Elements of Professional Python

Brian Welch

About me

- CTF player
- Find my projects at https://brianwel.ch

Before we begin

- Assuming some familiarity with Python
- All code samples assume a CPython 3.8 environment
- Dunder methods
  - __*init*__ == “dunder init”
  - __*new*__ == “dunder new”
  - etc.

Agenda

- Type Hinting
- Linting
- Avoiding Boilerplate
- Standard Library Hidden Gems

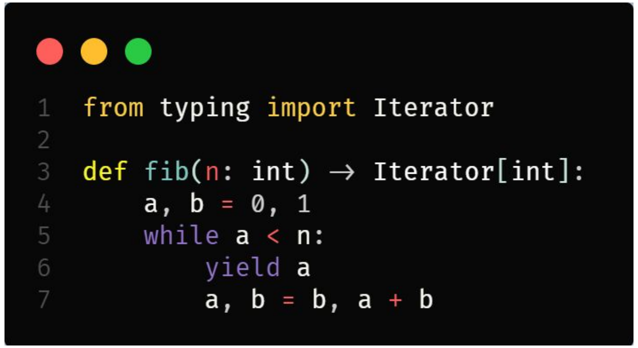
Overview

- Two ideas
  - Managing the quality of your code
  - Writing clearer, more Pythonic code
- Tools and libraries can make this easy

Type Hinting

PEP 484 -- Type Hints

- PEP 484 brought type hints to the Python language syntax
- Not enforced at runtime
- Leads to easier-to-read code
- The basis for type theory based static analysis tools
- Surprisingly expressive: support for nominative and structural subtyping

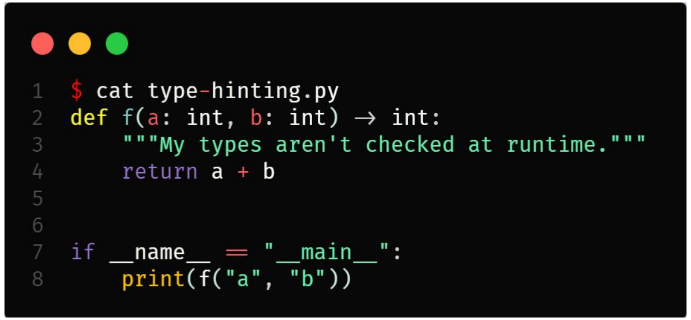


```python
from typing import Iterator

def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a + b
```

Type hinting example from the Mypy project’s readme

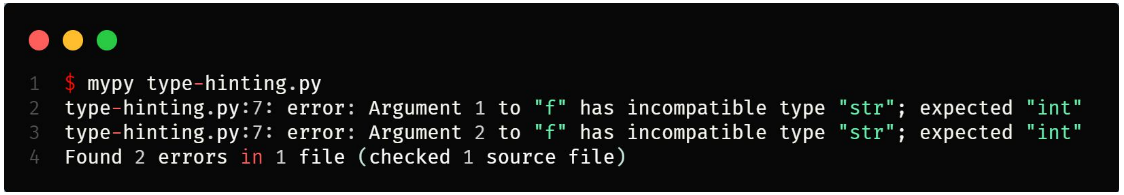
Type Checking with Mypy

- Mypy: a static type checker for Python
- Under active development, but mostly mature and robust
- Other options:
    - Pyright from Microsoft
    - Pyre from Facebook



```
$ cat type-hinting.py
def f(a: int, b: int) → int:
    """My types aren't checked at runtime."""
    return a + b


if __name__ == "__main__":
    print(f("a", "b"))
```

