

# [220 / 319] Strings

Meena Syamkumar  
Andy Kuemmel

- **Exam I Friday**
- **Please read email with subject: “[IMPORTANT]: CS220 / CS319: Exam I location”**

# Learning Objectives Today

## String Basics

- Comparison
- Common functions



## Method Syntax

## Sequences (a string is an example of a sequence)

- len
- indexing
- slicing
- for loop

Chapter 8+9 of  
Think Python

what we've learned  
about strings so far



what we'll learn today

# Today's Outline

## Comparison

String Methods

Sequences

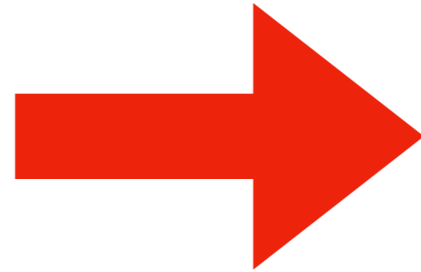
Slicing

for loop over sequence

for loop over range

# Comparison

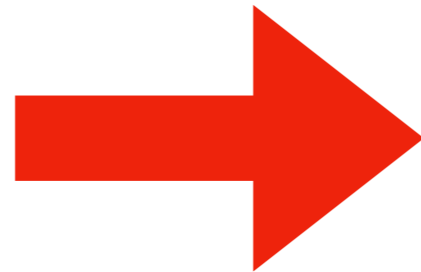
1 < 2



True

(because 1 is before 2)

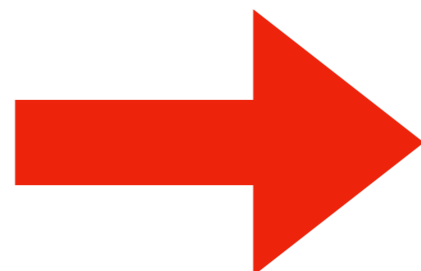
200 < 100



False

(because 200 is NOT before 100)

“cat” < “dog”



True

(because “cat” is before “dog” in the dictionary)



Python can also compare strings

# Comparison

“dog” < “doo doo” → True

What about strings that start with the same letter?

Look for the first letter that’s different, and compare those.

# Comparison

“dog” < “doo doo” → True

There are three gotchas:

- 1 case (upper vs. lower)
- 2 digits
- 3 prefixes

# I. Case rules

“A” < “B” < ... < “Y” < “Z”

**makes sense**

“a” < “b” < ... < “y” < “z”

**makes sense**

---

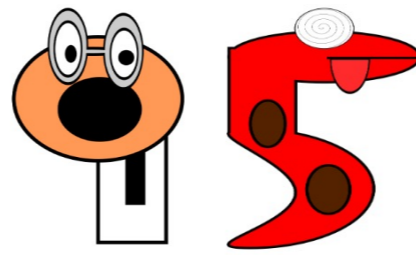
Any two characters are compared using their position in the ASCII table.

“C” < “b”  
“Z” < “a”

In the ASCII table, upper case is before lower case.

**To learn more, visit**  
<https://simple.wikipedia.org/wiki/ASCII>

## 2. Pesky digits



“0” < “1”      **makes sense**

“8” < “9”      **makes sense**

---

“11” < “2”  
“100” < “15”

remember to find the **FIRST** difference,  
and base everything on that



# 3. Prefixes

**String 1:** bat□  
**String 2:** batman



“” < “m”, so String 1 is first:

“bat” < “batman”

Do problem 1

# Today's Outline

Comparison

**String Methods**

Sequences

Slicing

for loop over sequence

for loop over range

# What is a method?


A special function associated variable/value

```
>>> msg = "hello"
```

```
>>> len(msg)
```

```
5
```

```
>>>
```



len is a normal function,  
it returns number  
of characters in string.

It returns the number of  
characters in a string

# What is a method?

A special function associated variable/value

```
>>> msg = "hello"  
>>> len(msg)  
5  
>>> msg.isdigit()  
False  
>>>
```

*type of msg*  
*method in str (similar to mod)*  
str.isdigit(msg)

*equivalent*

isdigit is a special function, called a method, that operates on the string in msg.

It returns a bool, whether the string is all digits

# What is a method?

A special function associated variable/value

```
>>> msg = "hello"  
>>> len(msg)  
5  
>>> msg.isdigit()  
False  
>>>
```

Both the regular function (`len`) and method (`isdigit`) are answering a question about the string in `msg`, but we call them slightly differently

# What is a method?

A special function associated variable/value

```
>>> msg = "hello"
```

```
>>> len(msg)
```

```
5
```

```
>>> msg.isdigit()
```

```
False
```

```
>>> msg.upper()
```

```
'HELLO'
```

 is upper a regular function or a method?

# What is a method?

A special function associated variable/value

```
>>> msg = "hello"
>>> len("220")
3
>>> "220".isdigit()
True
>>> "Hello World".upper()
'HELLO WORLD'
```

methods can be called with literal values as well as with values in variables



| String Method        | Purpose                                              |
|----------------------|------------------------------------------------------|
| s.upper()            | change string to all upper case                      |
| s.lower()            | opposite of upper()                                  |
| s.strip()            | remove whitespace (space, tab, etc) before and after |
| s.lstrip()           | remove whitespace from left side                     |
| s.rstrip()           | remove whitespace from right side                    |
| s.format(args...)    | replace instances of “{}” in string with args        |
| s.find(needle)       | find index of needle in s                            |
| s.startswith(prefix) | does s begin with the given prefix?                  |
| s.endswith(suffix)   | does s end with the given suffix?                    |
| s.replace(a, b)      | replace all instances of a in s with b               |

