November 2016 meeting.

PyRVA Sub-lightning talk

try/except v. if/else

slides: github.com/georgeflanagin/pyrva

Who am I?

George Flanagin (me@georgeflanagin.com)

- UR computer scientist, and I taught computer science down the street at VCU ending in 2004.
- My project is the critical infrastructure of the University's data integration.
- I work in Python 3 nearly every day (including weekends).

How do you decide if something is an int?

Suppose `s` is a string with data from the keyboard.

This always fails because Python objects are type-const:

```
    if isinstance(s, int): i = int(s)
```

We just said it was an str... so it cannot be both, right?

Well, you could get all clever....

And try this:

```
if s.isdigit(): i = int(s)
```

But what about the possibility that `s == '-1'` or it might be `s == '1E06'` ?

Python's rule is "just ***try*** it."

There is only one certain way to convert types:

```
try:
    i = int(s)
except Exception as e:
    do_something(e)
```

Only Python knows the range of valid string representations of int-s.

Emptiness ....

- Python defines `if(False)` loosely.
    - `False`
    - `None`
    - `int(0)`
    - empty string
    - dicts and lists with no members

But not loosely enough for my project ...

Result set from an Oracle query might have nothing, but we still get a list with an empty row. The rows are dicts with k-v pairs of column name and value:

```
result = [dict()]

if result: print("yes, it's true")
```

The problem is somewhat complicated ...

```python
if not dict():
    print("yes, an empty dict is false.")

if not list():
    print("yes, and so is an empty list.")

if list(dict()):
    print("but two falses make a truth")
```

I want an empty() function like PHP ...

(... ok, not really like PHP ...)

```
def empty(o:any) -> bool:
   if not o: return True # the existing test

   # now what?
```

You can "if" it ...

```
def empty(o:any) -> bool:
   if not o: return True

   if hasattr(o, "__iter__"):
      for oo in o:
         if not empty(oo): return False
      return True
   else:
      return False
```

But this is better:

```
def empty(o:any) -> bool:
    if not o: return True

    try:
        for oo in o:
            if not empty(oo): return False
        return True
    except:
        return False
```

And after more familiarity with the libraries:

```
from functools import reduce
from operator import and_

def empty(o:any) -> bool:
    if not o: return True

    try:
        return reduce(and_, [empty(oo) for oo in o])
    except:
        return False
```