

# Typing made easy

How to make mypy happy

*Hana -  
Backend dev @ octopus energy*



HanaPoulpe

Sometime you have something that works:

```
def main() -> None:  
    numbers = first_pi_decimal()  
    print(dict(count_individual_values(numbers)))
```

Sometime you have something that works:

```
def main() -> None:
    numbers = first_pi_decimal()
    print(dict(count_individual_values(numbers)))
```

But then this happens

demo.py:20: error: Argument 1 to "count\_individual\_values" has incompatible type "Generator[int, None, None]"; expected "List[int]"

Sometime you have something that works:

```
def main() -> None:
    numbers = first_pi_decimal()
    print(dict(count_individual_values(numbers)))
```

But then this happens

demo.py:20: error: Argument 1 to "count\_individual\_values" has incompatible type "Generator[int, None, None]"; expected "List[int]"

You're in a hurry and use this solution:

```
def main() -> None:
    numbers = first_pi_decimal()
    print(dict(count_individual_values(list(numbers))))
```

# But what happened?

```
def count_individual_values(values: list[int]) -> dict[int, int]:  
    individual_values = defaultdict(int)  
  
    for v in values:  
        individual_values[v] += 1  
  
    return individual_values
```

## But what happened?

```
def count_individual_values(values: list[int]) -> dict[int, int]:  
    individual_values = defaultdict(int)  
  
    for v in values:  
        individual_values[v] += 1  
  
    return individual_values
```

## How to fix it?

```
def count_individual_values(values: Iterable[int]) -> dict[int, int]:
```

My rule of thumb:

- 1. Function parameter should request the minimal implementation**
- 2. Function returns should be as detailed as possible**



ABC	Inherits from	Abstract Methods	Mixin Methods
<a href="#">Container</a> [1]		<code>__contains__</code>	
<a href="#">Hashable</a> [1]		<code>__hash__</code>	
<a href="#">Iterable</a> [1] [2]		<code>__iter__</code>	
<a href="#">Iterator</a> [1]	<a href="#">Iterable</a>	<code>__next__</code>	<code>__iter__</code>
<a href="#">Reversible</a> [1]	<a href="#">Iterable</a>	<code>__reversed__</code>	
<a href="#">Generator</a> [1]	<a href="#">Iterator</a>	<code>send</code> , <code>throw</code>	<code>close</code> , <code>__iter__</code> , <code>__next__</code>
<a href="#">Sized</a> [1]		<code>__len__</code>	
<a href="#">Callable</a> [1]		<code>__call__</code>	
<a href="#">Collection</a> [1]	<a href="#">Sized</a> , <a href="#">Iterable</a> , <a href="#">Container</a>	<code>__contains__</code> , <code>__iter__</code> , <code>__len__</code>	
<a href="#">Sequence</a>	<a href="#">Reversible</a> , <a href="#">Collection</a>	<code>__getitem__</code> , <code>__len__</code>	<code>__contains__</code> , <code>__iter__</code> , <code>__reversed__</code> , <code>index</code> , and <code>count</code>
<a href="#">MutableSequence</a>	<a href="#">Sequence</a>	<code>__getitem__</code> , <code>__setitem__</code> , <code>__delitem__</code> , <code>__len__</code> , <code>insert</code>	Inherited <a href="#">Sequence</a> methods and <code>append</code> , <code>reverse</code> , <code>extend</code> , <code>pop</code> , <code>remove</code> , and <code>__iadd__</code>
<a href="#">ByteString</a>	<a href="#">Sequence</a>	<code>__getitem__</code> , <code>__len__</code>	Inherited <a href="#">Sequence</a> methods



## Other cool tools:

```
class Parser(Protocol):  
    def parser(self, input: str) -> Any:  
        ...
```



# Python 3.12

## PEP 695: Type Parameter Syntax

Generic classes and functions under [PEP 484](#) were declared using a verbose syntax that left the scope of type parameters unclear and required explicit declarations of variance.

[PEP 695](#) introduces a new, more compact and explicit way to create [generic classes](#) and [functions](#):

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

In addition, the PEP introduces a new way to declare [type aliases](#) using the `type` statement, which creates an instance of `TypeAliasType`:



HanaPoulpe/DjangoCon2023