**Quick demos...**

**Do problem 2**

# Today's Outline

Comparison

String Methods

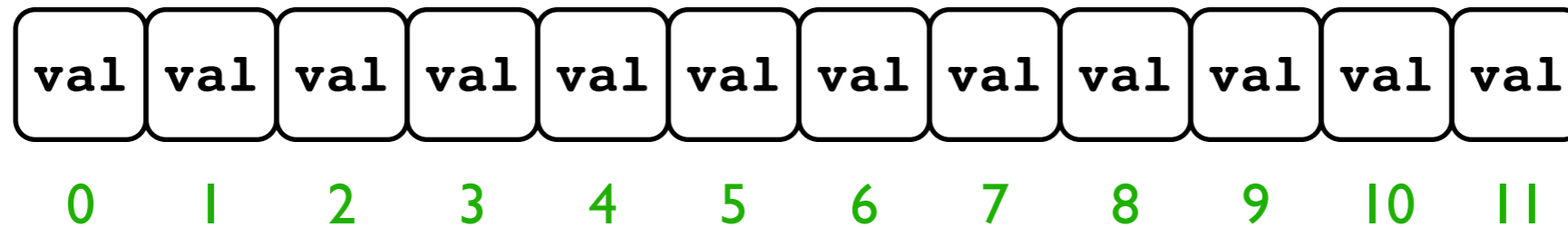
**Sequences**

Slicing

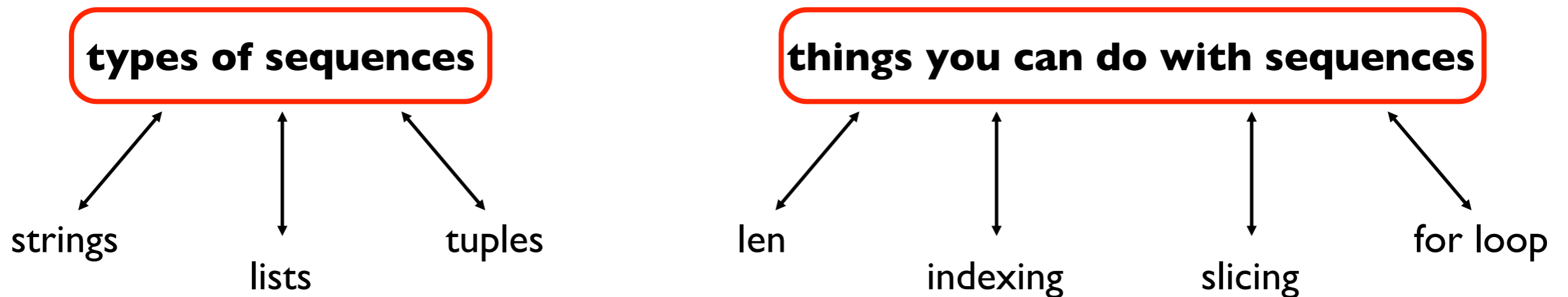
for loop over sequence

for loop over range

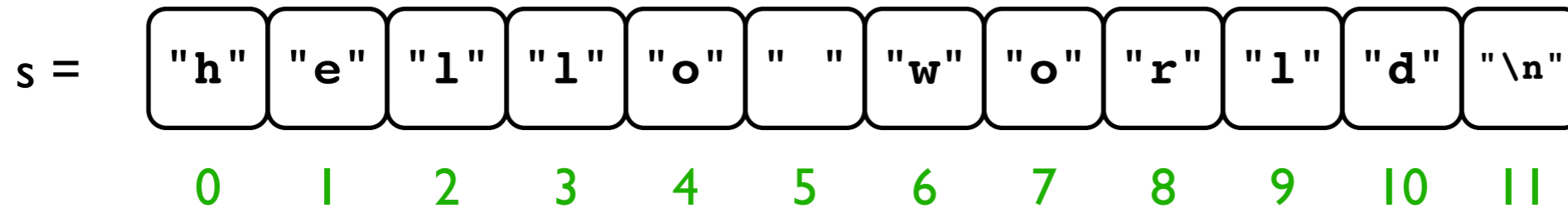
# Python Sequences



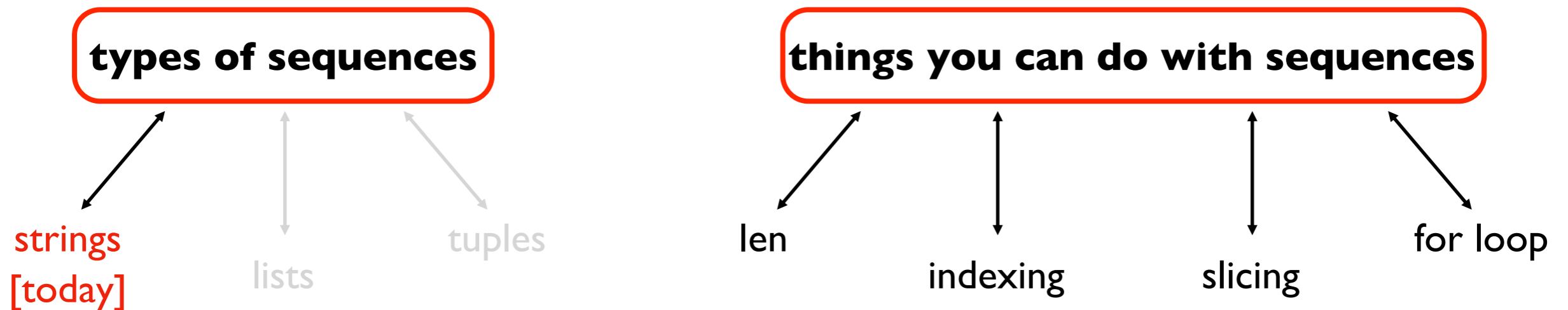
**Definition:** a *sequence* is a collection of numbered/ordered values



