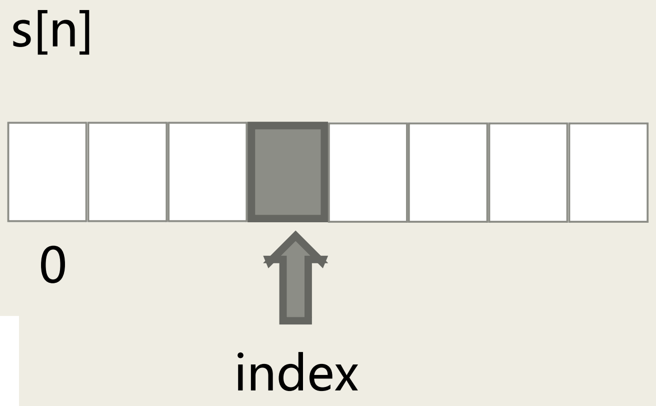
LECTURE 8: STRINGS

Hung-Yu Wei

String

- String: a sequence of characters
- []
- Index
  - *Integer*
  - *Starting from 0*



```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

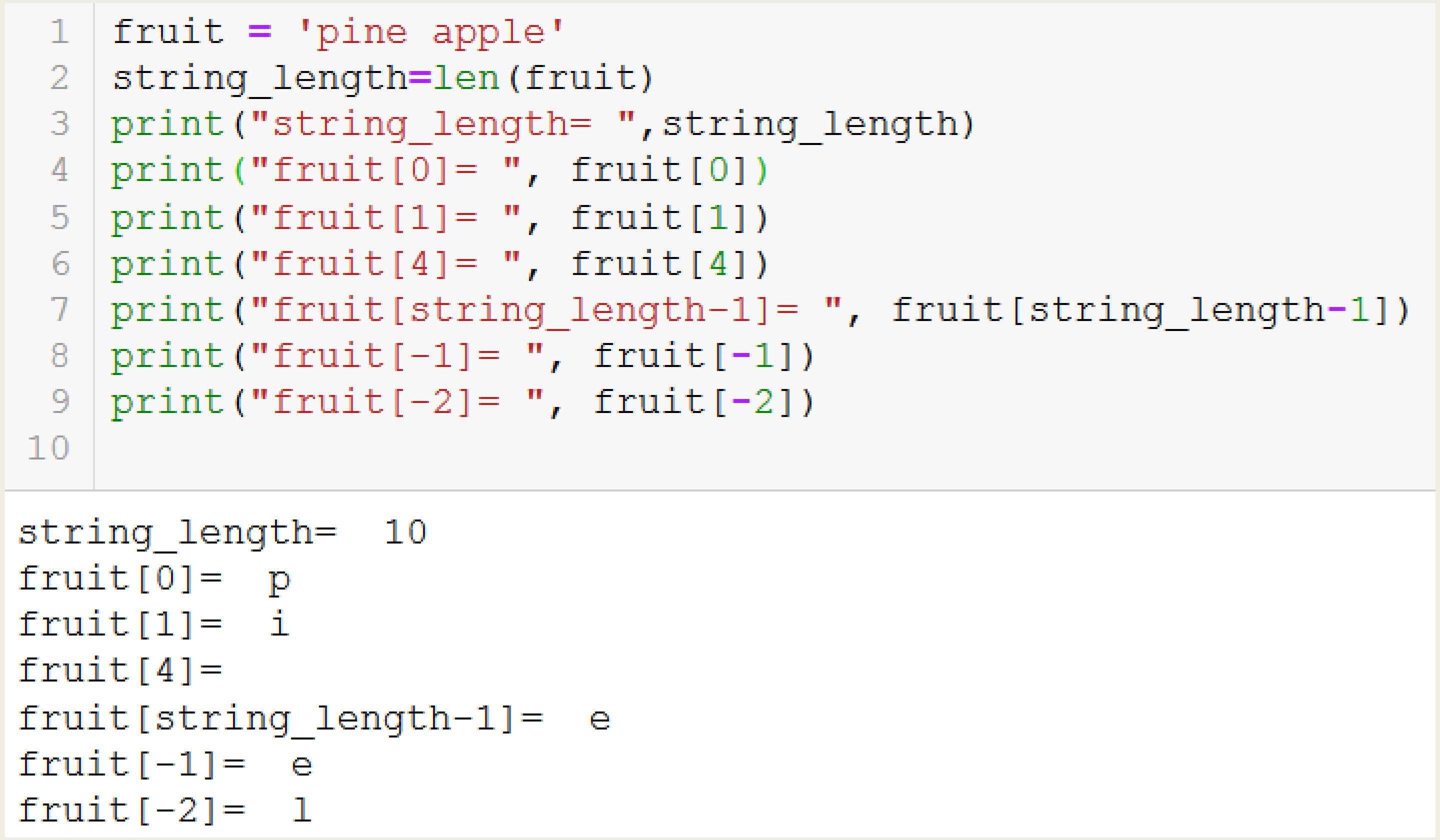
```
>>> letter
'a'
```

