Effortless Logging:

A deep dive into the logging module

PyCon 2018
May 13, 2018

Mario Corchero
Python Infrastructure @ Bloomberg
@mariocj89

TechAtBloomberg.com

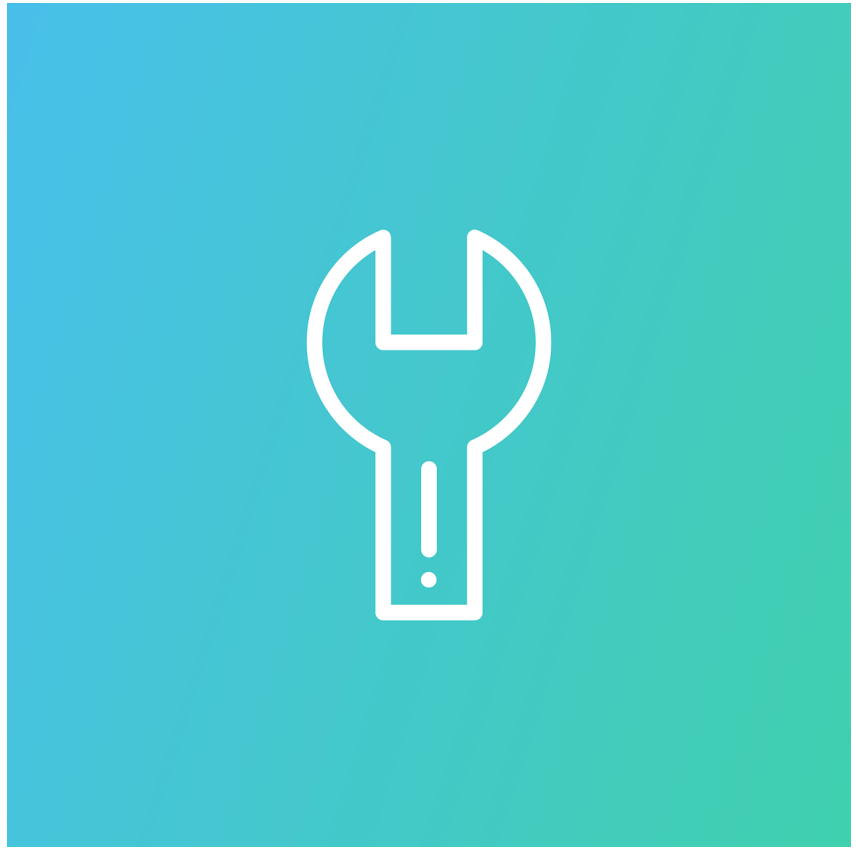
Bloomberg

Agenda

- Why logging matters
- How logging works
- How to use it
- How to configure it
- Sailing to the guts of logging
- Sample use cases
- Q&A

