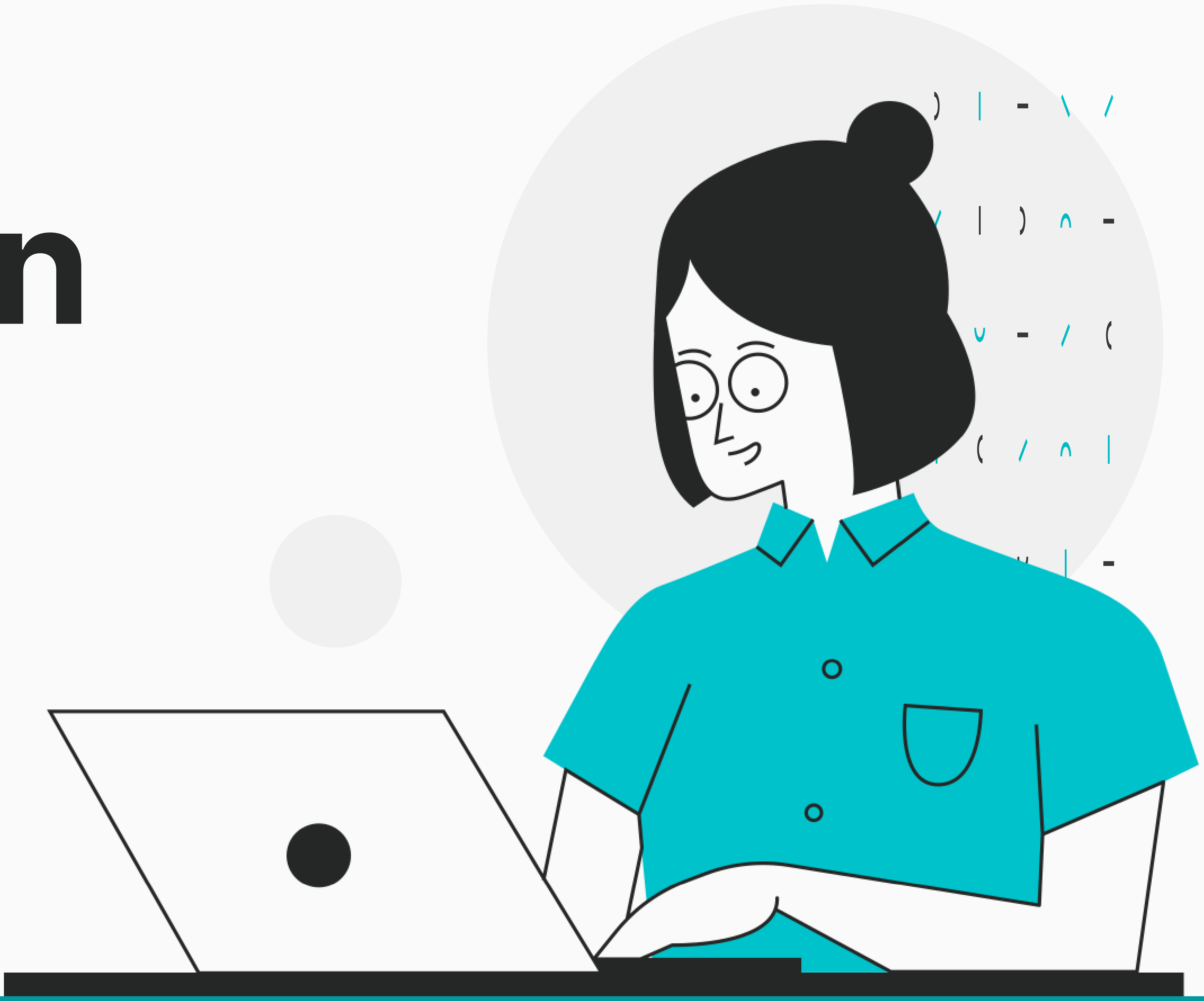
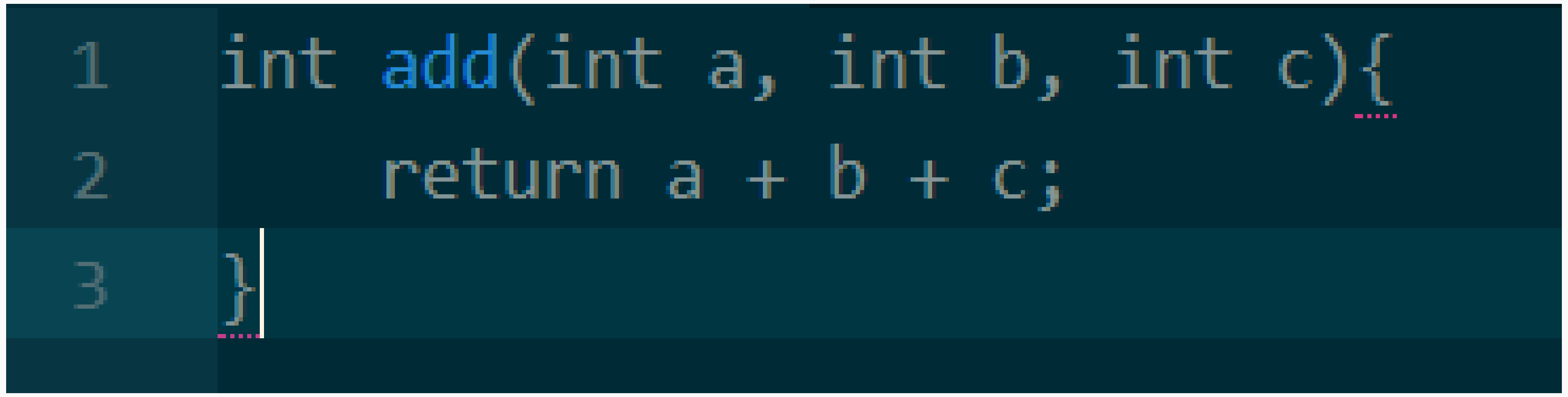
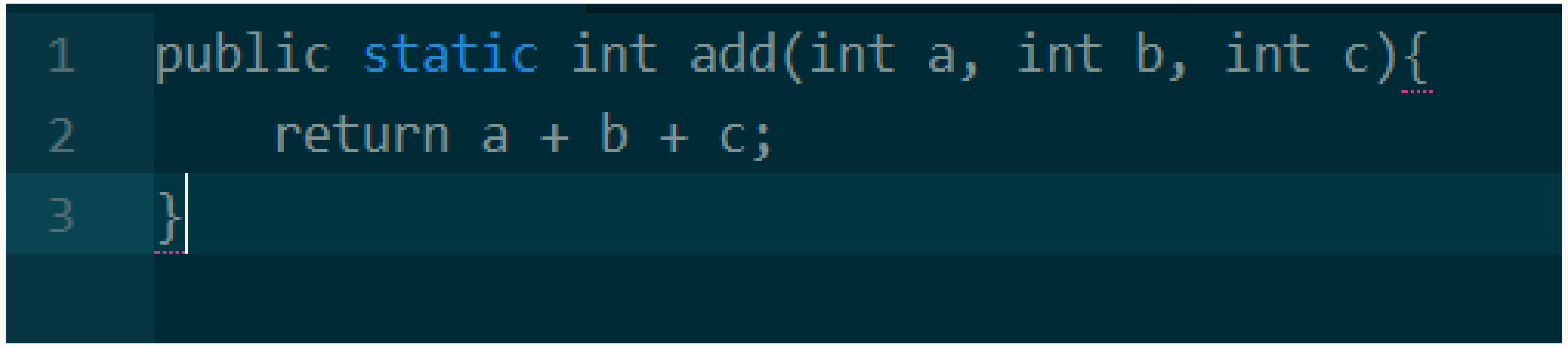
Static Typing in Python

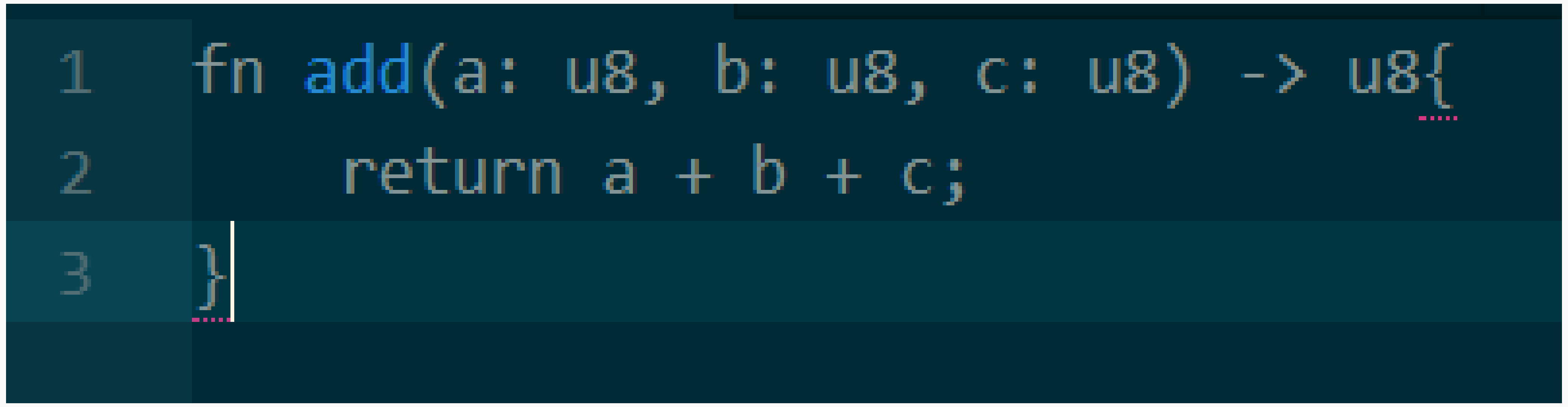




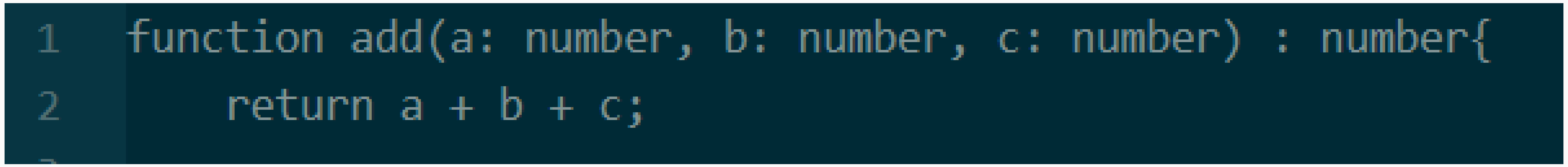
```
int add(int a, int b, int c){
    return a + b + c;
}
```



```java
public static int add(int a, int b, int c){
    return a + b + c;
}
```



```
fn add(a: u8, b: u8, c: u8) -> u8{
    return a + b + c;
}
```



```
function add(a: number, b: number, c: number) : number{
    return a + b + c;
```

Dynamic	Static
Ruby Closure JavaScript	Java Rust C/C++ TypeScript

What about Python?

Python is dynamically typed but can optionally be statically typed