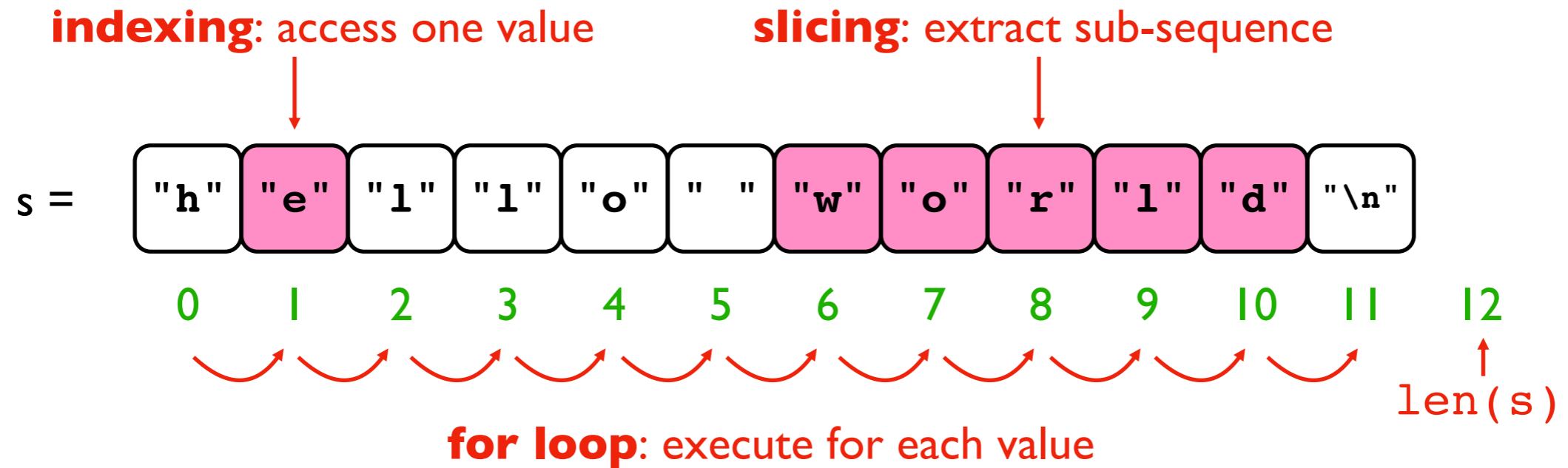
# Python Sequences



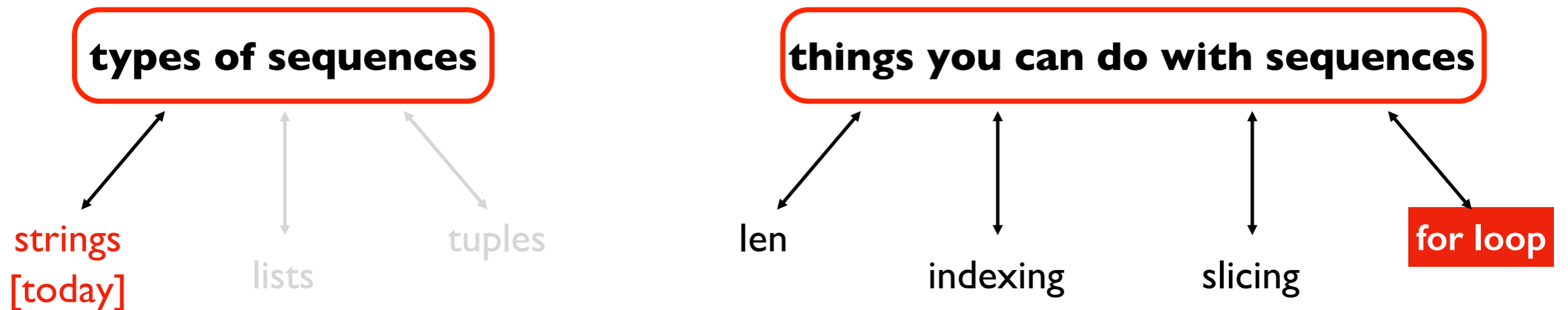
**Definition:** a *string* is a sequence of one-character strings



