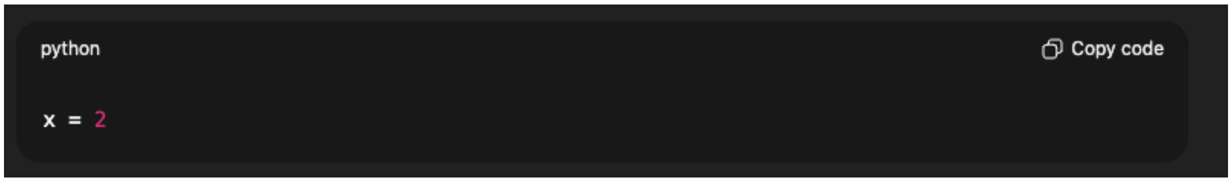
What is a Variable?

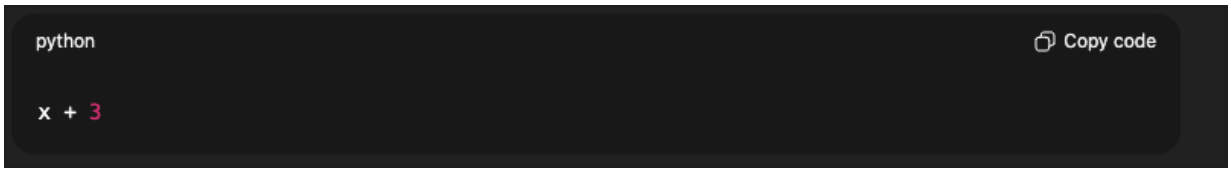
In Python, a **variable** is like a box or a labeled container that holds some kind of value such as a number or text.

We can imagine putting a value inside that box and giving the box a name, so we can use it later.



```python
x = 2
```

This line says: "Put the number **2** inside a box labeled **x**."
Now, anytime we use x, Python knows it means **2**.
We can use x in math expressions just like a number:

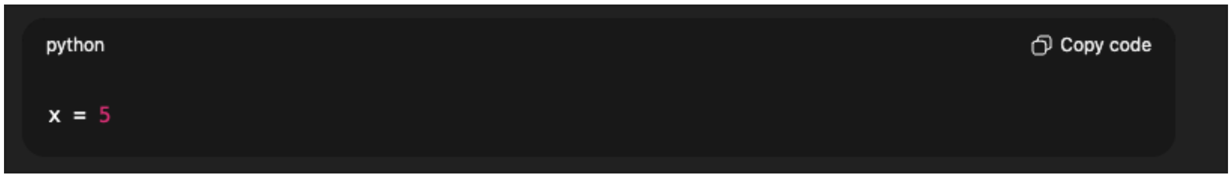


```python
x + 3
```

How Assignment Works

The = sign in Python doesn't mean "equal" like in math it means **assign**.
It tells Python to **store** the value on the right side into the variable on the left side.

So:

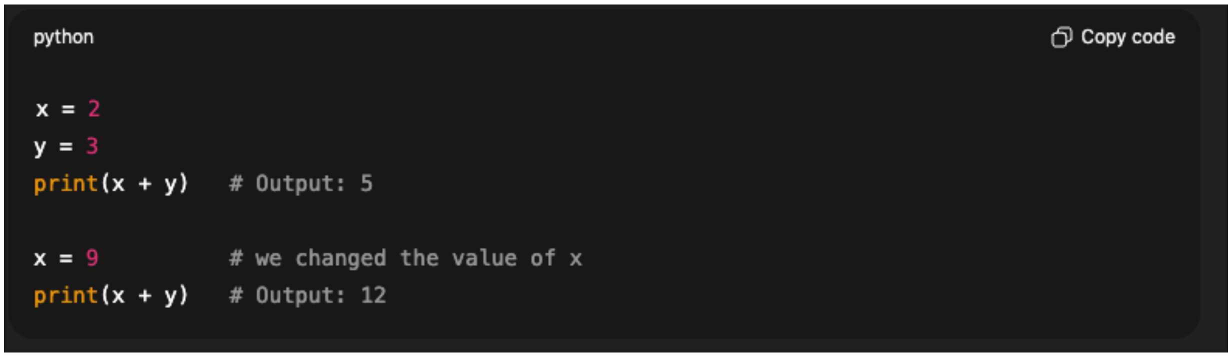


```python
x = 5
```

means "assign 5 to x," not "x equals 5" as a math statement.