```
>>>type(30)
<class 'int'>
>>>type(32.3)
<class 'float'>
>>>type('hey')
<class 'str'>
>>>type(['hey', 'there'])
<class 'list'>
```

```
>>>a = 30
30
>>>float(30)
30.0
>>>str(float(30))
'30.0'
>>>list(str(float(30)))
['3', '0', '.', '0']
```

```
>>> type(30) is int
True
>>> int
<class 'int'>
>>>isinstance(30, int)
True
```

Dynamic Typing

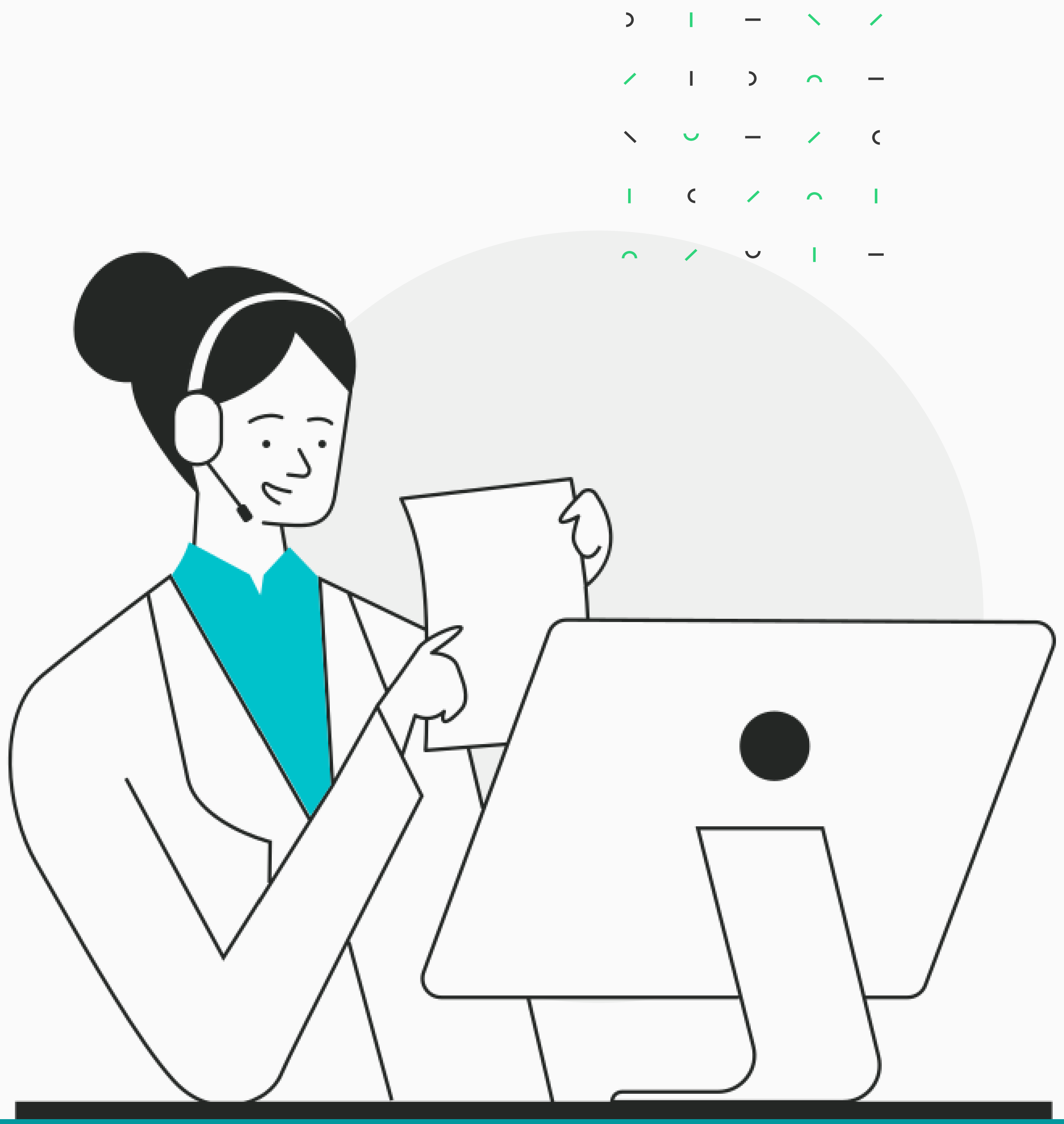
- Variables can be any type
- Arguments and Return values of functions can be any type

```
>>> import random
>>> n = random.choice([30, 30.0,
'30.0'])
>>>type(n)
```

```
def func (a, b, c):
return a + b + c
>>> func (3, 6, 9):
18
>>> func ('Hi', ' ', 'Pythonistas'):
