Estreitamento de tipo em Python

Aprimorando a Precisão no Código com Type Narrowing em Python

Jhonata Medeiros

- Desenvolvedor web a dois anos e meio
- Pythonista de coração

O que seria estreitamento de tipo?

O que seria estreitamento de tipo?*

*Type Narrowing

O estreitamento de tipo ocorre quando você convence um verificador de tipo de que um tipo mais amplo é, na verdade, mais específico, por exemplo, que um objeto do tipo ***Forma Geométrica*** *é, na verdade, do tipo mais estreito* ***Quadrado****.*

O estreitamento de tipo ocorre quando você convence um verificador de tipo de que um tipo mais amplo é, na verdade, mais específico, por exemplo, que um objeto do tipo ***Forma Geométrica*** *é, na verdade, do tipo mais estreito* ***Quadrado****.*

Documentação do MyPy

Para que serve?

- Segurança e precisão no código
- Evitar validações repetitivas
- Possibilita comportamentos diferenciados

Formas simples de se estreitar tipos

Formas simples de se estreitar tipos

- isinstance()
- issubclass()
- type(obj) is x
- obj is not None
- cast(list[int], [1])

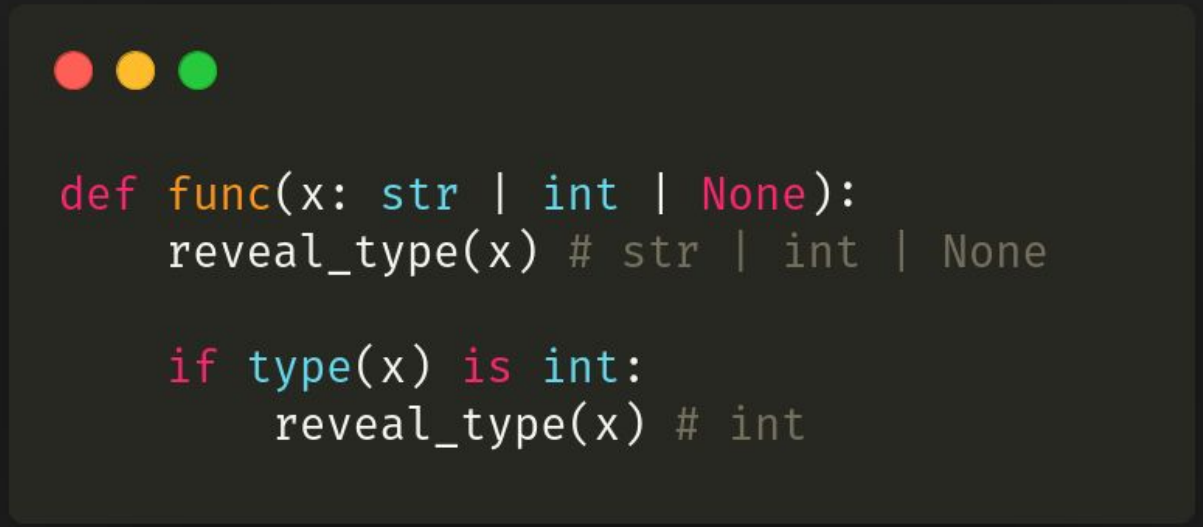
isinstance()

```python
def func(x: str | int):
    reveal_type(x) # Revealed type is "str | int"

    if isinstance(x, int):
        reveal_type(x) # Revealed type is "int"

    if isinstance(x, str):
        reveal_type(x) # Revealed type is "str"
```

issubclass()

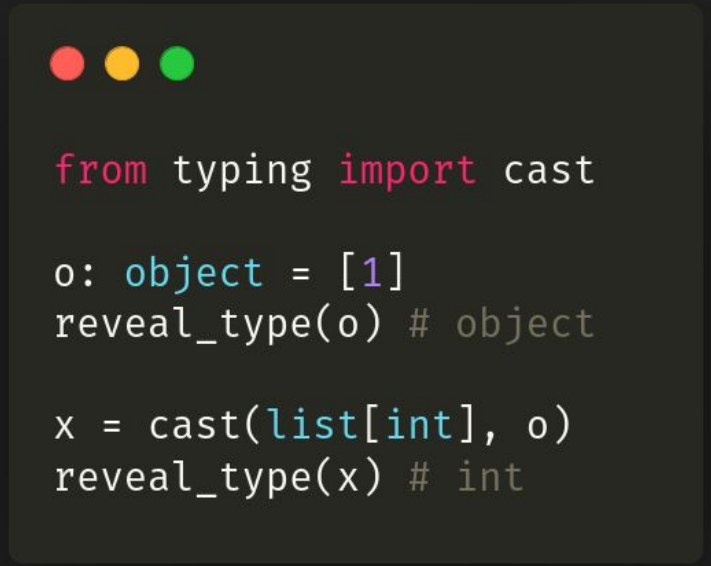
```python
class A: ...
class B(A): ...

def func(p: object):
    type_p = type(p)
    reveal_type(type_p) # type[object]

    if issubclass(type_p, A):
        reveal_type(type_p) # type[A]
    else:
        reveal_type(type_p) # type[object]
```

type(obj) is x



```python
def func(x: str | int | None):
    reveal_type(x) # str | int | None

    if type(x) is int:
        reveal_type(x) # int
```

cast(list[int], [1])



```python
from typing import cast

o: object = [1]
reveal_type(o) # object

x = cast(list[int], o)
reveal_type(x) # int
```

Formas avançadas

TypeGuard

Existem casos em que apenas informação estática não garante um estritamento de tipo preciso.

