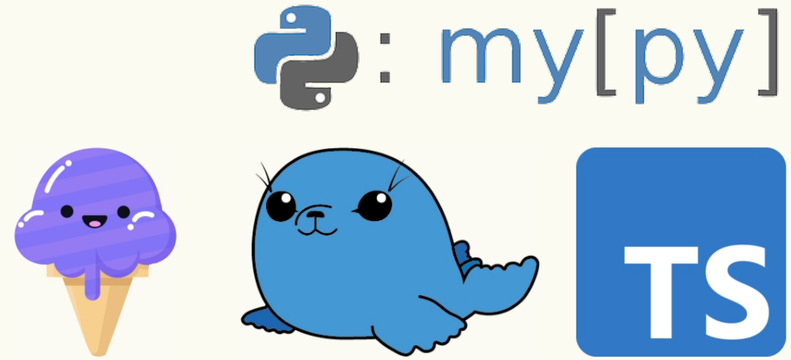
Toward a Corpus Study of the Dynamic Gradual Type

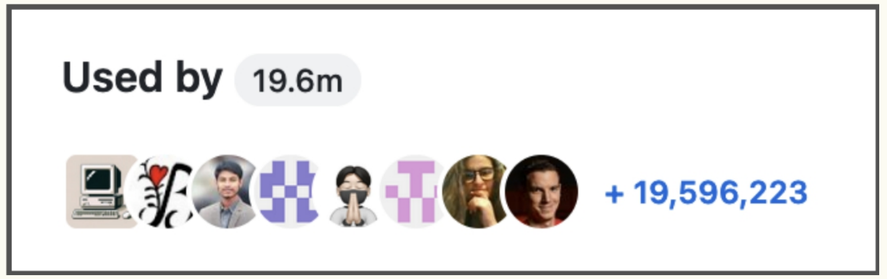
Dibri Nsofor
Ben Greenman

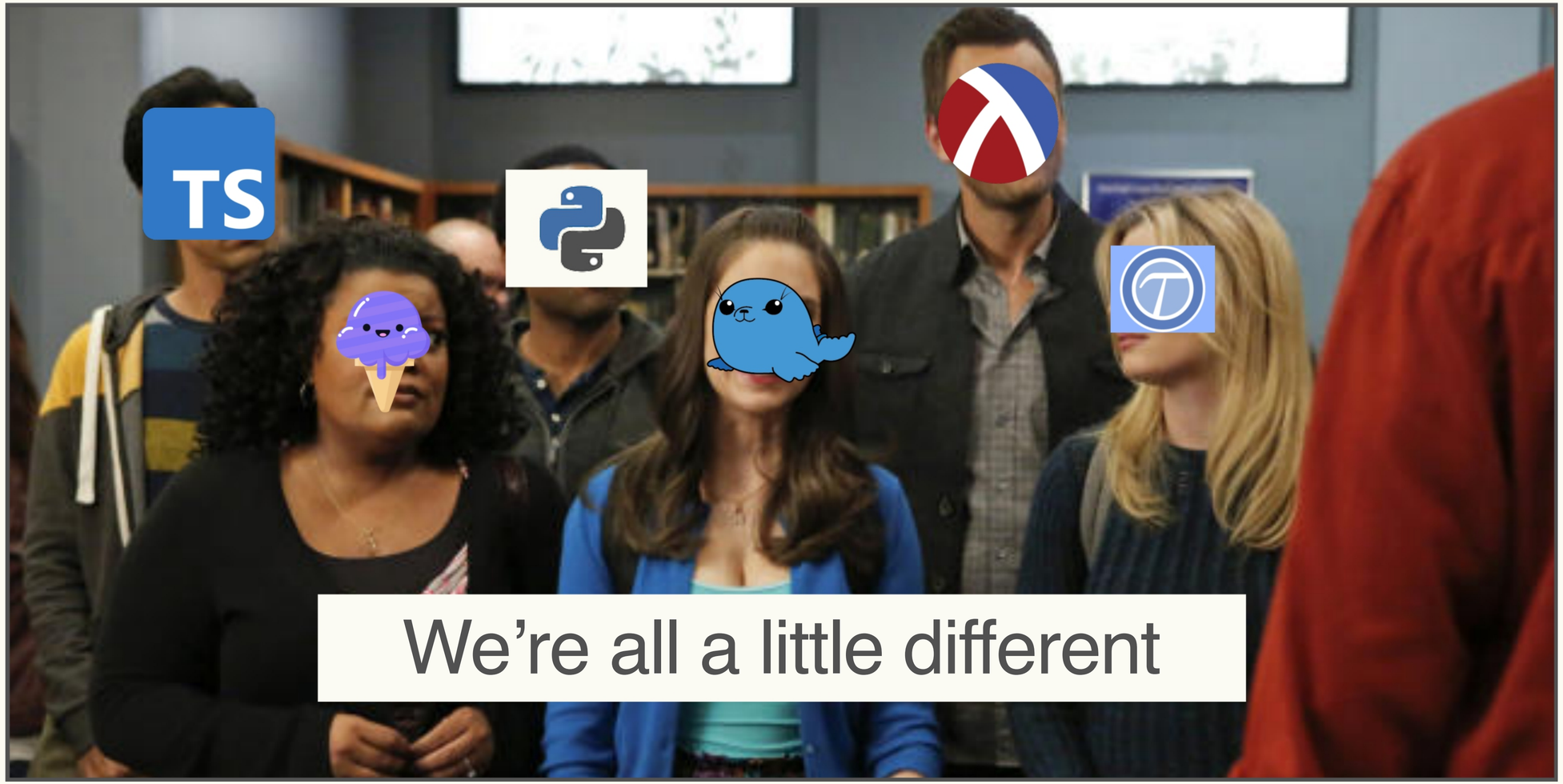

Static
Typing
Dynamic
Typing


Gradual Typing

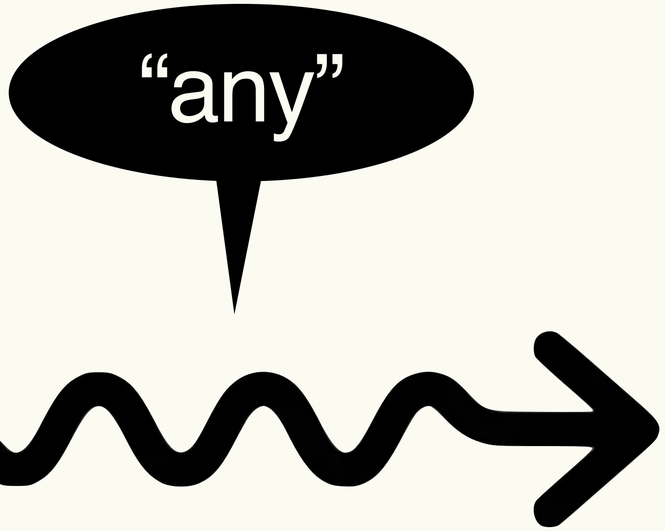
Developers love these gradual types






TS
We’re all a little different

But we have one similarity



Static Typing

Dynamic typing

Why is any important

```
def div(n: int, d: int) -> Any
  if d == 0:
    return “div0 error”
  else:
    return n/d

print(5 + div(2, 0)) # No Type Error
```

But Any is imprecise

```
delays = (2.0, 3.0)
makeReq(“https://grata", delays)
```

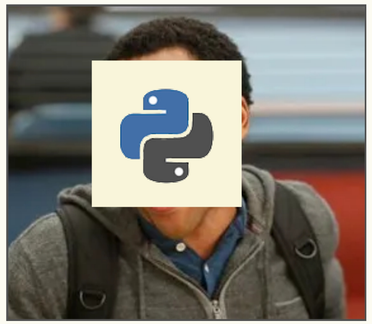
```
def makeReq(url: str) -> Dict[str, str]:
    return getUrl(url).json()
```

```
import requests

def getUrl(url: str, *args, **kws) -> Any:
    ...
```