A toy example that uses unexpected types



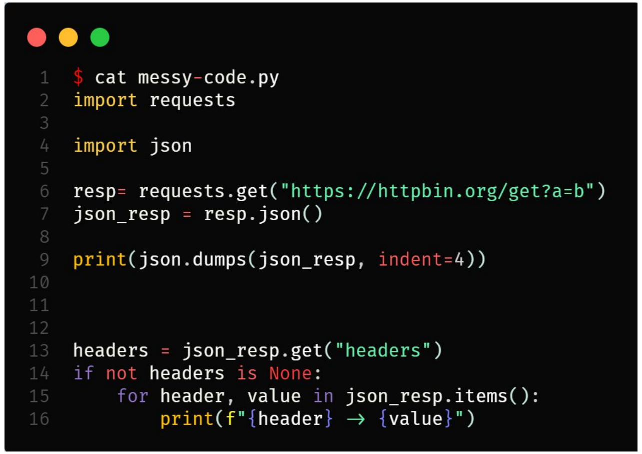
```
$ mypy type-hinting.py
type-hinting.py:7: error: Argument 1 to "f" has incompatible type "str"; expected "int"
type-hinting.py:7: error: Argument 2 to "f" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Mypy detects our invalid types through static analysis

Linting

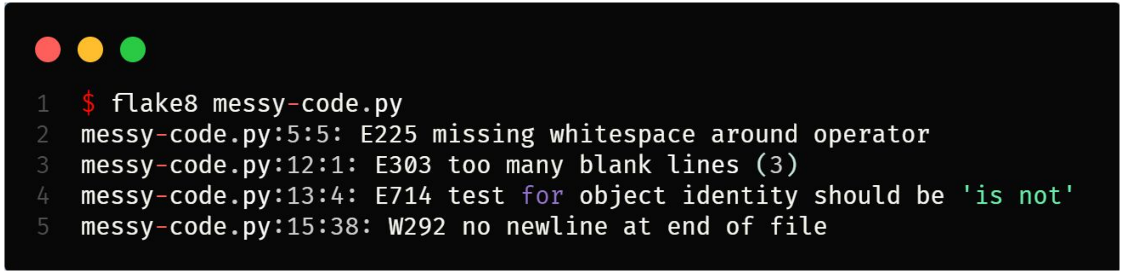
PEP 8 -- Style Guide for Python

- PEP8: The original Python style guide (almost 20 years old now)
- How can we check compliance with these standards?
  - flake8 -- a linter that combines stylistic and logical analysis
  - Pylint -- an alternative to flake8
  - Bandit -- checks for common security flaws



```
$ cat messy-code.py
import requests

import json

resp= requests.get("https://httpbin.org/get?a=b")
json_resp = resp.json()

print(json.dumps(json_resp, indent=4))



headers = json_resp.get("headers")
if not headers is None:
    for header, value in json_resp.items():
        print(f"{header} → {value}")
```

Some non-PEP8-compliant Python code

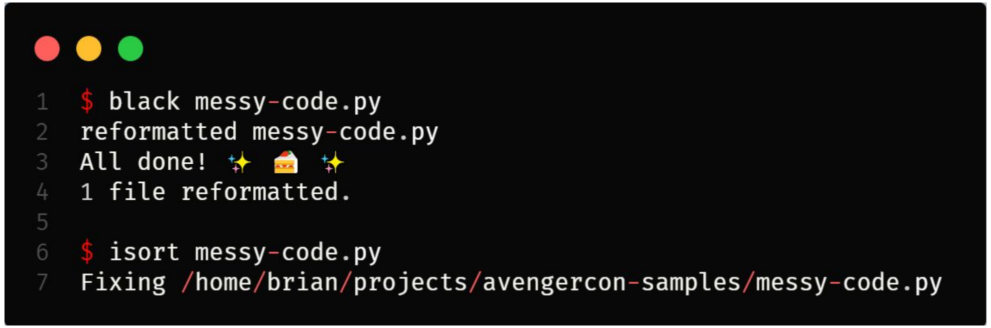


