

CSCI 145 · Introduction to Computing · Fall 2026

More Iteration

Lecture 7 — looping over strings, accumulators, input, nested loops

Outline

- Looping over strings

- The accumulator pattern

- Getting input from the user

- Nested loops

Part 1 of 4

Looping over strings

Anatomy of a `for` loop

REVISITED

keywords

`for` and `in`

loop variable

Holds the current item
You pick the name

sequence

Also called an **iterable**

Last time: a `range(...)`

Today: a string

repeat

Until the sequence
runs out

`for` *variable* `in` *sequence*:

statement 1

statement 2

...

body

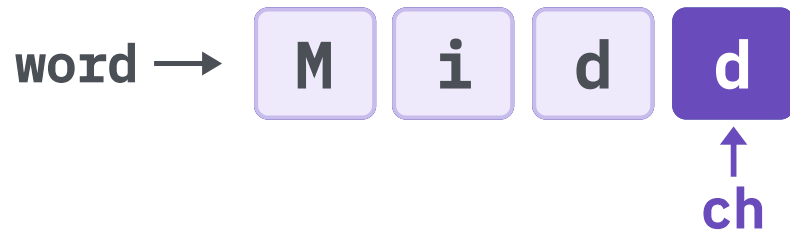
Runs once per item

No loop condition, no update

A `for` loop can step through a string

A string is a **sequence** of characters, so `for` can visit them one at a time

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



OUTPUT

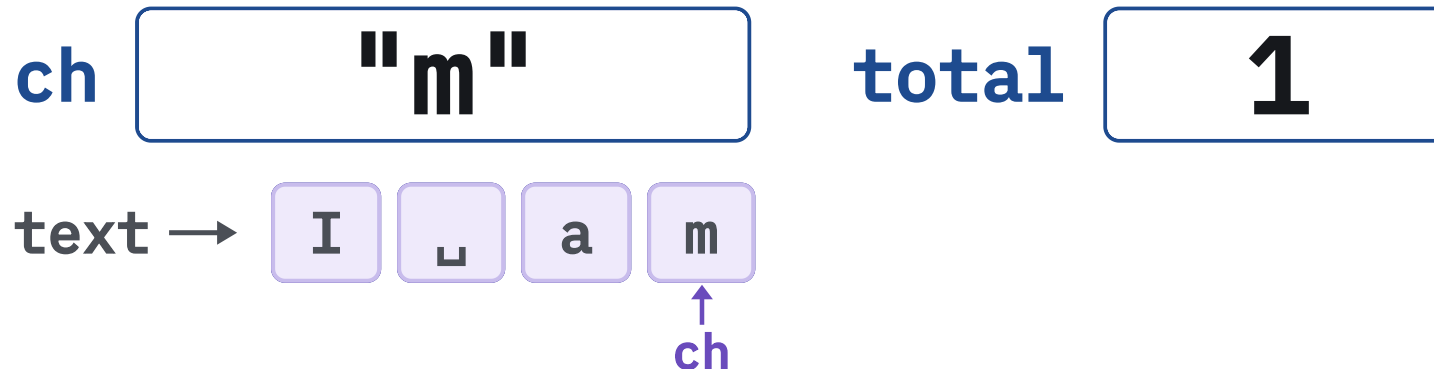
M
i
d
d

Each time through, `ch` is the **next character**

No index needed

Tracing a string loop: count the spaces

```
text = "I am"  
total = 0  
for ch in text:  
    if ch == " ":  
        total += 1  
print(total)
```



AFTER THE LOOP

total holds the
answer: one space

OUTPUT

1

Done!

Two ways to loop over a string

Each character

```
for ch in word:  
    print(ch)
```

Each index

```
for i in range(len(word)):  
    print(word[i])
```

Same output: `i` is 0, 1, 2, 3, and `word[i]` is each character

WHICH ONE?