Any breeds imprecision

Error: No JSON attr

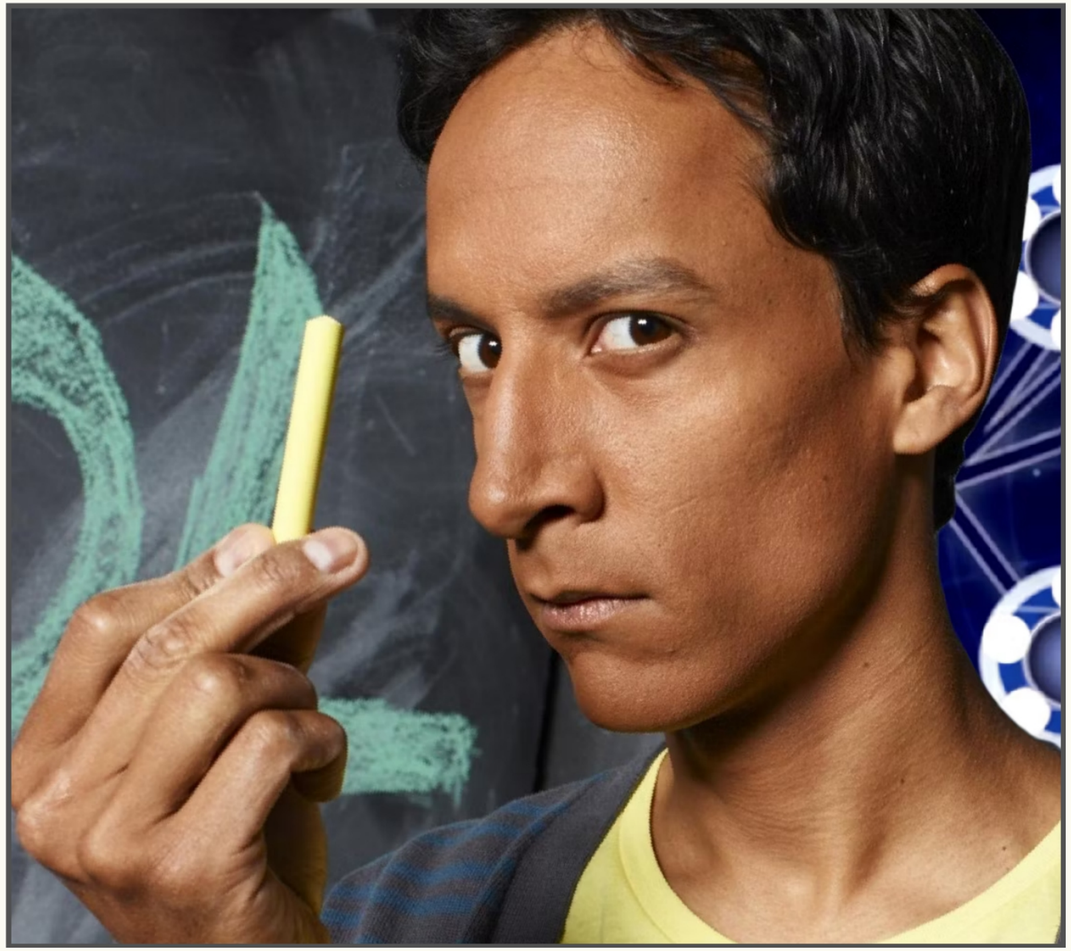
Anys, a blessing and a curse



Anys show up everywhere in Mypy

320,000 signatures across 221 projects

How do we design gradual types?