Changing Variable Values

Variables are **mutable**, meaning we can change what's inside them.
If we decide to update the value of x, we can simply reassign it:



```python
x = 2
y = 3
print(x + y)   # Output: 5

x = 9          # we changed the value of x
print(x + y)   # Output: 12
```

Now x no longer holds 2 — it holds 9.
This ability to reassign values makes variables flexible and powerful.

Naming Variables

You can name a variable almost anything you want, but there are some rules:

- Variable names can include letters, numbers, and underscores (_)
- They **cannot start with a number**
- They **cannot contain spaces** or special characters like -, /, or @
- Python is **case-sensitive**, so Name, name, and NAME are three different variables

✅ Good variable names:
age, total_price, x, my_number

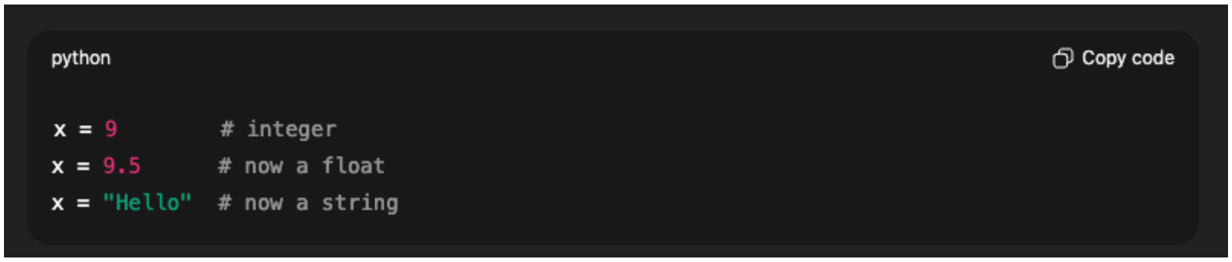
❌ Invalid variable names:
2x, my variable, price$

Choose names that describe what the value represents it makes code easier to read.

Python's Dynamic Typing

In some programming languages, you must say what kind of data a variable will hold.
In Python, you don't — Python figures it out automatically based on what you assign.

Example:



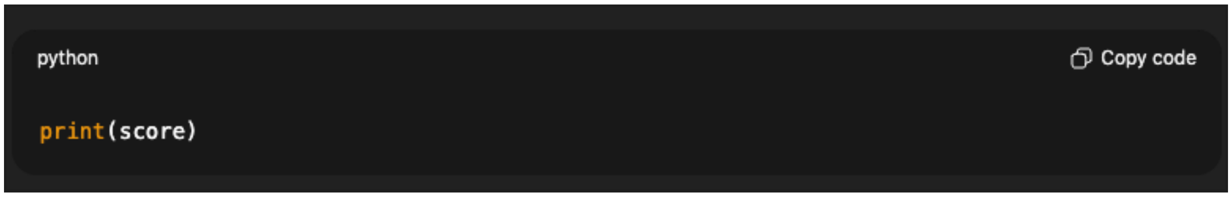
```python
x = 9        # integer
x = 9.5      # now a float
x = "Hello"  # now a string
```

Python changes the variable's **type** depending on what's stored inside it.
This is called **dynamic typing**, and it's one of the reasons Python is great for beginners you can focus on logic, not setup.

Undefined Variables and Errors

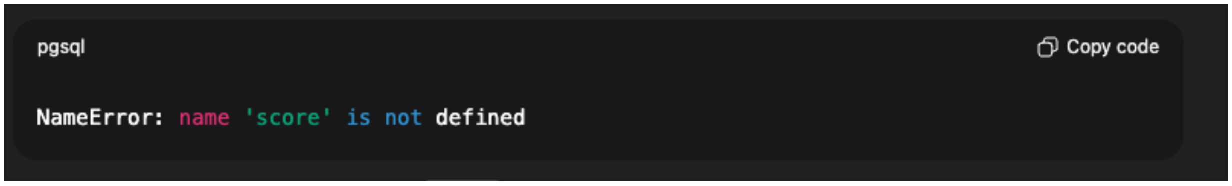
If you try to use a variable before it's created, Python will give you an error.