- `for ch in s`: you only need the characters (simpler)
- `for i in range(len(s))`: you need the **position** too

Part 2 of 4

The accumulator pattern

The accumulator pattern

Build up an answer one piece at a time as the loop runs

```
def count_spaces(text):  
    total = 0  
    for ch in text:  
        if ch == " "  
            total += 1  
    return total
```

1. before the loop: start it off

2. inside the loop: update it

3. after the loop: use it

- The variable that collects the answer is the **accumulator**
- Its start value depends on what you're building

A number accumulator: factorial

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

```
def factorial(n):  
    result = 1  
    for i in range(2, n + 1):  
        result = result * i  
    return result
```

i	result * i	result
–	–	1
2	1 * 2	2
3	2 * 3	6
4	6 * 4	24
5	24 * 5	120

Why start `result` at `1`, not `0`?

Anything times 0 is 0
The answer would always be 0

A string accumulator

Replace every `v` with a `5`: `cyberpunk("David")` returns `"Da5id"`

```
def cyberpunk(name):  
    result = ""  
    for ch in name:  
        if ch == "v" or ch == "V":  
            result += "5"  
        else:  
            result += ch  
    return result
```

ch	add	result
-	-	" "
"D"	"D"	"D"
"a"	"a"	"Da"
"v"	"5"	"Da5"
"i"	"i"	"Da5i"
"d"	"d"	"Da5id"

- Start with the **empty string** `" "` and add one piece per character
- `result += ch` is shorthand for `result = result + ch`

What does `count_a("banana")` return?

```
def count_a(word):  
    for ch in word:  
        count = 0  
        if ch == "a":  
            count += 1  
    return count
```

Go to **slido.com** and enter the code: `3674841`

Or [join directly here](#)



What does `count_a("banana")` return?

```
def count_a(word):  
    for ch in word:  
        count = 0  
        if ch == "a":  
            count += 1  
    return count
```

Returns 1, not 3

`count = 0` is **inside** the loop
It resets on every character

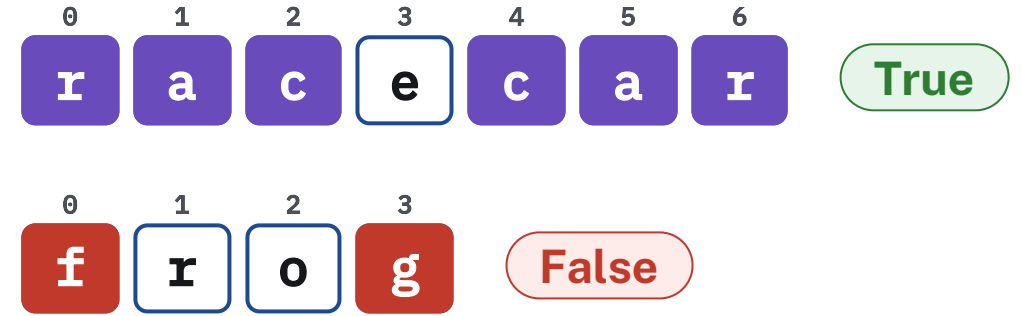
```
def count_a(word):  
    count = 0  
    for ch in word:  
        if ch == "a":  
            count += 1  
    return count
```

✓ Fix: start it before the loop

Set `count = 0` once, before the loop
Now it returns 3

Stopping early with `return`

```
def is_palindrome(word):  
    for i in range(len(word) // 2):  
        if word[i] != word[len(word) - 1 - i]:  
            return False  
    return True
```



- One mismatch is enough: `return False` right away
- `return` ends the function, and the loop with it
- `return True` only **after** the loop, once every pair has matched

What does `has_space("hi there")` return?

```
def has_space(s):  
    for ch in s:  
        if ch == " "  
            return True  
        else:  
            return False
```

Go to **slido.com** and enter the code: `3674841`

Or [join directly here](#)



What does `has_space("hi there")` return?

```
def has_space(s):  
    for ch in s:  
        if ch == " "  
            return True  
        else:  
            return False
```