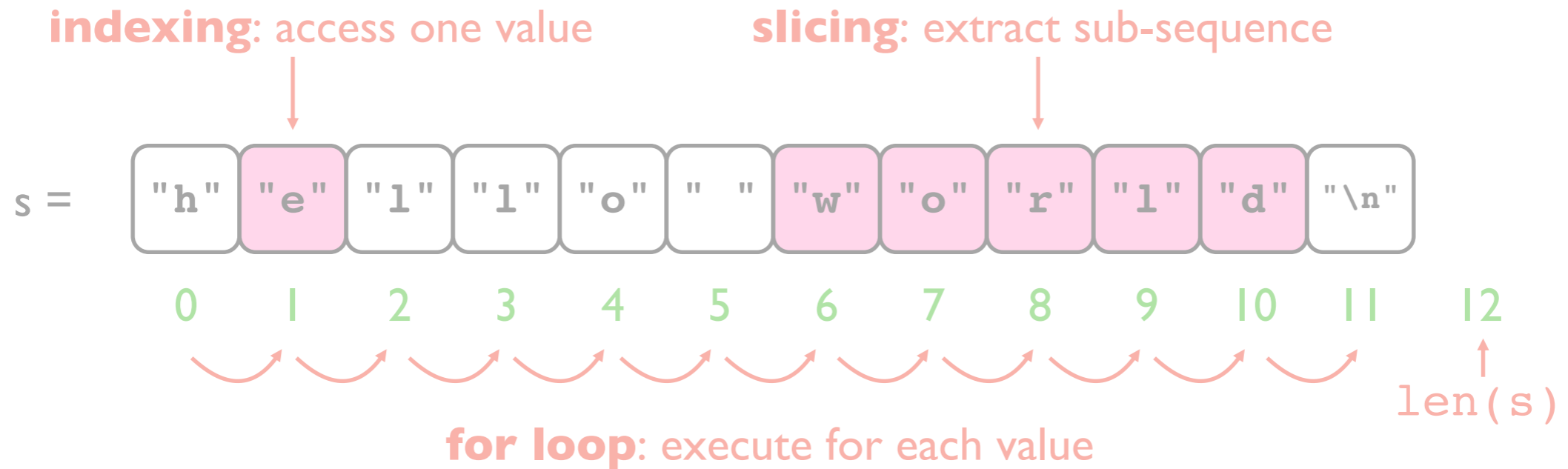
# Python Sequences



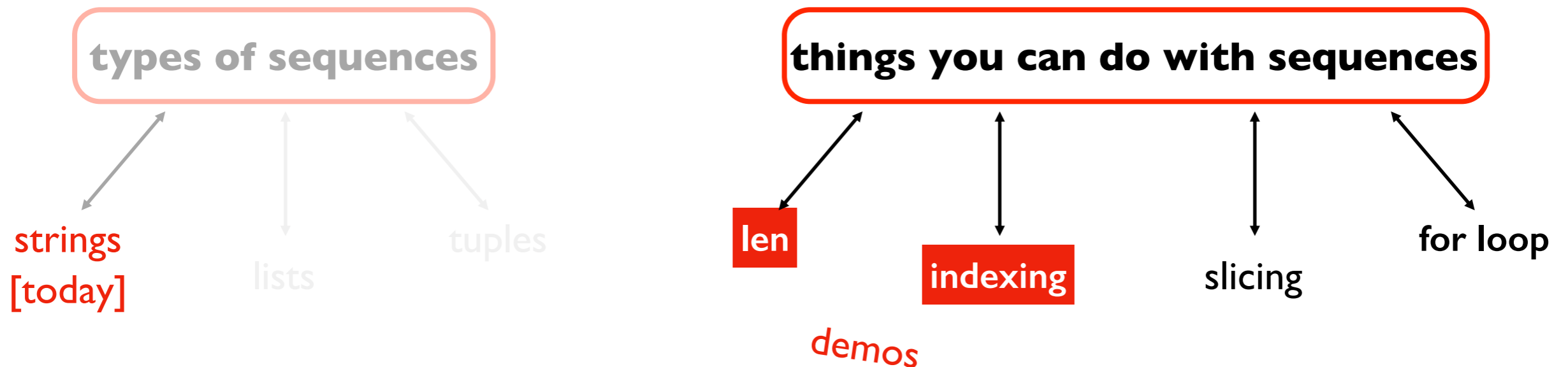
**Definition:** a *string* is a sequence of one-character strings



# Python Sequences



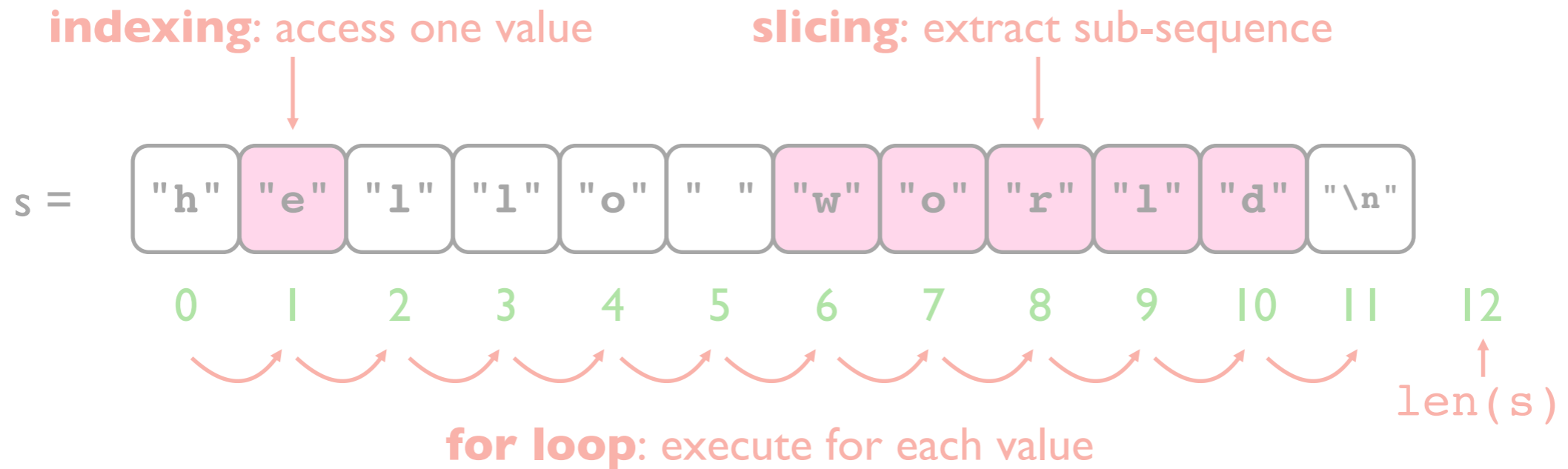
**Definition:** a *string* is a sequence of one-character strings