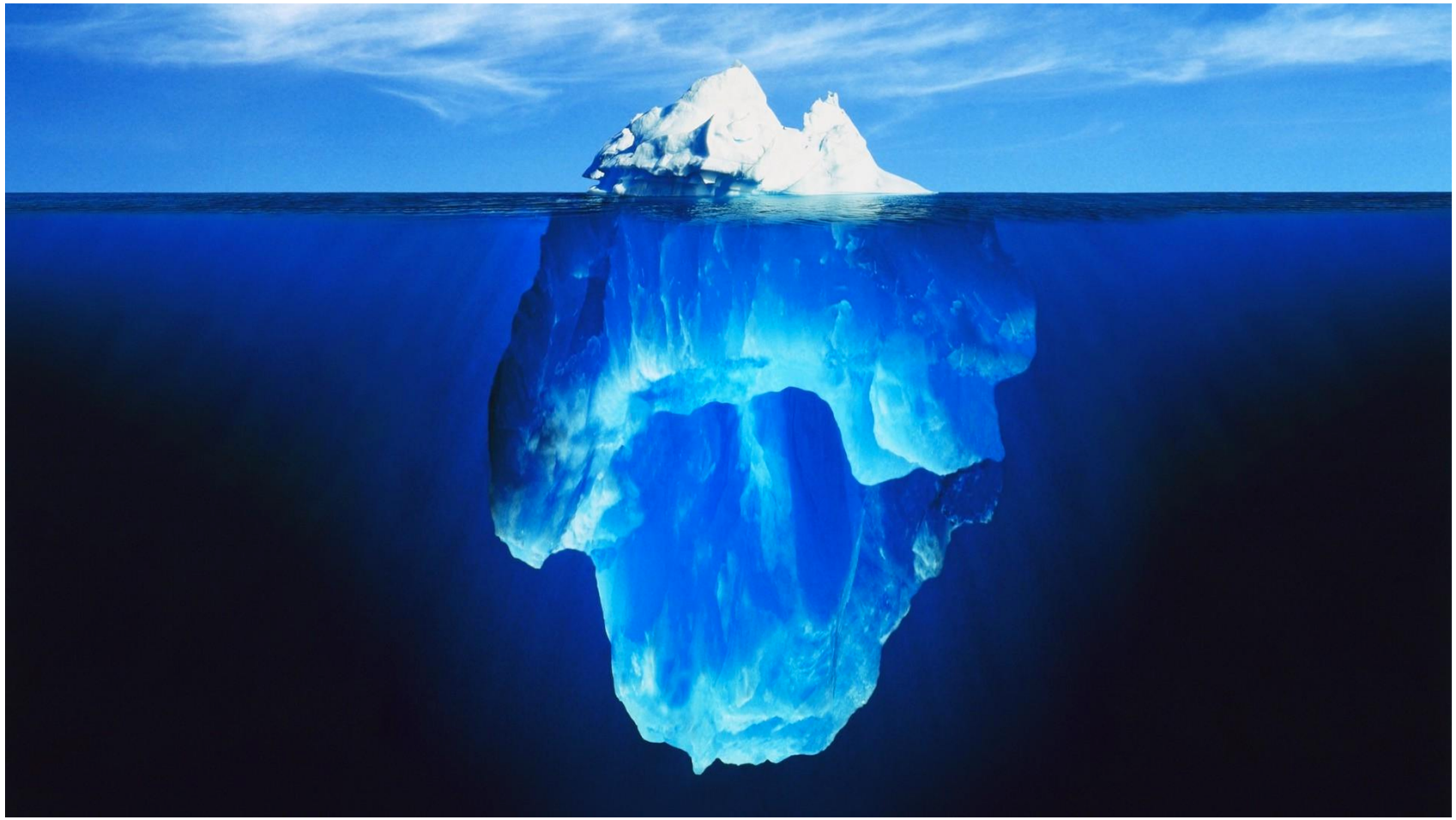
Why logging matters

Versatility & Configurability



Bloomberg

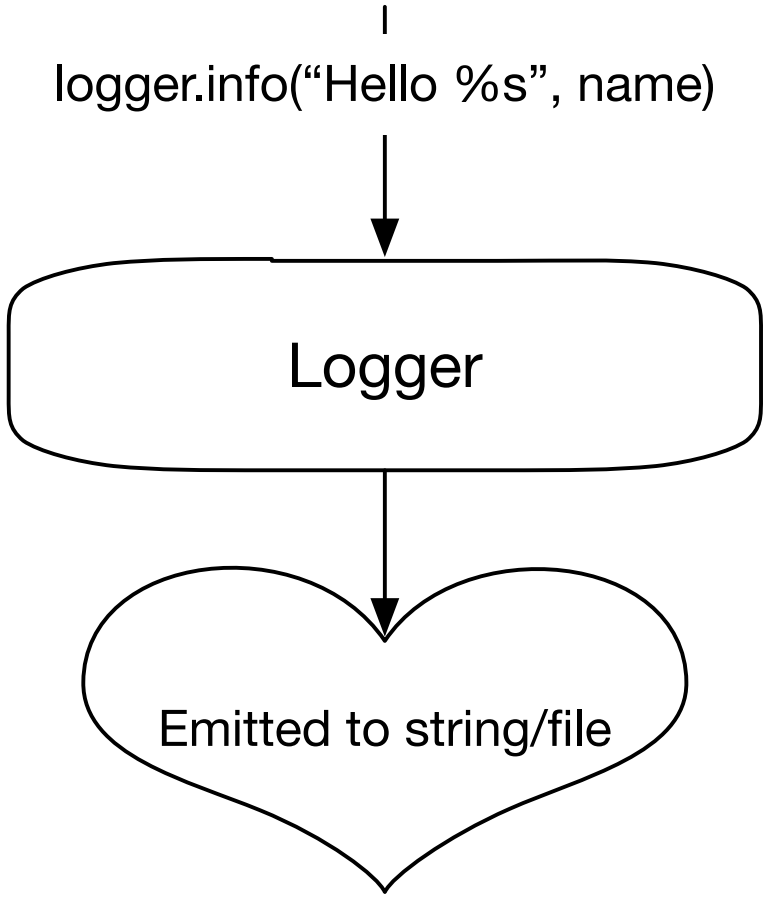
How logging works



Bloomberg

Logger

Logger



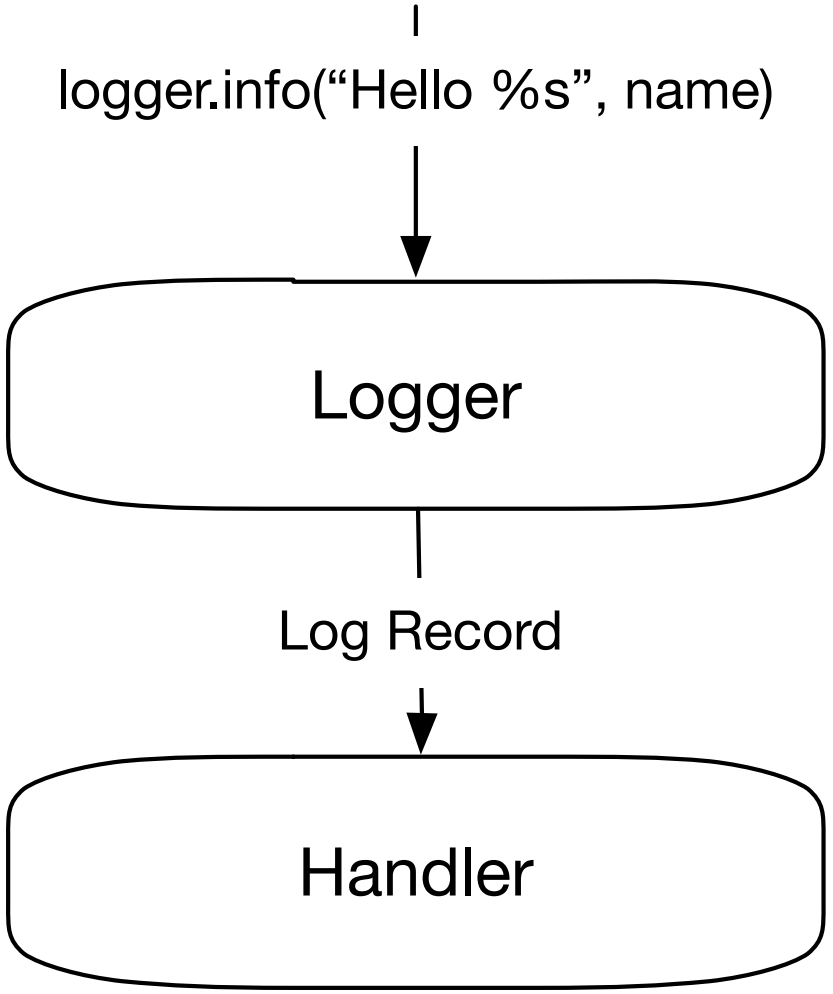
Bloomberg

Record

Bloomberg

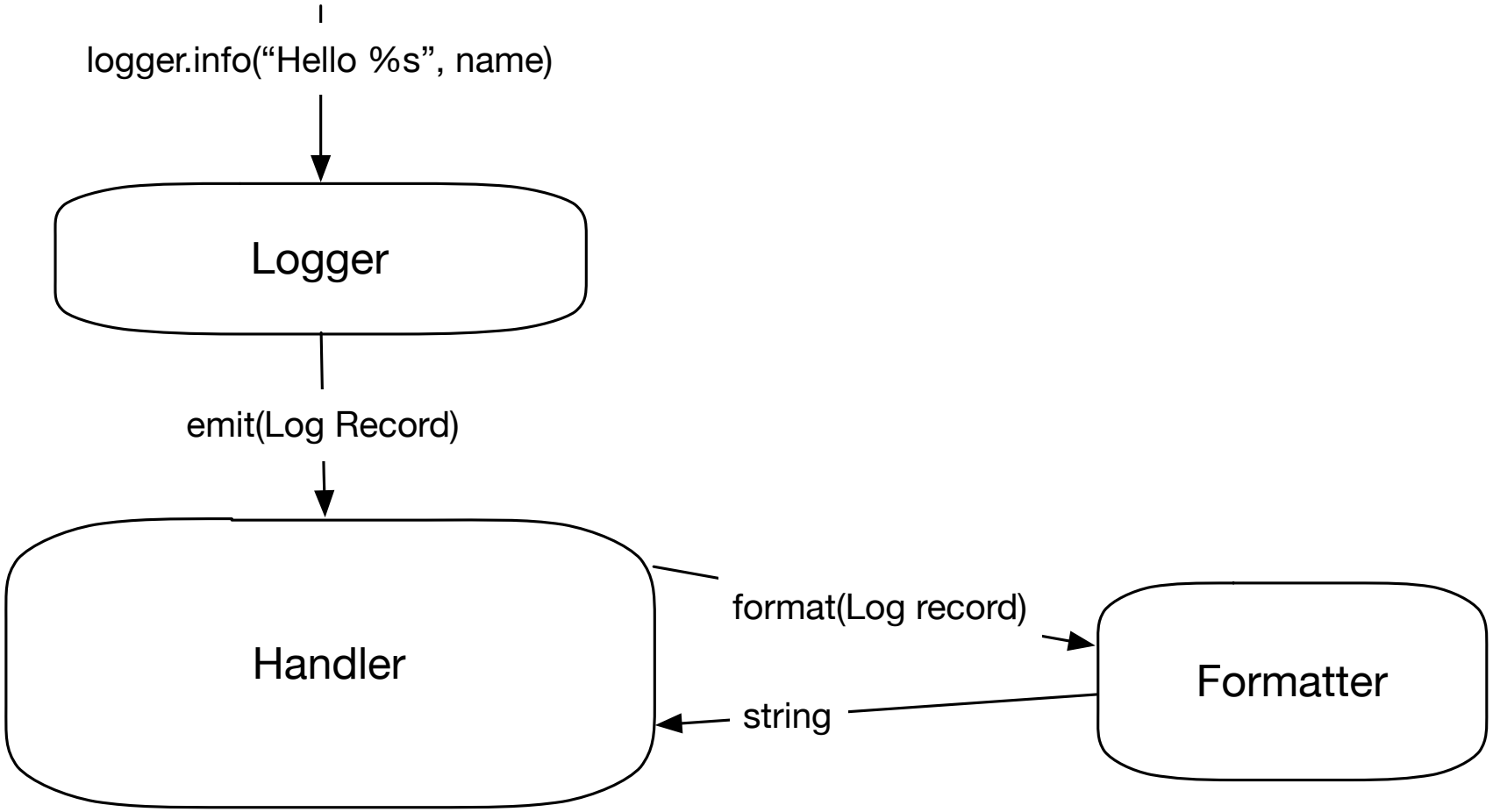
Handler

Handler



Formatter