'Hi Pythonistas'
```

```
def func (a, b, c):
return a + b + c
>>> func ('I', 'am', 10)
Traceback(most recent call last):
File"<stdin>", line 1 in <module>
File "<stdin>", line 1 in <func>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

How do I fix this?



How do I fix this?

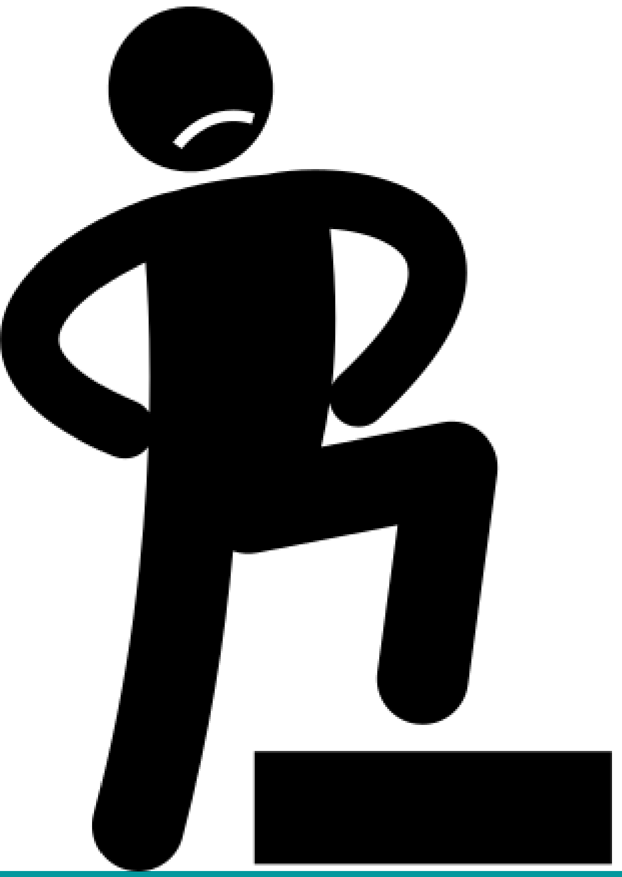
WRITE DOCSTRINGS!

How do I fix this?

ASSERT

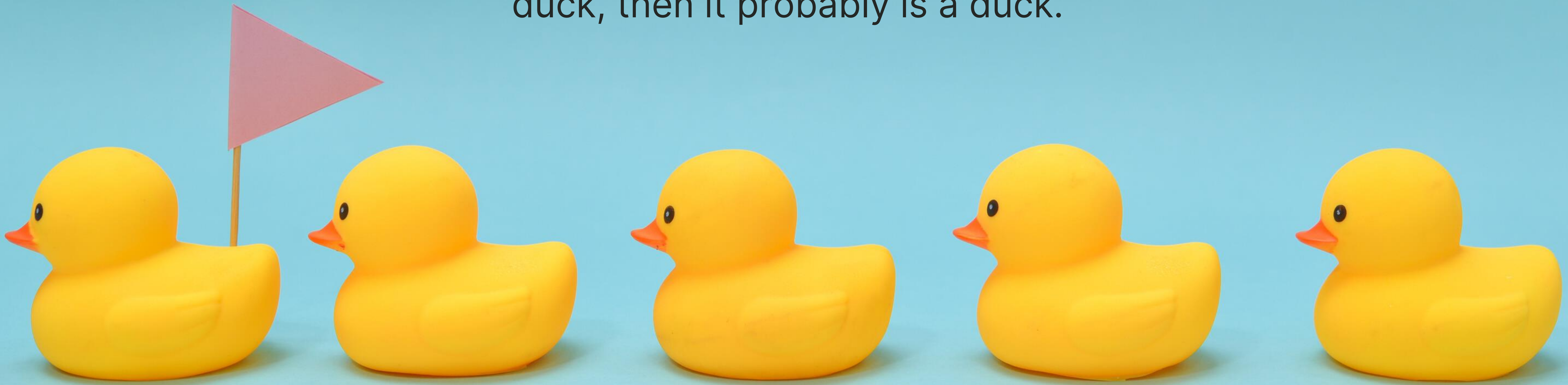
ASSERT

```
def func (a, b, c):
assert type(a) is int
assert type(b) is int
assert type(c) is int
answer = a + b + c
assert type(answer) is int
return answer
```



Duck Typing

If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.



So, how did this come into being?