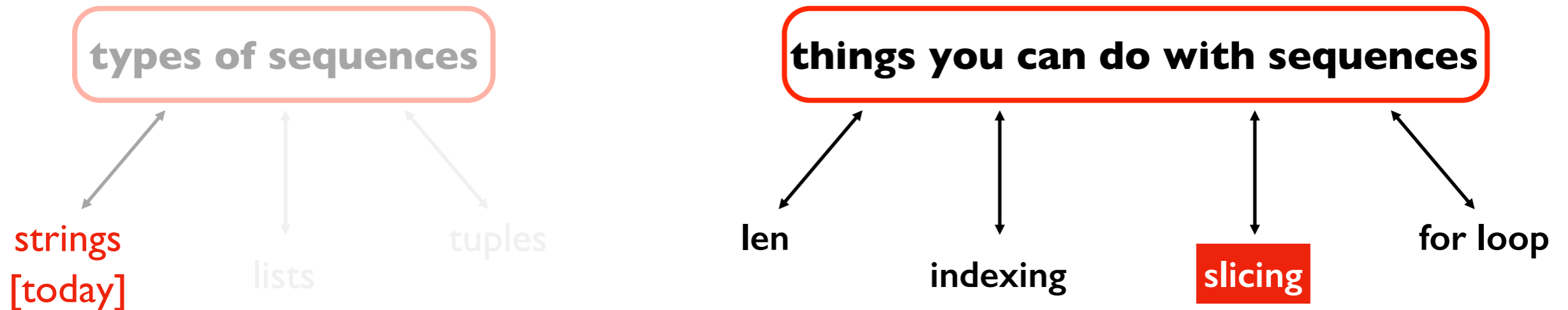
**Do problem 3**



# Python Sequences



**Definition:** a *string* is a sequence of one-character strings



# Today's Outline

Comparison

String Methods

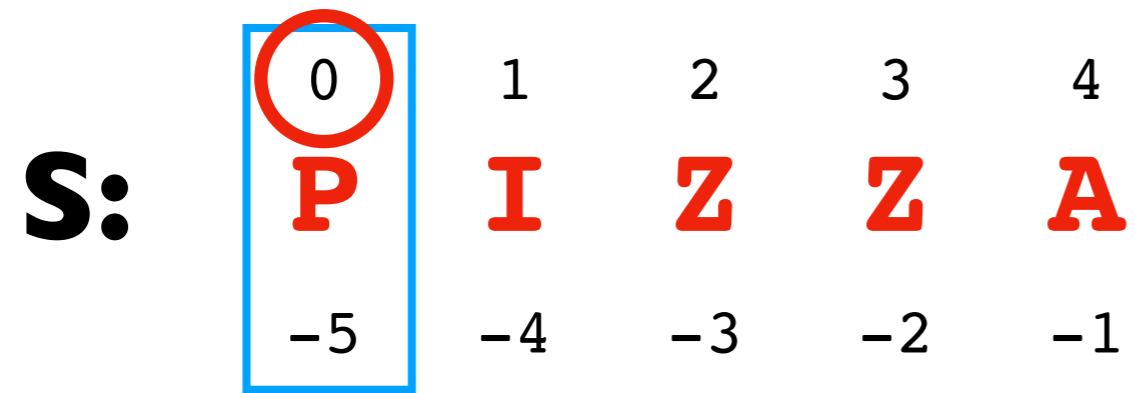
Sequences

**Slicing**

for loop over sequence

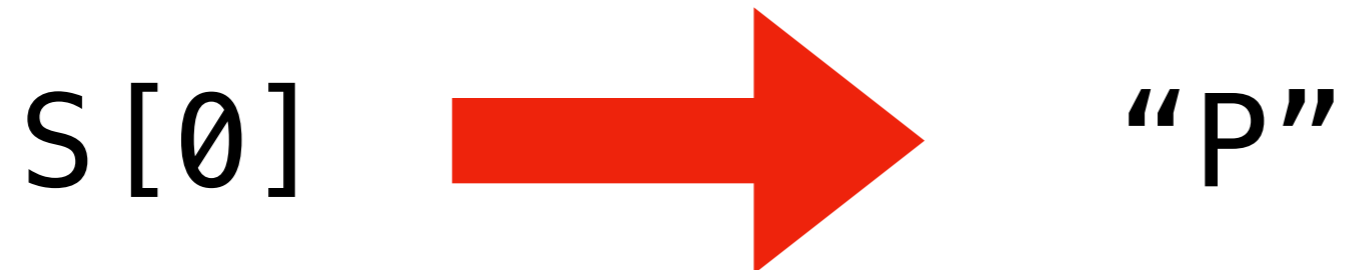
for loop over range

# Indexing

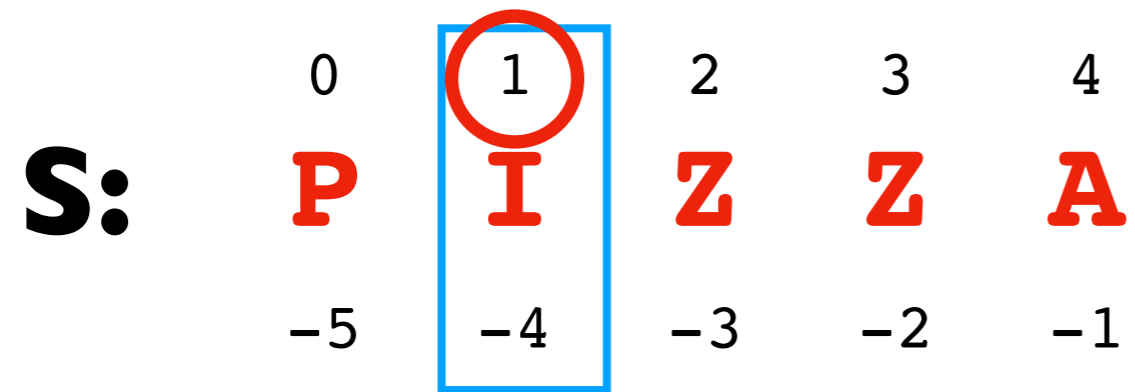


**Code:**

```
S = "PIZZA"
```

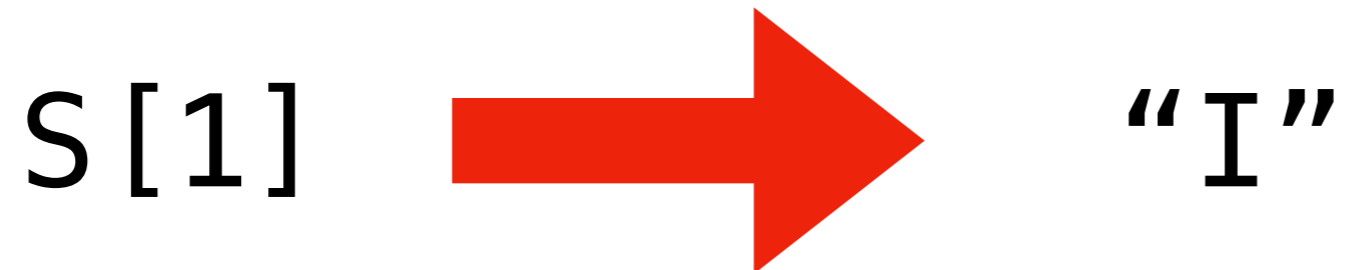


# Indexing



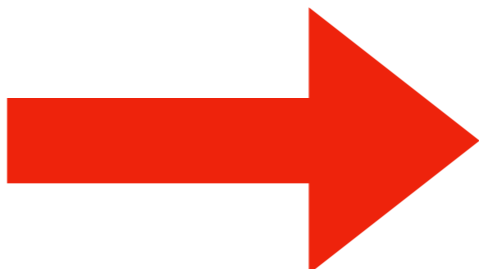
**Code:**

S = "PIZZA"