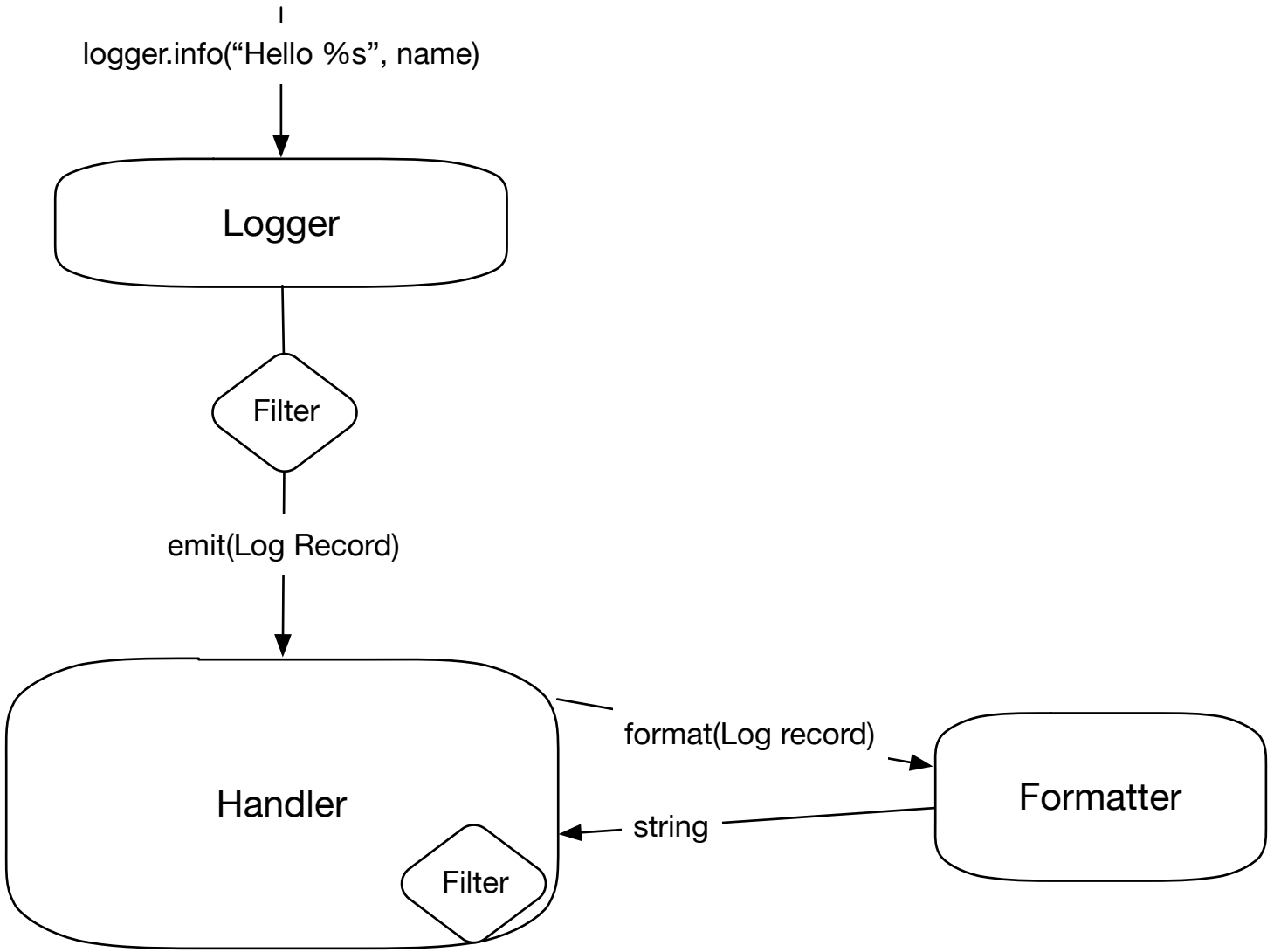
Formatter



Bloomberg

Filter

Filter

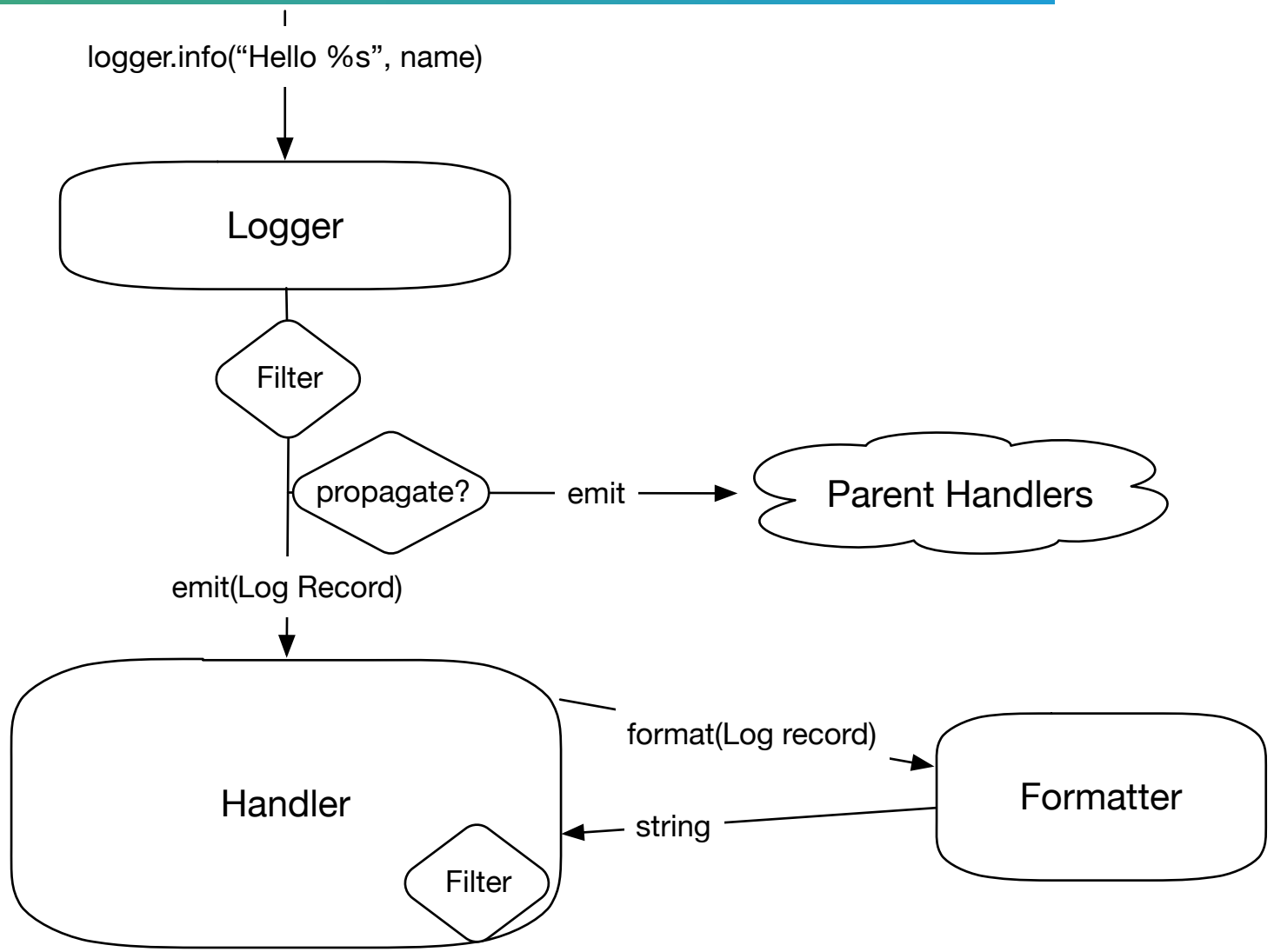


Bloomberg

The hierarchy

```
logger = logging.getLogger(”parent.child”)
```

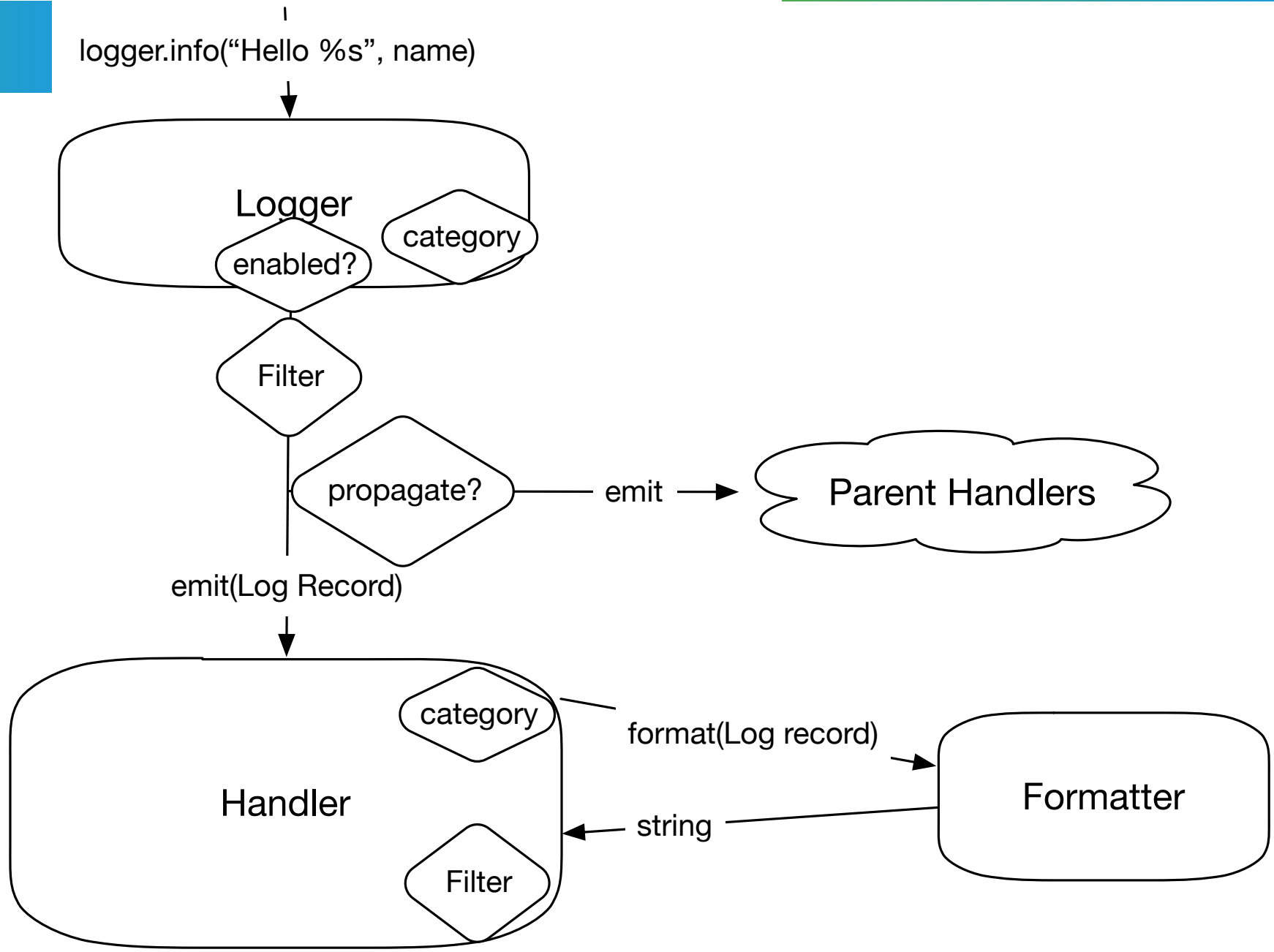
The hierarchy



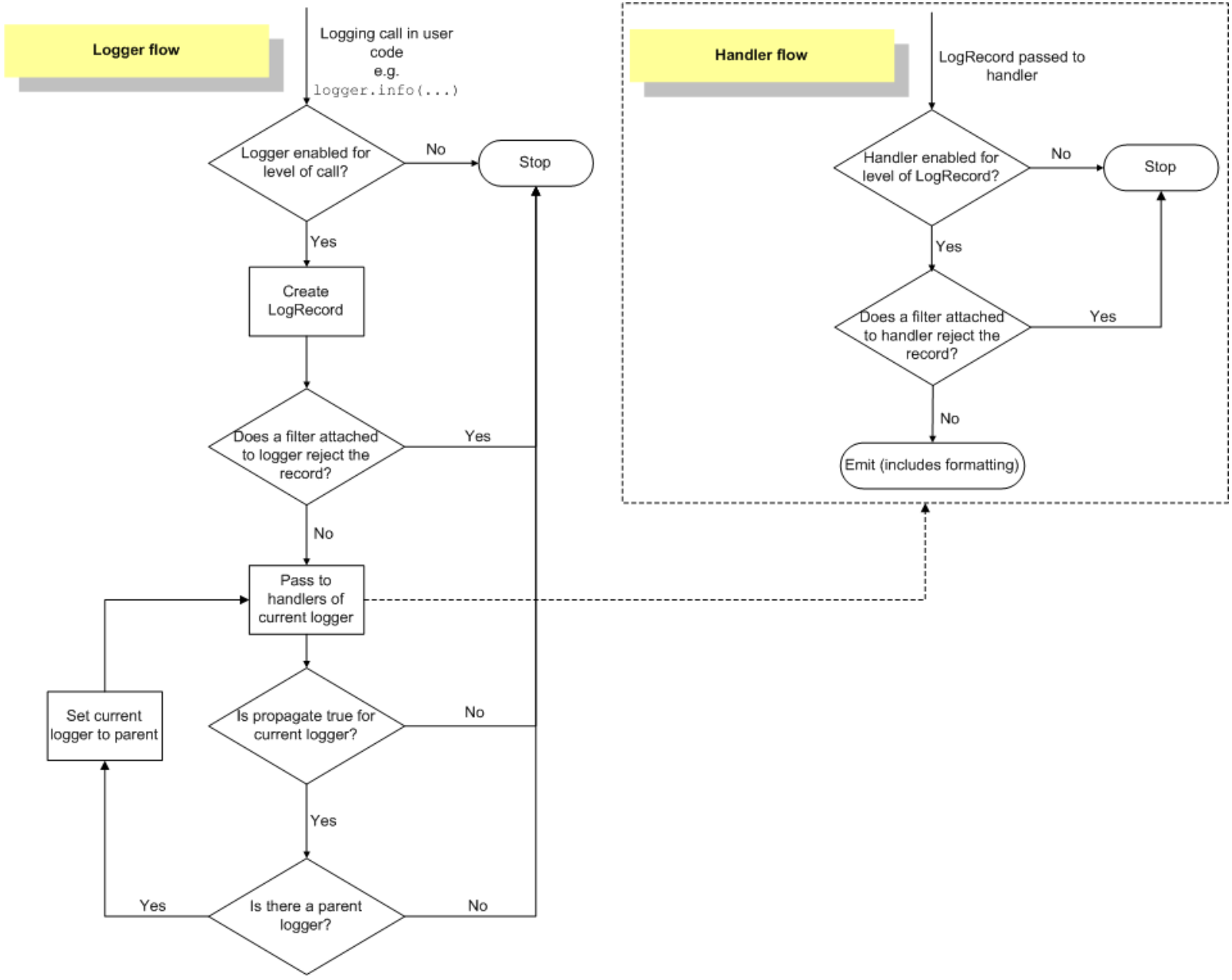
Bloomberg

The actual flow

