Typing Software Heritage with MyPy

. . . the codebase, not the archive!

Stefano Zacchiroli

Software Heritage

20 November 2019
Software Heritage, Inria
Paris, France

Outline

1. Static typing with MyPy
2. Typing the Software Heritage codebase
3. Tips, tricks, FAQs, . . .
4. Outlook

Outline

Credits

Reusing intro slides shamelessly from:

Diego Muñoz
Python static typing with MyPy
Python Madrid Meetup, 2018
https://slides.kartones.net/027.html


Type Hints

Type Hints

- For Python 3

```
from typing import Union


def my_method(an_integer: int, a_float: float) -> Union[None, float]:
    if a_float == 0:
        return None
    the_result: float = an_integer / a_float
    return the_result
```

Stub files

- `.pyi` files
- Code you cannot annotate directly (e.g. external library)
- `stubgen <module>`

```
from typing import Any, List
from django.views.generic import View


class MyLibView(object):

   http_method_names: List

   def __init__(self, **kwargs: Any) -> None: ...

   def as_view(self, cls: View, **initkwargs: Any) -> View: ...

   def sum_values(self, first: int, second: int) -> int: ...
```

- Primitives

```
from typing import Union


def my_method(an_integer: int, a_float: float) -> Union[None, float]:
    if a_float == 0:
        return None
    the_result: float = an_integer / a_float
    return the_result
```

- Unknown type

```
from typing import Any


class MyDjangoView(View):
   def post(self, request: WSGIRequest, **kwargs: Any) -> JsonResponse:
       pass
```

- Lists and Tuples

```
from typing import List, Tuple


def numeric_tuple_to_list(source_tuple: Tuple) -> List:
    return [item for item in source_tuple]


def numeric_tuple_to_list(source_tuple: Tuple[int]) -> List[int]:
    return [item for item in source_tuple]
```

- Dictionaries

```python
from typing import Dict


def my_method(input_data: Dict) -> None:
    for key, value in input_data.items():
        print(f"{key}:{value}")


def my_method_2(input_data: Dict[str, int]) -> None:
    for key, value in input_data.items():
        print(f"{key}:{value}")
```

- Multiple types

```
from typing import Union


def my_method(an_integer: int, a_float: float) -> Union[None, float]:
    if a_float == 0:
        return None
    the_result: float = an_integer / a_float
    return the_result
```

- “Nullable type” (type or `None`)

```python
from typing import Optional


def my_method(an_integer: int, a_float: float) -> Optional[float]:
    if a_float == 0:
        return None
    the_result: float = an_integer / a_float
    return the_result
```

Python 3

- Types can also be specified

```
from typing import Type


def validate_type(a_vehicle_class: Type) -> bool:
    return a_vehicle_class in [Car, Truck]
```

Python 3
- Iterators

```
from typing import Iterator


def find_items() -> Iterator[MyClass]:
    # ...
    yield item
```

Python 3

- Function/Method handlers

```python
from typing import Callable


def my_method1(func: Callable, value: int) -> int:
    return func(value)


def my_method2(func: Callable[[int], int], value: int) -> int:
    return func(value)
```

Python 3
- Type aliases

```
from typing import List

Vector = List[float]
```


MyPy

Annotations not enforced by Python

Left for type checkers to use (**gradual typing)**

Static type checker tool

How to run

- From command line

```
mypy myproj

myproj/models.py:46: error: Function is missing a type annotation for one or
more arguments

myproj/models.py:54: error: Unsupported operand types for >= ("int" and "str")

myproj/tests/models_test.py:253: error: Argument 2 to "my_method" of "MyClass"
has incompatible type "int"; expected "str"
```

Outline

Goal

Minimal Viable Product

- make “mypy swh” pass in every `swh-environment` repository...
- ...with no or minimal type annotations
  - i.e., “only” avoid blatant type errors
- integrate with CI
  - i.e., create a regression-free baseline for incrementally annotating our codebase in the medium/long term

Developer interface

```
cd swh-environment/swh-foo
```

`make typecheck` only run mypy

`make check` will *also* do the above (in addition to `flake8` & `codespell`)