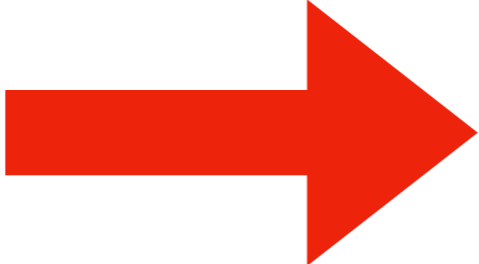
# Indexing

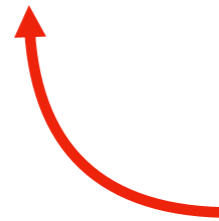
|           |          |          |          |          |          |
|-----------|----------|----------|----------|----------|----------|
|           | 0        | 1        | 2        | 3        | 4        |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
|           | -5       | -4       | -3       | -2       | -1       |

S[-1]  "A"

# Slicing

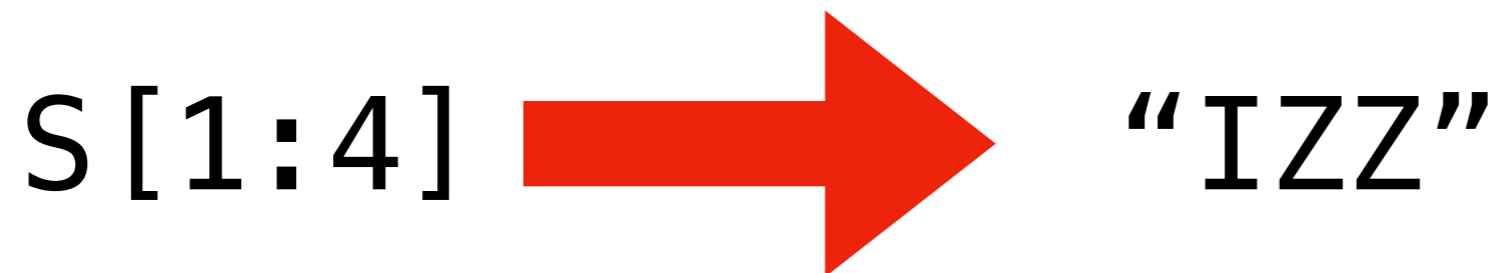
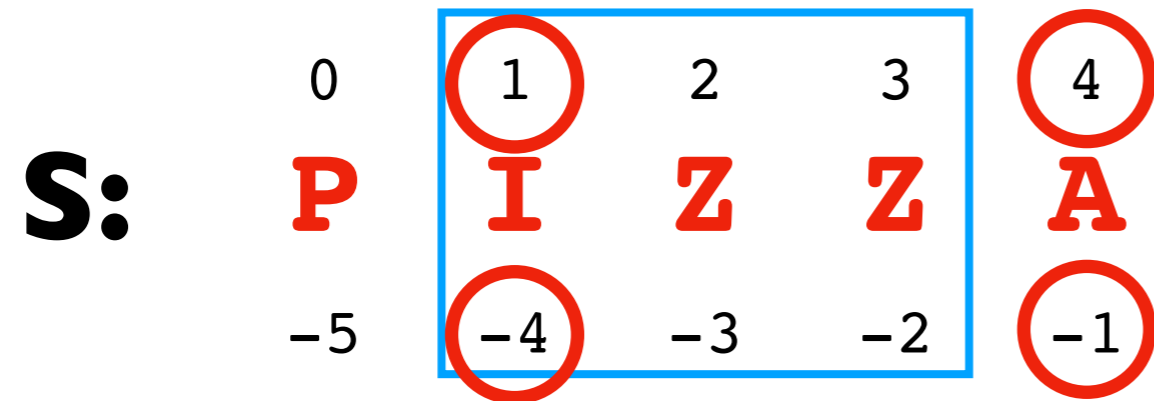
|           |          |          |          |          |          |
|-----------|----------|----------|----------|----------|----------|
|           | 0        | 1        | 2        | 3        | 4        |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
|           | -5       | -4       | -3       | -2       | -1       |

S [???]  "IZZ"



what to put if we want multiple letters,  
like "IZZ"?

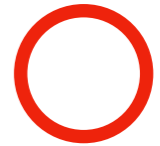
# Slicing



Many different slices give the same result:  
 $S[1:4] == S[1:-1] == S[-4:4] == S[-4:-1]$

# Slicing

|           |          |          |          |          |          |
|-----------|----------|----------|----------|----------|----------|
|           | 0        | 1        | 2        | 3        | 4        |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
|           | -5       | -4       | -3       | -2       | -1       |



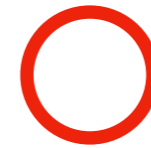
`S[1:100]`  "IZZA"

Slices don't complain about out-of-range numbers.  
You just don't get data for that part



# Slicing

|           |          |          |          |          |          |  |  |
|-----------|----------|----------|----------|----------|----------|--|--|
|           | 0        | 1        | 2        | 3        | 4        |  |  |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |  |  |
|           | -5       | -4       | -3       | -2       | -1       |  |  |



**S** [~~50~~ : ~~100~~]  ""

Slices don't complain about out-of-range numbers.  
You just don't get data for that part

# Slicing

**S:**

|          |          |          |          |          |
|----------|----------|----------|----------|----------|
| 0        | 1        | 2        | 3        | 4        |
| <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
| -5       | -4       | -3       | -2       | -1       |

**S** [ : **2** ]  "PI"

Feel free to leave out one of the numbers in the slice

# Slicing

|           |          |          |          |          |          |
|-----------|----------|----------|----------|----------|----------|
|           | 0        | 1        | 2        | 3        | 4        |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
|           | -5       | -4       | -3       | -2       | -1       |

S[2 : ]  "ZZA"