```
>>> i = 1
>>> fruit[i]
'a'
>>> fruit[i+1]
'n'
```

```
>>> letter = fruit[0]
>>> letter
'b'
```

len()

- Length of a string
- Syntax
  len(name_of_string)



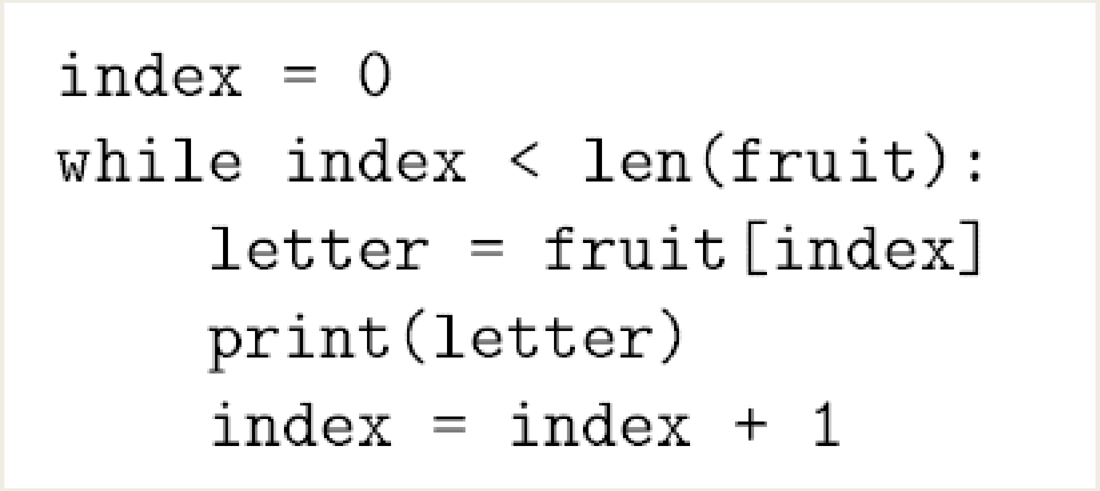
```python
fruit = 'pine apple'
string_length=len(fruit)
print("string_length= ",string_length)
print("fruit[0]= ", fruit[0])
print("fruit[1]= ", fruit[1])
print("fruit[4]= ", fruit[4])
print("fruit[string_length-1]= ", fruit[string_length-1])
print("fruit[-1]= ", fruit[-1])
print("fruit[-2]= ", fruit[-2])
```

