Python_3_error_handling_cheat_sheet

April 2, 2017

Cheat sheets assume you know what you are doing and only need a quick reference. If you don't understand something, read a tutorial instead.

1 Python 3 error handling sheet cheat - april 2017

1.1 Références

- Download this cheat sheet on http://encyclopython.com/pages/error-handling-cheat-sheet.html
- Python official doc tutorial on exceptions: https://docs.python.org/3/tutorial/errors.html
- Python official doc exception reference: https://docs.python.org/3/library/exceptions.html

1.2 Handling

1.2.1 Typical case

```
In [36]: try:
             1 / 0  # Operation that can fail
         except ZeroDivisionError: # React specifically to this error
             print('Something failed')

         # DO NOT use a "catch-all" like:

         try:
             1 / 0
         except Exception:
             print('Something failed')

         # or:

         try:
             1 / 0
         except BaseException:
             print('Something failed')

         # or:
```

```
        try:
            1 / 0
        except:
            print('Something failed')

        # Unless you really, REALLY know what you are doing. Those will make debugging much har
```

```
Something failed
Something failed
Something failed
Something failed
```

1.2.2 More complex logic

```
In [37]: import random

        try:
            # This code can fail in many ways
            res = str(1 / random.choice([1, "1", 0]))
            print(float(random.choice([res, 'a'])))

        except ZeroDivisionError as e:  # You can capture the exception in a variable
            print('Something failed:', e)

        # You can use several "except" blocks
        except (TypeError, ValueError):  # You can capture several exceptions at once
            print('Something else failed')

        else:  # You can execute code if there is NO error
            print("It's all good")

        finally:  # This will be executed in all cases, error or not
            print("Good bye")
```

```
Something failed: division by zero
Good bye
```

1.3 Using exceptions yourself

```
In [137]: raise ValueError('You can use any standard exception in your code')
```

```
        Traceback (most recent call last):


          File "<ipython-input-137-07c6eae0bc46>", line 1, in <module>
        raise ValueError('You can use any standard exception in your code')
```

```
ValueError: You can use any standard exception in your code
```

1.3.1 Use

- **ValueError** when there is not obvious way of dealing with a value. E.G: trying to convert a sound into a color.
- **TypeError** when the value is of the wrong type. E.G: trying to get a color from a sound.
- **RuntimeError** when the configuration prevent your from going any further. E.G: an key resource is missing.
- **NotImplementedError** when part of the code is missing, either because you haven't coded it, can't code it or require the user to code it. E.G: a parent class delegate a method implementation to its children.
- **TimeoutError** when an operation is taking too much time. E.G: the serveur is not responding.
- **LookupError** when a search failed. E.G: trying to get a car from a wrong brand.

1.3.2 You can and should create your own exceptions

In [39]:
```
import datetime

# It's a good idea to have a hierarchy in your exceptions
class YourBaseError(Exception):
    pass

# This can be dealt with "except" on YourPreciseError or YourBaseError
class YourPreciseError(YourBaseError):
    pass

# This can be dealt with "except" on YourCustomValueError, ValueError or YourBaseError
class YourCustomValueError(ValueError, YourBaseError):
    pass

# Exceptions are regular classes, you can make them do anything
class ExceptionWithParamsError(YourBaseError):
    def __init__(self, msg):
        msg = "{}: {}".format(datetime.datetime.now(), msg)
        super().__init__(msg)

# You can then raise any of those 4 exceptions
raise ExceptionWithParamsError('Custom error !')
```

```
Traceback (most recent call last):
```

```
  File "<ipython-input-39-d9dd630225c7>", line 22, in <module>
raise ExceptionWithParamsError('Custom error !')


ExceptionWithParamsError: 2017-03-31 14:20:58.607556: Custom error !
```

1.4 Intercepting exceptions then letting it crash

1.4.1 Reraise the original exception

```
In [4]: try:
            1 / 0
        except ZeroDivisionError:
            print('Something failed')
            raise  # An empty "raise" will raise the original ZeroDivisionError
```

```
Something failed
```

```
Traceback (most recent call last):


  File "<ipython-input-4-0da52efb6151>", line 2, in <module>
1 / 0


ZeroDivisionError: division by zero
```

