



Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 5: Sequences I (Strings and Lists)

Goals for today

- Create string literals using both single-quotes and double-quotes.
- Explain the purpose of and use backslash-escaping in string literals.
- Describe a string as an ordered *sequence* of characters.
- Explain and use sequence operators (indexing, slicing) to obtain subsequences (including individual characters) or transform a string.
- Apply knowledge of strings as a sequence to **lists**, a sequence of any type.

To follow along: create a **class05.py** script where we will write code together.
(in your **cs146** folder)

Follow-up from last class: we can also *nest* **for** loops!

How can we **print** these patterns?

```
% % % %  
% % % %  
% % % %  
% % % %  
% % % %  
% % % %
```

```
@  
@ @  
@ @ @
```

```
1 def print_rect(width, height, char):  
2     """Print an ASCII rectangle  
3  
4     Args:  
5         width: number of columns of rectangle  
6         height: number of rows of rectangle  
7         char: Character to print  
8     """  
9     for i in range(height):  
10         for j in range(width):  
11             print(char, end=" ")  
12         print() # new line at end of the row
```

```
1 def print_triangle(size, char):  
2     """Print an ASCII lower triangle  
3  
4     Args:  
5         size: number of rows/columns  
6         char: Character to print  
7     """  
8     for i in range(size):  
9         for j in range(i+1):  
10             print(char, end=" ")  
11         print()
```

Question 1: What will this code print?

```
1 sum = 0
2 for i in range(5):
3     for j in range(5):
4         sum = sum + i
5 print(sum)
```

- A. 10
- B. 50
- C. 225
- D. 500

i	
0	+ 0 + 0 + 0 + 0 + 0 + 0
1	+ 1 + 1 + 1 + 1 + 1 + 1
2	+ 2 + 2 + 2 + 2 + 2 + 2
3	+ 15
4	+ 20

0 + 5 + 10 + 15 + 20
= 50

Working with strings:

- Recap: `+` (concatenation: `str + str`), `*` (repetition: `int * str` or `str * int`).
- Strings can be delimited by either single- or double-quotes (e.g. `'hello'` or `"hello"`).
- This allows us to include the other in a string: `'Maybe a string has "quotes" in it'`
- We can "escape" characters using a backslash. This means double-quotes can be embedded into double-quote delimited strings with `\` (same idea with `\'`). Commonly used escape character is `\n` which means "start a new line in the string" (see `help(print)`).
- Accessing specific characters: use square brackets `[]` with *index*. Starts at `0`.
- `len(str)` returns number of characters. So we can use `[]` up to `len(s) - 1` (for some string `s`).
- We can also index the string with negative numbers: `s[-1]` gives the *last* character.
- Similar to `range(start, stop, step)` we can *slice* strings using `s[start:stop:step]`.
- We can think of strings as *sequences* of characters:

```
1 s = 'hello'
2 for i in range(len(s)):
3     # i is an integer
4     print(s[i])
```

```
1 s = 'hello'
2 for c in s:
3     # c is a character
4     print(c)
```

Question 2: What is the value of **s** after this code executes?

```
1 s = 'abc'  
2 s = 'd'*3+s  
3 s = s + ''*3  
4 s = s + 'q'
```

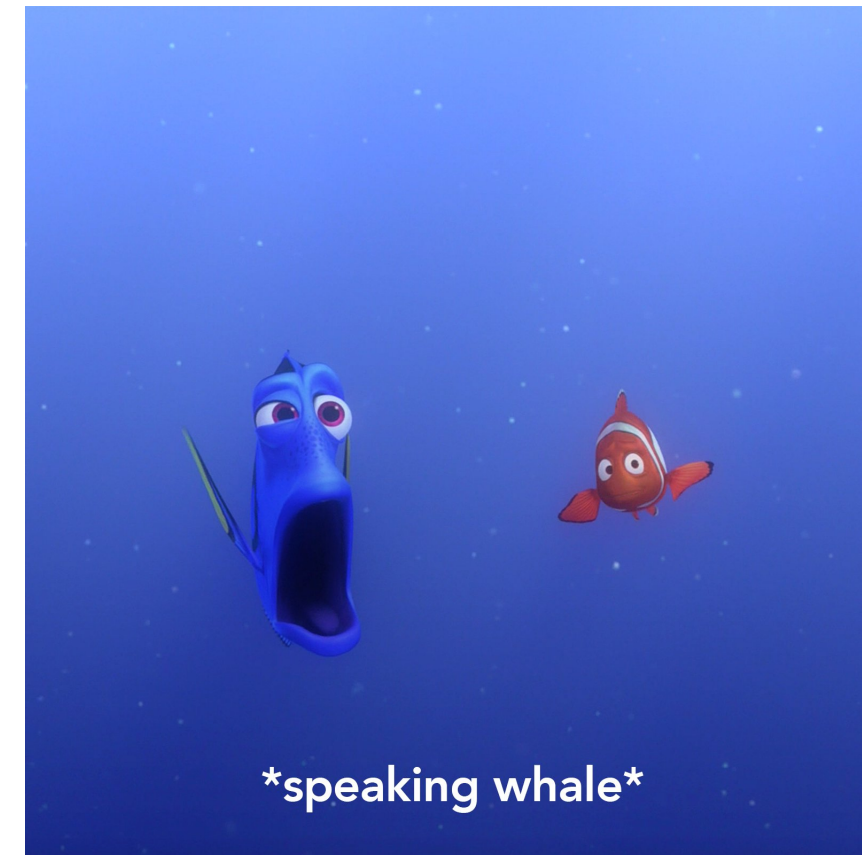
"dddabc"
no change
"dddabcq"

- A. "abcddd q"
- B. "abcddd' ' ' ' 'q"
- C. "abcdddq"
- D. "Odddabc"
- E. "dddabcq"

Investigate!