Our journey to type checking 4 million lines of Python

PEP 3107

Function Annotations

```
>>>def func(a: 'b', b: 2+3, c: []) -> max(3,
11):
return a + b + c

>>>func._ _annotations_ _
{'a' : 'b', 'b' : 5, 'c' : [], 'return': 11}
```

- Providing Typing Information
  - Type checking
  - Letting IDEs show the type a function expects/returns
  - Function overloading/ Generic functions
  - Foreign language bridges
  - Predicate logic functions
  - Database query mapping
  - RPC parameter marshaling

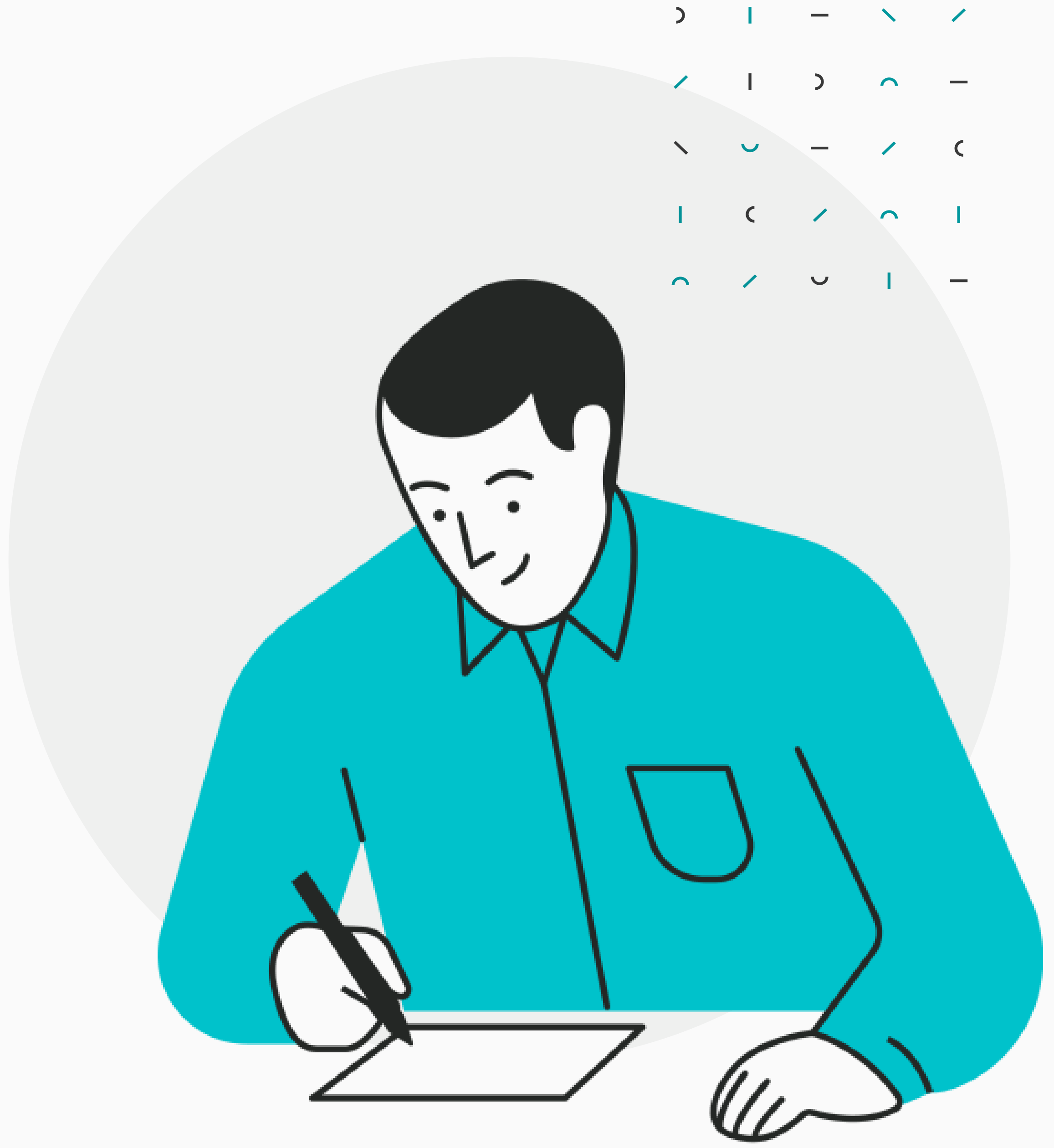
Documentation for parameters and return values

```
>>>def func(a: int, b: int, c: int) -> int:
return a + b + c

>>>func._ _annotations_ _
{'a' : int, 'b' : int, 'c' : int, 'return': int}