```
$ flake8 messy-code.py
messy-code.py:5:5: E225 missing whitespace around operator
messy-code.py:12:1: E303 too many blank lines (3)
messy-code.py:13:4: E714 test for object identity should be 'is not'
messy-code.py:15:38: W292 no newline at end of file
```

Errors printed by running flake8 over our example messy code from the left image

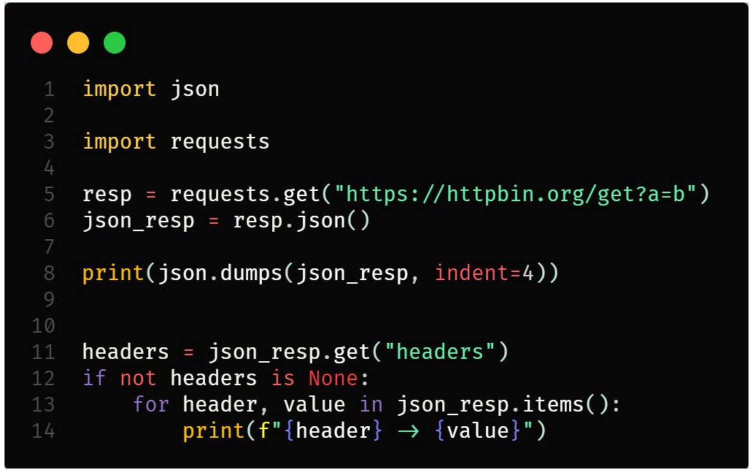
Automate PEP8 Compliance

- `black` -- automatically format your code
- `isort` -- automatically sort your import statements
- Similar idea to `go fmt`



```
$ black messy-code.py
reformatted messy-code.py
All done! ✨ 🍰 ✨
1 file reformatted.

$ isort messy-code.py
Fixing /home/brian/projects/avengercon-samples/messy-code.py
```

Example invocations of the `black` and `isort` command-line tools



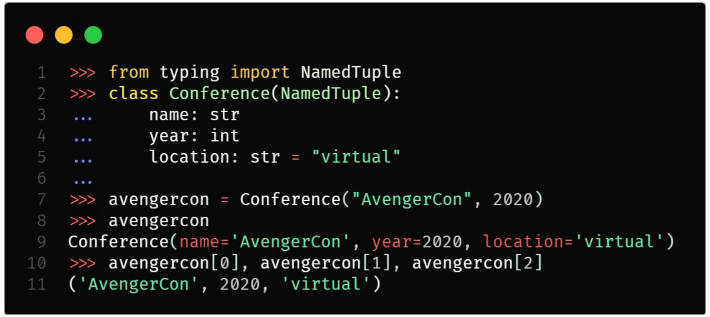
```python
import json

import requests

resp = requests.get("https://httpbin.org/get?a=b")
json_resp = resp.json()

print(json.dumps(json_resp, indent=4))


headers = json_resp.get("headers")
if not headers is None:
    for header, value in json_resp.items():
        print(f"{header} → {value}")
```

Our previous code sample, now PEP8-compliant

Avoiding Boilerplate

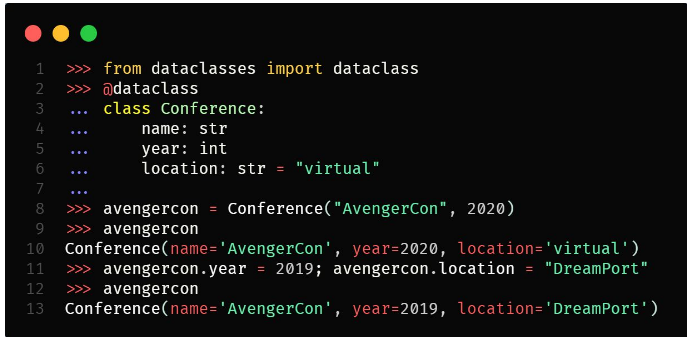
Quick and Easy Objects

- Don't let dictionaries be part of your object model
- `namedtuple`
  - Good for cleaning up existing code
  - Gives structure to things that would otherwise be raw tuples
- `dataclasses`
  - Removes the need for writing boilerplate `__init__` methods
  - Gives you lots of other dunder methods for free