Example:



```python
print(score)
```

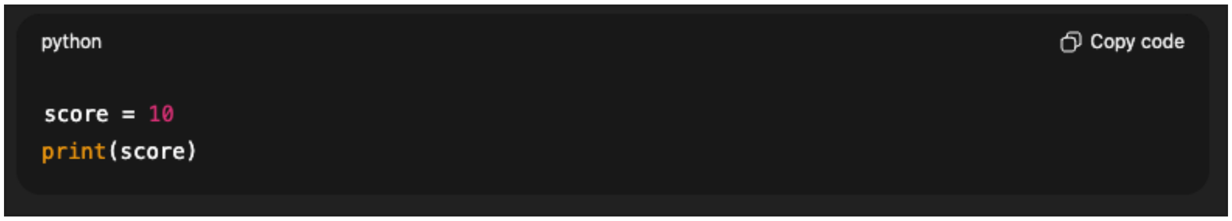
Output:



```
NameError: name 'score' is not defined
```

This means Python has no idea what score is because we haven't created it yet.

✅ Correct way:



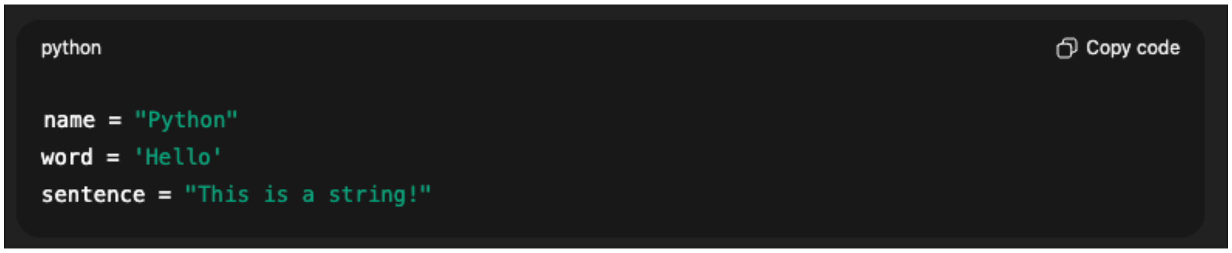
```python
score = 10
print(score)
```

What is a String?

So far, we've worked with **numbers** integers and floats.
Now we'll learn a new type of data: **strings**.

A **string** is simply a sequence of characters — letters, numbers, or symbols — surrounded by quotes.

Examples:

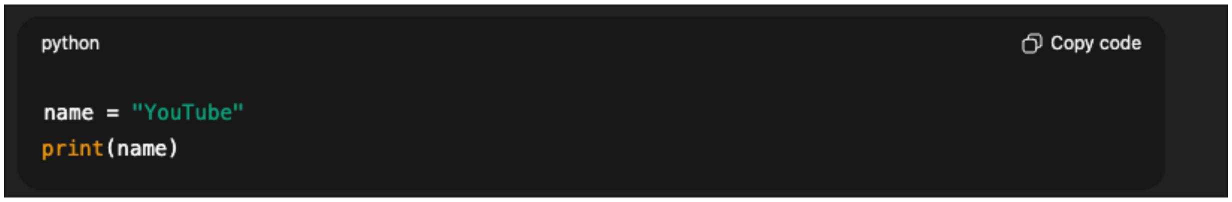


```python
name = "Python"
word = 'Hello'
sentence = "This is a string!"
```

You can use either **single quotes (' ')** or **double quotes (" ")**, but they must match.

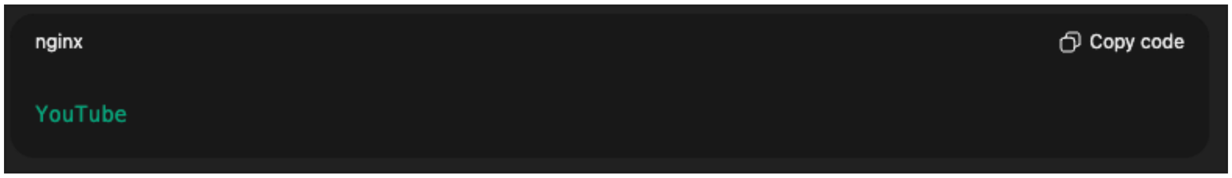
Printing Strings

When you print a string, Python shows the text inside:



```python
name = "YouTube"
print(name)
```

