Typing

Or:
How
I Learned
To
Stop
Worrying
And
Accept
Python

Stephen Catsamas
25/08/2025

What is Typing?

≈ the type of a variable describes what data it holds

example types: `int`, `str`, `list`, `dict`

```
ok : int = 7

not_ok : int = “abc”

also_ok : str = “abc”
```

Types tell us more about how we can use a variable without knowing where they came from

Different Languages Have Different Type Systems

Strong vs Weak

Static vs Dynamic

Manifest vs Inferred

The study of type systems is a branch of mathematics called type theory

LEAN4: theorem prover and programming language

Propositions are types
Proofs are variables

Why Should I Use Typing?

Catch Errors

Reason Clearly

Document Code

Aid Collaboration

Guide Design

```
def add4(x):
     return x + 4

add4(1) # 5
add4(“abc”) # !TypeError
```

Implementation needed to decern valid inputs

```
def typed_add4(x : int) -> int:
     return x + 4
```

Obvious “abc” invalid input

Separates implementation from behaviour

Generic Types

```
int_list : list = [1,2,3]
str_list : list = ["a", "ab", "abc"]
int_str_list : list = [1,"ab",3]
```

These are all `lists` but the elements of the lists have different types

```
def endpoint_diff(l : list) -> int:
    start = l[0]
    end = l[-1]

    return end - start
```

TypeError for `str_list` and `int_str_list`

Generic Types

Using Generics:

```
def typed_endpoint_diff(l : list[int]) -> int:
    start = l[0]
    end = l[-1]

    return end - start
```

Other Examples:

```
dict[str, int]
list[list[int]]
```

Typing as Documentation

Ideally behaviour specified in:

1. Code implementation
2. Unit testing
3. Documentation

Docstrings don't affect program behaviour

This means it will be out of date

Typing moves more documentation into code

Much like good naming but better

Types enforce stronger function "contracts"

```
def pop6(l):
    return l[0:-6]

def remove_space_cube(faces : list[Faces]) -> list[Faces]:
    '''
    removes the faces defining the space cube from a list of faces
    Parameters
    ----------
    faces : List[Faces]
        a list of faces with space cube faces as the final entries
    Returns
    -------
    : List[Faces]
        the list of faces with space cube faces removed
    '''
    N_SPACE_CUBE_FACES = 6
    return faces[0:-N_SPACE_CUBE_FACES]
```

This example is a bit verbose
contract still isn't ideal
It requires the programmer (you) to ensure that the last six entries of faces define the space cube
This is a source for bugs

Custom Types

```
def get_norad_launchdate(id : int) -> datetime:
def get_disco_launchdate(id : int) -> datetime:
```

NORAD ID ≠ DISCO ID! **Potential Bugs**

```
NoradId = NewType('NoradId', int)
DiscoId = NewType('DiscoId', int)

def get_norad_launchdate(id : NoradId) -> datetime:
def get_disco_launchdate(id : DiscoId) -> datetime:
```

Very powerful with `match` statement

```
match id:
        case NoradId():
                …
        case DiscoId():
                …
```

Custom Types

Sum types (`Union`): can hold either but not both

```
id : NoradId | DiscoId
```

A useful sum type is `Optional[T]`
Either hold a `T` or `None`

```
def get_norad_launchdate(id : NoradId) -> datetime:
```

What if the satellite has not launched?

```
def get_norad_launchdate(id : NoradId) -> Optional[datetime]:
```

Returns `None` if not launched
AND
tells caller this is possible

Custom Types

Product types: hold multiple values

Unnamed: `tuples`

```
satellite_ids : tuple[NoradId, DiscoId]
```

Named: `dataclass`

```
from dataclasses import dataclass

@dataclass
class Satellite:
      name : str
      age : float
      mass : float
      id : NoradId | DiscoId
      orbit : Orbit
```

Limitations

Python doesn’t check types

mypy is a static type checker

vscode python extension does

numpy has no fixed size arrays