```
>>> from typing import NamedTuple
>>> class Conference(NamedTuple):
...     name: str
...     year: int
...     location: str = "virtual"
...
>>> avengercon = Conference("AvengerCon", 2020)
>>> avengercon
Conference(name='AvengerCon', year=2020, location='virtual')
>>> avengercon[0], avengercon[1], avengercon[2]
('AvengerCon', 2020, 'virtual')
```

A simple class definition using `namedtuple`

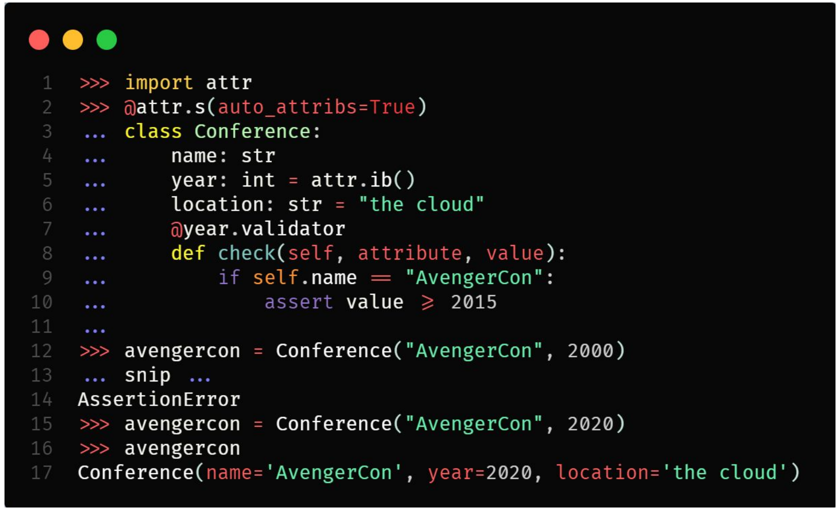


```
>>> from dataclasses import dataclass
>>> @dataclass
... class Conference:
...     name: str
...     year: int
...     location: str = "virtual"
...
>>> avengercon = Conference("AvengerCon", 2020)
>>> avengercon
Conference(name='AvengerCon', year=2020, location='virtual')
>>> avengercon.year = 2019; avengercon.location = "DreamPort"
>>> avengercon
Conference(name='AvengerCon', year=2019, location='DreamPort')
```

A simple class definition using `dataclass`

Quick and Easy Objects (cont.)

- `attrs`
  - Third-party option for quick Python objects
  - Great mypy integration
  - Pre-dates standard library options
  - Includes other features like validators
- Main advantages over `dataclasses`:
  - Validators
  - Converters
  - More control over `__slots__`
  - Not a part of the standard library



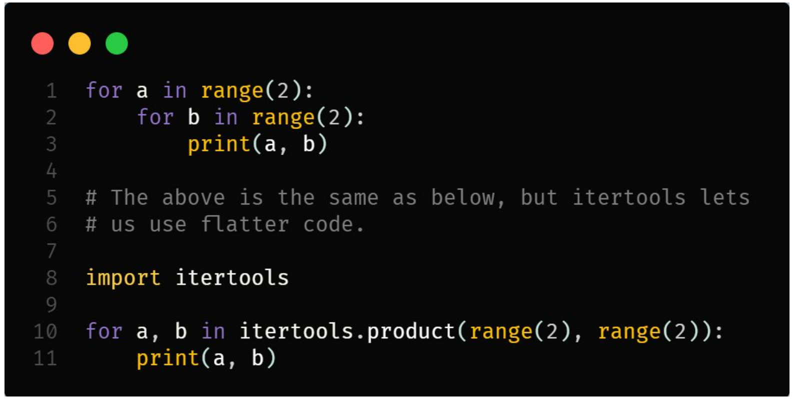
```
>>> import attr
>>> @attr.s(auto_attribs=True)
... class Conference:
...     name: str
...     year: int = attr.ib()
...     location: str = "the cloud"
...     @year.validator
...     def check(self, attribute, value):
...         if self.name == "AvengerCon":
...             assert value >= 2015
...
>>> avengercon = Conference("AvengerCon", 2000)
... snip ...
AssertionError
>>> avengercon = Conference("AvengerCon", 2020)
>>> avengercon
Conference(name='AvengerCon', year=2020, location='the cloud')
```

A simple class definition using the `attrs` library

Standard Library Hidden Gems

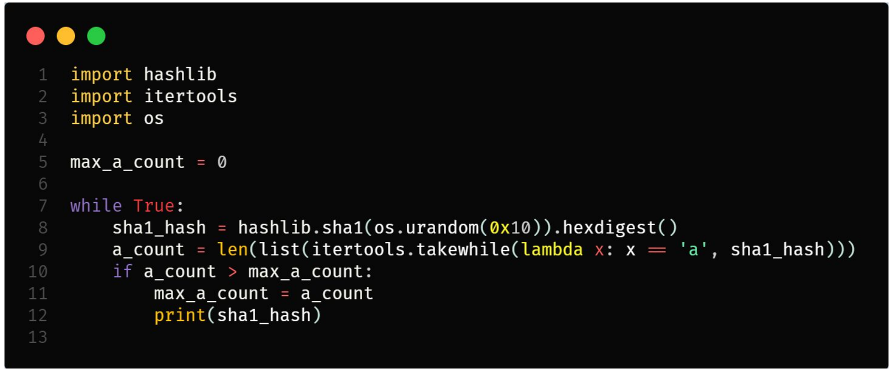
itertools

- itertools -- functions for creating efficient iterators
- Works well with functional programming styles
- Operates on iterators, rather than in-memory sequences
- Highlights: product, takewhile, tee, chain, and many more