Output:



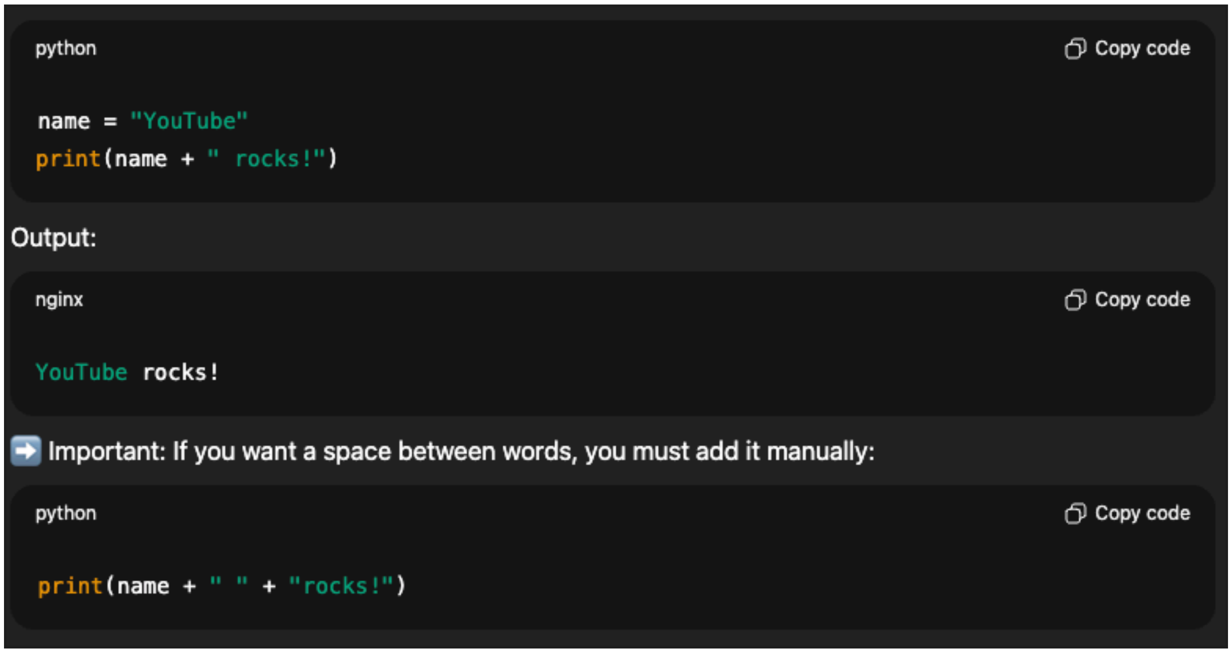
```
YouTube
```

Here, the variable name stores a string, and print(name) displays it.

Joining (Concatenating) Strings

You can **combine strings** using the + operator. This is called **concatenation**.

