

Como puedo hacer para que en el siguiente código en las funciones TextBoxHTMLControl y ButtonHTMLControl en vez de especificar que devuelve un string (cosa que si hace pero son elementos html) se pueda simular que regresa contenido html?

```
from abc import ABC, abstractmethod

class Control(ABC):
    @abstractmethod
    def print(self, text: str) -> str:
        pass

class TextBoxHTMLControl(Control):
    def print(self, text: str) -> str:
        return f"<input type='text' placeholder='{text}' />"

class ButtonHTMLControl(Control):
    def print(self, text: str) -> str:
        return f"<button>{text}</button>"
```

Buenísima pregunta 🙌 — esto es **100% un tema de tipado y diseño**, no de ejecución, y en Python hay varias formas “elegantes” de **expresar que lo que devuelves es HTML y no un string cualquiera**.

Primero, idea clave:

En Python **todo seguirá siendo un `str` en runtime**, lo que cambia es **cómo se lo comunicas al sistema de tipos (mypy, pyright, IDEs)**.

Te dejo **las opciones más usadas**, de más simple a más “pro”.

Opción 1: Crear un tipo semántico con `NewType` (simple y clara)

Esta es la más común cuando quieres **distinguir conceptualmente HTML de texto plano**.

python

```
from abc import ABC, abstractmethod
from typing import NewType
```

```
HTML = NewType("HTML", str)
```

```
class Control(ABC):
    @abstractmethod
    def print(self, text: str) -> HTML:
        pass
```

```
class TextBoxHTMLControl(Control):
    def print(self, text: str) -> HTML:
        return HTML(f"<input type='text' placeholder='{text}' />")
```

 Copiar código

```
class ButtonHTMLControl(Control):
    def print(self, text: str) -> HTML:
        return HTML(f"<button>{text}</button>")
```

Ventajas

- Muy legible
- IDEs y type checkers entienden que **no es un string cualquiera**
- Cero impacto en runtime

Desventaja


- Sigue siendo `str` internamente (pero eso es normal y esperado)

Opción 2: Usar una clase `Html` (más orientado a OOP)

Si quieres que el HTML sea un **objeto con identidad propia**, esta es buena.

python

```
class Html(str):
    pass
```

 Copiar código

python

```
class Control(ABC):
    @abstractmethod
    def print(self, text: str) -> Html:
        pass
```

```
class TextBoxHTMLControl(Control):
    def print(self, text: str) -> Html:
        return Html(f"<input type='text' placeholder='{text}' />")
```

```
class ButtonHTMLControl(Control):
    def print(self, text: str) -> Html:
        return Html(f"<button>{text}</button>")
```

 Copiar código

Ventajas

- Puedes agregar métodos luego (`escape()` , `render()` , etc.)
- Muy expresivo

Desventaja

- Es un poco más pesado que `NewType`

Opción 3: Usar Protocol (cuando quieres abstracción avanzada)

Útil si **no te importa la implementación**, solo que "se comporte como HTML".

python

```
from typing import Protocol

class HTML(Protocol):
    def __str__(self) -> str: ...
```

 Copiar código

python

```
class Control(ABC):
    @abstractmethod
    def print(self, text: str) -> HTML:
        pass
```

 Copiar código

Esto es más raro para este caso, pero existe.

Opción 4 (bonus): Si usas librerías web reales

Muchas librerías ya tienen tipos para esto:

- `markupsafe.Markup` (Flask, Jinja2)
- `LiteralString` (Python 3.11+, para HTML seguro)

Ejemplo con Markup:

python

```
from markupsafe import Markup
```

```
def print(self, text: str) -> Markup:
    return Markup(f"<button>{text}</button>")
```

 Copiar código

¿Cuál te recomiendo?

♦ Para tu ejemplo:

👉 `NewType("HTML", str)`

Es limpio, claro y exactamente lo que buscas: *“esto es HTML, no texto plano”*.

Si quieres, en el próximo mensaje puedo:

- Ajustar esto para **MVC / UI frameworks**
- Mostrar cómo evitar mezclar HTML con texto plano
- O hacer que `Control` sea genérico (`Control[T]`) 💡



Para que sirve Protocol?

Buenísima pregunta 😊 — Protocol suele confundir al principio porque **no sirve para lo mismo que una clase abstracta**, aunque se parezcan por fuera.

Vamos por partes, con intuición primero y técnica después.

¿Qué es Protocol ?

Protocol sirve para **tipado estructural**, también conocido como:

“Si camina como pato y hace cuac, es un pato” 🦆

Es decir:

- 👉 No importa de qué clase sea un objeto,
- 👉 importa qué métodos y atributos tenga.

Si los tiene → el type checker lo acepta.

