Exception Error Messages

Money Factory

A `Money` class + a factory helper function:

```
import re

class Money:
    def __init__(self, dollars, cents):
        self.dollars = dollars
        self.cents = cents
    # Plus other methods

def money_from_string(amount):
    # amount is a string like "$140.75"
    match = re.search(
        r'^\$(?P<dollars>\d+)\.(?P<cents>\d\d)$', amount)
    dollars = int(match.group('dollars'))
    cents = int(match.group('cents'))
    return Money(dollars, cents)
```

```
>>> cash = money_from_string("$99.21")
>>> cash.dollars, cash.cents
(99, 21)
```

Huh?

What happens if you pass it bad input? The error isn't very informative.

```
>>> money_from_string("$140.75")
Money(140,75)
>>> money_from_string("$12.30")
Money(12,30)
>>> money_from_string("Big money")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in money_from_string
AttributeError: 'NoneType' object has no attribute 'group'
```

Imagine finding this error deep in a stack trace. We have better things to do than decrypt this.

Better Errors

Add a check on the `match` object. If it's `None`, meaning `amount` doesn't match the regex, raise a `ValueError`.

```python
import re
def money_from_string(amount):
    match = re.search(
        r'^\$(?P<dollars>\d+)\.(?P<cents>\d\d)$', amount)
    # Adding the next two lines here
    if match is None:
        raise ValueError("Invalid amount: " + amount)
    dollars = int(match.group('dollars'))
    cents = int(match.group('cents'))
    return Money(dollars, cents)
```

More Understandable

```
>>> money_from_string("$140.75")
Money(140,75)
>>> money_from_string("$12.30")
Money(12,30)
>>> money_from_string("Big money")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in money_from_string
ValueError: Invalid amount: Big money
```

This is MUCH better. The exact nature of the error is immediately obvious.

Catch And Re-Raise

In an `except` block, you can re-raise the current exception.

Just write `raise` by itself:

```
try:
    do_something()
except ExceptionClass:
    handle_exception()
    raise
```

It's a shorthand, equivalent to this:

```
try:
    do_something()
except ExceptionClass as err:
    handle_exception()
    raise err
```

Interject Behavior

One pattern this enables: inject but delegate.

```
try:
    process_user_input(value)
except ValueError:
    logging.error("Invalid user input: %s", value)
    raise
```

It enables other patterns too.

Creating directories

`os.makedirs()` creates a directory.

```python
# Creates the directory "some-directory",
# relative to the current directory.
import os
os.makedirs("some-directory")
```

But if the directory already exists, it raises `FileExistsError`.

```
>>> os.makedirs("some-directory")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/lib/python3.6/os.py", line 220, in makedirs
    mkdir(name, mode)
FileExistsError: [Errno 17] File exists: 'some-directory'
```

Using In Code

Suppose if that happens, we want to log it, then continue:

```python
import os
import logging
UPLOAD_ROOT = "/var/www/uploads"

def create_upload_dir(username):
    userdir = os.path.join(UPLOAD_ROOT, username)
    os.makedirs(userdir)

def setup_user(username):
    try:
        create_upload_dir(username)
    except FileExistsError:
        logging.warning(
            "Upload dir for new user already exists")
```

This works, but the log message is not informative:

```
WARNING: Upload dir for new user already exists
```

Logging The directory

`FileExistsError` objects have an attribute called `filename`. Let's use that to create a useful log message:

```python
# Better version!
def setup_user(username):
    try:
        create_upload_dir(username)
    except FileExistsError as err:
        logging.warning("Upload dir already exists: %s",
            err.filename)
    # And other setup
```

MUCH better:

```
WARNING: Upload dir already exists: /var/www/uploads/joe
```

OSError

`FileExistsError` is only in Python 3. In Python 2, `os.makedirs()` instead raises `OSError`.

But `OSError` can indicate many other problems:

- filesystem permissions
- a system call getting interrupted
- a timeout over a network-mounted filesystem
- And the directory already existing.. the only one we care about.

How do you distinguish between these?

errno

`OSError` objects set an `errno` attribute. It's essentially the `errno` variable from C.

The standard constant for "file already exists" is `EEXIST`:

```
from errno import EEXIST
```

Game plan:

- Optimistically create the directory.
- if `OSError` is raised, catch it.
- Inspect the exception's `errno` attribute. If it's equal to `EEXIST`, this means the directory already existed; log that event.
- If `errno` is something else, it means we don't want to catch this exception here; re-raise the error.

create_upload_dir() in 2.x

```
# How to accomplish the same in Python 2.

import os
import logging
from errno import EEXIST

UPLOAD_ROOT = "/var/www/uploads/"

def create_upload_dir(username):
    userdir = os.path.join(UPLOAD_ROOT, username)
    os.makedirs(userdir)

def setup_user(username):
    try:
        create_upload_dir(username)
    except OSError as err:
        if err.errno != EEXIST:
            raise
        logging.warning("Upload dir already exists: %s",
            err.filename)
```