1.4.2 Raise a new exception but keep the original context

```
In [171]: class CustomException(Exception):
              pass

          try:
              1 / 0
          except ZeroDivisionError as e:
              # This will preserve the original stack trace.
              raise CustomException("This crashed because of this and that") from e
```

```
Traceback (most recent call last):
```

```
  File "<ipython-input-171-7332353a141f>", line 8, in <module>
    raise CustomException("This crashed because of this and that") from e


CustomException: This crashed because of this and that
```

** In a shell, the above exception does not reflect what will happen in a real program. This is what "raise from" would give you if used in a module: **

1.4.3 Catch all unhandled exceptions right before the program crashes

```
In [40]: import sys
         import logging

         import tempfile

         # You can use the standard Python logger to log exceptions.
         # This one is a dummy one that only prints on the console.
         logging.basicConfig()
         log = logging.getLogger()

         # Avoid erasing the previous exception handler.
         # You should ALWAYS do this.
         previous_hook = sys.excepthook

         def on_crash(type, value, tb):

             # Here you can do something right before the program crash.
             # You could even log the stack trace in a file or send an email.
             log.critical(
                  "Arrrg",
                  exc_info=(type, value, tb)
             )

             if previous_hook is sys.__excepthook__:
                 # Here you can do something if the previous handler
                 # is the standard Python one.
                 print('... yes, but not a standard arg !')

             # You could reuse the previous handler if you wish:
             # previous_hook(type, value, tb)  # The default handler displays the traceback

         sys.excepthook = on_crash

         1 / 0
```

```
Traceback (most recent call last):


  File "<ipython-input-40-86b8f9ec8425>", line 34, in <module>
1 / 0


ZeroDivisionError: division by zero
```

** In a shell, the above exception does not reflect what will happen in a real program. This is what it would give you if used in a module: **

1.5 Typical exceptions to catch

1.5.1 Missing import

```
In [207]: try:
              import something
          except ImportError:
              print("This does not exist")
```

```
This does not exist
```

1.5.2 Converting unsafe values

```
In [208]: try:
              a = int('a')
          except ValueError:
              print("This can't be converted")

          try:
              a = list(1)
          except TypeError:
              print("This can't be converted")
```

```
This can't be converted
This can't be converted
```

1.5.3 Dealing with a file

```
In [41]: import uuid

         # It is very unlikely that this file exists
         filename = str(uuid.uuid4())
```

```
        try:
            with open(filename) as f:
                print(f.read())

        # The folliwing error can happen for a lot of reasons: the file does not exist,
        # it's a dir, it's corrupted, it's already in use, lack of permissions...
        # Use one of the subclasses of OSError to target a specific case:
        # - BlockingIOError
        # - ChildProcessError
        # - ConnectionError
        # - BrokenPipeError
        # - ConnectionAbortedError
        # - ConnectionRefusedError
        # - ConnectionResetError
        # - FileExistsError
        # - FileNotFoundError
        # - InterruptedError
        # - IsADirectoryError
        # - NotADirectoryError
        # - PermissionError
        # - ProcessLookupError
        # - TimeoutError
        except OSError:
            print("OMG I can't even !")
```

```
OMG I can't even !
```

1.5.4 Missing module, function, class or variable

```
In [212]: try:
              print(wololo)
          except NameError:
              print('This does not exist')
```

```
This does not exist
```

1.5.5 Using the wrong index

```
In [42]: try:
             lyrics = ["99 beers on the wall..."]
             lyrics[1]
         except IndexError:
             print('Incomplete song')
```

```
Incomplete song
```

1.6 Shorcuts for exceptions handling

1.6.1 In file handling

```
In [214]: import tempfile

          # Dummy file for the example
          _, temp_filename = tempfile.mkstemp()

          # Replace:
          try:
              f = open(temp_filename, 'w')
              f.write('hello')
          finally:
              try:
                  f.close()
              except NameError:
                  pass

          # With:
          with open(temp_filename, 'w') as f:
              f.write('hello')
```