Ejemplo rápido

python

```
from typing import Protocol

class Printable(Protocol):
    def print(self) -> str:
    ...
```

 Copiar código

Ahora cualquier objeto que tenga un método `print()` que devuelva `str` **es considerado** `Printable`, aunque no herede de nada.

python

```
class A:
    def print(self) -> str:
        return "hola"
```

```
class B:
    def print(self) -> str:
        return "chau"
```

```
def render(x: Printable):
    print(x.print())
```

 Copiar código

```
render(A()) # OK
render(B()) # OK
```

 **Ninguna clase hereda de** Printable , pero funciona.

¿En qué se diferencia de una ABC?

Con ABC (abc . ABC)

python

```
class Control(ABC):
    @abstractmethod
    def print(self) -> str:
        pass
```

 Copiar código

- ✓ Requiere herencia explícita
- ✓ Se valida en runtime
- ✗ Más acoplado

Con Protocol

python

```
class Control(Protocol):
    def print(self) -> str:
```

 Copiar código

...

- ✓ No requiere herencia
- ✓ Más flexible
- ✓ Ideal para APIs y librerías
- ✗ Solo lo valida el type checker (mypy, pyright)

¿Cuándo usar Protocol ?

1 Cuando no controlas las clases

Ej: objetos de librerías externas

python

```
def draw(control: Control):
    print(control.print())
```

 Copiar código

No puedes forzar que esas clases hereden de tu ABC, pero sí puedes **describir lo que esperas**.

2 Cuando te importa el comportamiento, no la jerarquía

Muy común en:

- UI
- Plugins
- Adaptadores
- Código desacoplado

3 Para evitar herencia artificial

Sin Protocol :

python

```
class UserButton(Button, Clickable, Printable, Serializable):
```

 Copiar código

...


Con Protocol :

👉 ninguna herencia innecesaria

Tu caso del HTML (ejemplo realista)

python

```
class HTML(Protocol):
    def __html__(self) -> str:
        ...
```

 Copiar código

python

```
class Control(Protocol):
    def print(self, text: str) -> HTML:
        ...
```

 Copiar código

Ahora **cualquier objeto que sepa renderizar HTML** sirve:

- un str
- un Markup
- una clase custom
- un wrapper de seguridad

Resumen rápido

Concepto	ABC	Protocol
Herencia obligatoria	✓	✗
Validación en runtime	✓	✗
Tipado estructural	✗	✓
Ideal para APIs	😐	✓

Concepto	ABC	Protocol
Desacoplamiento	😐	🔥

Regla práctica 🧠

👉 Usa Protocol cuando diseñes contratos

👉 Usa ABC cuando diseñes jerarquías

Si quieres, puedo:

- Reescribir tu ejemplo HTML usando Protocol
- Mostrar un bug real que Protocol evita
- Compararlo con interfaces en Java / C#

Tú manda 😊



En python como puedo indicar que una variable puede ser de dos tipos diferentes con typing?

En Python se hace con Union (o con | , que es la forma moderna). Es súper común y muy útil 🙌

Forma moderna (Python 3.10+ ✅)

python

```
x: int | str
```

📋 Copiar código

Eso significa:

Ejemplo:

python

```
def parse(value: int | str) -> int:
    if isinstance(value, int):
        return value
    return int(value)
```

 Copiar código

Forma clásica (Python < 3.10)

python

```
from typing import Union
```

```
x: Union[int, str]
```

 Copiar código

Es exactamente lo mismo, solo más verboso.

Casos comunes y útiles

1 Valor opcional (None)

Muy típico:

python

```
name: str | None
```

 Copiar código

Equivale a:

python


```
Optional[str]
```

 Copiar código

Ejemplo:

python

```
def greet(name: str | None):  
    if name is None:  
        print("Hola desconocido")  
    else:  
        print(f"Hola {name}")
```

 Copiar código

2 Tipos más complejos

python

```
data: dict[str, int] | list[int]
```

 Copiar código

3 Con clases propias

python

```
class A: ...  
class B: ...
```

```
x: A | B
```

 Copiar código

4 Con Protocol (muy poderoso)

python

```
from typing import Protocol  
  
class Renderable(Protocol):  
    def render(self) -> str: ...  
  
def draw(x: Renderable | str):  
    ...
```

 Copiar código

Tipado + runtime (importante ⚠️)

El type hint **no hace validación automática**.

❌ Esto NO lanza error:

```
python
```

```
x: int | str = 3.14
```

 Copiar código

✓ El error lo detecta:

- mypy
- pyright
- el IDE

Si quieres validación real:

```
python
```

```
if not isinstance(x, (int, str)):
    raise TypeError()
```

 Copiar código

Mini resumen mental 🧠

- $A \mid B \rightarrow$ puede ser A o B
- None se tipa explícitamente
- $|$ es preferido hoy
- Union sigue existiendo para compatibilidad
- Es solo para **tipado estático**

Si quieres, dime:

- tu versión de Python
 - si lo estás usando con mypy o pyright
- y te muestro ejemplos más finos (genéricos, narrowing, overload, etc.) 😊



