# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])
```

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
```
⟹ None ⟹ 2

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
⟹ None     ⟹ 2
choose(None, "hello")
⟹ None     ⟹ "hello"
```

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
⟹ None     ⟹ 2
choose(None, "hello")
⟹ None     ⟹ "hello"

def partial(f, x):
    def p(y):
        return f(x, y)
    return p
```

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"

def partial(f, x):
    def p(y):
        return f(x, y)
    return p

p = partial(choose, None)
```

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"

def partial(f, x):
    def p(y):
        return f(x, y)
    return p

p = partial(choose, None)
p(2)
⟹ None      ⟹ 2
```

# A motivating example

```
def choose(a1, a2):
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"

def partial(f, x):
    def p(y):
        return f(x, y)
    return p

p = partial(choose, None)
p(2)
⟹ None      ⟹ 2
p("hello")
⟹ None      ⟹ "hello"
```

# A motivating example

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"

def partial[X, Y, Z](f: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]:
    def p(y: Y) -> Z:
        return f(x, y)
    return p

p = partial(choose, None)
p(2)
⟹ None      ⟹ 2
p("hello")
⟹ None      ⟹ "hello"
```

# A motivating example

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"

def partial[X, Y, Z](f: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]:
    def p(y: Y) -> Z:
        return f(x, y)
    return p

p = partial(choose, None)
p(2)
⟹ None      ⟹ 2
p("hello")
⟹ None      ⟹ "hello"
```

Hello computer, please invoke `multiplay`

# A motivating example

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

choose(None, 2)
⟹ None      ⟹ 2
choose(None, "hello")
⟹ None      ⟹ "hello"
```

```
def partial[X, Y, Z](f: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]:
    def p(y: Y) → Z:
        return f(x, y)
    return p

p = partial(choose, None)
p(2)
⟹ None      ⟹ 2
p("hello")
⟹ None      ⟹ "hello"
```

```
error[invalid-argument-type] Argument is incorrect: Expected `Y@partial |
error: Argument of type "Literal[2]" cannot be assigned to parameter of t
ERROR Argument `Literal[2]` is not assignable to parameter with type `Non
error: Argument 1 has incompatible type "int"; expected "None" [arg-type]
error: Argument 1 has incompatible type "int"; expected "None" [arg-type]
Any
```

# Constraint sets in ty

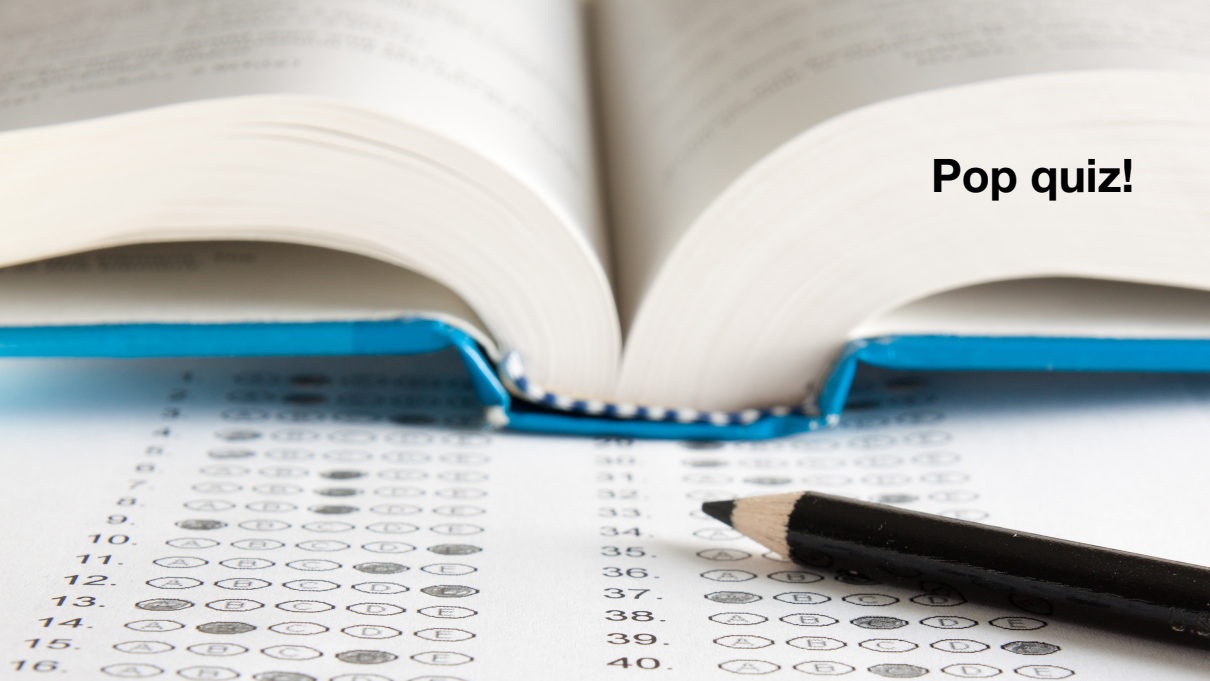
Douglas Creager
Astral

PyCon Typing Summit
May 14, 2026 – Long Beach, CA

# Constraint sets in ty

Douglas Creager
Astral
(an OpenAI joint)

PyCon Typing Summit
May 14, 2026 – Long Beach, CA


Pop quiz!

# Is this expression valid? What is its type?

# Is this expression valid? What is its type?