```

Jukka Lehtosalo

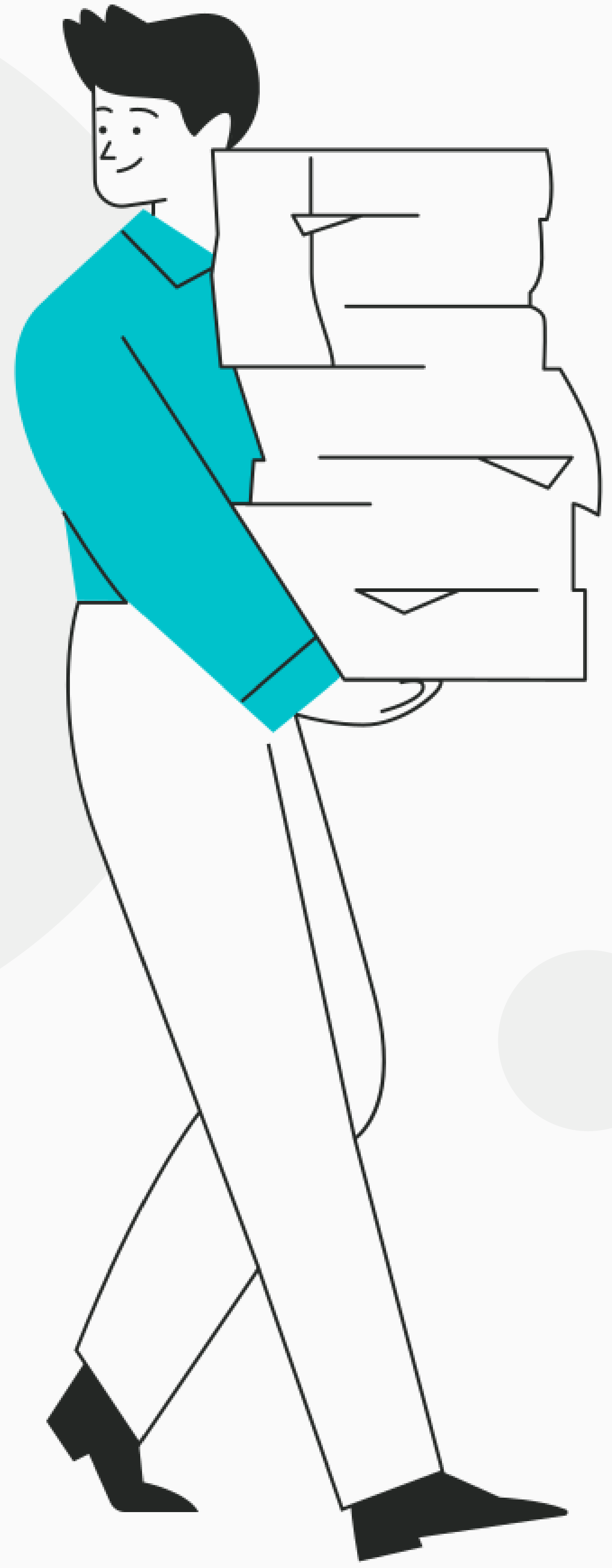
Unification of statically typed and dynamically typed languages



From tiny scripts

to

multi-million line codebases

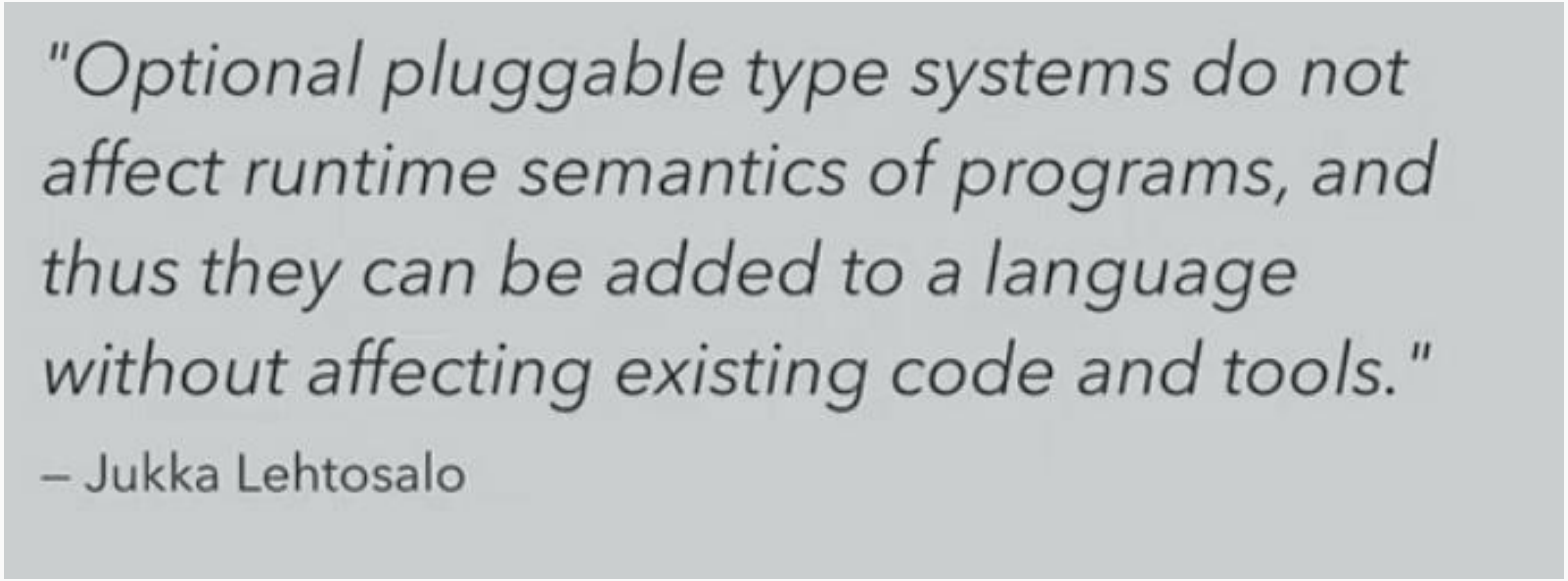


Gradual growth

from an untyped prototype to a statically typed product

"Adding a static type system to a dynamically-typed language can be an invasive change that requires coordinated modification of existing programs, virtual machines and development tools."

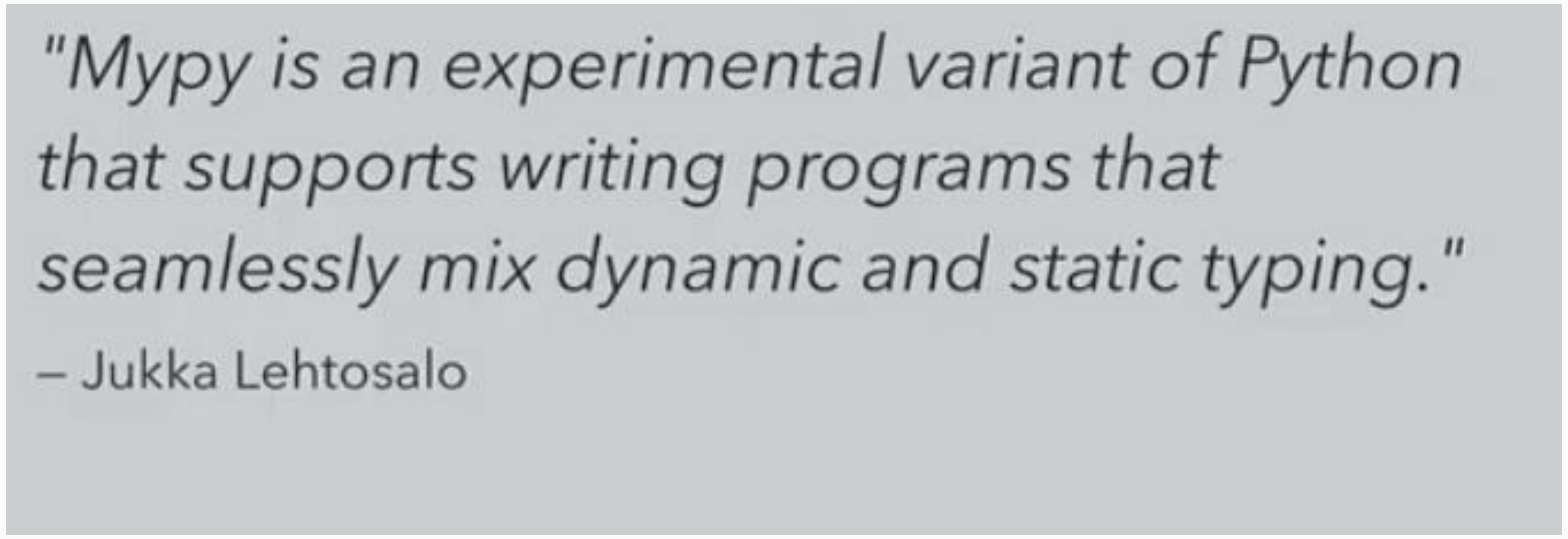
– Jukka Lehtosalo



"Optional pluggable type systems do not affect runtime semantics of programs, and thus they can be added to a language without affecting existing code and tools."