Example:



```python
name = "YouTube"
print(name + " rocks!")
```

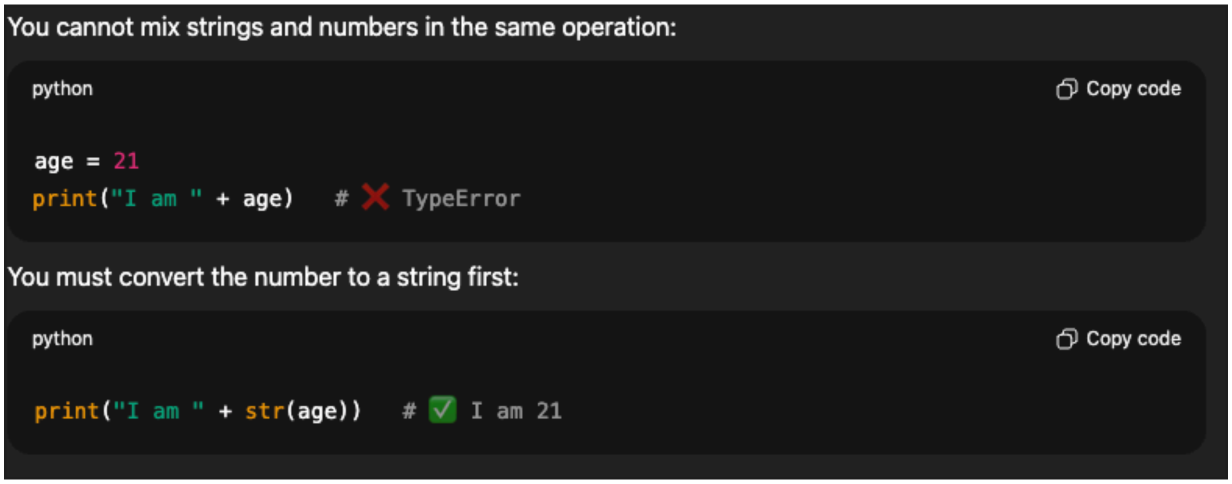
Output:

```nginx
YouTube rocks!
```

➡️ Important: If you want a space between words, you must add it manually:

```python
print(name + " " + "rocks!")
```

Strings vs. Numbers



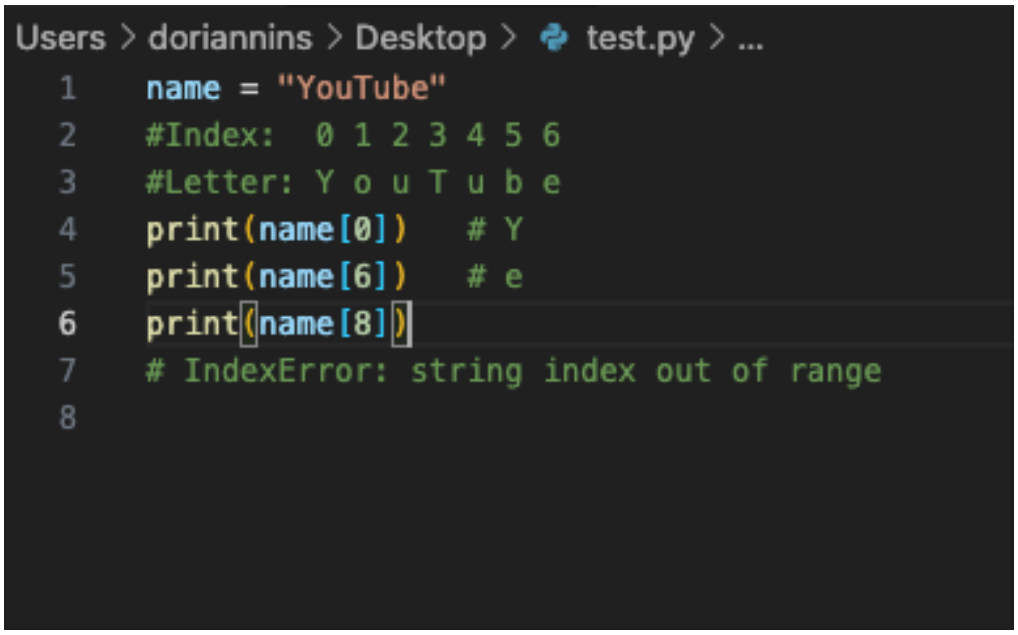
You cannot mix strings and numbers in the same operation:

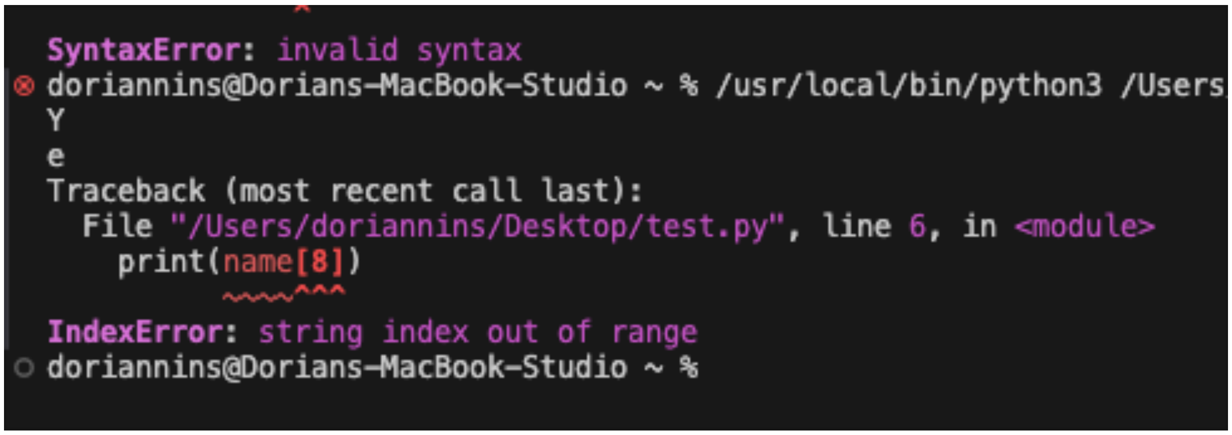
```python
age = 21
print("I am " + age)    # ❌ TypeError
```