Returns False

The first character, "h", isn't a space
So the `else` returns False right away

```
def has_space(s):  
    for ch in s:  
        if ch == " "  
            return True  
    return False
```

✓ Fix: decide False after the loop

Delete the `else`
Put `return False` after the loop

Part 3 of 4

Getting input from the user

Asking the user with `input()`

```
>>> name = input("Your name? ")
Your name? Ada
>>> name
'Ada'
```

```
>>> age = input("Age? ")
Age? 19
>>> age
'19'
```

```
>>> age = int(input("Age? "))
Age? 19
>>> age + 1
20
```

- Prints the prompt, then **waits** until the user presses Enter
- Returns what they typed
- Always returns a **string**, even if they type a number
- Convert with `int()` for math

Repeat until the answer is valid

We can't know how many tries the user needs, so this is a job for `while`

```
n = int(input("Pick 1-10: "))  
while n < 1 or n > 10:  
    print("Out of range.")  
    n = int(input("Pick 1-10: "))  
print("You picked", n)
```

SAMPLE RUN

```
Pick 1-10: 42  
Out of range.  
Pick 1-10: 0  
Out of range.  
Pick 1-10: 7  
You picked 7
```

ask once before the loop

loop while the answer is bad

ask again at the end of the body

A guessing game

```
def guessing_game():  
    number = random.randint(1, 20)  
    correct = False  
    while not correct:  
        guess = int(input("Guess 1-20: "))  
        if guess == number:  
            print("Correct!")  
            correct = True  
        elif guess < number:  
            print("Too low")  
        else:  
            print("Too high")
```

SAMPLE RUN

```
Guess 1-20: 10  
Too high  
Guess 1-20: 5  
Too low  
Guess 1-20: 7  
Correct!
```

- `randint(1, 20)` : 1 to 20, **including** 20
- `correct` is a **flag** that tells the loop when to stop

Picking a random character

`random.choice(s)` returns one character of the string `s`, chosen at random

```
>>> random.choice("+-*")
'*'
>>> random.choice("+-*")
'+'
>>> random.choice("aeiou")
'o'
```

- Today's activity: use both to build an arithmetic expression like `"7 * 3"`
- Lab 3 extends it to longer expressions

Part 4 of 4

Nested loops

Nested loops

- A loop's body can contain **another** loop
- The inner loop runs **all the way through** for each pass of the outer loop

```
for i in range(1, 4):  
    for j in range(1, 4):  
        print(i * j, end=" ")  
    print()
```

INDICES

```
i = 1, j = 1, 2, 3  
i = 2, j = 1, 2, 3  
i = 3, j = 1, 2, 3
```

OUTPUT

```
1 2 3  
2 4 6  
3 6 9
```

- 3 values of `i` × 3 values of `j` = 9 numbers printed
- `end=" "` keeps a row on one line; the empty `print()` ends the row

What does this print?

```
for i in range(1, 5):  
    row = ""  
    for j in range(i):  
        row = row + "*"   
    print(row)
```

INDICES

```
i = 1, j = 0  
i = 2, j = 0, 1  
i = 3, j = 0, 1, 2  
i = 4, j = 0, 1, 2, 3
```

OUTPUT

```
*  
**  
***  
****
```

- The inner loop runs `i` times, so each row is one star longer
- `row` is a string accumulator, restarted at `""` for each row

Activity

1. Download [day07_iteration.py](#) and save it in your `cs145/class` folder.
2. Launch Thonny. From the File menu, open `cs145/class/day07_iteration.py`.
3. For part 1, use the accumulator pattern in `count_char` and `remove_spaces`.
4. For part 2, complete `ask_yes_no` with `input()` and a `while` loop, and `random_problem` with `randint` and `choice`.
5. If you have time, complete `has_double(s)`. Which loop fits?

To do

- Review this slide deck and the corresponding notes: [Notes 7](#)
- [Homework 2](#) is due Thursday 10/8
- [Lab 3](#) is this Friday

Next time

Lecture 8 — lists