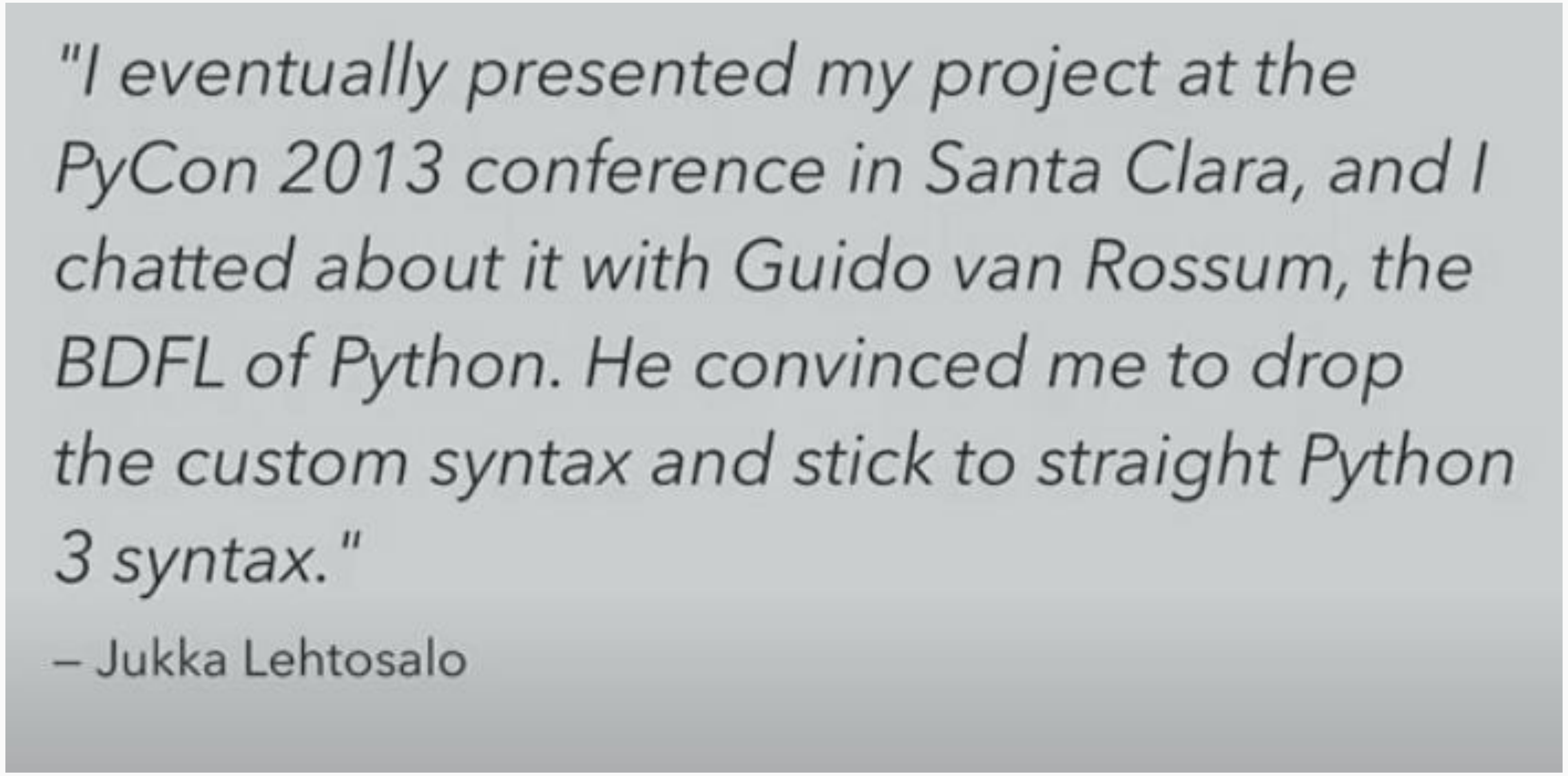
– Jukka Lehtosalo

Mypy



"Mypy is an experimental variant of Python that supports writing programs that seamlessly mix dynamic and static typing."

– Jukka Lehtosalo

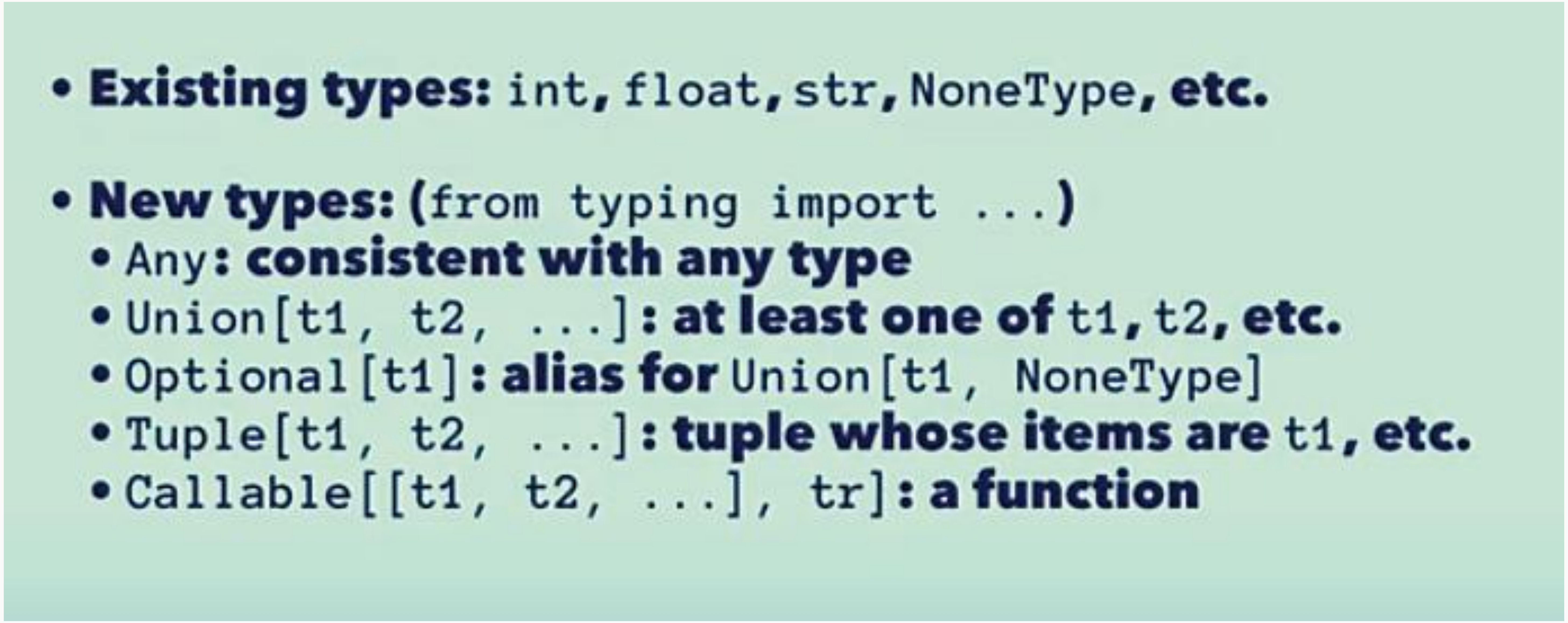


"I eventually presented my project at the PyCon 2013 conference in Santa Clara, and I chatted about it with Guido van Rossum, the BDFL of Python. He convinced me to drop the custom syntax and stick to straight Python 3 syntax."

