

# Strings II

# Review

- Strings are stored character by character.
- Can access each character individually by using an index:

0	1	2	3	4	5	6	7
"C"	"o"	"m"	"p"	"u"	"t"	"e"	"r"

# New

- Negative indexing can be used. (*Particularly useful for getting characters near the end of a string.*)

0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1
"C"	"o"	"m"	"p"	"u"	"t"	"e"	"r"

# The basic string for loop

- Use this whenever you need to process a string one character at a time.

```
# assume s is a string variable  
for pos in range(0, len(s)):  
    # do something with s[pos]
```

```
s = "banana"
total = 0
for pos in range(0, len(s)):
    if s[pos] == "a":
        total = total + 1
```

0    1    2    3    4    5

"b"	"a"	"n"	"a"	"n"	"a"
-----	-----	-----	-----	-----	-----

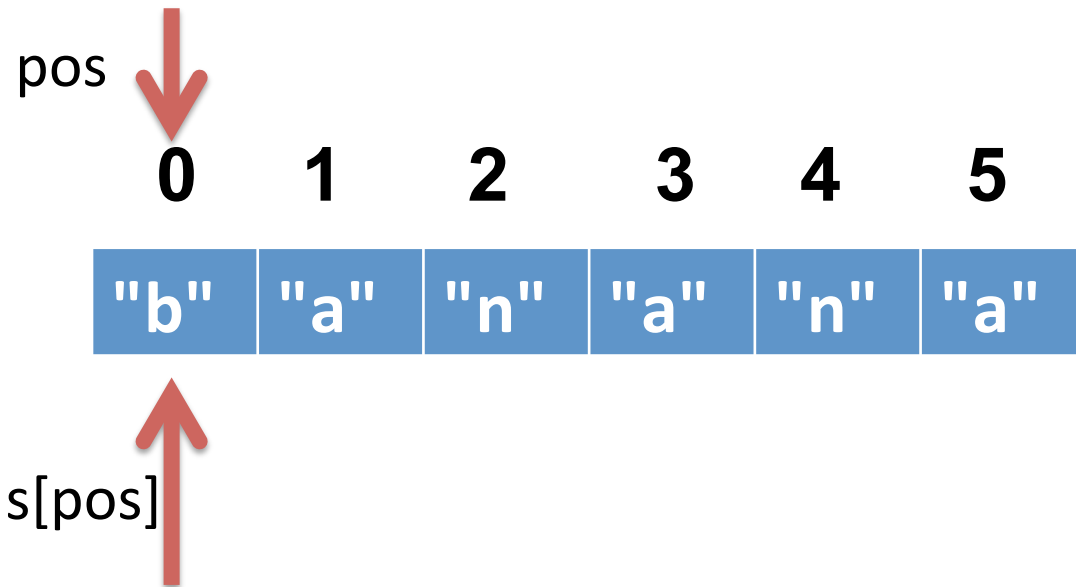
```
s = "banana"
```

```
total = 0
```

```
for pos in range(0, len(s)):
```

```
    if s[pos] == "a":
```

```
        total = total + 1
```



**1st iteration**

pos: 0

s[pos]: "b"

total: 0

```
s = "banana"
```

```
total = 0
```

```
for pos in range(0, len(s)):
```

```
    if s[pos] == "a":
```

```
        total = total + 1
```

pos

0 1 2 3 4 5

"b"	"a"	"n"	"a"	"n"	"a"
-----	-----	-----	-----	-----	-----

s[pos]

