

Sans I/O

Programming

Alex Chan

@alexwlchan

they/them



Hi, I'm Alex! aka @alexwlchan



“very important celebrity” ~ Daniele, this morning



I'm a software developer building digital preservation systems at Wellcome



I'm trans, genderfluid, and I use “they/them” pronouns



Euston Square Station

FREE EXHIBITION
A free introduction to the history of medicine
wellcome collection

BRAINS
The Mind as Matter
29 March - 17 June
wellcome collection

FREE EXHIBITION
A free introduction to the history of medicine
wellcome collection

EUSTON UNDERPASS
Ring road A 501
(A 41) (A 40)
Kilburn
Marylebone
West End
Westminster
Holloway
Camden Town
E 400

Food, books, gifts and comfortable seating inside... as well as free exhibitions, events and library. Curious? Come in. wellcome collection

Photo: Wellcome Collection



441

EPB Giant O/S
(Rare Books)
A1-J9 F.101 - F.363A

442

EPB Giant O/S
(Rare Books)
A1-B9 F.364 - F.433

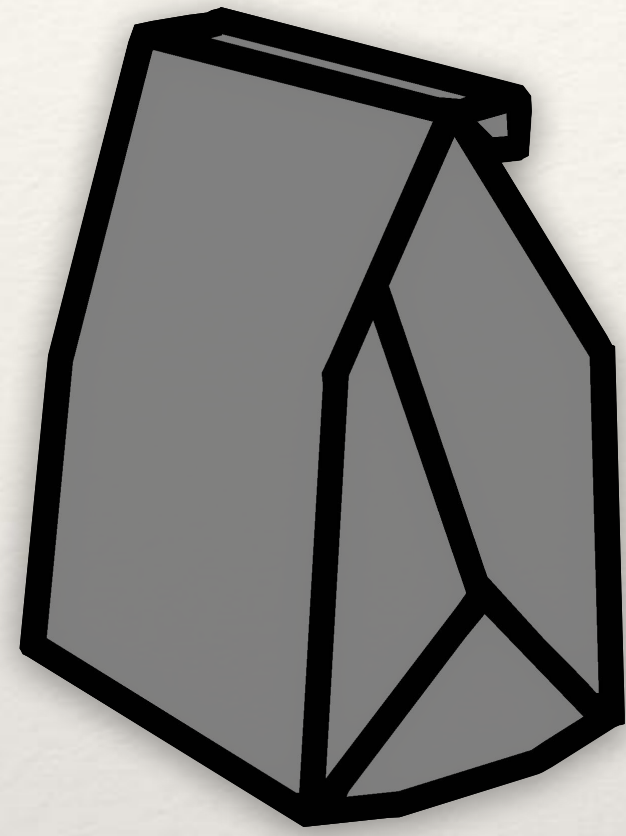
443

444

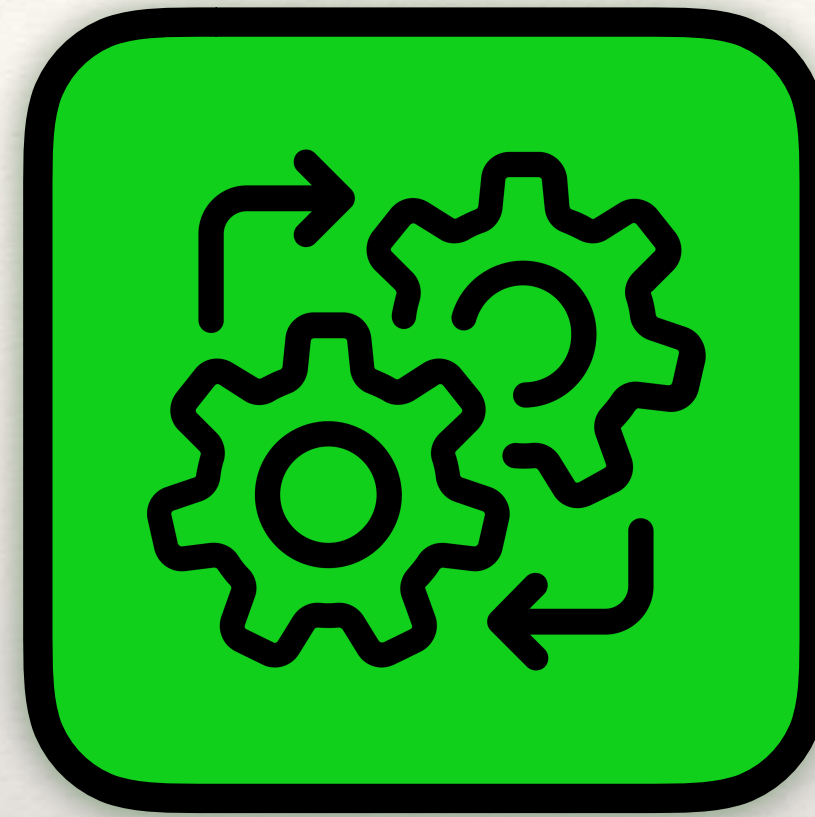
445

Photo: Wellcome Collection

Icons: Dorian Dawance, Alice Design
(The Noun Project), AWS Simple Icons



BagIt
package



Storage
service



Amazon S3

LibraryOfCongress/bagit-python

Work with Baglt packages from Python.

Updated on 5 Jul

● Python

★ 131

LibraryOfCongress/bagit-java

Java library to support the Baglt specification.

bagit

java

manifest

validation

checksum

archive

Updated 26 days ago 2 issues need help

● Java

★ 57

```
import bagit
```

```
bag = bagit.Bagit("/path/to/bag")
```

```
Path rootDir = Paths.get("/path/to/bag");
```

```
BagReader reader = new BagReader();
```

```
Bag bag = reader.read(rootDir);
```

We can't use these libraries
because they mix I/O
and parsing logic