```
string_length=  10
fruit[0]=  p
fruit[1]=  i
fruit[4]=
fruit[string_length-1]=  e
fruit[-1]=  e
fruit[-2]=  l
```

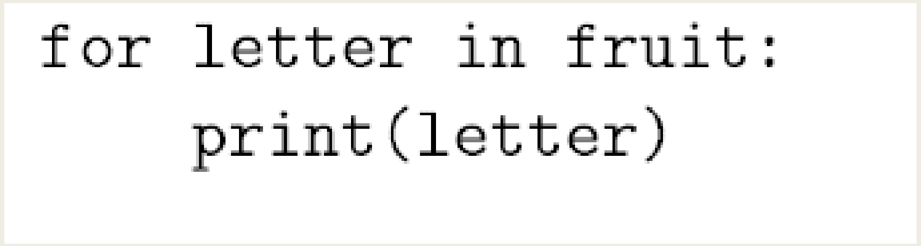
- String
  - *Beginning from element 0 to (length-1)*
  - *Negative index*
    - Counting from the end of string

Traversal of a string

- Traversal of a string
  - *Going through each element*
- Repetition
  - *while*
  - *for*



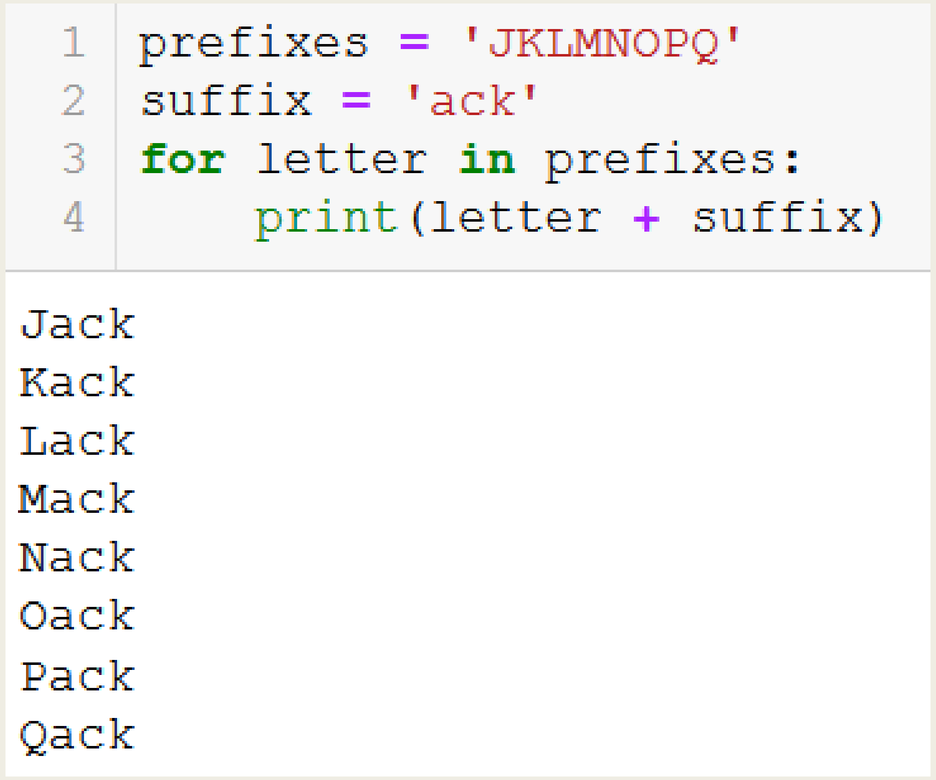
```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```



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

Example: concatenation using loop

- String addition (concatenation)
  - *For loop*
  - *string + string*

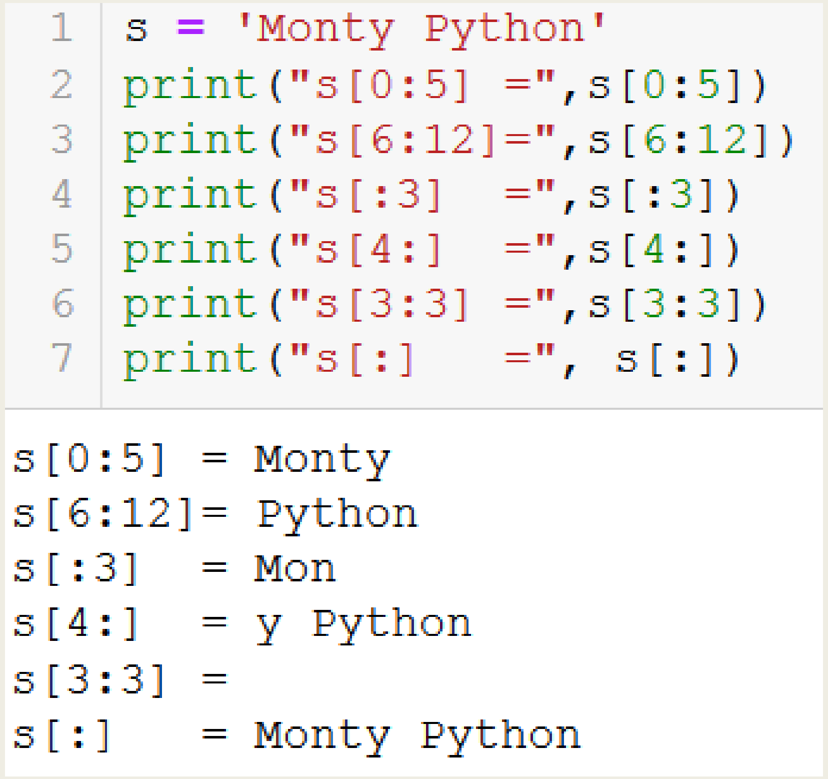


```python
prefixes = 'JKLMNOPQ'
suffix = 'ack'
for letter in prefixes:
    print(letter + suffix)
```