`tox -e mypy` like `make typecheck`, but in `tox`

`tox` all of it + tests, but in `tox`

Current status

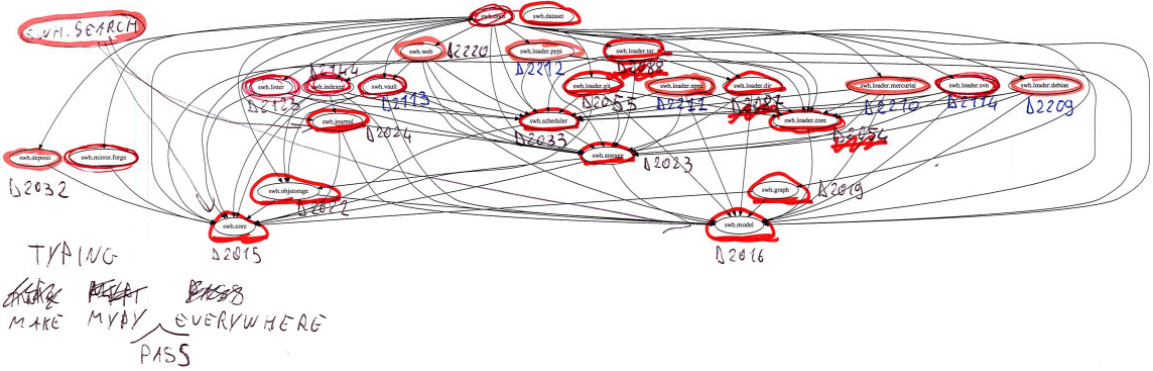
```
~/swh-environment $ make typecheck
make -C swh-core typecheck
make[1]: Entering directory '/home/zack/swh-environment/swh-core'
mypy swh
Success: no issues found in 41 source files
make[1]: Leaving directory '/home/zack/swh-environment/swh-core'

[...]

make[1]: Entering directory '/home/zack/swh-environment/swh-web'
Makefile.local:97: warning: overriding recipe for target 'check-mypy'
../Makefile.python:39: warning: ignoring old recipe for target 'check-mypy'
DJANGO_SETTINGS_MODULE=swh.web.settings.development mypy swh
Success: no issues found in 120 source files
make[1]: Leaving directory '/home/zack/swh-environment/swh-web'

~/swh-environment $ echo $?
0
```