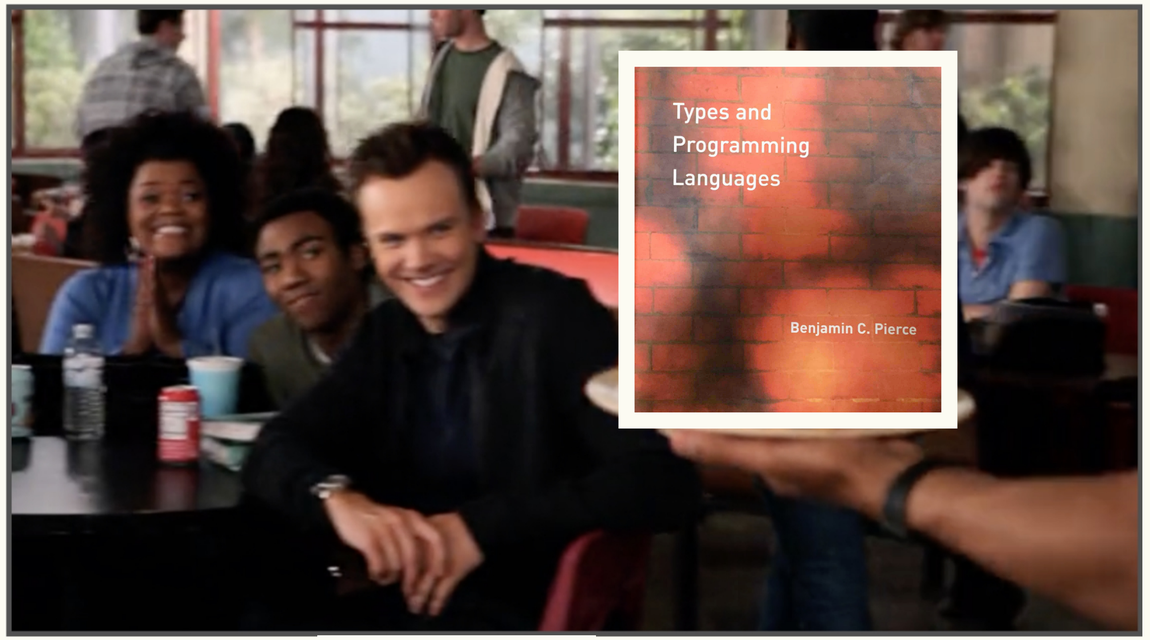


How do we design gradual types?

1 Textbook types + Any

✓ Easy

✗ Not Exhaustive



How do we design gradual types?

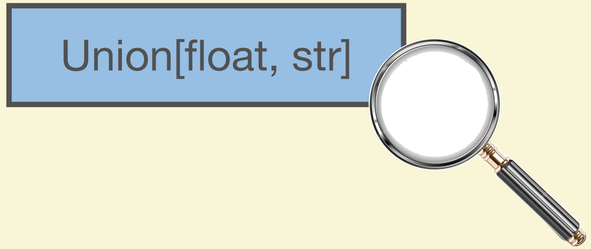
2 Wizard Approach

- ✓ High Quality
- ✗ Hard
- ✗ Error-prone
- ✗ What if language changes?

