

# TypeScript vs Python:

Static, Strong, and Structural Typing Explained

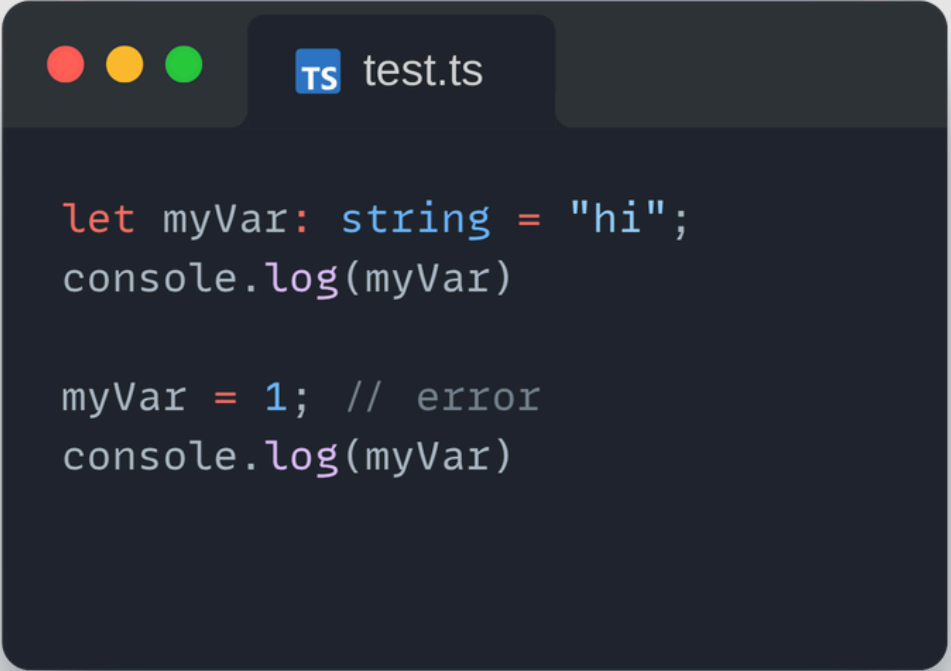


Next Slide



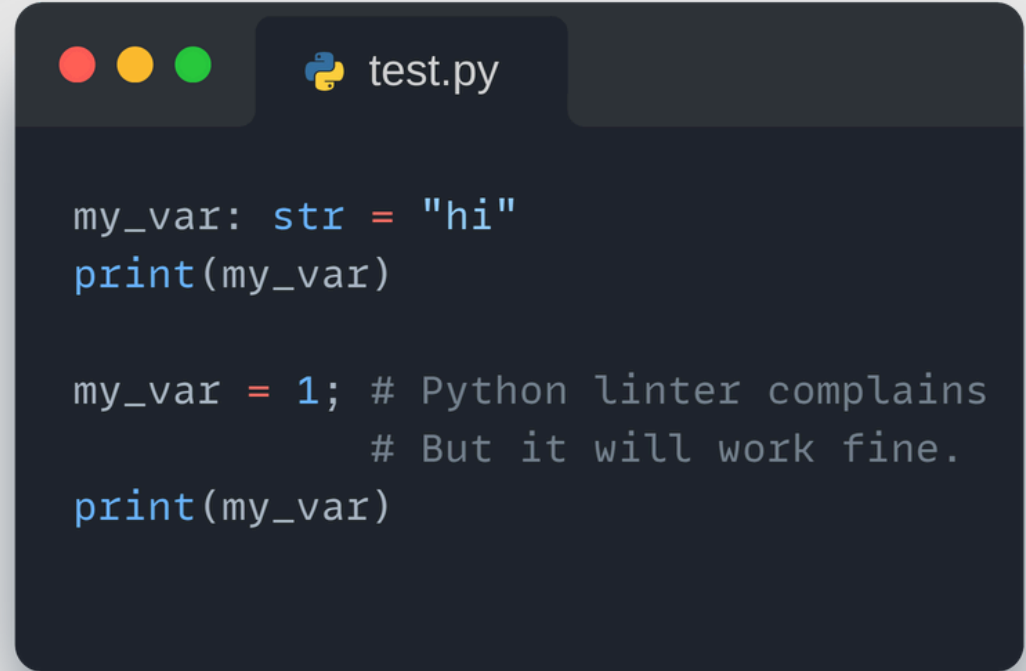
# Static vs Dynamic Typing

TypeScript is a statically typed language, whereas Python is dynamically typed. In TypeScript, a variable's type is determined at compile time and cannot change at runtime, though its value can. In contrast, Python allows variables to change both their values and types dynamically, meaning a variable can be reassigned to a value of a different type.