Feel free to leave out one of the numbers in the slice

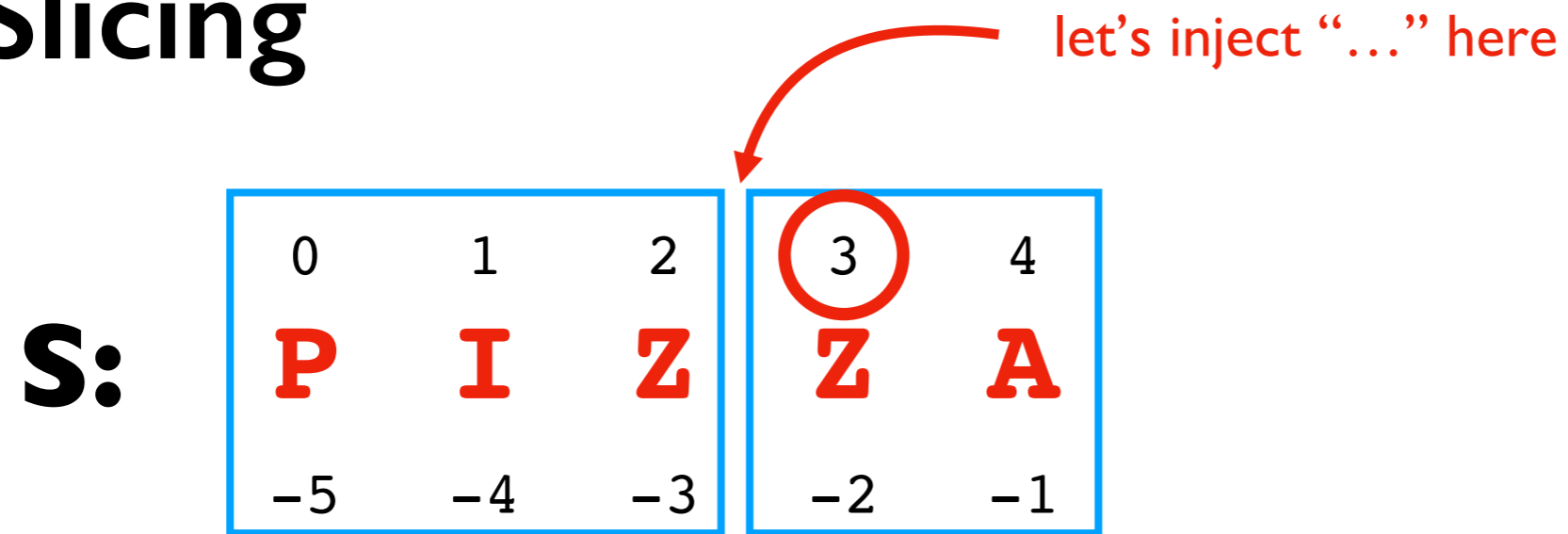
# Slicing

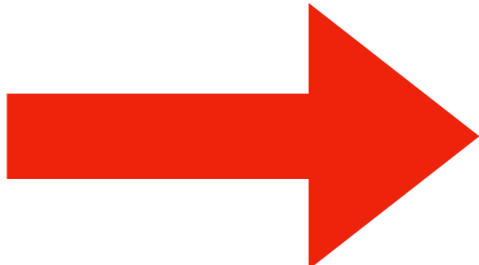
|           |          |          |          |          |          |
|-----------|----------|----------|----------|----------|----------|
|           | 0        | 1        | 2        | 3        | 4        |
| <b>S:</b> | <b>P</b> | <b>I</b> | <b>Z</b> | <b>Z</b> | <b>A</b> |
|           | -5       | -4       | -3       | -2       | -1       |

S[2 : ] → "ZZA"