The actual flow

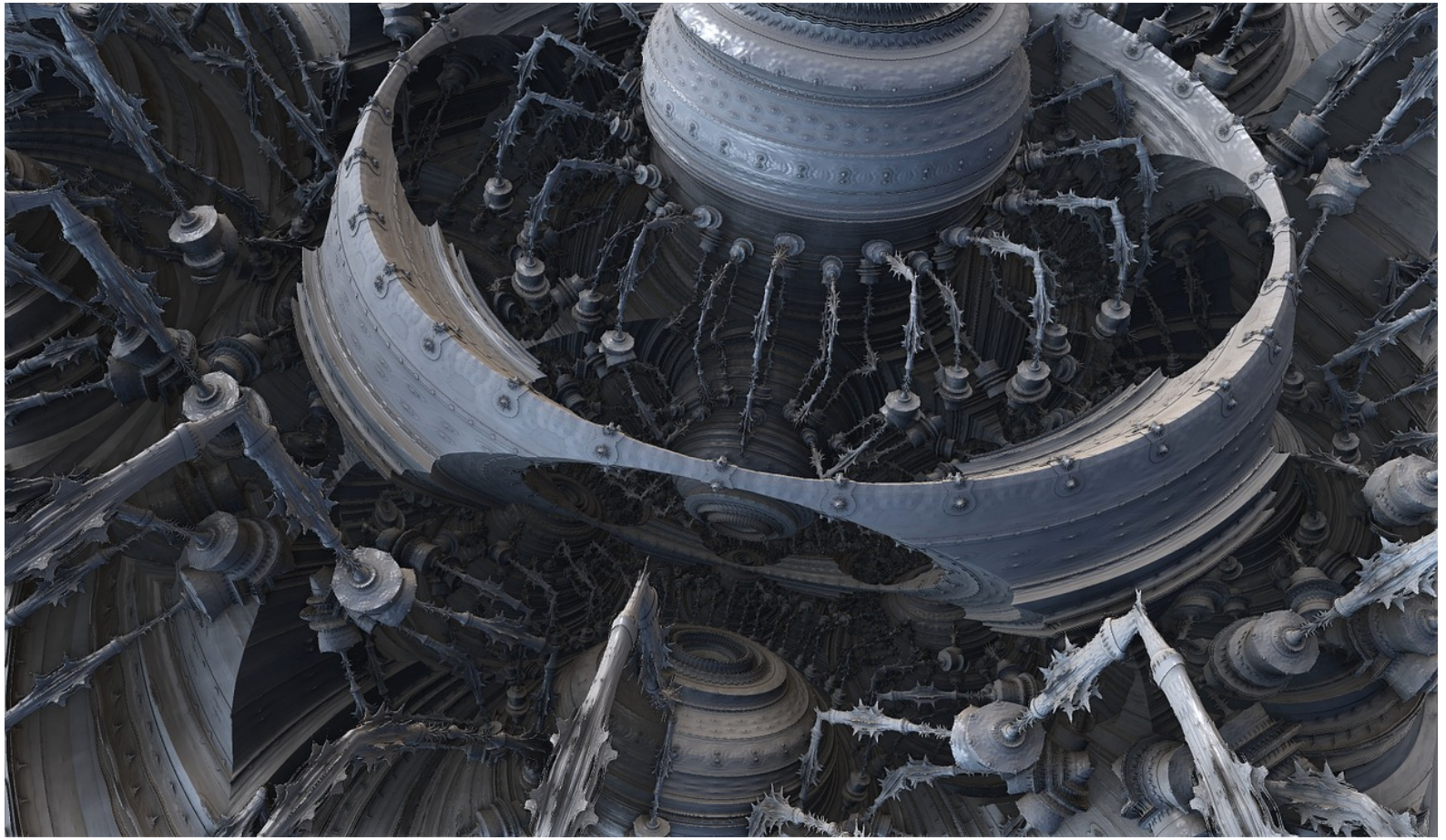


The actual flow



TechAtBloomberg.com

How to use it



Bloomberg

How to use it

```
import logging

def sample_function(secret_parameter):
    logger = logging.getLogger(__name__)
    logger.debug (”Going to perform magic with '%s’”,  secret_parameter)
    ...
    try:
        result = do_magic(secret_parameter)
    except IndexError:
        logger.exception("OMG it happened again, someone please tell Laszlo")
    except :
        logger.info("Unexpected exception", exc_info=True)
        raise
    else:
        logger.info("Magic with '%s' resulted in '%s'", secret_parameter, result, stack_info=True)
```


Bloomberg

Common misuses

Common misuses

```
logger.debug("Hello {}".format(name))
```

```
logger.debug("Hello %s", name))
```

Common misuses

```
except Exception as error:
    logging.info("A terrible error happened: %s", error)
```

```
except Exception:
    logging.info("A terrible error happened", exc_info=True)
```