Both approaches are limited

Why is any important

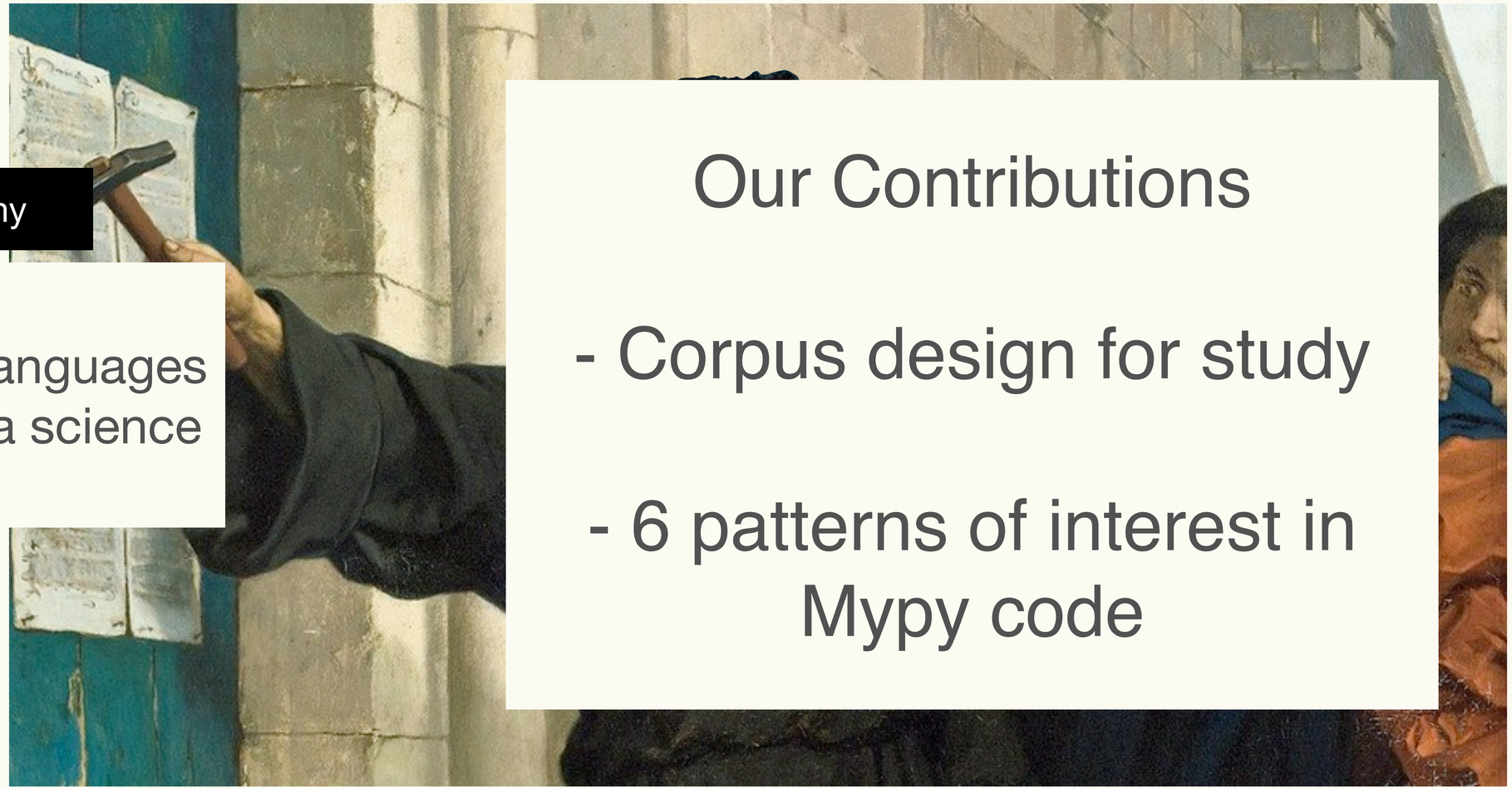
```
def div(n: int, d: int) ->
  if d == 0:
    return “div0 error”
  else:
    return n/d

print(5 + div(2, 0)) # Type Error
```