You must convert the number to a string first:

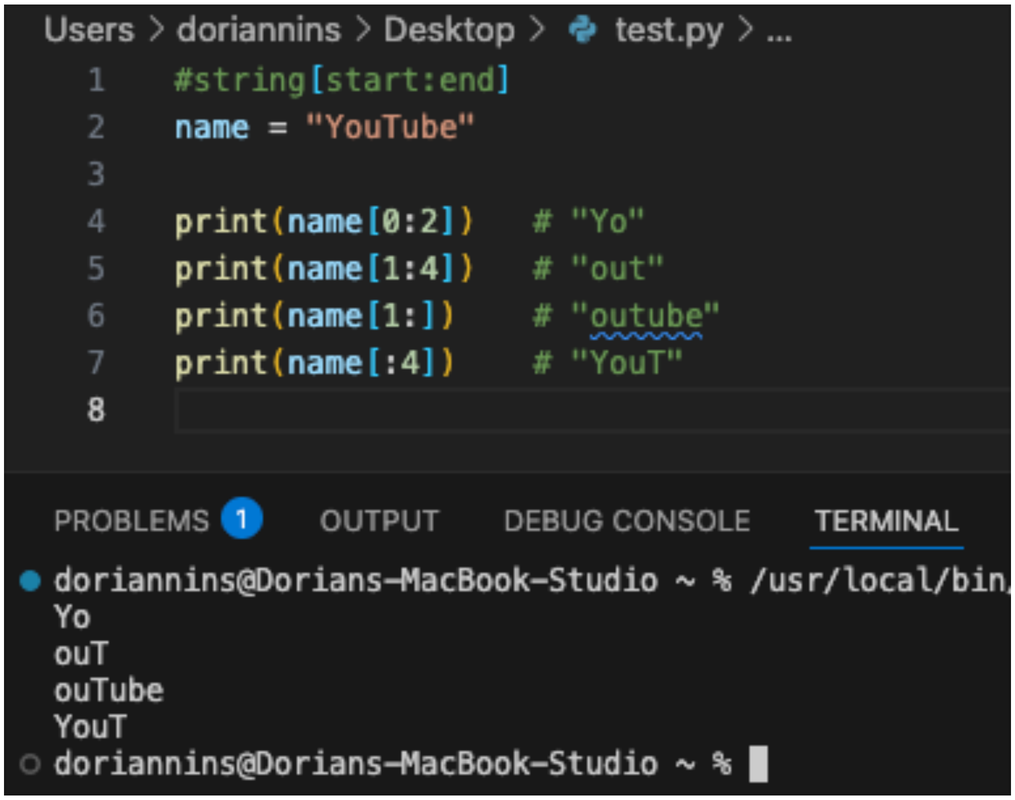
```python
print("I am " + str(age))    # ✅ I am 21
```

Accessing and Slicing Strings


```
Users > doriannins > Desktop > test.py > ...
name = "YouTube"
#Index:  0 1 2 3 4 5 6
#Letter: Y o u T u b e
print(name[0])    # Y
print(name[6])    # e
print(name[8])
# IndexError: string index out of range
```


```
SyntaxError: invalid syntax
doriannins@Dorians-MacBook-Studio ~ % /usr/local/bin/python3 /Users
Y
e
Traceback (most recent call last):
  File "/Users/doriannins/Desktop/test.py", line 6, in <module>
    print(name[8])
          ~~~~^^^
IndexError: string index out of range
doriannins@Dorians-MacBook-Studio ~ %
```