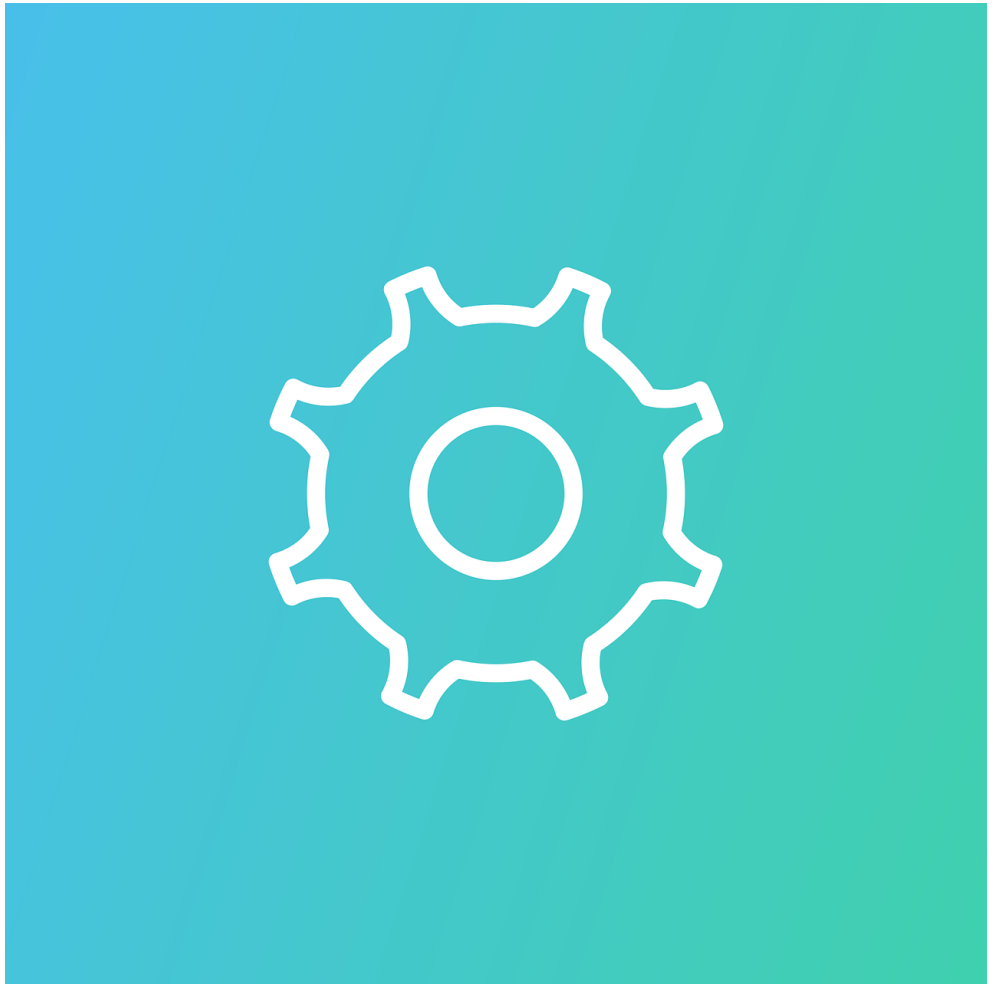

Bloomberg

Common misuses

```
Logging.getLogger("project_name")
```

```
Logging.getLogger(__name__)
```

How to configure it



Bloomberg

basicConfig

```
import logging
logging.basicConfig(level='INFO')
```

filename
filemode
format
datefmt
level
stream

dictConfig

```
config = {    # logging.config.dictConfig(config)
    'disable_existing_loggers': False,
    'version': 1,
    'formatters': {
        'short': {
            'format': '%(asctime)s %(levelname)s %(name)s: %(message)s'
        },
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'formatter': 'short',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        '': {
            'handlers': ['console'],
            'level': 'INFO',
        },
        'plugins': {
            'level': 'ERROR',
        }
    },
}
```

Demo!

```
import logging
logging.basicConfig(level='INFO')
logger = logging.getLogger(’logname')
logger.info('Hello %s', 'EuroPython')
```

Calling info

```
ipdb> l
      1 import logging
      2 logging.basicConfig(level='INFO')
      3 logger = logging.getLogger('logname')
----> 4 logger.info('Hello %s', 'EuroPython')
      5
```


Bloomberg

Logger.info

```
> /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/logging/__init__.py(1300)info()
   1291     def info(self, msg, *args, **kwargs):
   1292         """
   1293         Log 'msg % args' with severity 'INFO'.
   1294
   1295         To pass exception information, use the keyword argument exc_info with
   1296         a true value, e.g.
   1297
   1298         logger.info("Houston, we have a %s", "interesting problem", exc_info=1)
   1299         """
1> 1300         if self.isEnabledFor(INFO):
   1301             self._log(INFO, msg, args, **kwargs)
```

Logger._log

```
> /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/logging/__init__.py(1435)_log()
  1414     def _log(self, level, msg, args, exc_info=None, extra=None, stack_info=False):
  1415         """
  1416         Low-level logging routine which creates a LogRecord and then calls
  1417         all the handlers of this logger to handle the record.
  1418         """
  1419         sinfo = None
  1420         if _srcfile:
  1421             #IronPython doesn't track Python frames, so findCaller raises an
  1422             #exception on some versions of IronPython. We trap it here so that
  1423             #IronPython can use logging.
  1424             try:
  1425                 fn, lno, func, sinfo = self.findCaller(stack_info)
  1426             except ValueError: # pragma: no cover
  1427                 fn, lno, func = "(unknown file)", 0, "(unknown function)"
  1428         else: # pragma: no cover
  1429             fn, lno, func = "(unknown file)", 0, "(unknown function)"
  1430         if exc_info:
  1431             if isinstance(exc_info, BaseException):
  1432                 exc_info = (type(exc_info), exc_info, exc_info.__traceback__)
  1433             elif not isinstance(exc_info, tuple):
  1434                 exc_info = sys.exc_info()
2> 1435         record = self.makeRecord(self.name, level, fn, lno, msg, args,
  1436                                  exc_info, func, extra, sinfo)
  1437         self.handle(record)
  1433             elif not isinstance(exc_info, tuple):
  1434                 exc_info = sys.exc_info()
```