```
def takes_int(x: int) -> int: ...

takes_int(2)
```

# Is this expression valid? What is its type?

```
def takes_int(x: int) -> int: ...

takes_int(2)
```

int

# Is this expression valid? What is its type?

```
def takes_int(x: int) -> int: ...

takes_int(None)
```

# Is this expression valid? What is its type?

```
def takes_int(x: int) -> int: ...

takes_int(None)
```

```
[invalid-argument-type]
```

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

int

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

int
Literal[2]

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

int
Literal[2]
int | None

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

int
Literal[2]
int | None
object

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

identity(2)
```

```
int
Literal[2]
int | None
object
SupportsIndex
⋮
```

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

result: int = identity(2)
```

# Is this expression valid? What is its type?

```
def identity[T](x: T) -> T: ...

result: int = identity(2)
```

int
Literal[2]
~~int | None~~
~~object~~
~~SupportsIndex~~
⋮

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(2)
```

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(2)
```

$$\texttt{Literal[2]} \preceq \texttt{int}$$

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(2)
```

always

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(None)
```

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(None)
```

$$\texttt{None} \preceq \texttt{int}$$

# *When* is this expression valid?

```
def takes_int(x: int) -> int: ...

takes_int(None)
```

never

# *When* is this expression valid?

```
def identity[T](x: T) → T: ...

identity(2)
```

# *When* is this expression valid?

```
def identity[T](x: T) → T: ...

identity(2)
```

$$\texttt{Literal[2]} \preceq \texttt{T}$$

# *When* is this expression valid?

```
def identity[T](x: T) → T: ...

identity(2)
```

$$\texttt{Literal[2]} \preceq \texttt{T}$$

# *When* is this expression valid?

```
def identity[T](x: T) → T: ...

result: int = identity(2)
```

# *When* is this expression valid?

```
def identity[T](x: T) -> T: ...

result: int = identity(2)
```

$$\texttt{Literal[2]} \preceq \texttt{T}$$

# *When* is this expression valid?

```
def identity[T](x: T) → T: ...

result: int = identity(2)
```

$$\texttt{Literal[2]} \preceq \texttt{T} \wedge \texttt{T} \preceq \texttt{int}$$

# *When* is this expression valid?

```
def identity[T](x: T) -> T: ...

result: int = identity(2)
```

$$\texttt{Literal[2]} \preceq \texttt{T} \land \texttt{T} \preceq \texttt{int}$$

# *When* is this expression valid?

| | |
|---|---|
| `takes_int(2)` | always |
| `takes_int(None)` | never |
| `identity(2)` | $\texttt{Literal[2]} \preceq \texttt{T}$ |
| `result: int = identity(2)` | $\texttt{Literal[2]} \preceq \texttt{T} \wedge \texttt{T} \preceq \texttt{int}$ |