http://

Wasted time and effort





Duplicate bugs

Harder to experiment



Harder to optimise your code



Mixing I/O and business logic causes (some) problems:

- No code reuse
- Wasted time and effort
- Duplicate bugs
- Less room for experiments
- Harder to optimise your code


How do we fix
this problem?

Never do I/O.

How do we fix
this problem?

**Never mix I/O
and business logic.**

Create an “I/O sandwich”



The diagram illustrates an 'I/O sandwich' structure. It consists of three main components arranged horizontally. On the left is a green, rounded rectangular shape with a concave left edge and a convex right edge, containing the text 'Inbound I/O'. In the center is a larger, bright green rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Business logic'. On the right is another green, rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Outbound I/O'. The three shapes are positioned such that their curved edges interlock, creating a sandwich-like appearance.

Inbound
I/O

Business
logic

Outbound
I/O

Build a toolbox of I/O implementations



This isn't everything,
but it's a good start.

**What are the
benefits?**

You get *much* simpler code



Your code is easier to test...



...and test at 100% coverage

100
==

===== 1417 passed in 32.48 seconds =====

py37 run-test: commands[1] | coverage report

Name	Stmts	Miss	Branch	BrPart	Cover	Missing

h2/__init__.py	2	0	0	0	100%	
h2/config.py	38	0	6	0	100%	
h2/connection.py	600	0	162	0	100%	
h2/errors.py	23	0	0	0	100%	
h2/events.py	140	0	8	0	100%	
h2/exceptions.py	45	0	0	0	100%	
h2/frame_buffer.py	52	0	18	0	100%	
h2/settings.py	119	0	38	0	100%	
h2/stream.py	440	0	90	0	100%	
h2/utilities.py	219	0	124	0	100%	
h2/windows.py	36	0	12	0	100%	

