**Fixing a Variable's Type in Python**

Python is dynamically typed, meaning a variable's type can change at runtime. However, if you want to enforce a fixed type, you can use type hints, assertions, decorators, or custom classes.

**1. Using Type Hints (Only Advisory, Not Enforced)**

Python’s type hints provide guidance but do not enforce type restrictions at runtime.

def set\_age(age: int):

return age

x: int = 10

x = "hello"

This method helps with static type checking using tools like MyPy but does not prevent type changes at runtime.

**2. Using isinstance() or assert for Runtime Type Checking**

You can use assert or isinstance() to ensure a variable stays of a fixed type.

x = 10

if not isinstance(x, int):

raise TypeError("x must be an integer")

# OR using assert

assert isinstance(x, int), "x must be an integer"

x = "hello" # This would raise an error in the above check

This method enforces type safety at runtime but requires explicit checks.

**3. Using Properties in a Class**

A better way is to use a class with property setters to enforce type constraints.

class FixedType:

def \_\_init\_\_(self, value: int):

self.\_value = None

self.value = value # Use setter

@property

def value(self):

return self.\_value

@value.setter

def value(self, new\_value):

if not isinstance(new\_value, int):

raise TypeError("Value must be an integer")

self.\_value = new\_value

# Usage

obj = FixedType(100)

print(obj.value) # Output: 100

obj.value = "hello" # TypeError: Value must be an integer

This method encapsulates type enforcement within a class and prevents accidental type changes.

**Best Approach**

* For small scripts, use assert isinstance()
* For functions, use type hints (def func(x: int) -> None)
* For class attributes, use properties or dataclass