# Sets of assignments

# Sets of assignments

$$\texttt{Literal[2]} \preceq \texttt{T} = \left\{ \begin{array}{l} \texttt{T = int}, \\ \texttt{T = Literal[2]}, \\ \texttt{T = int | None}, \\ \texttt{T = object}, \\ \texttt{T = SupportsIndex}, \\ \vdots \end{array} \right\}$$

# Sets of assignments

$$\texttt{Literal[2]} \preceq \texttt{T} = \left\{ \begin{array}{l} \texttt{T = int}, \\ \texttt{T = Literal[2]}, \\ \texttt{T = int | None}, \\ \texttt{T = object}, \\ \texttt{T = SupportsIndex}, \\ \vdots \end{array} \right\}$$

$$\texttt{Literal[2]} \preceq \texttt{T} \wedge \texttt{T} \preceq \texttt{int} = \left\{ \begin{array}{l} \texttt{T = int}, \\ \texttt{T = Literal[2]} \end{array} \right\}$$

# Partial application

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

choose(None, 2)
```

# Partial application

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
```

$$\texttt{None} \preceq \texttt{A}$$

# Partial application

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
```

$$\texttt{None} \preceq \texttt{A} \wedge \texttt{Literal[2]} \preceq \texttt{A}$$

# Partial application

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
```

$$\texttt{None} \preceq \texttt{A} \wedge \texttt{Literal[2]} \preceq \texttt{A}$$

# Partial application

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
```

$$\texttt{None} \preceq \texttt{A} \wedge \texttt{Literal[2]} \preceq \texttt{A}$$

$$\{\texttt{A = None | Literal[2]}, \texttt{A = None | int}, \ldots\}$$

# Partial application

$$\{\texttt{A = None | Literal[2]}, \texttt{A = None | int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

choose(None, 2)
```

$$\texttt{None} \preceq \texttt{A} \wedge \texttt{Literal[2]} \preceq \texttt{A}$$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$Callable[[A, A], A] \preceq Callable[[X, Y], Z]$$

# Partial application

$$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X$$

# Partial application

$$\{\texttt{A = None | Literal[2]}, \texttt{A = None | int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \texttt{None} \preceq X$$

# Partial application

$$\{\texttt{A = None | Literal[2]}, \texttt{A = None | int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\texttt{X} \preceq \texttt{A} \wedge \texttt{Y} \preceq \texttt{A} \wedge \texttt{A} \preceq \texttt{Z} \wedge \texttt{None} \preceq \texttt{X}$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[Y], Z]$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[Y], Z]$$

$$Z = \text{None}$$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

$Callable[[Y], Z]$

$Y = A$

$Z = None$

# Partial application

$$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

$$Callable[[Y], Z]$$

$$Y = None$$
$$Z = None$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$\text{Callable}[[\text{None}], \text{None}]$

$Y = \text{None}$
$Z = \text{None}$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

Callable[[None], None]
[invalid-argument-type]

Y = None
Z = None

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[Y], Z]$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2],\ A = \text{None} \mid \text{int},\ \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[Y], Z]$$

$$Y = A$$
$$Z = A$$

# Partial application

$$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

$$Callable[[A], A]$$

$$Y = A$$
$$Z = A$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[A], A] \qquad Y = A$$

$$\{A = \text{Literal}[2], A = \text{int}, \ldots\} \qquad Z = A$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$\text{Callable}[[A], A]$ $\quad Y = A$

$\{A = \text{Literal}[2], A = \text{int}, \ldots\}$ $\quad Z = A$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge None \preceq X \wedge None \preceq A \wedge None \preceq Z$$

$$Callable[[Y], Z]$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$\text{Callable}[[Y], Z]$ $\qquad$ $Y \preceq Z \wedge \text{None} \preceq Z$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$$\text{Callable}[[Y], Z] \quad \text{where} \quad Y \preceq Z \wedge \text{None} \preceq Z$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$X \preceq A \wedge Y \preceq A \wedge A \preceq Z \wedge \text{None} \preceq X \wedge \text{None} \preceq A \wedge \text{None} \preceq Z$$

$\text{Callable}[[Y], Z] \quad \text{where} \quad Y \preceq Z \wedge \text{None} \preceq Z$