Inclusive start and exclusive end makes it easier to split and inject things

# Slicing

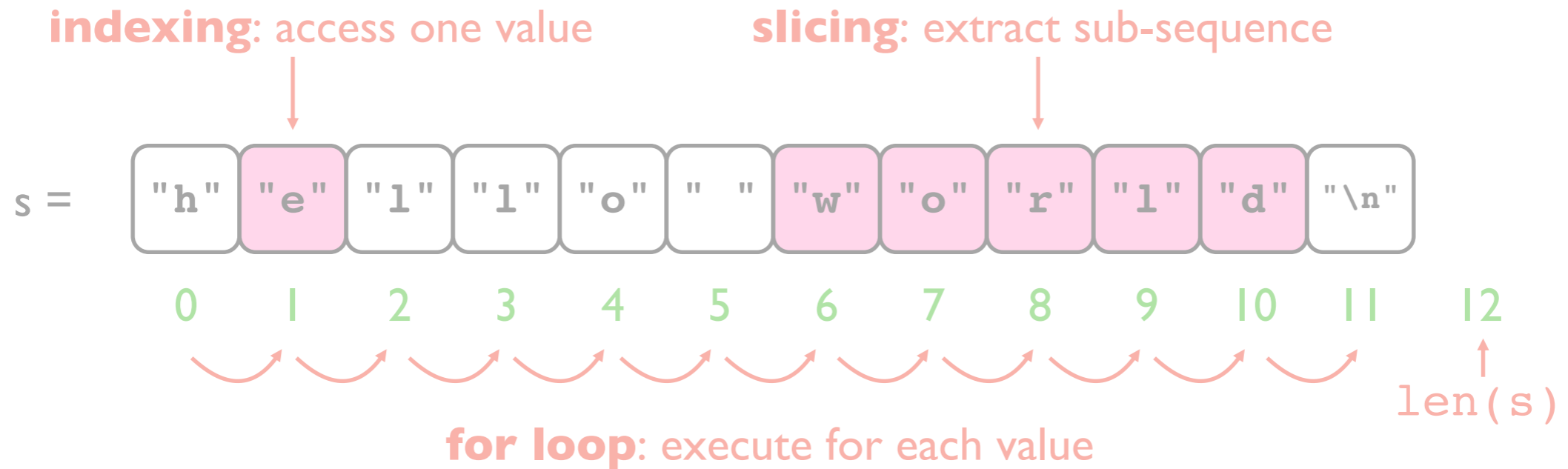


`S[:3] + "..." + S[3:]`  `"PIZ...ZA"`

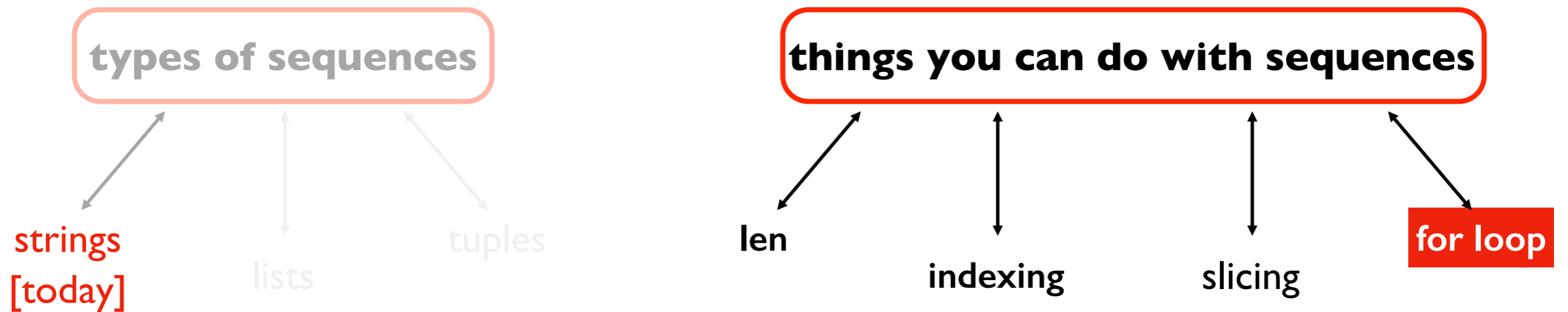
Inclusive start and exclusive end makes it easier to split and inject things

**Do problem 4**

# Python Sequences



**Definition:** a *string* is a sequence of one-character strings



# Today's Outline

Comparison

String Methods

Sequences

Slicing

**for loop over sequence**

for loop over range



# Motivation

```
msg = "hello"
```

```
# let's say we want to print
```

```
# each letter on its own line
```

# Motivation

```
msg = "hello"
```

```
i = 0
```

```
while i < len(msg):
```

```
    ???
```

```
    i += 1
```

indexing starts at 0, so msg[0] is 'h',  
so we want to start i at 0

last letter (o) has index 4,  
or len(msg)-1

we don't want to skip any letters

# Motivation

```
msg = "hello"
```

```
i = 0
```

```
while i < len(msg):
```

```
    letter = msg[i]
```

```
    print(letter)
```

```
    i += 1
```

get the letter for the current index



this is the only interesting part  
(we just want to print each letter!)



Code like this for sequences is so common  
that Python provides an easier way, with the **for loop**

# while vs. for

## while loop

```
msg = "hello"

i = 0
while i < len(msg):
    letter = msg[i]
    print(letter)
    i += 1
```

← *this happens automatically now*

## for loop

```
for letter in msg:
    print(letter)
```

*they do the same thing!*

# for syntax

**for  
loop**

```
for letter in msg:  
    print(letter)
```

basic syntax always used

# for syntax

do PythonTutor example

automatically initialized to a  
different item on each iteration  
("h" on 1st, "e" on 2nd, etc)

the sequence  
(e.g., "hello")

**for**  
**loop**

```
for letter in msg:  
    print(letter)
```

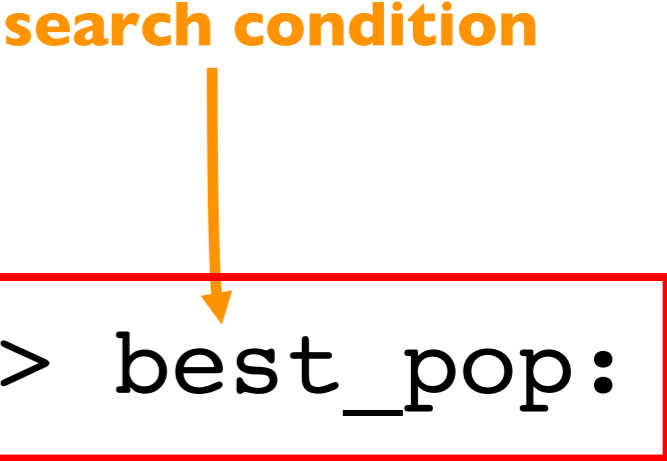
specify a variable name to use inside the loop,  
and the sequence you want to loop over

# Design Pattern: search **something** in data

```
best_pop = None
best_pop_state = None

for row in get_rows(data):
    pop = get_population(row)
    state = get_state(row)
    if best_pop == None or pop > best_pop:
        best_pop = pop
        best_pop_state = state
```

**search condition**



**Functions:**  
get\_rows(...)  
get\_state(...)  
get\_population(...)  
...

| State | Population | Area |
|-------|------------|------|
| WI    | 5.795      | ...  |
| CA    | 39.54      | ...  |
| MN    | 5.577      | ...  |
| ...   | ...        | ...  |

**Do problem 5**



# Today's Outline

Comparison

String Methods

Sequences

Slicing

for loop over sequence

**for loop over range**

# for with range

```
msg = "01234"
```

```
for item in msg:  
    print(item * 3)
```

**Output:**

000

111

222

333

444



what if we want to iterate over the integers  
0 to 4 (instead of string digits "0" to "4")?

# for with range

```
for item in range(5):  
    print(item * 3)
```

**Output:**

0  
3  
6  
9  
12



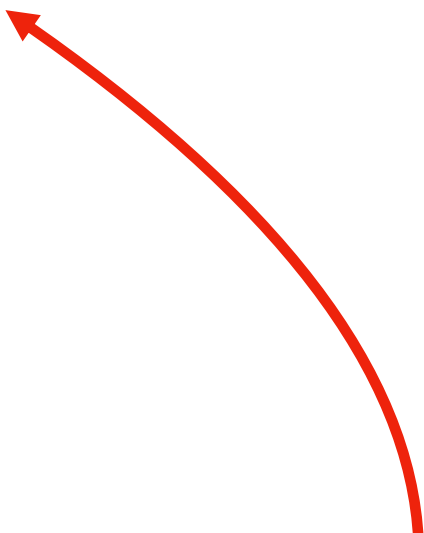
what if we want to iterate over the integers  
0 to 4 (instead of string digits "0" to "4")?

# for with range

```
for item in range(5):  
    print(item * 3)
```

**Output:**

0  
3  
6  
9  
12



using range(N) with a for loop will  
iterate with these values for item:  
0, 1, 2, ..., N-2, N-1

**Do problem 6**