String Slicing



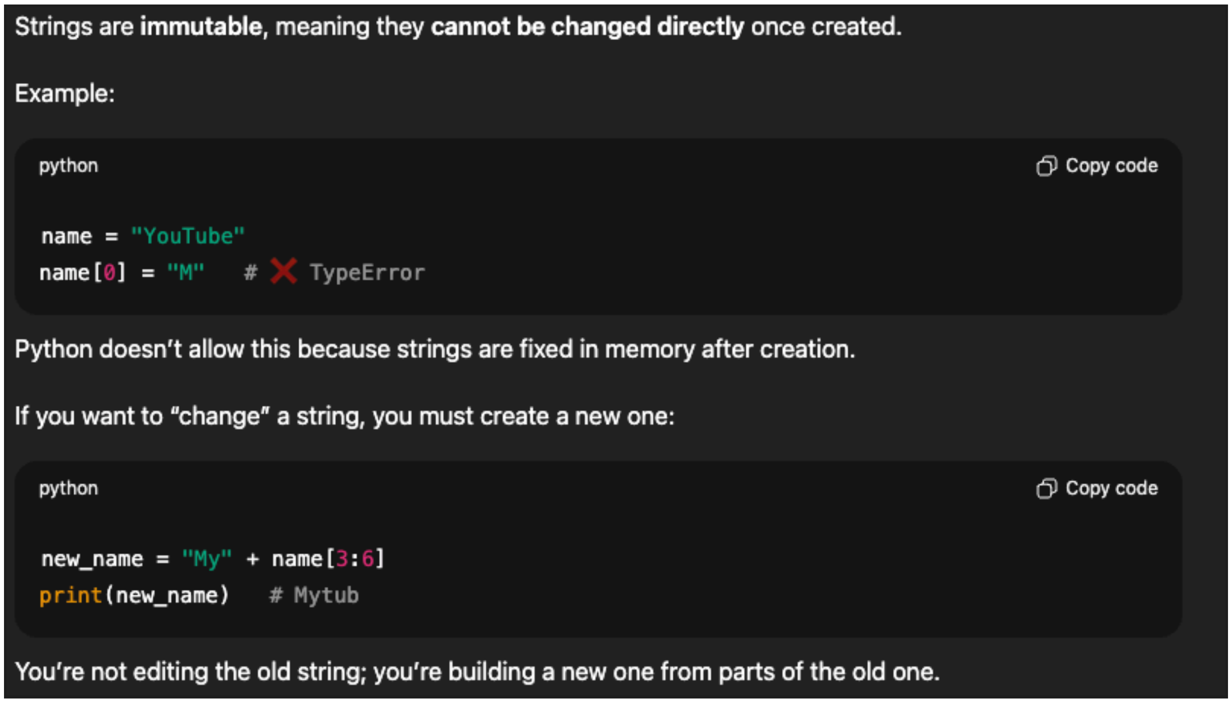
```python
#string[start:end]
name = "YouTube"

print(name[0:2])    # "Yo"
print(name[1:4])    # "out"
print(name[1:])     # "outube"
print(name[:4])     # "YouT"

```

```
doriannins@Dorians-MacBook-Studio ~ % /usr/local/bin
Yo
ouT
ouTube
YouT
doriannins@Dorians-MacBook-Studio ~ %
```

String Immutability



Strings are **immutable**, meaning they **cannot be changed directly** once created.

Example:

```python
name = "YouTube"
name[0] = "M"    # ❌ TypeError
```

Python doesn't allow this because strings are fixed in memory after creation.

If you want to "change" a string, you must create a new one:

```python
new_name = "My" + name[3:6]
print(new_name)    # Mytub
```

You're not editing the old string; you're building a new one from parts of the old one.

Concept	Description	Example
Variable	A name that stores a value	x = 5

Assignment	Giving a variable a value with =	name = "Bob"
Dynamic Typing	Python guesses the type automatically	x = 9.5
NameError	Happens when using an undefined variable	print(x) before assigning
String	A sequence of text in quotes	"Hello"
Concatenation	Joining strings with +	"Hi" + " there"
Indexing	Getting one character using brackets	name[0]
Negative Indexing	Count characters from the end	name[-1]
Slicing	Extracting parts of a string	name[1:4]
Immutability	Strings cannot be changed directly	name[0] = "a" ❌
len()	Finds the number of characters	len("Python") → 6