# Partial application

$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$Callable[[Y], Z]$$

$$Y \preceq Z \wedge None \preceq Z$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\texttt{Callable[[Y], Z]}$$

$$Y \preceq Z \wedge \text{None} \preceq Z \wedge \text{Literal}[2] \preceq Y$$

# Partial application

$\{A = \text{None} \mid \text{Literal}[2], A = \text{None} \mid \text{int}, \ldots\}$

```
def choose[A](a1: A, a2: A) → A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) → Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\text{Callable}[[Y], Z]$$

$$Y \preceq Z \wedge \text{None} \preceq Z \wedge \text{Literal}[2] \preceq Y$$

# Partial application

$$\{A = None \mid Literal[2], A = None \mid int, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$Callable[[Y], Z]$$

$$Y \preceq Z \wedge None \preceq Z \wedge Literal[2] \preceq Y \wedge Literal[2] \preceq Z$$

# Partial application

$$\{A = \texttt{None} \mid \texttt{Literal[2]}, A = \texttt{None} \mid \texttt{int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\texttt{Callable[[Y], Z]}$$

$$Y \preceq Z \wedge \texttt{None} \preceq Z \wedge \texttt{Literal[2]} \preceq Y \wedge \texttt{Literal[2]} \preceq Z$$

# Partial application

$$\{A = \text{None} \mid \text{Literal}[2],\ A = \text{None} \mid \text{int},\ \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\text{Callable}[[Y], Z]$$

$$Y \preceq Z \wedge \text{None} \preceq Z \wedge \text{Literal}[2] \preceq Y \wedge \text{Literal}[2] \preceq Z$$

$$\left\{ \begin{array}{l} Y = \text{None} \mid \text{Literal}[2] \\ Z = \text{None} \mid \text{Literal}[2] \end{array} , \begin{array}{l} Y = \text{None} \mid \text{int} \\ Z = \text{None} \mid \text{int} \end{array} , \ldots \right\}$$

# Partial application

$$\{\texttt{A = None | Literal[2]}, \texttt{A = None | int}, \ldots\}$$

```
def choose[A](a1: A, a2: A) -> A:
    return random.choice([a1, a2])

def partial[X, Y, Z](fn: Callable[[X, Y], Z], x: X) -> Callable[[Y], Z]: ...

p = partial(choose, None)
p(2)
```

$$\texttt{Callable[[Y], Z]}$$

$$\texttt{Y} \preceq \texttt{Z} \wedge \texttt{None} \preceq \texttt{Z} \wedge \texttt{Literal[2]} \preceq \texttt{Y} \wedge \texttt{Literal[2]} \preceq \texttt{Z}$$

$$\left\{ \begin{array}{l} \texttt{Y = None | Literal[2]} \\ \texttt{Z = None | Literal[2]} \end{array} , \begin{array}{l} \texttt{Y = None | int} \\ \texttt{Z = None | int} \end{array} , \ldots \right\}$$

# Open questions

# Open questions

Are there other examples where “constrained callables” are useful?

# Open questions

Are there other examples where “constrained callables” are useful?

Are they worth the extra complexity?

# Open questions

Are there other examples where “constrained callables” are useful?

Are they worth the extra complexity?

How are they implemented?

# Picture credits

Slide 3 David Brossard, “Fenceposts in Argyllshire”
CC BY-SA 2.0, https://flic.kr/p/2nagXBQ

Slide 4 Geoffrey Whiteway, “School test”
Equalicense, https://freerangestock.com/photos/27702/school-test.html

Slide 18 Dr. Matthias Ripp, “Any Questions?”
CC BY 2.0, https://flic.kr/p/pqiJNt

# Scratch

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

`Callable[[list[T]], T]` $\preceq$ `Callable[[A], B]`

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) → B:
    return fn(value)

def head[T](xs: list[T]) → T:
    return xs[0]

invoke(head, [1, 2])
```

$$\texttt{Callable[[list[T]], T]} \preceq \texttt{Callable[[A], B]}$$

$$\texttt{A} \preceq \texttt{list[T]}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) → B:
    return fn(value)

def head[T](xs: list[T]) → T:
    return xs[0]