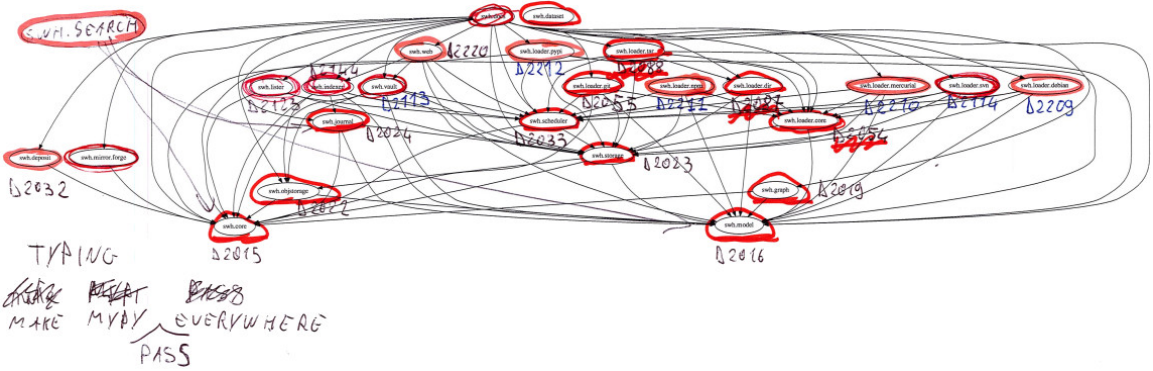
The journey



Elapsed (wall clock) time: 1.5 months

- D2015 (swh-core): submitted 2019-09-20
- D2230 (swh-search): closed 2019-11-07

The journey



Elapsed (wall clock) time: 1.5 months

- D2015 (swh-core): submitted 2019-09-20
- D2230 (swh-search): closed 2019-11-07

Source tree layout changes

```
diff --git a/mypy.ini b/mypy.ini
new file mode 100644
index 0000000..51a26dc
--- /dev/null
+++ b/mypy.ini
@@ -0,0 +1,18 @@
+[mypy]
+namespace_packages = True
+warn_unused_ignores = True
+
+# 3rd party libraries without stubs (yet)
+
+[mypy-pkg_resources.*]
+ignore_missing_imports = True
+
+[mypy-pytest.*]
+ignore_missing_imports = True
```

...

(more on this later)

Source tree layout changes (cont.)

```
diff --git a/.gitignore b/.gitignore
index b0e734a..3c564c7 100644
--- a/.gitignore
+++ b/.gitignore
@@ -9,3 +9,4 @@ build/
 dist/
 version.txt
 .tox
+.mypy_cache/
```

Source tree layout changes (cont.)

```
diff --git a/swh/search/py.typed b/swh/search/py.typed
new file mode 100644
index 0000000..1242d43
--- /dev/null
+++ b/swh/search/py.typed
@@ -0,0 +1 @@
+# Marker file for PEP 561.

diff --git a/MANIFEST.in b/MANIFEST.in
index 1271b63..8c63402 100644
--- a/MANIFEST.in
+++ b/MANIFEST.in
@@ -3,3 +3,4 @@ include requirements.txt
 include requirements-swh.txt
 include version.txt
 include README.md
+recursive-include swh py.typed
```

Source tree layout changes (cont.)

```
diff --git a/swh/search/py.typed b/swh/search/py.typed
new file mode 100644
index 0000000..1242d43
--- /dev/null
+++ b/swh/search/py.typed
@@ -0,0 +1 @@
+# Marker file for PEP 561.

diff --git a/MANIFEST.in b/MANIFEST.in
index 1271b63..8c63402 100644
--- a/MANIFEST.in
+++ b/MANIFEST.in
@@ -3,3 +3,4 @@ include requirements.txt
 include requirements-swh.txt
 include version.txt
 include README.md
+recursive-include swh py.typed
```

Source tree layout changes (cont.)

```
diff --git a/tox.ini b/tox.ini
index 7f7bc76..1c535aa 100644
--- a/tox.ini
+++ b/tox.ini
@@ -1,5 +1,5 @@
 [tox]
-envlist=flake8,py3
+envlist=flake8,mypy,py3

@@ -16,3 +16,11 @@ deps =
+
+[testenv:mypy]
+skip_install = true
+deps =
+  .[testing]
+  mypy
+commands =
+  mypy swh
```

Source tree layout changes (cont.)

```
diff --git a/swh/__init__.py b/swh/__init__.py
index 69e3be5..f14e196 100644
--- a/swh/__init__.py
+++ b/swh/__init__.py
@@ -1 +1,4 @@
-__path__ = __import__('pkgutil').extend_path(__path__, __name__)
+from pkgutil import extend_path
+from typing import Iterable
+
+__path__: Iterable[str] = extend_path(__path__, __name__)
```

Bestiary of encountered issues

- missing django kwarg: D2031
- bytes/str mismatch in CRAN lister: D2124
- drop obsolete CLI alaises: D2013, D2000
- click_required=1: D1985, D1986, D1987, D1988, D1989
- missing import json, shadowed by unrelated NoQA marker: D2022
- import loop swh-storage↔swh-journal
- do not explode when TEST_DB_DUMP=None: D2012
- unused format string parameter in SQL query formatting: D2144
- actual dynamic typing of int/str/datetime in default_min_bound of github lister
  - ⇒ swept under the carpet with `type: Any` for now

Outline

When should modules be ignored?

