

CSC 141 / FALL 2026

Strings: indexing, slicing, and traversal

CSC 141 · Class 16

Thursday, October 22, 2026

Dr. Si Chen

Where we are

Loops: while, for, nested loops, accumulator and counting.

So far the loops ran over numbers from `range()`.

Today: a string is a sequence of characters — loops run over it too.

Plus two new tools: indexing (one character) and slicing (a piece).

Today's learning goals

Read one character with an index, including negative indexes.

Take a piece of a string with a slice.

Loop over a string with for, with and without an index.

Count, search, and build new strings with loop patterns.

A string is a sequence

```
word = "python"
```

character	p	y	t	h	o	n
index	0	1	2	3	4	5
negative	-6	-5	-4	-3	-2	-1

`len(word)` is 6; the last index is 5.

Indexing: one character

```
word = "python"
print(word[0])      # p
print(word[5])      # n
print(word[-1])     # n
print(word[-2])     # o
print(len(word))    # 6
print(word[len(word) - 1]) # n
print(word[6])      # IndexError
```

`word[i]` is the character at position `i`.

`word[-1]` is the last one.

`IndexError`:
string index out of range.

Slicing: a piece of a string

```
word = "python"
print(word[0:3])      # pyt
print(word[2:5])      # tho
print(word[:2])       # py
print(word[3:])       # hon
print(word[-3:])      # hon
print(word[::-1])     # nohtyp
```

`word[start:stop]`

start included,
stop excluded,
just like `range()`.

Leave start
empty: from the
beginning. Leave
stop empty: to
the end.

Slicing rules

slice	result	why
<code>word[0:3]</code>	<code>pyt</code>	indexes 0, 1, 2
<code>word[1:4]</code>	<code>yth</code>	indexes 1, 2, 3
<code>word[:4]</code>	<code>pyth</code>	missing start = 0
<code>word[4:]</code>	<code>on</code>	missing stop = end
<code>word[2:2]</code>	<code>(empty)</code>	nothing between 2 and 2
<code>word[0:100]</code>	<code>python</code>	never an IndexError

Traversal: for over the characters

```
word = "python"  
for ch in word:  
    print(ch)
```

ch takes each character in order.

No index, no range().

Use this when you only need the characters.

Traversal: for with an index

```
word = "python"  
for i in range(len(word)):  
    print(i, word[i])
```

Output:

0 p

1 y

2 t

...

Use this when
you need the
position too.

Pattern: counting characters

```
def count_vowels(s):  
    count = 0  
    for ch in s:  
        if ch in "aeiou":  
            count += 1  
    return count  
  
print(count_vowels("programming")) # 3  
print(count_vowels(""))           # 0
```

The counting pattern over a string.

`ch in "aeiou"`
asks: is `ch` one of these?

Empty string: the loop runs zero times.

Pattern: building a new string

```
word = "python"
```

```
result = ""  
for ch in word:  
    result = ch + result  
print(result)          # nohtyp
```

```
loud = ""  
for ch in word:  
    loud += ch.upper()  
print(loud)            # PYTHON
```

The accumulator
with a string.

Start with ""
(empty).

ch + result puts
each character
in FRONT.

Trace: building the reverse

ch	result before	result after
p	""	"p"
y	"p"	"yp"
t	"yp"	"typ"
h	"typ"	"htyp"
o	"htyp"	"ohtyp"
n	"ohtyp"	"nohtyp"

Strings cannot be changed in place

```
word = "python"  
word[0] = "P"           # TypeError
```

Strings are
immutable.

```
# make a NEW string instead  
word = "P" + word[1:]  
print(word)             # Python
```

Indexing reads; it
cannot write.

Build a new
string and assign
it back.

Useful string methods

call	result	notes
<code>"Hello".upper()</code>	<code>"HELLO"</code>	
<code>"Hello".lower()</code>	<code>"hello"</code>	comparing
<code>" hi ".strip()</code>	<code>"hi"</code>	trims
<code>"banana".count("a")</code>	<code>3</code>	
<code>"banana".find("n")</code>	<code>2</code>	-1 if none
<code>"banana".replace("a","o")</code>	<code>"bonono"</code>	new string

Pattern: searching with a loop

```
def first_index(s, target):  
    for i in range(len(s)):  
        if s[i] == target:  
            return i  
    return -1
```

```
print(first_index("banana", "n"))    # 2  
print(first_index("banana", "z"))    # -1
```

return inside the
loop is right
HERE: stop at
the first match.

return -1 after
the loop: nothing
found.

This is what
find() does.

Common string bugs

Off by one in a slice: `word[0:3]` has 3 characters, indexes 0, 1, 2.

Case: `"A" in "aeiou"` is `False` — use `ch.lower()` first.

`input()` always returns a string: `"5" + 5` is a `TypeError`; use `int()`.

Forgetting the result: `s.upper()` returns a new string, it does not change `s`.

Guided build: is_palindrome

Write `is_palindrome(s)`: True if `s` reads the same backwards.

Ignore case: "Level" counts.

Plan: 1. make a lowercase copy. 2. build (or slice) the reverse. 3. compare.

Which tool: a loop that builds the reverse, or `s[::-1]`?

Either works.

is_palindrome test cases

input	expected
"racecar"	True
"Level"	True
"python"	False
""	True
"ab"	False

is_palindrome solution

```
def is_palindrome(s):  
    s = s.lower()  
    reverse = ""  
    for ch in s:  
        reverse = ch + reverse  
    return s == reverse  
  
print(is_palindrome("Level"))      # True  
print(is_palindrome("python"))    # False
```

lower() first, so
"Level" works.

The reverse is
the string
accumulator.

Shortcut: return
`s == s[::-1]`

Independent practice

Use `string_practice_starter.py`.

`count_char(s, c)`: how many times `c` appears in `s`

`count_char("banana", "a")` is 3

`first_half(s)`: the first half of `s`, `len(s) // 2`

`first_half("python")` is "pyt"

`shout(s)`: upper case, "!" after each character

`shout("hi")` is "H!!!"

Exit ticket

What is `"python"[1:4]`?

What is `"python"[-1]`?

Write the loop header that visits each character of a string `s`.

Why does `word[0] = "P"` fail?

Before next class

Finish the independent practice and run every test.

Playground, Class 16: every slide program plus both starters.

PokéLab: finish any Loop Trails mission still open.

Next week: lists — a sequence that CAN be changed.