```python
for a in range(2):
    for b in range(2):
        print(a, b)

# The above is the same as below, but itertools lets
# us use flatter code.

import itertools

for a, b in itertools.product(range(2), range(2)):
    print(a, b)
```

Using itertools.productto flatten nested loops

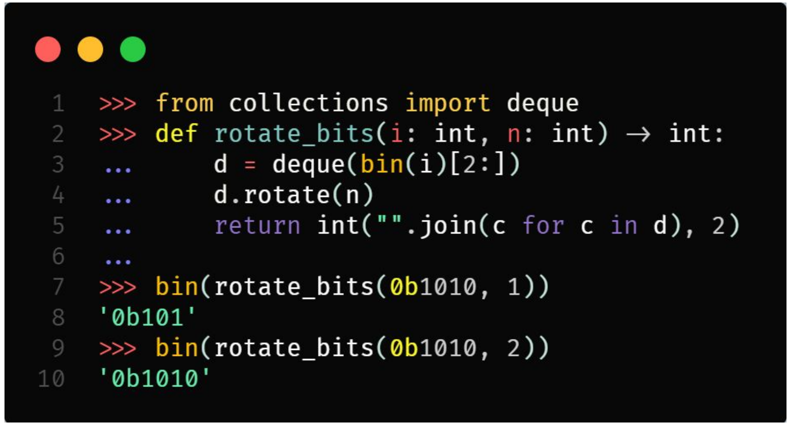


```python
import hashlib
import itertools
import os

max_a_count = 0

while True:
    sha1_hash = hashlib.sha1(os.urandom(0x10)).hexdigest()
    a_count = len(list(itertools.takewhile(lambda x: x == 'a', sha1_hash)))
    if a_count > max_a_count:
        max_a_count = a_count
        print(sha1_hash)

```

Using itertools.takewhileto find SHA1 hashes starting with consecutive "a" hex characters

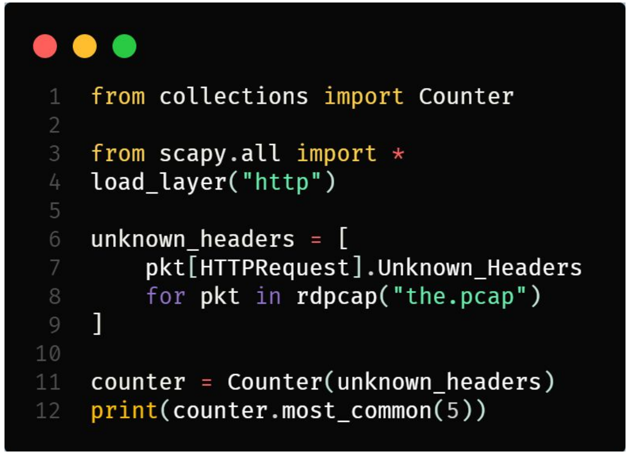
collections

- collections -- performant container types
- Some highlights include:
    - deque -- double-ended queue implementation
    - Counter -- count frequency of elements in sequences