Logger.handle

```
> /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/logging/__init__.py(1446)handle()
   1439     def handle(self, record):
   1440         """
   1441         Call the handlers for the specified record.
   1442
   1443         This method is used for unpickled records received from a socket, as
   1444         well as those created locally. Logger-level filtering is applied.
   1445         """
3> 1446         if (not self.disabled) and self.filter(record):
   1447             self.callHandlers(record)
   1448
```

Logger.callHandlers

```
> /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/logging/__init__.py(1506)callHandlers()
  1493     def callHandlers(self, record):
  1503         c = self
  1504         found = 0
  1505         while c:
4> 1506             for hdlr in c.handlers:
  1507                 found = found + 1
6 1508                 if record.levelno >= hdlr.level:
  1509                     hdlr.handle(record)
5 1510             if not c.propagate:
  1511                 c = None    #break out
  1512             else:
  1513                 c = c.parent
  1514         if (found == 0):
  1515             if lastResort:
  1516                 if record.levelno >= lastResort.level:
  1517                     lastResort.handle(record)
  1518             elif raiseExceptions and not self.manager.emittedNoHandlerWarning:
  1519                 sys.stderr.write("No handlers could be found for logger"
  1520                                  " \"%s\"\n" % self.name)
  1521                 self.manager.emittedNoHandlerWarning = True
```

Handler.handle

```
> /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/logging/__init__.py(854)handle()
    845     def handle(self, record):
    846         """
    847         Conditionally emit the specified logging record.
    848
    849         Emission depends on filters which may have been added to the handler.
    850         Wrap the actual emission of the record with acquisition/release of
    851         the I/O thread lock. Returns whether the filter passed the record for
    852         emission.
    853         """
7-> 854         rv = self.filter(record)
    855         if rv:
    856             self.acquire()
    857             try:
8   858                 self.emit(record)
    859             finally:
    860                 self.release()
    861         return rv
```

Sample recipes

Multiple handlers

```
'loggers': {
    'request': {
        'handlers': ['request_logs'],
        'level’: ‘DEBUG',
        'propagate': False,
    },
    '': {
        'handlers': ['info_log_file', 'error_log_file', 'debug_log_file', 'mail_admins'],
        'level': 'DEBUG',
    },
}
```

Logging JSON

```
import logging
import logging.config
import json
ATTR_TO_JSON = ['created', 'filename', 'funcName', 'levelname', 'lineno', 'module', 'msecs', 'msg',
'name', 'pathname', 'process', 'processName', 'relativeCreated', 'thread', 'threadName']
class JsonFormatter:
    def format(self, record):
        obj = {attr: getattr(record, attr)
                for attr in ATTR_TO_JSON}
        return json.dumps(obj, indent=4)

handler = logging.StreamHandler()
handler.formatter = JsonFormatter()
logger = logging.getLogger(__name__)
logger.addHandler(handler)
logger.error("Hello")
```

Using filters to add context

```
import logging
import logging.config
GLOBAL_SCARY_STUFF = 1

class ContextFilter(logging.Filter):
    def filter(self, record):
        global GLOBAL_SCARY_STUFF
        GLOBAL_SCARY_STUFF += 1
        record.crazy_global_data = GLOBAL_SCARY_STUFF
        return True

handler = logging.StreamHandler()
handler.formatter = logging.Formatter("%(crazy_global_data)s %(message)s")
handler.addFilter(ContextFilter())
logger = logging.getLogger(__name__)
logger.addHandler(handler)

logging.getLogger(__name__).error("Hi1")
logging.getLogger(__name__).error("Hi2")
```

Buffering

```
import logging
import logging.handlers

class SmartBufferHandler(logging.handlers.MemoryHandler):
    … init ...
    def emit(self, record):
        if len(self.buffer) == self.capacity - 1:
            self.buffer.pop(0)
            super().emit(record)
handler = SmartBufferHandler(buffered=2, target=logging.StreamHandler(), flushLevel=ERROR)
logger = logging.getLogger(__name__)
logger.setLevel("INFO")
logger.addHandler(handler)

logging.getLogger(__name__).error("Hello1")
logging.getLogger(__name__).info("Hello2")
logging.getLogger(__name__).info("Hello3")
logging.getLogger(__name__).error("Hello4")
```


Bloomberg

Non blocking handling of records

```
que = queue.Queue(-1) # no limit on size

queue_handler = QueueHandler(que)
handler = logging.StreamHandler()
listener = QueueListener(que, handler)

root = logging.getLogger()
root.addHandler(queue_handler)

listener.start()
root.warning('Look out!')
listener.stop()
```

Console output

```
class MaxLevelFilter:
    def __init__(self, max_level=None):
        self.max_level = max_level
    def filter(self, record):
        return record.levelno <= self.max_level

import sys
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
stdout = logging.StreamHandler(sys.stdout)
stdout.setLevel("DEBUG")
stdout.addFilter(MaxLevelFilter("INFO"))
stderr = logging.StreamHandler(sys.stderr)
stderr.setLevel(logging.WARNING)
logger.addHandler(stdout)
logger.addHandler(stderr)

logger.info('INFO')      # to stdout only
logger.error('ERROR') # to stderr
```

Lazy evaluation

```
import logging
def expensive_call():
    print("This was really expensive")
    return "Hi”

class LazyLog:
    def __init__(self, func, *args, **kwargs):
        self.func = func
        self.args = args
        self.kwargs = kwargs
    def __str__(self):
        return self.func(*self.args, **self.kwargs) or None

logging.basicConfig(level="INFO")
logging.debug(LazyLog(expensive_call))
logging.info(LazyLog(expensive_call))
```

Quiz time!

kahoot.it

Bloomberg

Links

Some useful links about the talk

- https://opensource.com/article/17/9/python-logging
- https://docs.python.org/3.6/howto/logging.html
- https://docs.python.org/3/howto/logging-cookbook.html
- https://github.com/python/cpython/blob/master/Lib/logging/__init__.py

Questions

TechAtBloomberg.com

Bloomberg