

Re: What c.l.py's opinions about Soft Exception?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2008-03/msg01043.html>

- *From:* Steven D'Aprano <steve@xx>
 - *Date:* Sun, 09 Mar 2008 12:54:05 -0000
-

On Sun, 09 Mar 2008 00:30:51 -0800, Lie wrote:

(3) Informing codes above it about what's currently happening inside, the thing is just a mundane report that might be useful to codes above

Which might be a useful place to use SoftExceptions

Okay, now we're getting somewhere.

So, I have a function foo() which raises a HardException on error, but it also raises a SoftException if it wants to notify me of something "mundane".

```
def foo(sequence):
    if args == []:
        raise SoftException("empty list")
    return len(args)
```

Now, how do I use that?

```
try:
    x = foo(something)
except TypeError:
    print "you should pass a sequence"
    sys.exit() # unrecoverable error
except SoftException:
    print "you passed an empty list"
print x
```

Am I close?

But there's a problem. Once the SoftException is caught, execution will continue from the line "print x" — but the function foo() never got a chance to actually return a result!

In order to make that work, you would need a significant change to Python's internals. I don't know how difficult that would be, but I'm

Re: What c.l.py's opinions about Soft Exception?

guess that it would be a lot of work for not much benefit.

But even if that happened, it would mean that the one mechanism has TWO different effects:

```
try:  
x = foo(sequence)  
except SoftException:  
print x # this is okay, because foo() did return  
except TypeError:  
print x # this is NOT okay, because foo() never returned
```

That is a recipe for confusion.

[...]

No, you're misunderstanding the purpose of Soft Exception, it's not for silencing errors and not so much for exceptional cases. It's for the more mundane tasks such as:

[...]

Perhaps relabeling it as Warning, and renaming raise SoftException as give Warning might make it more acceptable?

Do you realise that Python already has a warnings module?

And I agree that the probability for misuse is quite high, but the benefits is also quite high, it's just that you can't see it since you're not used to using such exceptions. The benefit of SoftExceptions lies mostly on the regular programmings tasks, not the exceptional programming tasks

Practical Example:

This example takes `_case ideas_` from this simple gravity simulator <http://www.pygame.org/project/617/> BUT `_no` line of code is taken from it_. I only give this link so you can easily know what the case is about without lengthy explanation.

A particle machine.

The particle machine calculates gravity created by the particles in a field. Additionally, it clumps together two particles that happens to be near enough (less than the sum of their radiuses).

The force two particle is expressing to each other is calculated with:

Re: What c.l.py's opinions about Soft Exception?

```
def calculateforce(P1, P2):  
    return (P1.mass - P2.mass) / distance(P1, P2)
```

and this is done to every particle in the field against the current particle.

And the distance is calculated by:

```
def distance(P1, P2)  
    return (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2
```

The problem happens when the distance is small enough and we want to clump them together.

A possible solution to this problem might be to check whether distance is less than $P1.radius + P2.radius$ in the calculateforce. But, this obfuscate the code since we have to separate distance calculation from the main formula (see !s),

I don't agree that this obfuscates the code.

and this also insist that clumping be done on force calculation level (see @s), shortly this piece of code is plain bad:

```
def distance(P1, P2):  
    return (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2
```

```
def calculateforce(P1, P2):  
    ## Separating this dist calculation into its own line is  
    ## necessary if we want to check the value of dist  
    ## Personally I think that's a bit obfuscated.  
    ## Well known formulas should be kept to one line if possible  
    ! dist = distance(P1, P2)
```

```
    if dist <= P1.radius + P2.radius:  
        ## Calling clump() here is bad, because  
        ## there are occasions where we only want to  
        ## calculate force but doesn't want to  
        ## clump it  
        @ clump(P1, P2)  
    else:  
        ! return (P1.mass - P2.mass) / dist
```

I agree that calling clump() there is bad.

Re: What c.l.py's opinions about Soft Exception?

```
## Codes calling calculateforce()
# Note: this code is located inside a loop

F = calculateforce(P1, P2)
# Do something else, acceleration calculation, movement
calculations, etc
```

A better refactoring would be like this, but this requires calculating distance twice (see !s):

That's not better. Don't do it.

```
def distance(P1, P2):
    return (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2

def calculateforce(P1, P2):
    ## Here distance is calculated once
    ! return (P1.mass - P2.mass) / distance(P1, P2)

## Codes calling calculateforce()
# Note: this code is located inside a loop

## Here distance is calculated again
! if distance(P1, P2) <= P1.radius + P2.radius:
    clump(P1, P2)
    break
F = calculateforce(P1, P2)
# Do something else, acceleration calculation, movement
calculations, etc
```

And here's a way to do it that doesn't calculate anything twice, and doesn't require any exceptions:

```
def calculateforce(P1, P2, dist):
    return (P1.mass - P2.mass)/dist
```

And then for all pairs of particles:

```
dist = distance(P1, P2)
if dist <= P1.radius + P2.radius:
    clump(P1, P2)
    break
F = calculateforce(P1, P2, dist)
```

Re: What c.l.py's opinions about Soft Exception?

Re: What c.l.py's opinions about Soft Exception?

```
A much better solution would be to use SoftException
def distance(P1, P2):
    D = (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2
    if D <= P1.radius + P2.radius:
        raise Collision
    return D
```

But look at that line, "raise Collision". You could replace that with a callback function, and have the same result:

```
DO_NOTHING = lambda : None

def distance(P1, P2, callback=DO_NOTHING):
    D = (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2
    if D <= P1.radius + P2.radius:
        callback()
    return D
```

That gives you virtually everything you want. If you want to ignore the signal, you simply call distance(P1, P2). If you want to do something on the signal, you set the callback.

But frankly, the function distance() is NOT the place for that. Why should the distance() function decide what's a special result and what isn't? With your plan, you end up with functions like this:

```
def distance(P1, P2):
    D = (P1.X - P2.X) ** 2 - (P1.Y - P2.Y) ** 2
    if D <= P1.radius + P2.radius:
        raise Collision
    elif D > 1000:
        raise FredsModuleEscape
    elif D <= 1:
        raise MagnetismModuleUnity
    elif D == 0:
        raise AnotherFunctionSingularity
    elif ...
    ...
    ...
    else:
        raise NothingSpecialHappened
    return D
```

Every module and function that calls distance() will start demanding that

Re: What c.l.py's opinions about Soft Exception?

it raises the SoftExceptions that *it* wants, and before you know it, your distance() function has a hundred SoftExceptions covering all sorts of things that most people don't care about.

No. This is a terrible idea. If the caller wants to treat a particular result as special, the caller should be responsible for detecting that, not the callee.

This results in a cleaner code.

I don't agree.

And it also allow _other part of codes_
that uses calculate distance and force to easily ignore or handle the Collision Exception.

And demand their own SoftExceptions.

Okay, now that I understand your intention better, I've gone from -1 to -2 on the idea. I no longer think it's a bad idea, I think it's a TERRIBLE idea.

Just out of curiosity, are there any existing languages that do something like this, or did you invent it yourself?

—
Steven

.