A code editor window with a dark background. The title bar shows three colored circles (red, yellow, green) and a tab labeled 'TS test.ts'. The code is written in a light blue font.

```
let myVar: string = "hi";  
console.log(myVar)
```

```
myVar = 1; // error  
console.log(myVar)
```

A code editor window with a dark background. The title bar shows three colored circles (red, yellow, green) and a tab labeled 'test.py'. The code is written in a light blue font.

```
my_var: str = "hi"  
print(my_var)
```

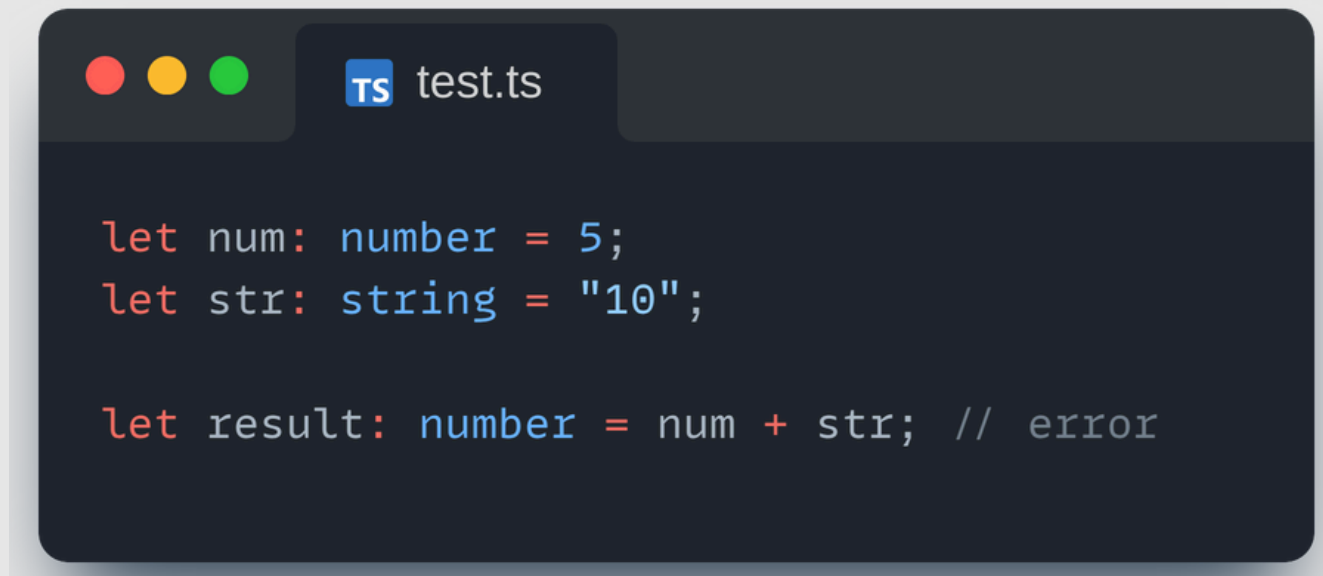
```
my_var = 1; # Python linter complains  
           # But it will work fine.  
print(my_var)
```

Next Slide



# Strong vs Weak Typing

There is no real agreement on what "strongly typed" means, although the most widely used definition in the professional literature is that in a "strongly typed" language, it is not possible for the programmer to work around the restrictions imposed by the type system. TypeScript and Python are both strongly typed languages. The following piece of code will raise an error in TypeScript.

A code editor window with a dark background and a title bar showing three colored circles (red, yellow, green) and a tab labeled 'test.ts'. The code inside is as follows:

```
let num: number = 5;  
let str: string = "10";  
  
let result: number = num + str; // error
```

Weak typing implies that the compiler does not enforce a typing discipline, or perhaps that enforcement can easily be subverted. An example of weakly typed language is JavaScript and you see the above piece of code work just fine in JavaScript.

Next Slide



# Structural vs Nominal Typing

- In a nominal system, comparisons between types are based on names and declarations. The Python type system is mostly nominal and to achieve structural typing, use protocols.
- In a structural system, comparisons between types are based on structure, e.g. in TypeScript, two types that have the same shape are effectively interchangeable. For nominal typing, use tagged unions.

