Mypy Cheatsheet

Introduction

`mypy` is a static type checker for Python. It combines the expressive power of dynamic typing with the reliability of static typing.

Basic Type Annotations

```
# Variables
x: int = 1
name: str = "Alice"
is_active: bool = True

# Functions
def add(x: int, y: int) -> int:
    return x + y

# Empty returns
def log(message: str) -> None:
    print(message)
```

Built-in Collections (Standard Types)

(For Python 3.9+, use built-in types directly; for older versions, import from `typing`)

```
# Lists
names: list[str] = ["Alice", "Bob"]
# Dictionaries
scores: dict[str, int] = {"Alice": 90,
"Bob": 85}
# Tuples
point: tuple[int, int] = (10, 20)
fixed: tuple[int, str, float] = (1,
"msg", 3.14)
# Sets
unique_ids: set[int] = {1, 2, 3}
```

Type Unions and Optional

```
from typing import Union, Optional

# Can be either type
def process(data: Union[int, str]) ->
None: ...

# Shorthand for Union[T, None]
def find_user(id: int) -> Optional[str]:
    # Returns a string or None
    ...
```

Classes and OOP

```
from typing import ClassVar

class BankAccount:
    def __init__(self, account_name:
str, initial_balance: int = 0) -> None:
        self.account_name = account_name
        self.balance = initial_balance

    def deposit(self, amount: int) ->
None:
        self.balance += amount

# Type-aware inheritance
class AuditedAccount(BankAccount):
    audit_log: list[str] = [] # Class
variable (shared)

    # ClassVar explicit annotation
    log_limit: ClassVar[int] = 100
```

Handling Confusion (Any and Cast)

```
from typing import Any, cast

# Avoid where possible; allows any
operation
x: Any = untyped_library_call()

# Forced override of type inference
(Static only, no runtime check)
a: list[int] = [1, 2, 3]
b = cast(list[float], a)

# Silence specific line errors
x = problematic_line() # type: ignore
```

Standard "Duck Types"

Use these for broader compatibility in function arguments.

```
from typing import Iterable, Sequence,
Mapping

# Anything that can be used in a 'for'
loop
def f(ints: Iterable[int]) -> None: ...

# Anything that supports 'len' and
'__getitem__'
def f(list_like: Sequence[int]) ->
None: ...

# Anything dict-like (read-only)
def f(map_like: Mapping[str, int]) ->
None: ...
```

Forward References

```
from __future__ import annotations #
Python 3.7+

# Refer to a class before it is defined
def process_node(node: Node) ->
None: ...

class Node: ...
```

Pro Tips

Reveal Type

Use `reveal_type(expression)` inside your code and run mypy to see what type is currently inferred. Remember to remove before running the code.

Type Checking Only Code

```
from typing import TYPE_CHECKING
if TYPE_CHECKING:
    # This block only executes during
static analysis
    from my_large_module import
ComplexType
```

Configuration

Manage global settings (strictness, ignore, etc.) in `pyproject.toml` or `mypy.ini`.

```
[tool.mypy]
python_version = "3.10"
strict = true
ignore_missing_imports = true
```