Mypy looks for type information:

- in the `.py` files themselves
  - for third party modules this is so *provided that* the module declares itself to be typed shipping `py.typed` (PEP 561)
- in third party `.pyi` interface files
  - locally installed to annotate 3rd party libraries, e.g., `django-stubs`
  - available from typeshed `https://github.com/python/typeshed/`

```
swh/search/elasticsearch.py:9: error: No library stub file for module 'elasticsearch'
swh/search/elasticsearch.py:10: error: No library stub file for module 'elasticsearch.helpers'
```

⇒

```
diff --git a/mypy.ini b/mypy.ini
--- a/mypy.ini
+++ b/mypy.ini
+[mypy-elasticsearch.*]
+ignore_missing_imports = True
```

When should modules be ignored?

Mypy looks for type information:

- in the `.py` files themselves
  - for third party modules this is so *provided that* the module declares itself to be typed shipping `py.typed` (PEP 561)
- in third party `.pyi` interface files
  - locally installed to annotate 3rd party libraries, e.g., `django-stubs`
  - available from typeshed `https://github.com/python/typeshed/`

```
swh/search/elasticsearch.py:9: error: No library stub file for module 'elasticsearch'
swh/search/elasticsearch.py:10: error: No library stub file for module 'elasticsearch.helpers'
```

⇒

```
diff --git a/mypy.ini b/mypy.ini
--- a/mypy.ini
+++ b/mypy.ini
+[mypy-elasticsearch.*]
+ignore_missing_imports = True
```

Docstrings

Do not duplicate type information between docstrings and type hints, sphinx+napoleon will Do The Right Thing™.

```python
def __init__(self, record_size: int, fname: str, mode: str = 'rb',
             length: int = None):
    """open an existing on-disk map

    Args:
        record_size: size of each record in bytes
        fname: path to the on-disk map
        mode: file open mode, usually either 'rb' for read-only maps, 'wb' for
            creating new maps, or 'rb+' for updating existing ones (default: 'rb')
        length: map size in number of logical records; used to initialize
            writable maps at creation time. Must be given when mode is 'wb' and
            the map doesn't exist on disk; ignored otherwise
    """
```

C.f. https://docs.softwareheritage.org/devel/apidoc/swh.graph.pid.html#swh.graph.pid._OnDiskMap

Postel's law & type hints

- Q Should I return `Dict` or `Mapping`?
- Q Should I accept `Sequence` or `List`?

Definition (Robustness principle)

Be conservative in what you send, be liberal in what you accept.

⇓

Return concrete types, accept abstract base classes.

```python
from typing import Any, Dict, Mapping, Sequence
def main_loop(conf: Mapping[str, Any],
              targets: Sequence[str]) -> Dict[str, int]
    pass
```

- note: `Mapping`, `Sequence` (input) v. `Dict` (output)
- keep `https://docs.python.org/3/library/collections.abc.html` handy!

Postel's law & type hints

- Q Should I return `Dict` or `Mapping`?
- Q Should I accept `Sequence` or `List`?

Definition (Robustness principle)

Be conservative in what you send, be liberal in what you accept.

⇓

Return concrete types, accept abstract base classes.

```
from typing import Any, Dict, Mapping, Sequence
def main_loop(conf: Mapping[str, Any],
              targets: Sequence[str]) -> Dict[str, int]
    pass
```

- note: `Mapping`, `Sequence` (input) v. `Dict` (output)
- keep `https://docs.python.org/3/library/collections.abc.html` handy!

Postel's law & type hints

- Q Should I return `Dict` or `Mapping`?
- Q Should I accept `Sequence` or `List`?

Definition (Robustness principle)

Be conservative in what you send, be liberal in what you accept.

⇓

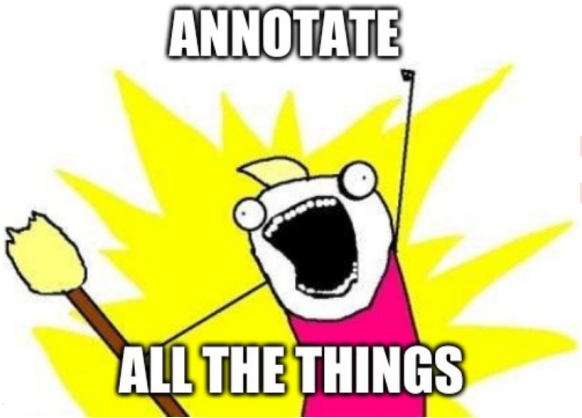
Return concrete types, accept abstract base classes.

```
from typing import Any, Dict, Mapping, Sequence
def main_loop(conf: Mapping[str, Any],
              targets: Sequence[str]) -> Dict[str, int]
    pass
```

- note: `Mapping`, `Sequence` (input) v. `Dict` (output)
- keep `https://docs.python.org/3/library/collections.abc.html` handy!