2<sup>nd</sup> iteration

pos: 1

s[pos]: "a"

total: 1

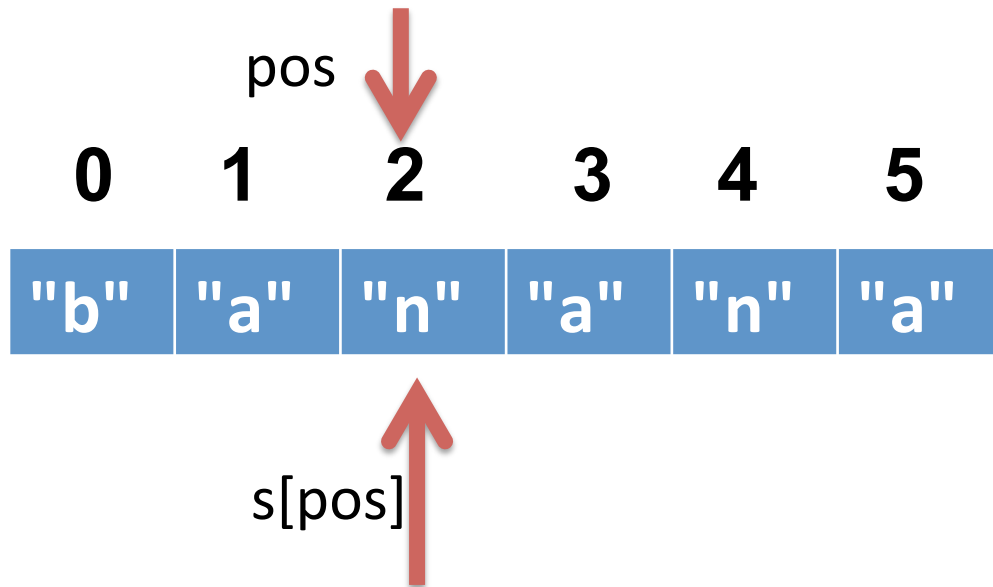
```
s = "banana"
```

```
total = 0
```

```
for pos in range(0, len(s)):
```

```
    if s[pos] == "a":
```

```
        total = total + 1
```



3<sup>rd</sup> iteration

pos: 2

s[pos]: "n"

total: 1



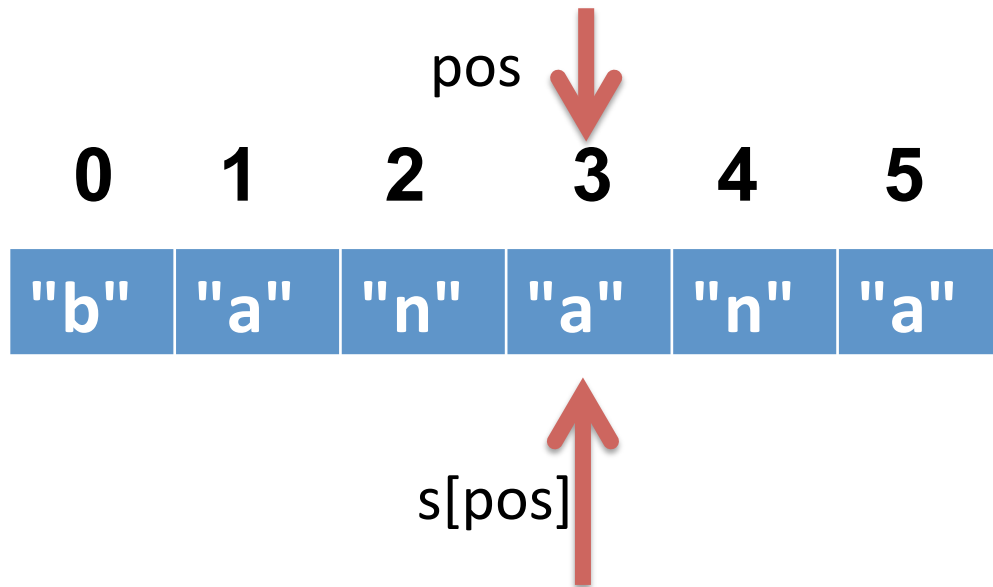
```
s = "banana"
```

```
total = 0
```

```
for pos in range(0, len(s)):
```

```
    if s[pos] == "a":
```

```
        total = total + 1
```