TOTAL	1714	0	458	0	100%	

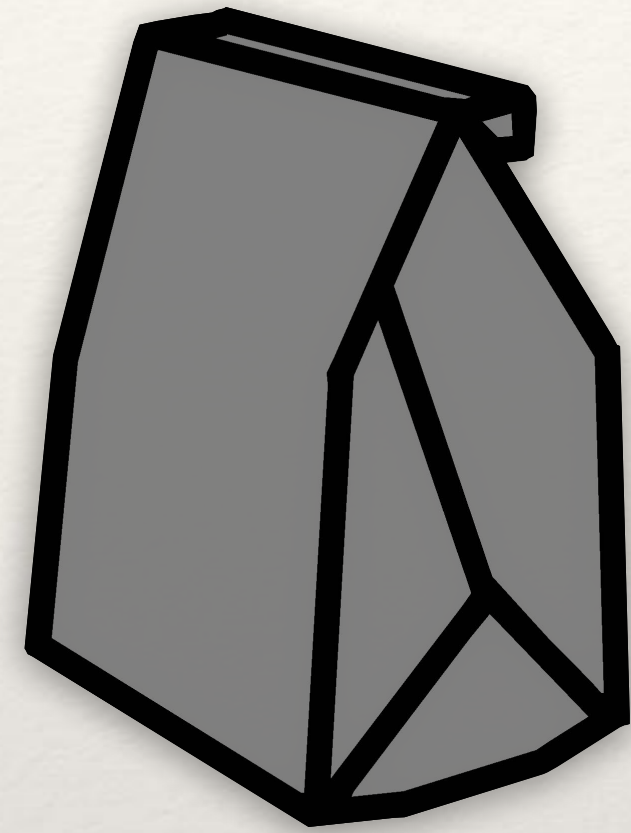
You have less bugs



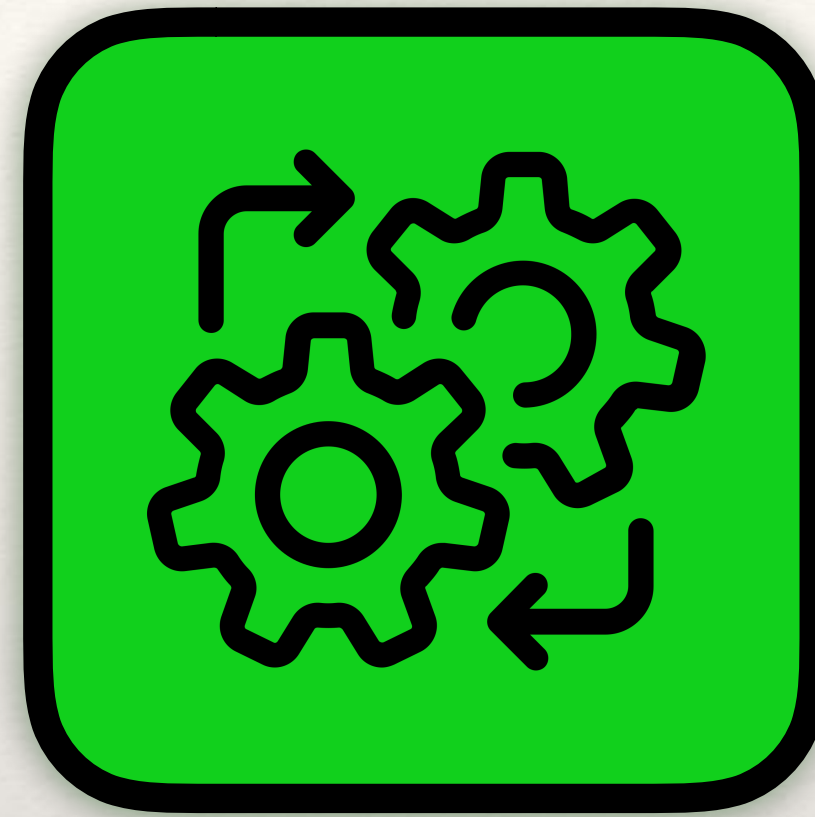
Your code is easier to reuse



Icons: Dorian Dawance, Alice Design
(The Noun Project), AWS Simple Icons



BagIt
package



Storage
service

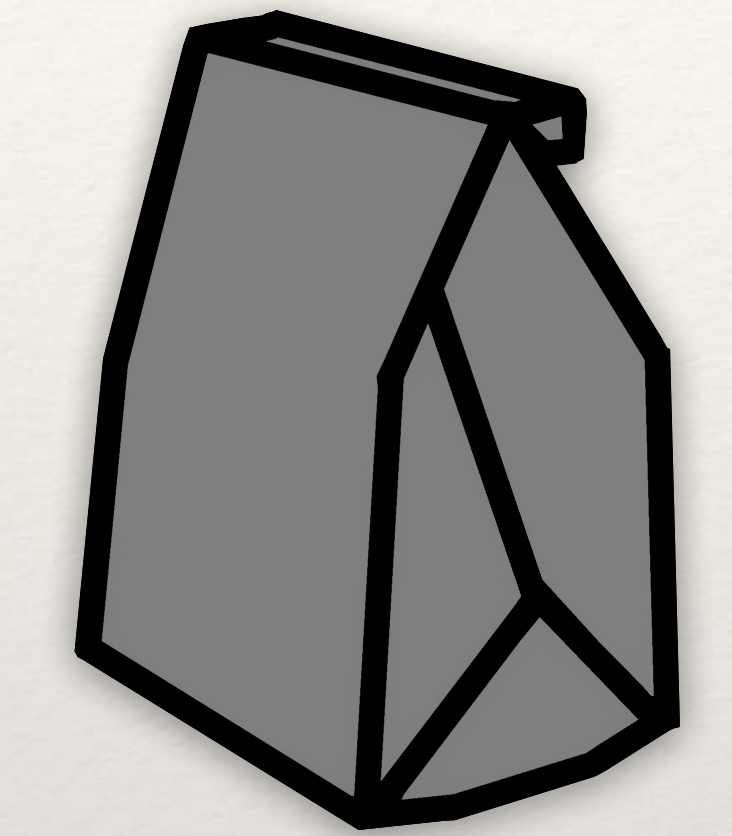


Amazon S3

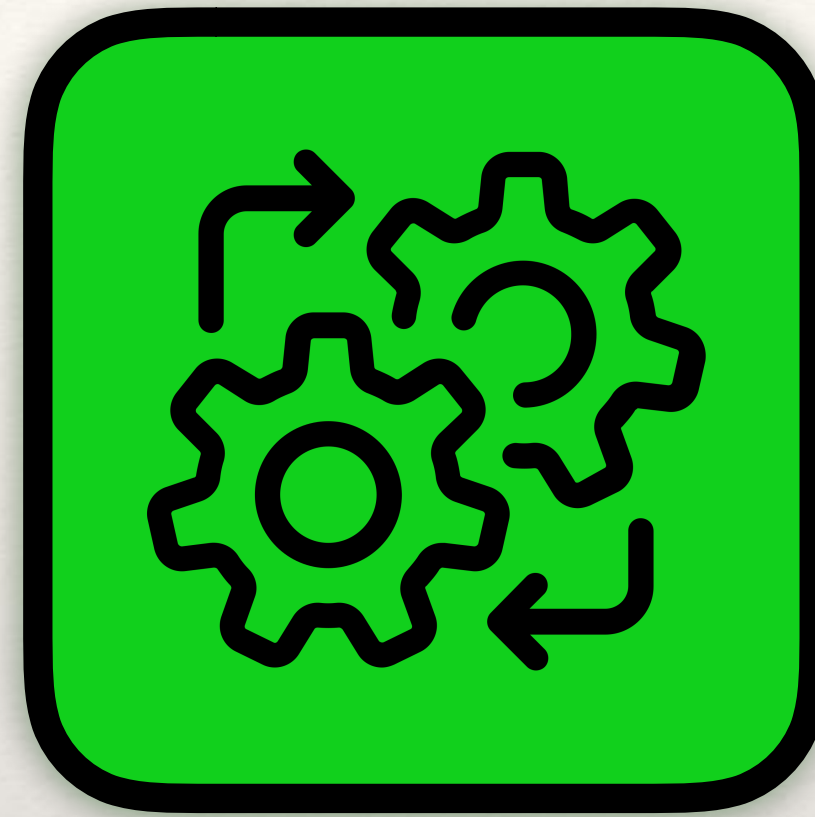
<https://thenounproject.com/term/paper-bag/28579>

<https://thenounproject.com/term/process/2473979>

Icons: Dorian Dawance, Alice Design
(The Noun Project), AWS Simple Icons



BagIt
package



Storage
service



Amazon S3




Azure Blob Storage

The benefits of separating I/O and business logic:

- Much simpler code
- Your code is easier to test (at 100% coverage!)
- You have less bugs
- Easier to reuse code

Create an “I/O sandwich”



The diagram illustrates an 'I/O sandwich' structure. It consists of three main components arranged horizontally. On the left is a green, rounded rectangular shape with a concave left edge and a convex right edge, containing the text 'Inbound I/O'. In the center is a larger, bright green rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Business logic'. On the right is another green, rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Outbound I/O'. The three shapes are positioned such that their curved edges interlock, creating a sandwich-like structure.