TypeGuard

Existem casos em que apenas informação estática não garante um estritamento de tipo preciso.

```python
from typing import TypeGuard

def is_str_list(val: list[object]) → TypeGuard[list[str]]:
    '''Determina se todos os objetos na lista são strings'''
    return all(isinstance(x, str) for x in val)

def func1(val: list[object]):
    if is_str_list(val):
        reveal_type(val) # list[object]

        print(' '.join(val)) # Error: invalid type
```

TypeGuard

Para resolver esse tipo de problema. O `TypeGuard` serve para adicionar uma "proteção de tipo definida pelo usuário"

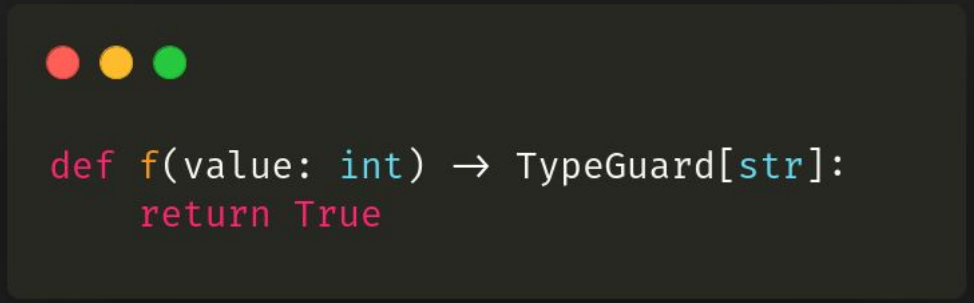
TypeGuard

Para resolver esse tipo de problema. O `TypeGuard` serve para adicionar uma "proteção de tipo definida pelo usuário"

```python
from typing import TypeGuard

def is_str_list(val: list[object]) → TypeGuard[list[str]]:
    '''Determina se todos os objetos na lista são strings'''
    return all(isinstance(x, str) for x in val)

def func1(val: list[object]):
    if is_str_list(val):
        reveal_type(val) # list[str]

        print(' '.join(val)) # Error: invalid type
```

Um adendo

Foi notado que sem impor um estreitamento estrito, seria possível quebrar a segurança de tipo. Uma função de proteção de tipo mal escrita poderia produzir resultados inseguros ou até mesmo sem sentido. Por exemplo:



Um adendo

No entanto, há muitas maneiras pelas quais um desenvolvedor determinado ou desinformado pode subverter a segurança de tipos – mais comumente usando castou Any. Se um desenvolvedor Python dedicar um tempo para aprender e implementar proteções de tipos definidas pelo usuário em seu código, é seguro assumir que ele está interessado em segurança de tipos e não escreverá suas funções de proteção de tipos de uma forma que prejudique a segurança de tipos ou produza resultados sem sentido.

-- PEP 647

TypeIs

- A partir da 3.13
- O retorno PRECISA ser um TypeIs
- Comportamento próximo ao isinstance()
- Um resultado *True* consegue inferir um tipo mais preciso

TypeIs

- A partir da 3.13
- O retorno PRE
- Comportament
- Um resultado 7

```python
from typing import TypeGuard
from typing_extensions import TypeIs

# A partir da versão 3.13
# from typing import TypeIs

def is_str_type_is(x: object) -> TypeIs[str]:
    return isinstance(x, str)

def is_str_type_guard(x: object) -> TypeGuard[str]:
    return isinstance(x, str)

def f(x: str | int) -> None:
    if is_str_type_is(x):
        reveal_type(x) # str
    else:
        reveal_type(x) # int

    if is_str_type_guard(x):
        reveal_type(x) # str
    else:
        reveal_type(x) # str | int
```

TypeIs

```python
from typing import TypeGuard, reveal_type, final
from typing_extensions import TypeIs

# A partir da versão 3.13
# from typing import TypeIs

class Base: ...
class Child(Base): ...
@final
class Unrelated: ...

def is_base_typeguard(x: object) -> TypeGuard[Base]:
    return isinstance(x, Base)

def is_base_typeis(x: object) -> TypeIs[Base]:
    return isinstance(x, Base)

def use_typeguard(x: Child | Unrelated) -> None:
    if is_base_typeguard(x):
        reveal_type(x)  # Base
    else:
        reveal_type(x)  # Child | Unrelated

def use_typeis(x: Child | Unrelated) -> None:
    if is_base_typeis(x):
        reveal_type(x)  # Child
    else:
        reveal_type(x)  # Unrelated
```

Hora do código!!

Obrigado
a todos!