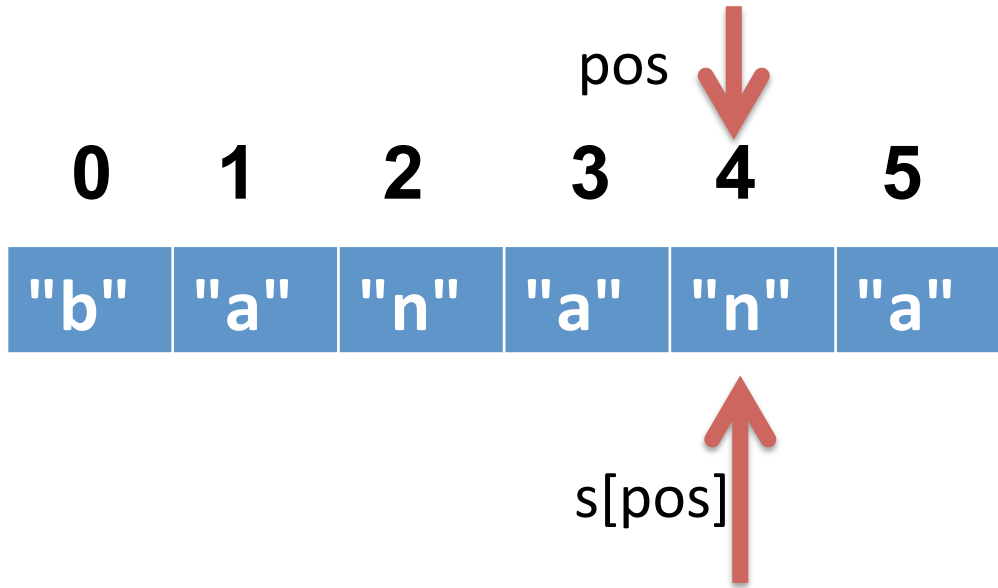
4<sup>th</sup> iteration

pos: 3

s[pos]: "a"

total: 2

```
s = "banana"
total = 0
for pos in range(0, len(s)):
    if s[pos] == "a":
        total = total + 1
```



5<sup>th</sup> iteration  
 pos: 4  
 s[pos]: "n"  
 total: 2

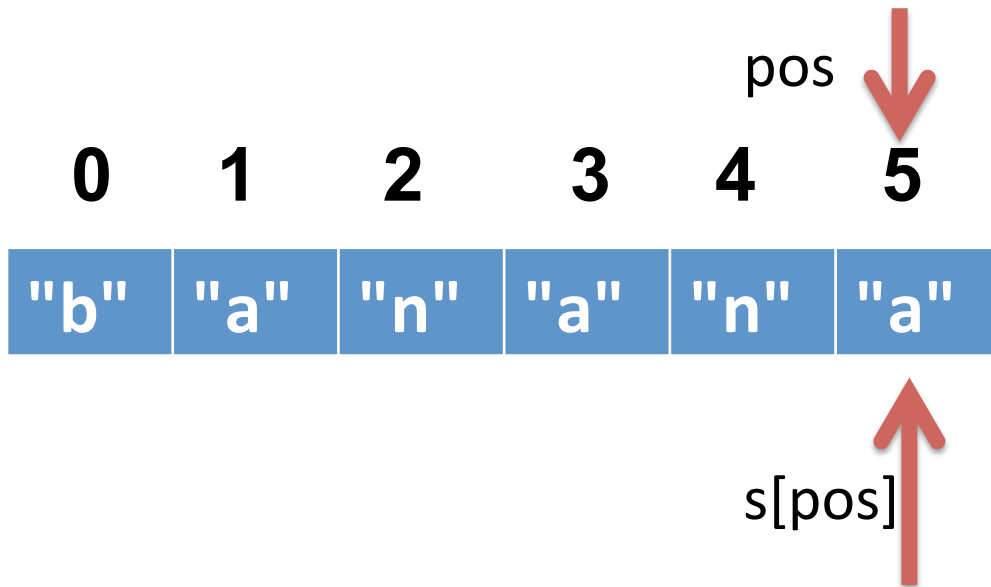
```
s = "banana"
```

```
total = 0
```

```
for pos in range(0, len(s)):
```

```
    if s[pos] == "a":
```

```
        total = total + 1
```