Inbound
I/O

Business
logic

Outbound
I/O

Examples!

Example:

**Creating an HTTP/2 server
with hyper-h2**


```
import h2.connection

conn = h2.connection.H2Connection(
    client_side=False
)

conn.initiate_connection()
out_bytes = conn.data_to_send()
```

Example:

**Calling the Slack API
with *slack-sansio***

```
import slack.methods

query = (
    slack.methods.CHAT_POST_MESSAGE,
    {"channel": "CM579FC82", "text": "Hello, PyCon UK!"}
)
```

```
import slack.methods
```

```
query = (  
    slack.methods.CHAT_POST_MESSAGE,  
    {"channel": "CM579FC82", "text": "Hello, PyCon UK!"}  
)
```

```
import slack.io.abc
```

```
class SlackAPIWrapper(abc.SlackAPI):  
    async def _request(self, method, url, headers, body):  
        # implementation goes here
```

```
import slack.methods
```

```
query = (  
    slack.methods.CHAT_POST_MESSAGE,  
    {"channel": "CM579FC82", "text": "Hello, PyCon UK!"}  
)
```

```
import requests
```

```
from slack.io.requests import SlackAPI
```

```
session = requests.Session()
```

```
client = SlackAPI(token="...", session=session)
```

```
client.query(*query)
```

```
import slack.methods
```

```
query = (  
    slack.methods.CHAT_POST_MESSAGE,  
    {"channel": "CM579FC82", "text": "Hello, PyCon UK!"}  
)
```

```
import asks
```

```
import curio
```

```
from slack.io.curio import SlackAPI
```

```
session = asks.Session()
```

```
client = SlackAPI(token="...", session=session)
```

```
curio.run(client.query(*query))
```

```
import slack.methods
```

```
query = (  
    slack.methods.CHAT_POST_MESSAGE,  
    {"channel": "CM579FC82", "text": "Hello, PyCon UK!"}  
)
```

```
import aiohttp, asyncio  
from slack.io.aiohttp import SlackAPI
```

```
loop = asyncio.get_event_loop()  
session = aiohttp.ClientSession(loop=loop)  
client = SlackAPI(token="...", session=session)
```

```
loop.run_until_complete(client.query(*query))
```

Example:

Refactoring a function in bagit-python

LibraryOfCongress/bagit-python

Work with Baglt packages from Python.

Updated on 5 Jul

● Python

★ 131

LibraryOfCongress/bagit-java

Java library to support the Baglt specification.

bagit

java

manifest

validation

checksum

archive

Updated 26 days ago 2 issues need help


● Java

★ 57

```
def load_tag_file(tag_file_name):  
    with open_text_file(tag_file_name) as tag_file:  
        tags = {}  
        for name, value in _parse_tags(tag_file):  
            # ... code omitted  
            # ... store (name, value) in tags  
  
        return tags
```

```
def load_tag_file(tag_file_name):  
    with open_text_file(tag_file_name) as tag_file:  
        return load_tag_file_from_lines(tag_file)  
  
def load_tag_file_from_lines(lines):  
    tags = {}  
    for name, value in _parse_tags(lines):  
        # ... code omitted  
        # ... store (name, value) in tags  
  
    return tags
```

Create an “I/O sandwich”



The diagram illustrates an 'I/O sandwich' structure. It consists of three main components arranged horizontally. On the left is a green, rounded rectangular shape with a concave left edge and a convex right edge, containing the text 'Inbound I/O'. In the center is a larger, bright green rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Business logic'. On the right is another green, rounded rectangular shape with a convex left edge and a concave right edge, containing the text 'Outbound I/O'. The three shapes are positioned such that their curved edges interlock, creating a sandwich-like structure.

Inbound
I/O

Business
logic

Outbound
I/O

Sans I/O Programming

Alex Chan
@alexwlchan
they/them