Thesis

Gradual languages need data science

Thesis



Any

Gradual languages need data science

Our Contributions

- Corpus design for study

- 6 patterns of interest in Mypy code

Big picture goal

79,000 REPOS

T ++

Projects from Github with Mypy options and at least 80 stars

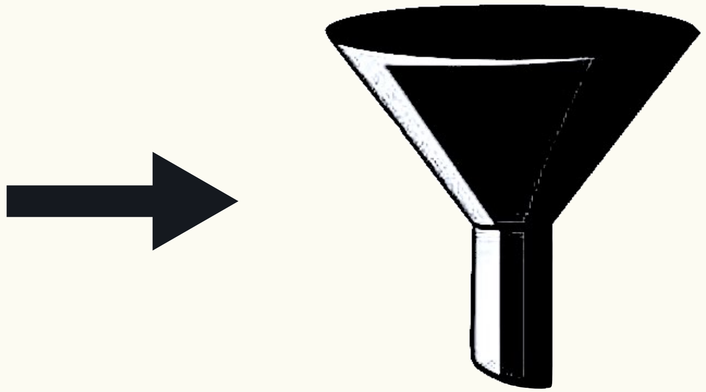
The Fault in Our Stars

Designing Reproducible Large-scale Code Analysis Experiments

Petr Maj
Czech Technical University, Prague, Czech Republic

Our approach - pre corpus study

221 REPOS



Filter for signatures with the gradual type

Manually inspect signatures

STUDY PATTERNS

Manual Study

Source code

```
def getitem(*items: Any) -> Callable[[Any], Any]:
    return functools.partial(_getitem_closure,
    items=items)
```

```
def getitem(*items: Any) -> Callable[[Any], Any]
```

FIltered type Signature with Any

Manual Study

250+ stars

5000
SIGNATURES
10
PROJECTS

Author A

92
SIGNATURES
30
PROJECTS

Author B

Mypy Overview

Python static type checker

Using PEP 484 (type hints)

```
Url = str

def retry(url: Url, retry_count: int) -> None:
    pass


retry(5, “notdibri.com”)
```

```
➤ ~ mypy retry.py --strict
retry.py:7: error: Argument 1 to "retry" has incompatible type "int"; expected "str"  [arg-type]
retry.py:7: error: Argument 2 to "retry" has incompatible type "str"; expected "int"  [arg-type]
Found 2 errors in 1 file (checked 1 source file)
```

Mypy Overview

Python static type checker

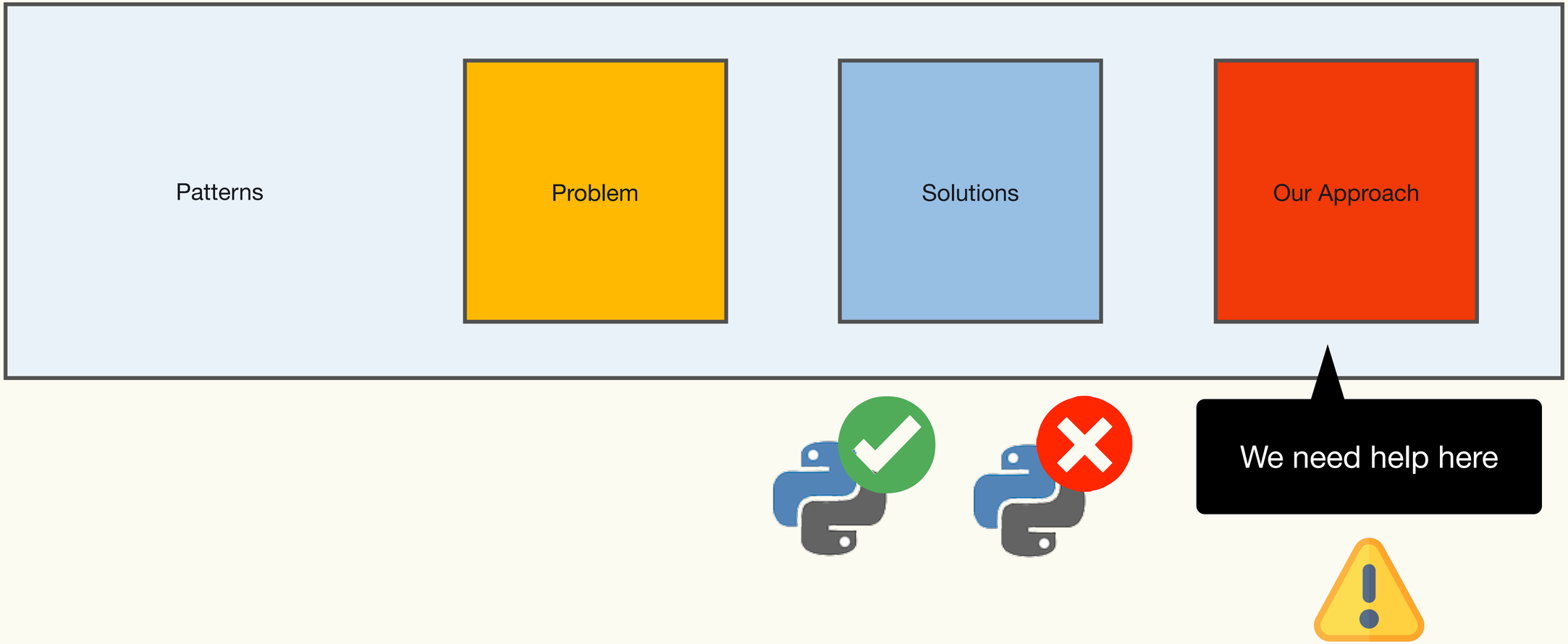
```python
T = TypeVar('T')

def a() -> List[T]:
  ...

reveal_type(a())  # Type: list[Any]
```

```python
def process_data(value: int) -> int:
  ...

data: int = None # Type: ignore
process_data(data) # No Error
```