– Jukka Lehtosalo

PEP 483
The Theory of Type Hints

- Optional Typing - only gets in your way if you want it to
- Gradual Typing - not do it all at once
- Variable Annotations - Annotating just not functions
- Type hinting for Python 2
- Special Type Constructs - fundamental building blocks for static typing



- **Existing types:** `int`, `float`, `str`, `NoneType`, **etc.**
- **New types: (**`from typing import ...`**)**
  - `Any`: **consistent with any type**
  - `Union[t1, t2, ...]`: **at least one of** `t1`, `t2`, **etc.**
  - `Optional[t1]`: **alias for** `Union[t1, NoneType]`
  - `Tuple[t1, t2, ...]`: **tuple whose items are** `t1`, **etc.**
  - `Callable[[t1, t2, ...], tr]`: **a function**

- Container Types - for defining types inside container classes
- Generic Types - when a class or a function behaves in a generic manner (like iterable)
- Relationships - between types, subtypes and classes

PEP 484

Type Hints

Python 3.5

Released: September 13, 2015

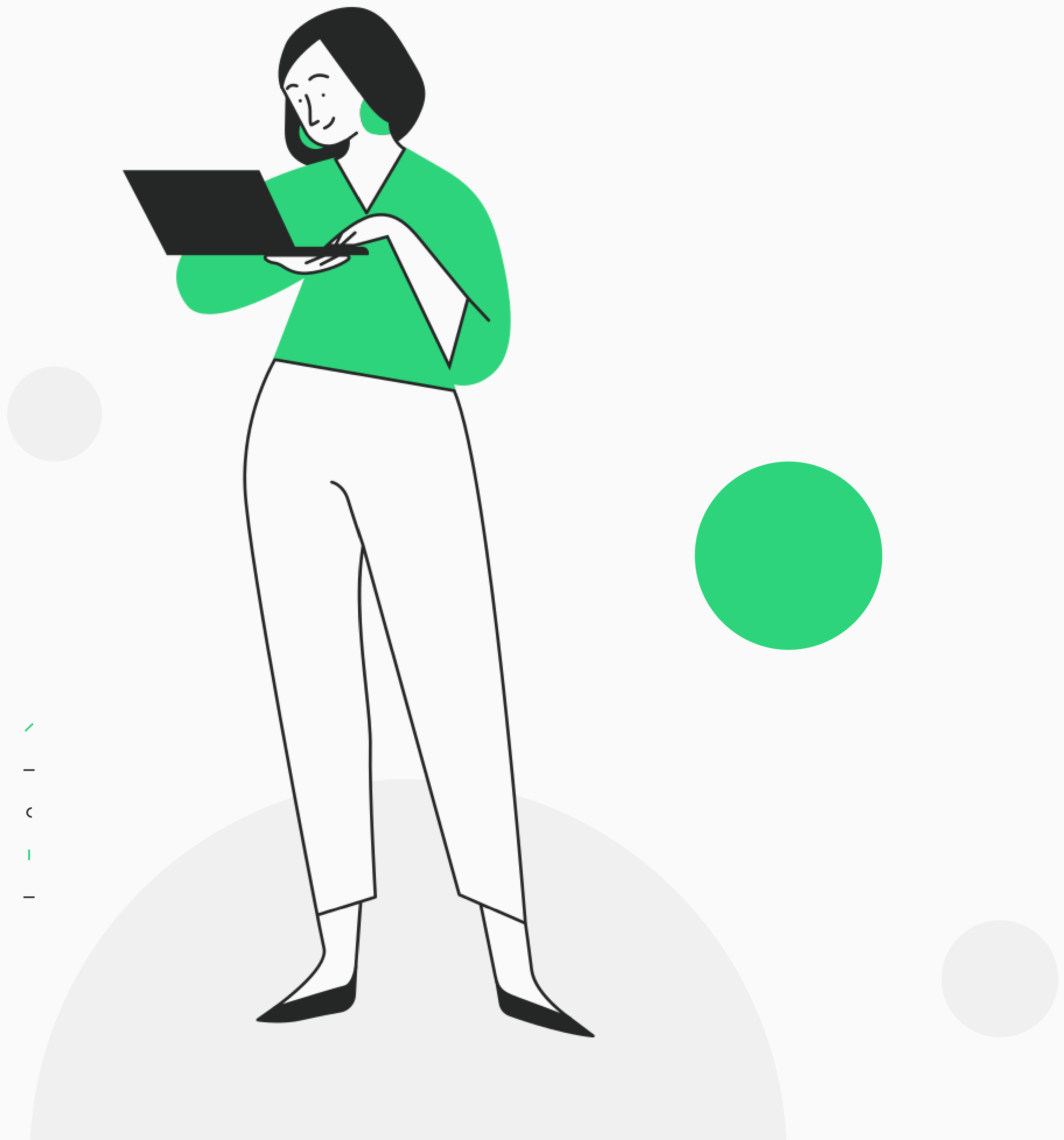
PEP 526

Syntax for Variable Annotations

```
# 'primes' is a list of integers
primes = []  # type: List[int]
```

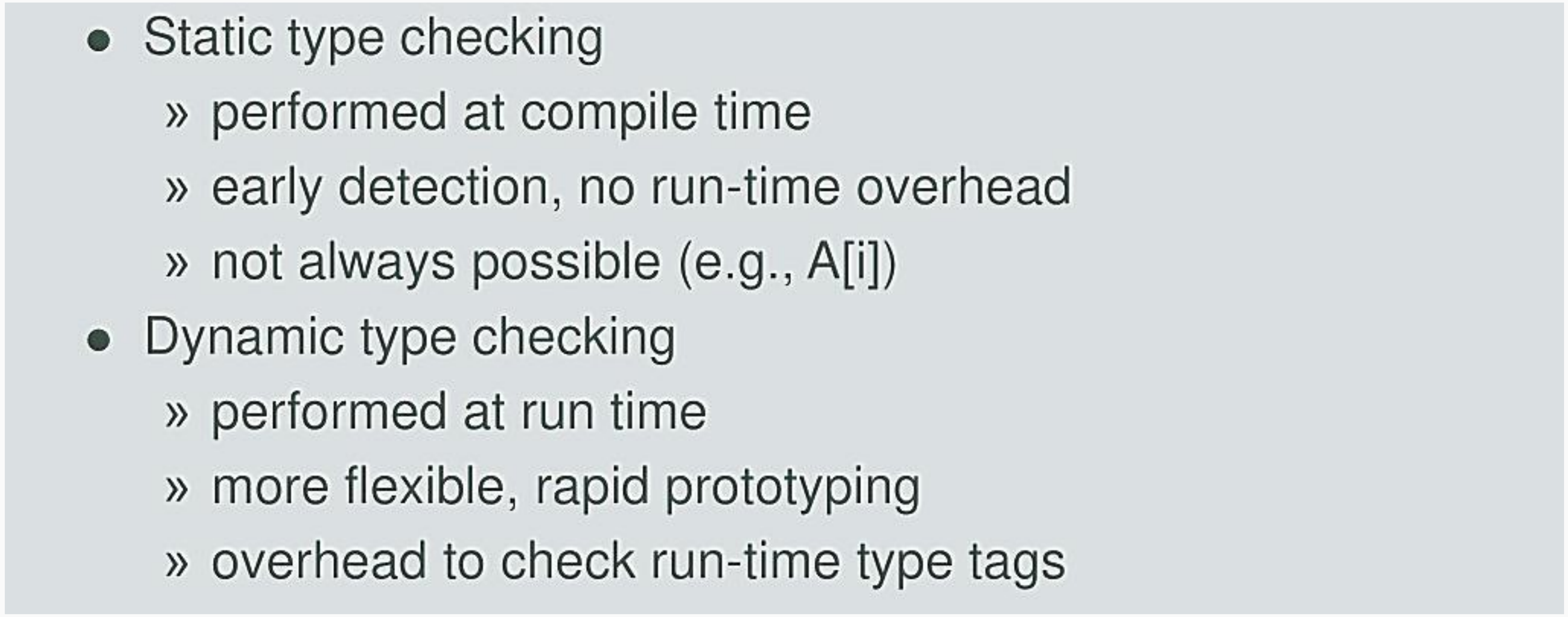
```
# 'primes' is a list of integers
primes: List[int] = []
```

MyPy



Type Checkers

Static vs Dynamic



- Static type checking
  - » performed at compile time
  - » early detection, no run-time overhead
  - » not always possible (e.g., A[i])
- Dynamic type checking
  - » performed at run time
  - » more flexible, rapid prototyping
  - » overhead to check run-time type tags

But why??

