Static typing

February 19, 2026

1 Static typing

Python is a strongly typed and dynamically typed programming language.

Dynamic typing means that variables can refer to different types of objects at different times and that collections can contain references to different types of objects, and that these types can vary at runtime.

However, these properties are rarely used. Most variables don't change type over the lifetime of a program and most collections have uniform types.

For this reason it makes sense to include *type hints* in Python programs for readability reasons and for improved code quality. We eliminate a whole category of bugs if we can make a static evaluation of types in the places where we use operators. Please note that in Python, function calls are operators. It turns out that adding type hints to function signatures gives us the best return on investment.

Python implements **incremental typing**. This means that code without type hints is not checked. Only the places where we have added hints get checked. The various checking tools also make **type inferences** where types can be calculated.

There are several tools that do **static typechecking**. The best ones are currently:

- **pyright** - also known as **pylance**, when used as a pluging to VSCode -created by Microsoft
- **mypy** - created by Dropbox when Guido van Rossum worked there
- **pyrefly** - a typechecker from Facebook, implementation in Rust
- **ty** - a typechecker from Astral, implemented in Rust

https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

1.0.1 Pydantic

Typing information is available as metadata in the Python interpreter, when it executes a program. Some tools have started using this information to make the developers life easier. The most commonly used of these is Pydantic, which allows you to define a data storage layer by adding static typing information to the definition of the classes you want to store.

https://docs.pydantic.dev/latest/

[1]:
```
from datetime import datetime

from pydantic import BaseModel, PositiveInt
```

```python
class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]


external_data = {
    'id': 123,
    'signup_ts': '2019-06-01 12:22',
    'tastes': {
        'wine': 9,
        b'cheese': 7,
        'cabbage': '1',
    },
}

user = User(**external_data)

print(user.id)
#> 123
print(user.model_dump())
"""
{
    'id': 123,
    'name': 'John Doe',
    'signup_ts': datetime.datetime(2019, 6, 1, 12, 22),
    'tastes': {'wine': 9, 'cheese': 7, 'cabbage': 1},
}
"""
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[1], line 3
      1 from datetime import datetime
----> 3 from pydantic import BaseModel, PositiveInt
      6 class User(BaseModel):
      7     id: int

File ~/src/miniconda3/lib/python3.11/site-packages/pydantic/__init__.py:5
      2 from typing import TYPE_CHECKING
      3 from warnings import warn
----> 5 from ._migration import getattr_migration
      6 from .version import VERSION, _ensure_pydantic_core_version
      8 _ensure_pydantic_core_version()
```

```
File ~/src/miniconda3/lib/python3.11/site-packages/pydantic/_migration.py:4
      1 import sys
      2 from typing import Any, Callable
----> 4 from pydantic.warnings import PydanticDeprecatedSince20
      6 from .version import version_short
      8 MOVED_IN_V2 = {
      9     'pydantic.utils:version_info': 'pydantic.version:version_info',
     10     'pydantic.error_wrappers:ValidationError': 'pydantic:
 ↪ValidationError',
   (…)
     15     'pydantic.generics:GenericModel': 'pydantic.BaseModel',
     16 }

File ~/src/miniconda3/lib/python3.11/site-packages/pydantic/warnings.py:5
      1 """Pydantic-specific warnings."""
      3 from __future__ import annotations as _annotations
----> 5 from .version import version_short
      7 __all__ = (
      8     'PydanticDeprecatedSince20',
      9     'PydanticDeprecatedSince26',
   (…)
     18     'TypedDictExtraConfigWarning',
     19 )
     22 class PydanticDeprecationWarning(DeprecationWarning):

File ~/src/miniconda3/lib/python3.11/site-packages/pydantic/version.py:7
      3 from __future__ import annotations as _annotations
      5 import sys
----> 7 from pydantic_core import __version__ as __pydantic_core_version__
      9 __all__ = 'VERSION', 'version_info'
     11 VERSION = '2.12.5'

File ~/src/miniconda3/lib/python3.11/site-packages/pydantic_core/__init__.py:6
      3 import sys as _sys
      4 from typing import Any as _Any
----> 6 from typing_extensions import Sentinel
      8 from ._pydantic_core import (
      9     ArgsKwargs,
     10     MultiHostUrl,
   (…)
     29     to_jsonable_python,
     30 )
     31 from .core_schema import CoreConfig, CoreSchema, CoreSchemaType,␣
 ↪ErrorType
```

```
ImportError: cannot import name 'Sentinel' from 'typing_extensions' (/home/jaco /
↪.local/lib/python3.11/site-packages/typing_extensions.py)
```

Below is an example of how you can retreive the type of an object at runtime.

[ ]:
```
from typing import reveal_type
i = 1
reveal_type(i)
```