you can achieve the same results by using try-except-finally, I would need to add an extra variable to control whether or not the exception occurred or not, while using try-except-else makes it more readable. +1 – Juan Antonio Gomez Moriano Apr 23 '13 at 4:53

If using exceptions is a good habit, it shouldn't have been discouraged by the `try-except-else-finally` being so awkward to use, given the Python indentation rules. – Elazar Jun 26 '13 at 14:15

add a comment

Python doesn't subscribe to the idea that exceptions should only be used for exceptional cases, in fact the idiom is 'ask for forgiveness, not permission'. This means that using exceptions as a routine part of your flow control is perfectly acceptable, and in fact, encouraged.

This is generally a good thing, as working this way helps avoid some issues (as an obvious example, race conditions are often avoided), and it tends to make code a little more readable.

Imagine you have a situation where you take some user input which needs to be processed, but have a default which is already processed. The `try: ... except: ... else: ...` structure makes for very readable code:

```
try:
   raw_value = int(input())
except ValueError:
   value = some_processed_value
else: # no error occured
   value = process_value(raw_value)
```

Compare to how it might work in other languages:

```
raw_value = input()
if valid_number(raw_value):
    value = process_value(int(raw_value))
else:
    value = some_processed_value
```

Note the advantages. There is no need to check the value is valid and parse it separately, they are done once. The code also follows a more logical progression, the main code path is first, followed by 'if it doesn't work, do this'.

The example is naturally a little contrived, but it shows there are cases for this structure.

edited Apr 22 '13 at 1:59
jamylak
38.8k ●7 ●76 ●124

answered Apr 22 '13 at 1:47
Lattyware
36.7k ●3 ●50 ●89

just note that *'ask for forgiveness, not permission'.* is actually not part of the zen – jamylak Apr 22 '13 at 1:53

@jamylak -- True, but it is in the documentation – mgilson Apr 22 '13 at 1:55

@jamylak True, it's essentially a natural expansion of *Errors should never pass silently.* Either way, I've changed the link to mgilson's suggestion, which is more apt. – Lattyware Apr 22 '13 at 1:55

Can't you just achieve the same result using try-except-finally? – Juan Antonio Gomez Moriano Apr 22 '13 at 2:04

3 @JuanAntonioGomezMoriano the `finally` clause is executed regardless of whether an error was raised or

3 @JuanAntonioGomezMoriano the `finally` clause is executed regardless of whether an error was raised or not. `else` only runs if no error occurs. Similar to a `for` loop `else` clause which runs only if no `break` s occured, that might be worth mentioning in this answer since having `else` in all of these constructs makes the language more complete – jamylak Apr 22 '13 at 2:14

show **5** more comments

You should be careful about using the finally block, as it is not the same thing as using an else block in the try, except. The finally block will be run regardless of the outcome of the try except.

```
In [10]: dict_ = {"a": 1}

In [11]: try:
   ....:     dict_["b"]
   ....: except KeyError:
   ....:     pass
   ....: finally:
   ....:     print "something"
   ....:
something
```

As everyone has noted using the else block causes your code to be more readable, and only runs when an exception is not thrown

```
In [14]: try:
             dict_["b"]
         except KeyError:
             pass
         else:
             print "something"
   ....:
```

edited Apr 24 '13 at 0:53
dcrosta
11.4k ●2 ●28 ●48

answered Apr 22 '13 at 2:31
Greg
1,440 ●5 ●11

I know that finally is always executed, and that is why it can be used to our advantage by always setting a default value, so in case of exception it is returned, if we do not want to return such value in case of exception, it is enough to remove the final block. Btw, using pass on an exception catch is something I would never ever do :) – Juan Antonio Gomez Moriano Apr 22 '13 at 4:55

@Juan Antonio Gomez Moriano , my coding block is for example purposes only. I probably would never use

pass either – Greg Apr 22 '13 at 6:13

add a comment

Do a search for "duck typing". It is considered a very pythonic way of programming even though it is not the norm in other languages. Duck typing can result in more readable and more reliable code than checking the type explicitly when used properly.

One caveat is that it is not consered good form to except any error. It is highly recomended to except explicit errors such as:

```
except TypeError:
```

or

```
except (TypeError, IndexError):
```

answered Apr 22 '13 at 1:49
dansalmo
3,365 ●3 ●12 ●22

What does duck typing have to do with the question? – Michael0x2a Apr 22 '13 at 2:47

The question was broad enough to be addressed in part by duck typing. – dansalmo Apr 22 '13 at 3:47

add a comment