When should I not use static typing?

- Not a replacement for unit tests

When should I use static typing?

- When writing millions of lines of codes

"At our scale–millions of lines of Python–the dynamic typing in Python made code needlessly hard to understand and started to seriously impact productivity."

– Jukka Lehtosalo

When should I use static typing?

- When your code is confusing

When should I use static typing?

- When your code is for the public

When should I use static typing?

- Before migrating

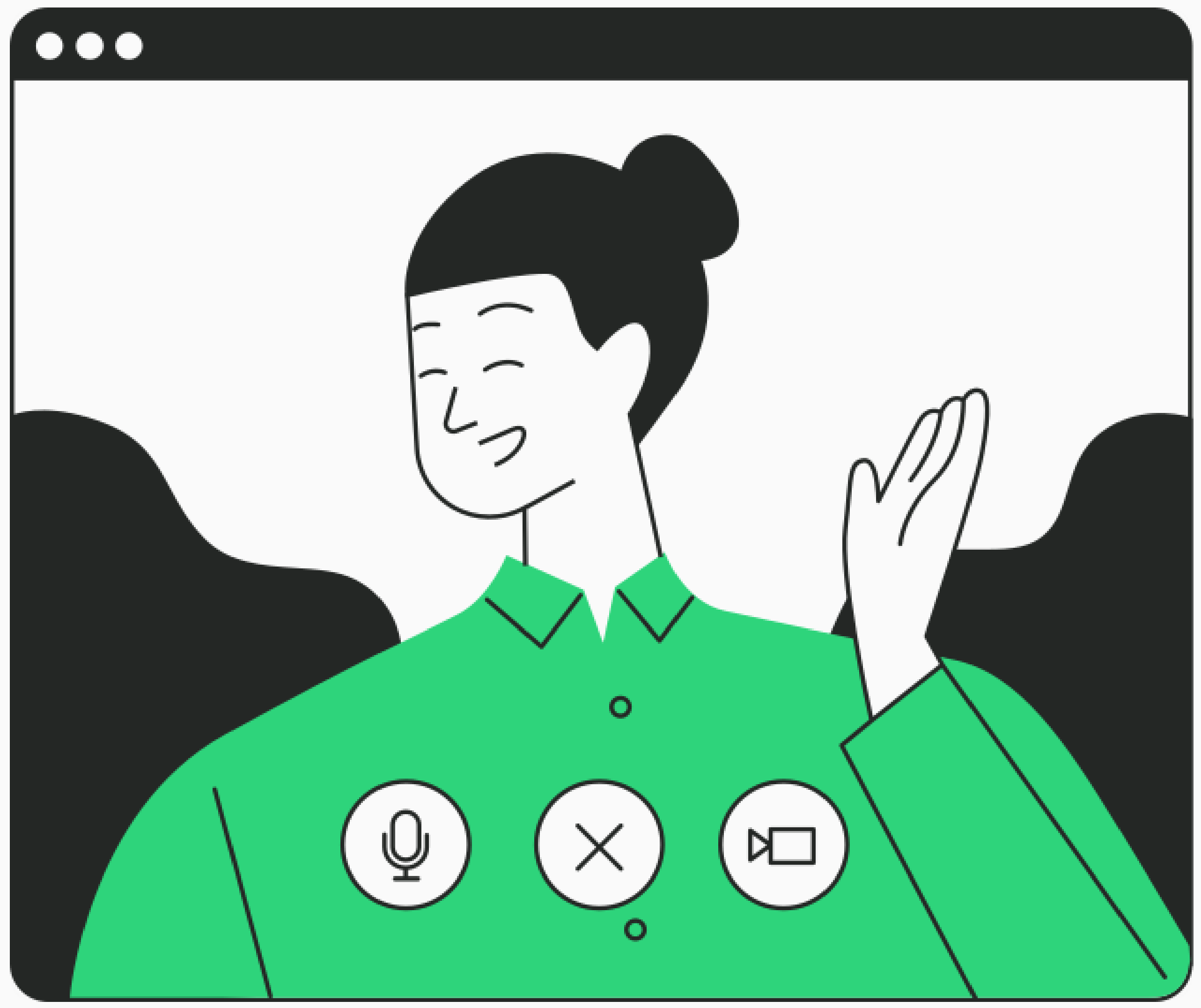
When should I use static typing?

- To experiment with static typing

How to get started

- MIGRATE TO PYTHON 3.6+
- INSTALL A TYPECHECKER LOCALLY
- START OPTIONALLY TYPING YOUR CODEBASE
- RUN A TYPECHECKER WITH YOUR LINTING
- CONVINCE OTHER DEVELOPERS TO JOIN IN

Thank you!



EMAIL

shubhikhanna31.sk@gmail.com

LINKEDIN

linkedin.com/in/shubhikhanna31/

TWITTER

@ShubhiKhanna3