Outline

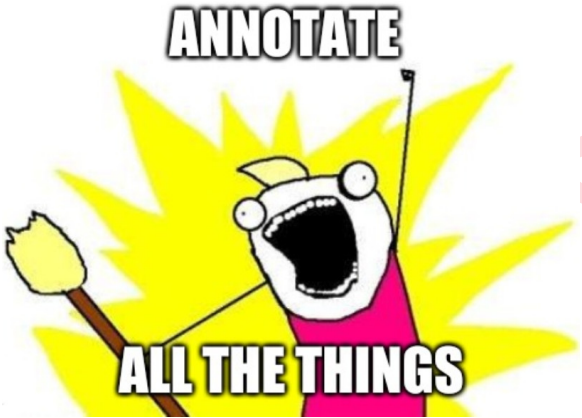
Next steps for typing



?

- if so: bottom-up, starting from core interfaces/functionalities and walking our way up towards external modules
- might be a good chance to design more abstract types/clear contracts for our architectural layers

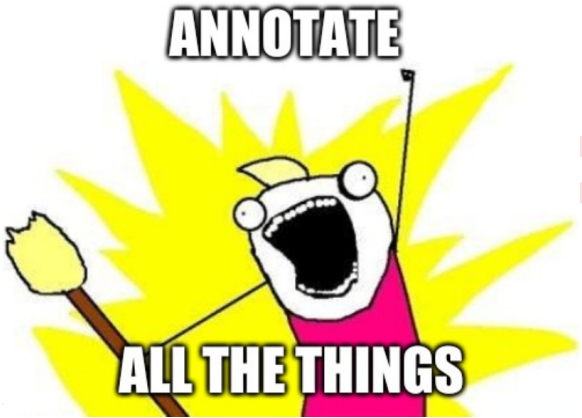
Next steps for typing



?

- if so: bottom-up, starting from core interfaces/functionalities and walking our way up towards external modules
- might be a good chance to design more abstract types/clear contracts for our architectural layers

Next steps for typing



?

- if so: bottom-up, starting from core interfaces/functionalities and walking our way up towards external modules
- might be a good chance to design more abstract types/clear contracts for our architectural layers

On doing cross-cutting changes to our codebase

- quite painful
- [short of monorepo] can we do better?
- unclear to me, some ideas
  - CI support for: “test this and all its reverse dependencies”
  - move CI/linting configuration *out* of individual packages, so that we have one central place where to change cross-cutting quality standards
    - price to pay: giving up on independent testability
  - non-uniform pre-commit rules is still a pain (WTH am I the only one hitting `codespell` issue?)
    - ⇒ go ahead with the switch to `https://pre-commit.com/` (T1881)

The good

- CI is a great help
- `git tag` to release is awesome

- (minimally) typing SWH with Mypy has been a positive experience—learned some stuff, squashed some bugs
- my take: we should now go ahead and (maximally) type SWH with Mypy
- it is complementary to current linting & testing and raises opportunities to define clear(er) invariants/contracts across our entire software stack

MyPy
Type hints cheat sheet
https://mypy.readthedocs.io/en/latest/cheat_sheet_py3.html

Dropbox
Our journey to type checking 4 million lines of Python
https://blogs.dropbox.com/tech/2019/09/our-journey-to-type-checking-4-million-lines-of-python/

Bernat Gabor
Type Hints: Inside the Snake Pit
https://gaborbernat.github.io/pycon-us-2019/

about the slides:
available at https://upsilon.cc/~zack/talks/2019/2019-11-20-swh-mypy.pdf