1.6.2 In anything having a .close() method

```
In [170]: import random
          from contextlib import closing

          # Plenty of things can go wrong before closing: http request, db connections...
          # Let's simulate one:
          class DangerousStuffThatCloses:
              def dangerous(self):
                  return 1 / random.randint(0, 1)
              def close(self):
                  print('Closing !')

          # You then can replace:
          try:
              stuff = DangerousStuffThatCloses()
              stuff.dangerous()
          finally:
              try:
                  stuff.close()
              except NameError:
                  pass

          # with :
          with closing(DangerousStuffThatCloses()) as stuff:
              stuff.dangerous()
```

```
Closing !
Closing !
```

1.6.3 In Dictionaries (avoid KeyError)

```
In [23]: thac0 = {
             "sword": -3,
             "axe": -2,
             "staff": -1
         }
```

```
In [17]: # Is this key in this dict ?
         "sword" in thac0
```

```
Out[17]: True
```

```
In [24]: # Get value if it exists or get default value
         print(thac0.get('sword', -5))
         print(thac0.get('bow', -5))
```

```
-3
-5
```

```
In [25]: # Get value if it exists or get default value
         # Create any missing key
         print(thac0.setdefault('sword', -5))
         print(thac0.setdefault('bow', -5))
         print(thac0)
```

```
-3
-5
{'sword': -3, 'axe': -2, 'staff': -1, 'bow': -5}
```

```
In [26]: # Count stuff without dealing with missing initial values
         from collections import Counter
         hp = Counter()
         print(hp['wolf'])
         hp['unicorn'] -= 1
         print(hp)
```

```
0
Counter({'unicorn': -1})
```

```
In [27]: # Generate a missing key on the fly
         from collections import defaultdict
         char_sheet = defaultdict(list) # Can be any callable. here list() creates an empty list
         char_sheet['classes'].append('rogue')
         print(char_sheet)
```

```
defaultdict(<class 'list'>, {'classes': ['rogue']})
```

1.6.4 In iterables (avoid IndexError and TypeError)

```
In [28]: # Get the first element or a default value
         print(next(iter(["first element"])))
         print(next(iter([]), "default value"))
```

```
first element
default value
```

```
In [3]: # islice can slice any iterables, even the ones without a size
        from itertools import islice

        # You can't use [:] on a generator
        generator = (x * x for x in range(10))

        # Like [2: 5] but wiht islice
        print(list(islice(generator, 2, 5)))
```

```
[4, 9, 16]
```

1.6.5 When getting attributes (avoid AttributeError)

```
In [34]: import random

         class Foo:
             bar = True

         obj = random.choice([None, Foo()])
         print(getattr(obj, 'bar', "Default value"))
```

```
True
```

1.6.6 When encoding or decoding (avoid UnicodeXXXError)

```
In [1]: import random
        charset = random.choice(['ascii', 'utf8'])
        # 'ignore' skip on errors, 'replace' put a "?" instead. Both work on decode() and encode
        # and won't do anything if you use the proper character set.
        print("I'm learning french: crèpe de fromage".encode(charset, errors="replace"))
        print(b"I'm learning french: cr\xc3\xa8pe de fromage".decode(charset, errors="ignore"))
```

```
b"I'm learning french: cr?pe de fromage"
I'm learning french: crpe de fromage
```

1.7 Silencing

```
In [4]: import warnings
        from contextlib import suppress

        # Ignore one error
        with suppress(ZeroDivisionError):
            1 / 0

        print('All good')

        # Silence warnings. Exists with a context manager as well.
        warnings.warn('Now you see me', category=DeprecationWarning)
        # This functions has a lot of parameters. RTFM.
        warnings.filterwarnings(action='ignore', category=DeprecationWarning)
        warnings.warn("Now you don't", category=DeprecationWarning)
```

```
All good
```

```
/home/user/.local/lib/python3.6/site-packages/ipykernel/__main__.py:11: DeprecationWarning: Now
```

You can also control warnings by passing:

```
action:message:category:module:lineno
```

to either:

```
the "-w" option of the "python" command
```

or:

```
the "PYTHONWARNINGS" environment variable
```