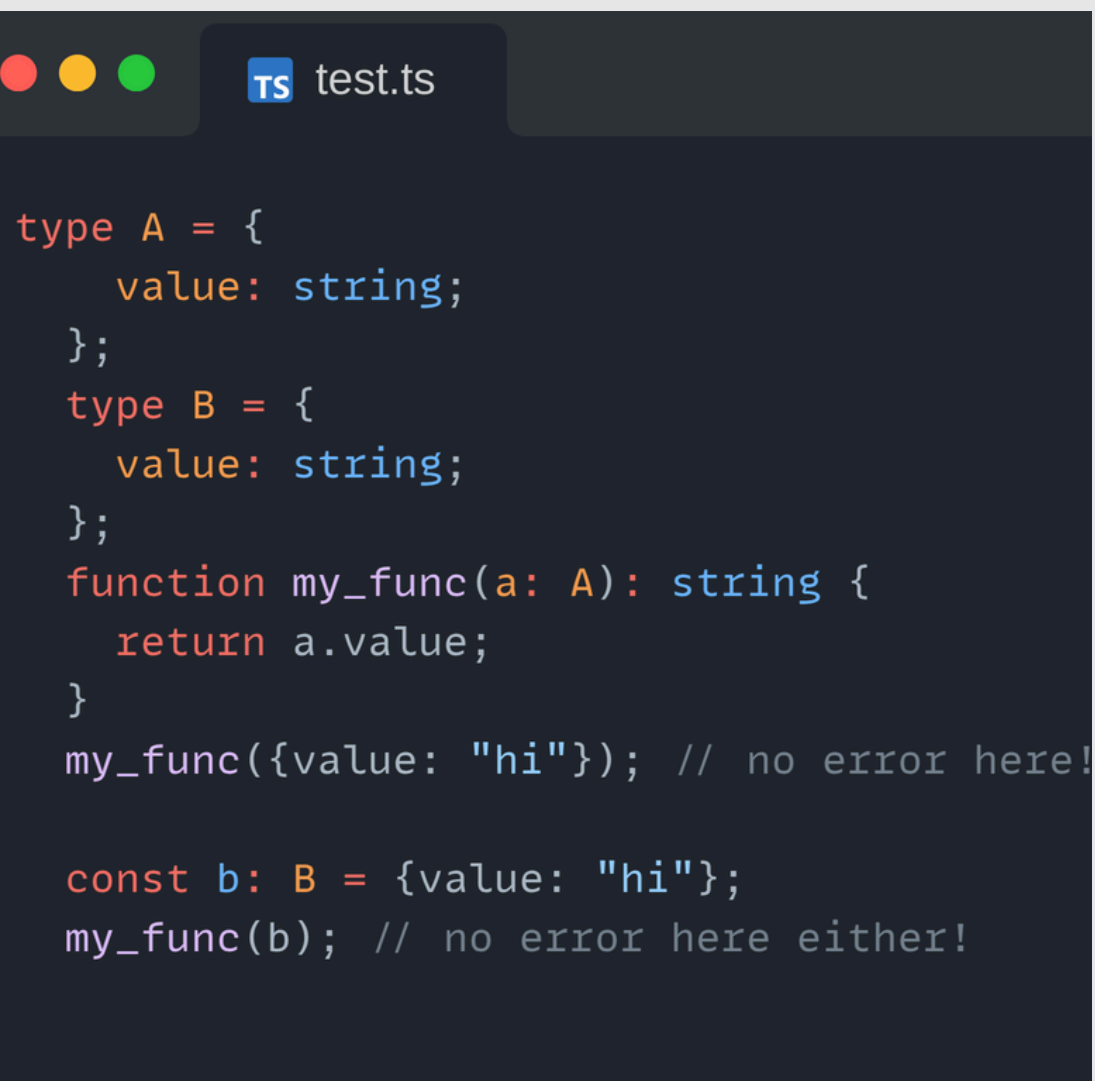
	Structural typing	Nominative typing
TypeScript	default behavior	Use tagged unions
Python with mypy	Use protocols	default behavior