Unbounded Type Variables

Type ignore comments

Patterns from Pre-corpus study



1.1

Stand In For Self Type

```
class Shape:
  def move(self, dist: int) -> Any:
    self.position += dist
    return self

class Circle(Shape):
  pass

Circle().move(4)
```

Imprecise return type

1.1

Stand In For Self Type

```python
class Shape:
  def move(self, dist: int) -> Self:
    self.position += dist
    return self

class Circle(Shape):
  pass

Circle().move(4) # Type: Circle
```

Self Type

1.2

Stand In For Self Type

How do we flag this?

```
class Shape:
  def move(self, dist: int) -> Any:
    self.position += dist
    return self

class Circle(Shape):
  pass

Circle().move(4)
```

Methods that return instance objects but typed with Any

2.1

For Dependent Dictionaries

```
def get_discount(item: Dict[str, Any]):
  if "price" in item:
    discount = item["price"] * 0.15

    return item["price"] - discount
```

Keys point to values of different types.

2.2

For Dependent Dictionaries

```
Config = Dict[str, Any]

def filter(cfg: Config, k: List[str]) -> Config:
  ...
  return {k[0]: cfg[k[0]]}
```

Depends on values in k

2.2

For Dependent Dictionaries

```
Config = Dict[str, Any]

def filter(cfg: Config, k: Config.keys) -> Config:
  ...
  return {k[0]: cfg[k[0]]}
```

Indexed Access Types

2.3

For Dependent Dictionaries

How do we flag this?

```
Config = Dict[str, Any]

def filter(cfg: Config, k: List[str]) -> Config:
  ...
  return {k[0]: cfg[k[0]]}
```

Mapping types whose values are typed with Any

3.1

Instead of a Type Variable

```
def eq(a: Any, b: Any) -> bool:
  return a == b
```

All calls could be valid

eq(13, “harden”)

3.2

Instead of a Type Variable

```
def zip(xs : list[Any], ys: list[Any]) -> list[(Any, Any)]:
    res = []
    for i in range(min(len(XS), len(ys))):
      res.append((xs[i], ys[i]))
    return res
```

Both lists could be of the same kind

3.2

Instead of a Type Variable

```
from typing import TypeVar, List

Q = TypeVar("Q")
T = TypeVar("T")

def zip(xs : List[T], ys: List[Q]) -> List[(T, Q)]:
    res = []
    for i in range(min(len(XS), len(ys))):
      res.append((xs[i], ys[i]))
    return res
```

Type Variables

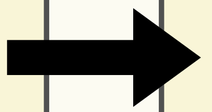
3.3

Instead of a Type Variable

How do we flag this? Not sure.