Using the string `message = 'Do you speak whale?'`

1. `message[13:18]`
2. `message[11:6]`
3. `message[11:6:-1]`
4. `message[::-1]`
5. `message[:5] + message[5:]`
6. `message[1::2]`



1. `'whale'`
2. `''` (an empty string)
3. `'kaeps'`
4. `'?elahw kaeps uoy oD'`
5. `'Do you speak whale?'`
6. `'oyusekwae?'`

Question 3: Which of the following would turn the string `s = "I love CS!"` into `"CS I love!"`?

A. `s[-4:-1] + s[1] + s[:6] + s[8]`

B. `s[-3:-1] + s[1] + s[-1]`

C. `s[-3:-1] + s[1] + s[:6] + s[-1]`



Example: writing a function **reverse** that returns the reverse of some input string **s** with/without slicing.

```
1 def reverse(s):
2     """
3     Returns the reverse of a string s
4
5     Args:
6         s: input string to reverse
7
8     Returns:
9         String with the characters of s reversed.
10    """
11    r = ''
12    for i in range(len(s)):
13        r = r + s[-(i + 1)]
```

```
1 def reverse_with_slice(s): # and no docstring
2     return s[::-1]
```

A **str** is a type of *sequence*.

And a *sequence* is an "Abstract Data Type" which means that we can expect certain functionalities to be implemented in any type that claims to be a sequence.

Think of the concept of a bag. What kinds of functions would we like from a bag?

- Hold things!
- Add/remove an item from the bag.
- Tell us how many items are in the bag.
- Empty the bag.
- A certain bag might be good at some of these functions, but not others.



Sequence (bag) is an ADT, **str** (backpack) is a data structure that implements the sequence interface.

Question 4: what does this function do?

```
1 def mystery(s):  
2     new_s = ""  
3     for c in s:  
4         new_s = c + new_s  
5     return new_s
```

- A. Return a copy of **s**
- B. Return the reverse of **s**
- C. Return a string consisting of only the first character of **s**
- D. Return a string consisting of only the final character of **s**

'hello'
↑ ↑
c ↑
new_s = 'h'
new_s = 'e' + new_s
= 'eh'
new_s = 'l' + new_s
= 'leh'
⋮
'olleh'

Question 5: What is the value of **val** after the above code executes?

```
1 val = 0
2 for i in 'abc':
3     for j in 'cde':
4         val += 1 # equivalent to val = val + 1
```

Handwritten annotations: Above line 1, 'a' is green, 'b' is blue, 'c' is orange, and a purple '3' is written. Above line 2, a purple bracket spans the inner loop, with a purple '3' written above it. A purple arrow points from the word 'check' to the inner loop. To the right of the code, a purple bracket spans lines 3 and 4, with a green '+3' written next to it. Below this, a blue '+3' and an orange '+3' are written, followed by a horizontal line and the number '9'.

- A. 1
- B. 3
- C. 6
- D. 9
- E. 27

A **list** is another data structure that implements the interface we expect for a sequence.

The main difference: **list** can store ANYTHING.

- Access items with **[]** and an index.
- Can get the number of items via **len**.
- We can iterate through items in a **list**, similar to **range** and **str**.
- We can also create new **lists** by *slicing*.
- We can even assign values into a **list** with **[]** and an index (unlike **str**).
This raises the concept of "mutability" (whether we can "mutate" or "change" an object).

```
>>> l = [7, 4, 3, 6, 1, 2]
[7, 4, 3, 6, 1, 2]
>>> list("abcd") # converts an input string to a list by splitting characters
['a', 'b', 'c', 'd']
>>> l = [1, 2.0, True, [3, 2, 4, 5]] # lists can store ANYTHING !!!
[1, 2.0, True, [3, 2, 4, 5]]
```

In the last example, we can retrieve the **4** using **l[3][2]**.

Remember: a **list** is mutable, so we can change the items, but **str** are not mutable.

```
>>> names = ["mike", "Sulley", "Randall"]
>>> names[2] = "Boo"
>>> names
['mike', 'Sulley', 'Boo']

>>> names[0][0] = "Mike" # try to fix capitalization
TypeError: 'str' object does not support item assignment
```

- We can also use **+** to concatenate lists and ***** to duplicate lists.

Summary

- Recall the difference between Abstract Data Type and Data Structure. We will see other data structures, and there is a whole class devoted to data structures (CSCI 201)!
- Practice with slicing (either `lists` or `strs`).
- Programming Assignment 2 due Thursday (initial submission). Read the instructions carefully!
- Help hours: see [go/smith](#) (Smith Gakuya, ASI) and [go/cshelp](#).