Next Slide



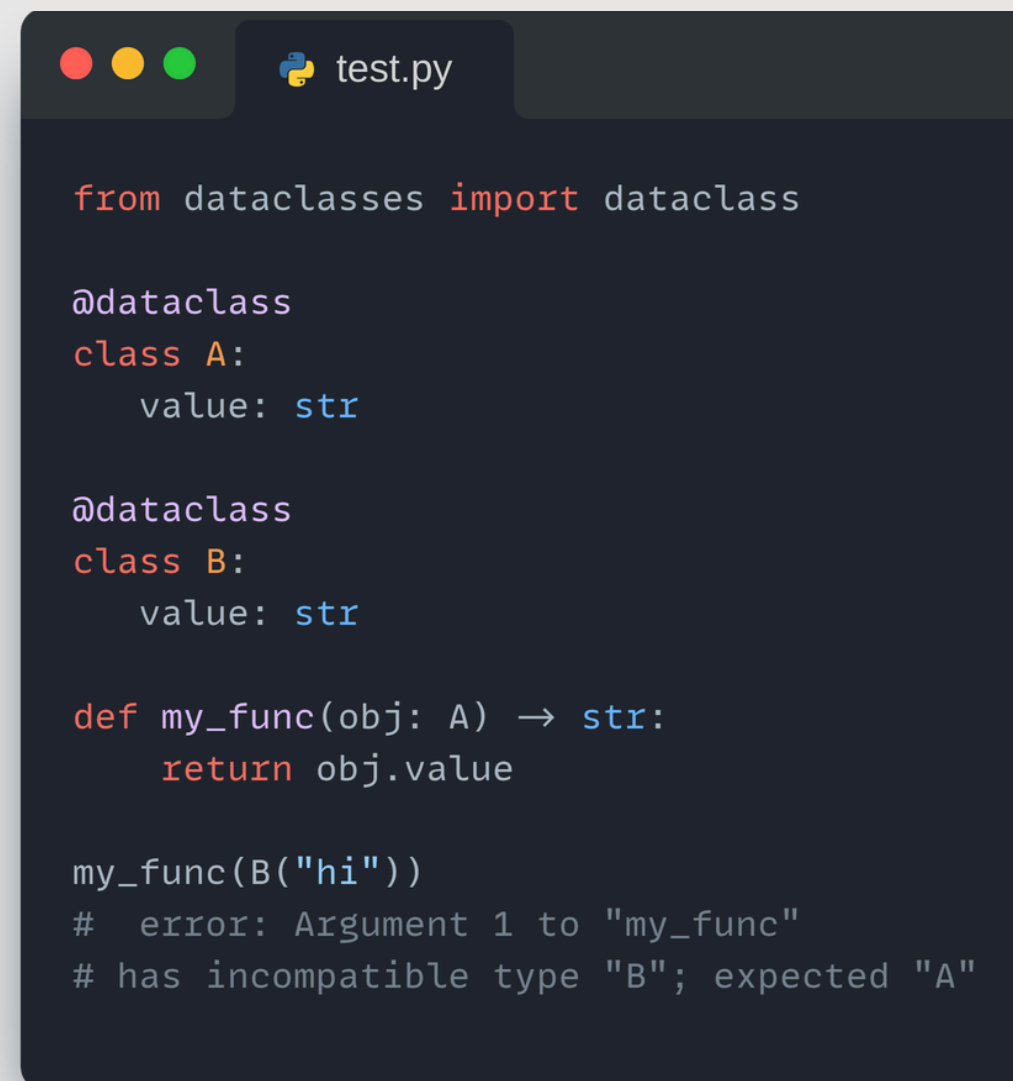
# Default Behaviors

## Structural Typing in TypeScript

A screenshot of a code editor window titled 'test.ts' with a TypeScript icon. The code defines two interfaces, A and B, both with a 'value' property of type 'string'. A function 'my\_func' is defined to take an argument of type A and return a string. Two calls to 'my\_func' are shown: one with a literal object and one with a variable 'b' of type B. Both calls are commented as having no error.

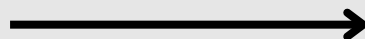
```
type A = {  
  value: string;  
};  
type B = {  
  value: string;  
};  
function my_func(a: A): string {  
  return a.value;  
}  
my_func({value: "hi"}); // no error here!  
  
const b: B = {value: "hi"};  
my_func(b); // no error here either!
```

## Nominal Typing in Python

A screenshot of a code editor window titled 'test.py' with a Python icon. The code uses 'dataclass' to define two classes, A and B, both with a 'value' attribute of type 'str'. A function 'my\_func' is defined to take an argument of type A and return a string. A call to 'my\_func' with a B object is shown, followed by a comment indicating a runtime error because the argument's type 'B' is incompatible with the expected type 'A'.

```
from dataclasses import dataclass  
  
@dataclass  
class A:  
    value: str  
  
@dataclass  
class B:  
    value: str  
  
def my_func(obj: A) → str:  
    return obj.value  
  
my_func(B("hi"))  
# error: Argument 1 to "my_func"  
# has incompatible type "B"; expected "A"
```

Next Slide



# Useful Links

- [TypeScript for Pythonistas](#)
- [Python Type Checking \(Guide\)](#)
- [Why is TypeScript surpassing Python?](#)
- [Static vs Dynamic Typing: A Detailed Comparison](#)
- [Type Checking With Mypy](#)
- [What is the difference between a strongly typed language and a statically typed language?](#)
- [Structural vs. Nominal Types](#)
- [Typed Python For TypeScript Developers](#)