More Advanced Logging

Why not use print()?

If logging is new you to you, you may wonder why we don't just sprinkle the code with `print()` statements.

Essentially: some of those `print()` statements are more important than others. Python's logging module makes that hierarchy explicit.

And: you can change the threshold as needed.

Also: `print()` only writes to stdout. The logging module allows other destinations.

Efficiency

Freely write as many logging statements as you want, without affecting performance. Any lines below the threshold are cheaply ignored.

```
# Only log INFO messages or higher
logging.basicConfig(log_level = logging.INFO)
# Python essentially skips over this statement.
logging.debug("Received a message from X")
```

Often you will introduce many `debug()` statements while troubleshooting, and leave them in after the issue is resolved.

Modern String Formatting

String Formatting means inserting parameters into a template string at runtime, to get a final, calculated string.

In modern Python, you'll mainly use either f-strings, or `str.format()`.

```
>>> item = "cup of coffee"
>>> price = 1.75
>>> print(f"Your {item} costs ${price:0.2f}.")
Your cup of coffee costs $1.75.
```

```
>>> template = "Your {} costs ${:0.2f}."
>>> output = template.format("cup of coffee", 1.75)
>>> print(output)
Your cup of coffee costs $1.75.
```

Percent Formatting

Python's original string interpolation. Inspired by C.

Uses `%s` for string; `%f` for float; `%d` for integer.

```
>>> template = "Your %s costs $%0.2f."
>>> output = template % ("cup of coffee", 1.75)
>>> print(output)
Your cup of coffee costs $1.75.

>>> # Or as a dictionary:
... data = {"item": "burger", "price": 6.95}
>>> print("Your %(item)s costs $%(price)0.2f." % data)
Your burger costs $6.95.
```

Largely outdated now, but still important in logging. Even in the latest Python 3.

Log Message Parameters

In many (most) log messages, you’ll want to inject runtime data.

There’s a good way and a bad way. The good way uses a *variant* of percent formatting:

```
logging.info("Your %s costs $%0.2f.", item, price)
```

- First argument: a percent-style template string
- 2nd and subsequent arguments: the parameters

The Bad Way

The good way again:

```
logging.info("Your %s costs $%0.2f.", item, price)
```

The bad way:

```
logging.info("Your %s costs $%0.2f." % (item, price))
```

Why is that bad?

Avoid Unnecessary Calculations

Imagine the log threshold is set to `logging.WARNING`. Then neither of these will emit any message:

```
logging.info("Your %s costs $%0.2f.", item, price)
logging.info("Your %s costs $%0.2f." % (item, price))
```

The second incurs a computational cost, to calculate the log message that’s thrown away. But the first is very cheap.

This removes any hesitation to introducing log statements. Especially for INFO and lower, more tends to be better.

Why Percent Formatting?

Basically: Legacy constraints.

Original plan was to remove percent formatting completely. But it's not practical to convert old logging code to `str.format()`.

Rather than render large groups of Python programs practically impossible to upgrade to Python 3, percent formatting is here to stay.

It's less painful to use the alternatives in NEW logging code. But I recommend you just use percent formatting.

Log Formats

Each call to `logging.debug()`, `.warning()`, etc. creates one log record - one line in the log file.

What info that record includes, and in what order, is determined by the *log format*.

```
logging.basicConfig(
    format="%(asctime)s %(levelname)s %(message)s")
logging.warning("Collision imminent")
```

Run this as a program, and you get the following log line:

```
2019-12-31 10:56:00,959 WARNING Collision imminent
```

Contrast with the default format:

```
WARNING:root:Collision imminent
```

Log Format Attributes

Attribute	Format	Description
asctime	`%(asctime)s`	Human-readable date/time
funcName	`%(funcName)s`	Function containing the logging call
lineno	`%(lineno)d`	The line number of the logging call
message	`%(message)s`	The log message
pathname	`%(pathname)s`	Full pathname of the source file of the logging call
levelname	`%(levelname)s`	Text logging level for the message (*DEBUG*, *INFO*, *WARNING*, *ERROR*, *CRITICAL*)
name	`%(name)s`	The logger's name

Lab: Basic Interface

Lab file: `logging/logging_basic.py`

- In `labs` folder
- When you are done, study the solution - compare to what you wrote.
- ... and then optionally do `logging/logging_basic_extra.py`