

Strings

Python Reference

Indexing

A string is text inside quotes. Each character has a position, called its index . The first index is 0, not 1.

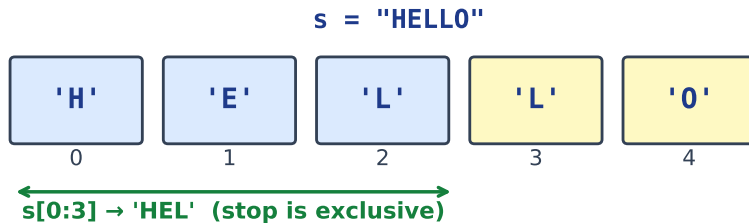
Read one character with square brackets:

```
word = "Python"
print(word[0])    # P  (the first character)
print(word[2])    # t
print(len(word))  # 6  (how many characters)
```

- Counting starts at 0, so the last index is `len(word) - 1`.
- A negative index counts back from the end: `word[-1]` is the last character.

```
word = "Python"
print(word[-1])   # n
print(word[-2])   # o
```

- An index that is too large gives an error (an `IndexError`).



Exam tip: slice is half-open [start, stop); omit ends for defaults; step is optional

`s[start:stop]` is a half-open window of characters

Slicing

A slice takes a part of a string. Write `word[start:end]`. The slice keeps `start` but stops **before** `end`.

```
word = "Python"
print(word[0:3])  # Pyt  (positions 0, 1, 2)
print(word[2:5])  # tho
```

- Leave out `start` to begin at 0; leave out `end` to go to the end.

```
word = "Python"
print(word[:3])   # Pyt
print(word[3:])   # hon
```

- A third number is the step . `word[::-1]` reverses the string.

```
print("Python"[::-1])  # nohtyP
```

String methods & length

A method is a function that belongs to a value. You call it with a dot:

```
name = "mei chen"
print(name.upper())    # MEI CHEN
print(name.title())    # Mei Chen
print(len(name))       # 8
```

Strings are immutable : a method returns a **new** string and never changes the original one.

```
name = "mei"
print(name.upper())    # MEI  (the returned value)
print(name)            # mei  (the original is unchanged)
```

Common methods (each returns a new value):

Method	Meaning	Example	Result
<code>.upper()</code> / <code>.lower()</code>	change case	<code>"Hi".lower()</code>	hi
<code>.strip()</code>	remove edge spaces	<code>" hi ".strip()</code>	hi
<code>.replace(a, b)</code>	swap text	<code>"cat".replace("c", "b")</code>	bat
<code>.split(sep)</code>	break into a list	<code>"a,b".split(",")</code>	['a', 'b']

Join strings with `+`. This is called concatenation :

```
first = "Mei"  
last = "Chen"  
print(first + " " + last)    # Mei Chen
```

f-strings

An f-string builds text from values. Put `f` before the quote, then write `{...}` around a value.

```
name = "Mei"  
age = 17  
print(f"{name} is {age} years old")    # Mei is 17 years old
```

- Any expression can go inside the braces.
- `{value:.2f}` rounds to 2 decimal places .

```
price = 9.5  
print(f"Two cost {price * 2}")          # Two cost 19.0  
print(f"Pi is about {3.14159:.2f}")    # Pi is about 3.14
```

Common mistakes

- Strings cannot be changed in place: `s[0] = "x"` raises an error. Build a new string instead.
- Indexing starts at 0; the last character is `s[-1]`, and `s[len(s)]` is out of range.
- A slice `s[a:b]` includes `a` but stops **before** `b`.
- String methods return a NEW string: `s.upper()` does nothing unless you store the result.