Replace Anys with Type Variables and unify types

```
def eq(a: T1, b: T2) -> bool:
  return a == b
```



```
def eq(a: T, b: T) -> bool:
  return a == b
```

4.1

Due To Unconstrained Type Variables

```
Car = TypeVar("Car") # Car is unconstrained / unbounded

Traffic = Union[Car, List["Traffic"]]

def count_cars(x: Traffic, car: Car) -> int:
  if isinstance(x, List):
    x.append(car)
   return len(x)

count_cars(["FJ40", "Baja Buggy"], 5) # No Type Error
```

`Car` is instantiated with multiple types (string and int)

4.1

Due To Unconstrained Type Variables

```
Car = TypeVar("Car", str)

Traffic = Union[Car, List["Traffic"]]

def count_cars(x: Traffic, car: Car) -> int:
  if isinstance(x, List):
    x.append(car)
   return len(x)

count_cars(["FJ40", "Baja Buggy"], 5) # Type Error
```

Bounded Type Variables

4.2

Due To Unconstrained Type Variables

How do we flag this?

```
Car = TypeVar("Car")

Traffic = Union[Car, List["Traffic"]]

def count_cars(x: Traffic, car: Car) -> int:
  if isinstance(x, List):
    x.append(car)
return len(x)

count_cars(["FJ40", "Baja Buggy"], 5) # Type Error
```

Look for unbounded or unconstrained type variables

5.1

For Method Overrides

```
class BinaryIO(IO[bytes]):
  def write(self, s: Union[bytes, bytearray]) -> int:
    pass

class FileOpener(BinaryIO):
  def write(self, s: bytes) -> int: # Type: ignore[override]
    return self.fdesc.write(s)
```

Ignore comments weaken type analysis

5.1

For Method Overrides

```
class BinaryIO(IO[bytes]):
  def write(self, s: Union[bytes, bytearray]) -> int:
    pass

class FileOpener(BinaryIO):
  def write(self, s: bytes) -> int:
    return self.fdesc.write(s)
```

```
# No Type Error
```

Ignore by default

5.2

For Method Overrides

How do we flag this?

```
class BinaryIO(IO[bytes]):
  def write(self, s: Union[bytes, bytearray]) -> int:
    pass

class FileOpener(BinaryIO):
  def write(self, s: bytes) -> int: # Type: ignore[override]
    return self.fdesc.write(s)
```

Find ignore comments
and parent class

6.1

Uncategorized Dynamic Types

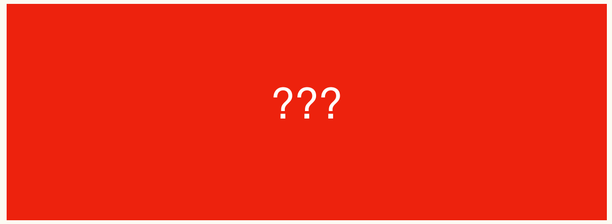
Any to mask implicit return

```
def on_log_message(self, message: str) -> Any:
    self.state.add_agent_log(message)
```

Tools like MonkeyType

```
def on_log_message(self, message: str) -> None:
    self.state.add_agent_log(message)
```

How do we flag this? Not Sure



Road to 79k

79,000
REPOS

τ ++

Where we are

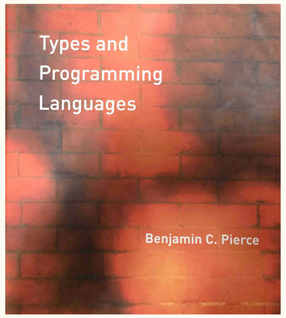
Identified 8 patterns of the Any type

Designed analyses for 5 patterns

Need help with 3 patterns

Better types, tools, and documentation


Design
UX


Types and
Programming
Languages
Benjamin C. Pierce