invoke(head, [1, 2])
```

$$\texttt{Callable[[list[T]], T]} \preceq \texttt{Callable[[A], B]}$$

$$\texttt{A} \preceq \texttt{list[T]} \wedge \texttt{T} \preceq \texttt{B}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) → B:
    return fn(value)

def head[T](xs: list[T]) → T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1]} \preceq L \land \texttt{Literal[2]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) → B:
    return fn(value)

def head[T](xs: list[T]) → T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1]} \preceq L \land \texttt{Literal[2]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1]} \preceq L \land \texttt{Literal[2]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1, 2]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \land T \preceq B \land \texttt{list[L]} \preceq A \land \texttt{Literal[1, 2]} \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$\texttt{A} \preceq \texttt{list[T]} \wedge \texttt{T} \preceq \texttt{B} \wedge \texttt{list[L]} \preceq \texttt{A} \wedge \texttt{Literal[1, 2]} \preceq \texttt{L} \wedge \texttt{list[L]} \preceq \texttt{list[T]}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge \texttt{list[L]} \preceq \texttt{list[T]}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge L \preceq T \wedge T \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge L \preceq T \wedge T \preceq L$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$\texttt{A} \preceq \texttt{list[T]} \wedge \texttt{T} \preceq \texttt{B} \wedge \texttt{list[L]} \preceq \texttt{A} \wedge \texttt{Literal[1, 2]} \preceq \texttt{L} \wedge \texttt{L} \preceq \texttt{T} \wedge \texttt{T} \preceq \texttt{L} \wedge \texttt{L} \preceq \texttt{B}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge L \preceq T \wedge T \preceq L \wedge L \preceq B$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$\texttt{A} \preceq \texttt{list[T]} \wedge \texttt{T} \preceq \texttt{B} \wedge \texttt{list[L]} \preceq \texttt{A} \wedge \texttt{Literal[1, 2]} \preceq \texttt{L} \wedge \texttt{L} \preceq \texttt{T} \wedge \texttt{T} \preceq \texttt{L} \wedge \texttt{L} \preceq \texttt{B} \wedge \texttt{Literal[1, 2]} \preceq \texttt{B}$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge L \preceq T \wedge T \preceq L \wedge L \preceq B \wedge \texttt{Literal[1, 2]} \preceq B$$

# Invariant types

```
def invoke[A, B](fn: Callable[[A], B], value: A) -> B:
    return fn(value)

def head[T](xs: list[T]) -> T:
    return xs[0]

invoke(head, [1, 2])
```

$$A \preceq \texttt{list[T]} \wedge T \preceq B \wedge \texttt{list[L]} \preceq A \wedge \texttt{Literal[1, 2]} \preceq L \wedge L \preceq T \wedge T \preceq L \wedge L \preceq B \wedge \texttt{Literal[1, 2]} \preceq B$$

$$\left\{ \begin{array}{l} A = \texttt{list[Literal[1, 2]]} \\ B = \texttt{Literal[1, 2]} \end{array} , \begin{array}{l} A = \texttt{list[int]} \\ B = \texttt{int} \end{array} , \ldots \right\}$$

# Math!

$$\begin{aligned} \mathcal{C} &\triangleq \texttt{always} \\ &\mid \texttt{never} \end{aligned}$$

# Math!

$$
\begin{aligned}
\mathcal{C} \triangleq\ & \texttt{always} \\
\mid\ & \texttt{never} \\
\mid\ & \texttt{type} \preceq \texttt{T} && \text{(lower bound)} \\
\mid\ & \texttt{T} \preceq \texttt{type} && \text{(upper bound)}
\end{aligned}
$$

# Math!

$$
\begin{array}{lll}
\mathcal{C} \triangleq & \texttt{always} & \\
\mid & \texttt{never} & \\
\mid & \texttt{type} \preceq \texttt{T} & \text{(lower bound)} \\
\mid & \texttt{T} \preceq \texttt{type} & \text{(upper bound)} \\
\mid & \neg\, \mathcal{C} & \text{(not)} \\
\mid & \mathcal{C} \wedge \mathcal{C} & \text{(and)} \\
\mid & \mathcal{C} \vee \mathcal{C} & \text{(or)}
\end{array}
$$