6<sup>th</sup> iteration

pos: 5

s[pos]: "a"

total: 3

# Algorithm -> Function

- Counting the number of a certain character in a string seems like a good candidate for a function.

```
def count_a(s):  
    total = 0  
    for pos in range(0, len(s)):  
        if s[pos] == "a"  
            total = total + 1  
    return total
```

```
def count_a(s):
```

```
    total = 0
```

```
    for pos in range(0, len(s)):
```

```
        if s[pos] == "a":
```

```
            total = counter + 1
```

```
    return counter
```

```
def main():
```

```
    name = input("What is your name? ")
```

```
    freq = count_a(name)
```

```
    print("Your name has", freq, "A's in it.")
```

- Step 1: Change the count function so it takes a second argument called **letter**. The function should count the number of times that **letter** occurs in the string (instead of only lowercase a's).
- Step 2: Change the main function so that the user can type in their name and a letter and the program prints the frequency of that letter in their name.
- **Challenge:** Write a function `count_dups` that counts (and returns) all occurrences of consecutive duplicated letters in a string.
  - e.g., `count_dups("balloon")` returns 2.

Not all string problems are solved with for loops.

```
def get_initial(firstname):  
    first_init = firstame[0]  
    return first_init
```

# String Concatenation

- Combines two strings into a new, longer string.
- Uses the same plus sign as addition.

```
s1 = "CS141"
```

```
s2 = "rocks!"
```

```
bigstring = s1 + s2
```

```
print(bigstring)
```

```
# prints CS141rocks!
```



# String Concatenation

- Unlike `print()`, string concatenation does not put spaces between your strings.

```
s1 = "CS141"
```

```
s2 = "rocks!"
```

```
bigstring = s1 + " " + s2
```

```
print(bigstring)
```

```
# prints CS141 rocks!
```

# Sample problem

- All professor email addresses at Rhodes are constructed from the professor's last name, followed by the initial letter of their first name.
- We want to design a function that takes a prof's first and last name and returns their email address.

```
def make_prof_email(first, last):  
    init = first[0]  
    address = last + init + "@rhodes.edu"  
    return address  
  
def main():  
    firstname = input("First name: ")  
    lastname = input("Last name: ")  
    addr = make_prof_email(firstname, lastname)  
    print("Email:", addr)
```

# You try it

- Write a function `make_student_email` that creates (and returns) a student email address.
- The function should take four parameters: first name, last name, middle name, and class year.
- Challenge: Modify the function so it takes only two parameters: someone's full name (one string with first, middle, and last names within it) and class year.

- A fundamental problem when using strings is computing a *substring*, or a string *slice*.
- We want to tell Python
  - take some string,
  - give me all the characters starting from one index,
  - and ending at another index.
- Fortunately, this is built into Python!

- Two ways to use square brackets.
- 1 number inside the brackets:
  - returns ***exactly one*** character of a string.
  - if s = "Computer, then s[0] returns "C"
- 2 numbers inside the brackets:
  - returns a ***substring*** or string ***slice***.

**s[a:b]** gives you a string slice of string **s** starting from index **a** and ending at index **b-1**.

0	1	2	3	4	5	6	7
"C"	"o"	"m"	"p"	"u"	"t"	"e"	"r"

s[0:1] -> "C" just like s[0]

s[0:2] -> "Co"

s[0:7] -> "Compute"

s[3:6] -> "put"

s[0:8] -> "Computer"

# More fun with indices

- Indices can also be negative.
- A negative index counts from the right side of the string, rather than the left.

```
s = "Computer"  
print(s[-1])           # prints r  
print(s[-3:len(s)])   # prints ter  
print(s[1:-1])        # prints ompute
```



- Slices don't need both left and right indices.
- Missing left index:
  - Python assumes you meant 0 [far left of string]
- Missing right index:
  - Python assumes you meant `len(s)` [far right of string]

```
s = "Computer"  
print(s[1:])           # prints omputer  
print(s[:5])          # prints Compu  
print(s[-2:])         # prints er
```

# Indices don't have to be literal numbers

Say we have this code:

```
name = input("type in your name: ")  
x = int(len(s) / 2)  
print(name[0:x])
```

What does this print?