4.1 - Errors and Exception Handling

Ha Khanh Nguyen (hknguyen)

---

1. `try/except`

- Many functions only work on certain kinds of input.
- For example, the `int()` function fails with string that contains anything other than number!

```
int('430')
```

```
## 430
```

```
int('430.12')
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): ValueError: invalid lit
eral for int() with base 10: '430.12'
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

- Suppose we want a version of `int()` that instead of throwing an error, it just simply returns the input string!
- This can be done by writing a function that enclosed the call to `int()` in a `try/except` block:
    - The code follows `except` will only be executed if `int()` raises an exception.

```
# define the function
def attempt_int(s):
  try:
    return int(s)
  except:
    return s
```

```
# call the function
attempt_int('430')
```

```
## 430
```

```
attempt_int('430.12')
```

```
## '430.12'
```

- You might notice that `int()` can raise exceptions other than `ValueError`:

```
int((1, 2))
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): TypeError: int() argume
nt must be a string, a bytes-like object or a number, not 'tuple'
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

- You might want to only suppress `ValueError`, since a `TypeError` (the input was not a string or numeric value) might indicate a bigger bug. To do that, write the exception type after `except`:

```
def attempt_int(s):
  try:
    return int(s)
  except ValueError:
    return s
```

```
attempt_int((1, 2))
attempt_int('123.12')
```

- You can catch multiple exception types by writing a tuple of exception types instead (the parentheses are required):

```
def attempt_int(s):
  try:
    return int(s)
  except (ValueError, TypeError):
    return s
```

---

2. `finally`

- There are cases where even though there is an exception, you still want some code to be executed regardless.
- To do this, use `finally` keyword:

```
f = open('a path to a file', 'w')

try:
  write_to_file(f)
finally:
  f.close()
```

- Similarly, you can have code that executes only if the `try:` block succeeds using `else`:

```
f = open(path, 'w')

try:
  write_to_file(f)
except:
  print('Failed')
else:
  print('Succeeded')
finally:
  f.close()
```

3. Exercise

Let s be a string that contains a simple mathematical expression, e.g.,

```
s = '1.5 + 2.1'
```

```
s = '10.0-1.6'
```

```
s = '3.1*5.8'
```

```
s = '4.7 /7.2'
```

The expression will only have 2 operands and the operator will be one of the following: +, -, * and /.

Write a **function** that interprets the expression, then evaluates it and returns the result. If the input string `s` does not follow the format above, simply return a `None` object.

This lecture note is modified from Chapter 3 of Wes McKinney's Python for Data Analysis 2nd Ed (https://www.oreilly.com/library/view/python-for-data/9781491957653/).