```
>>> from collections import deque
>>> def rotate_bits(i: int, n: int) → int:
...     d = deque(bin(i)[2:])
...     d.rotate(n)
...     return int("".join(c for c in d), 2)
...
>>> bin(rotate_bits(0b1010, 1))
'0b101'
>>> bin(rotate_bits(0b1010, 2))
'0b1010'
```

Using a `deque` to rotate the bits in an integer

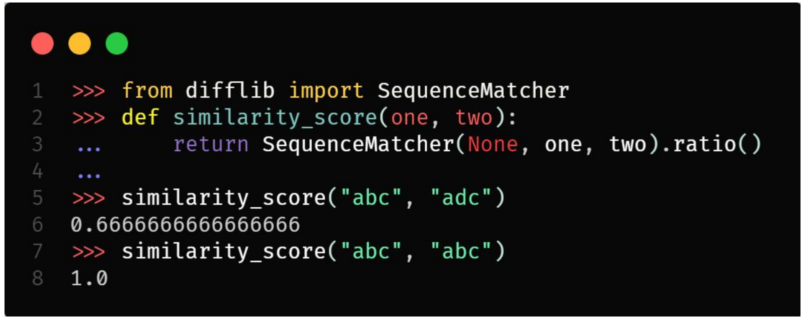


```
from collections import Counter

from scapy.all import *
load_layer("http")

unknown_headers = [
    pkt[HTTPRequest].Unknown_Headers
    for pkt in rdpcap("the.pcap")
]

counter = Counter(unknown_headers)
print(counter.most_common(5))
```

Using `Counter` to find most common headers in a packet capture

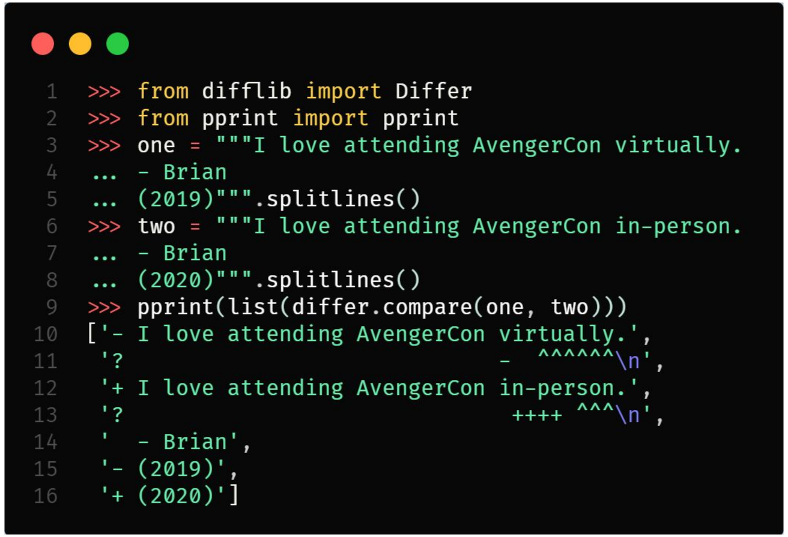
`difflib`

- `difflib` -- utilities for comparing textual sequences
- Lets you do things like:
  - Get the similarity score of two sample texts
  - Create patch diffs for bodies of text



```
>>> from difflib import SequenceMatcher
>>> def similarity_score(one, two):
...     return SequenceMatcher(None, one, two).ratio()
...
>>> similarity_score("abc", "adc")
0.6666666666666666
>>> similarity_score("abc", "abc")
1.0
```

Example of computing the similarity ratio between two short text samples



```
>>> from difflib import Differ
>>> from pprint import pprint
>>> one = """I love attending AvengerCon virtually.
... - Brian
... (2019)""".splitlines()
>>> two = """I love attending AvengerCon in-person.
... - Brian
... (2020)""".splitlines()
>>> pprint(list(differ.compare(one, two)))
['- I love attending AvengerCon virtually.',
 '?                             -  ^^^^^^\n',
 '+ I love attending AvengerCon in-person.',
 '?                              ++++ ^^^\n',
 '  - Brian',
 '- (2019)',
 '+ (2020)']
```

Example of finding the differences in two sample texts

Thanks for your time

Find these slides at
https://slides.brianwel.ch/elements-prof-python