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

String slice

- Slice
  - *A segment of string*
- [n:m]
  - *Start from n-th element*
  - *End before m-th element*
    - Not include s[m]



```
s = 'Monty Python'
print("s[0:5] =",s[0:5])
print("s[6:12]=",s[6:12])
print("s[:3]  =",s[:3])
print("s[4:]  =",s[4:])
print("s[3:3] =",s[3:3])
print("s[:]   =", s[:])
```

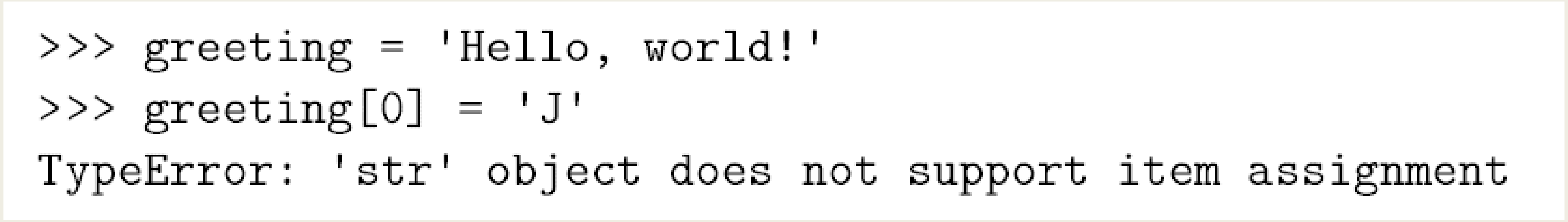
```
s[0:5] = Monty
s[6:12]= Python
s[:3]  = Mon
s[4:]  = y Python
s[3:3] =
s[:]   = Monty Python
```

String is immutable

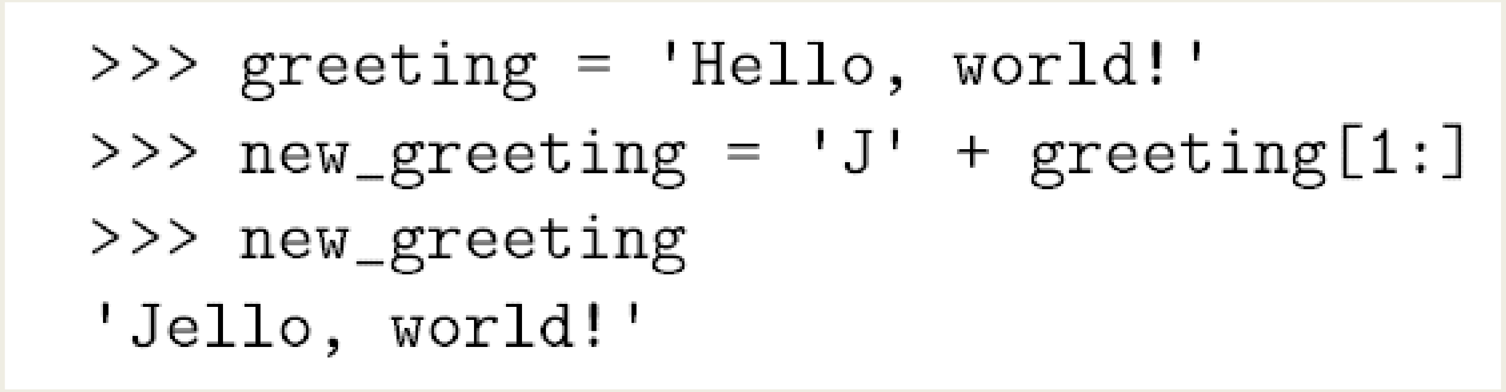
- Strings are **immutable**
  - *Cannot change an existing string*

TypeError

- If you want to modify a string,
  - *Create a new string, and fill in with desirable contents*



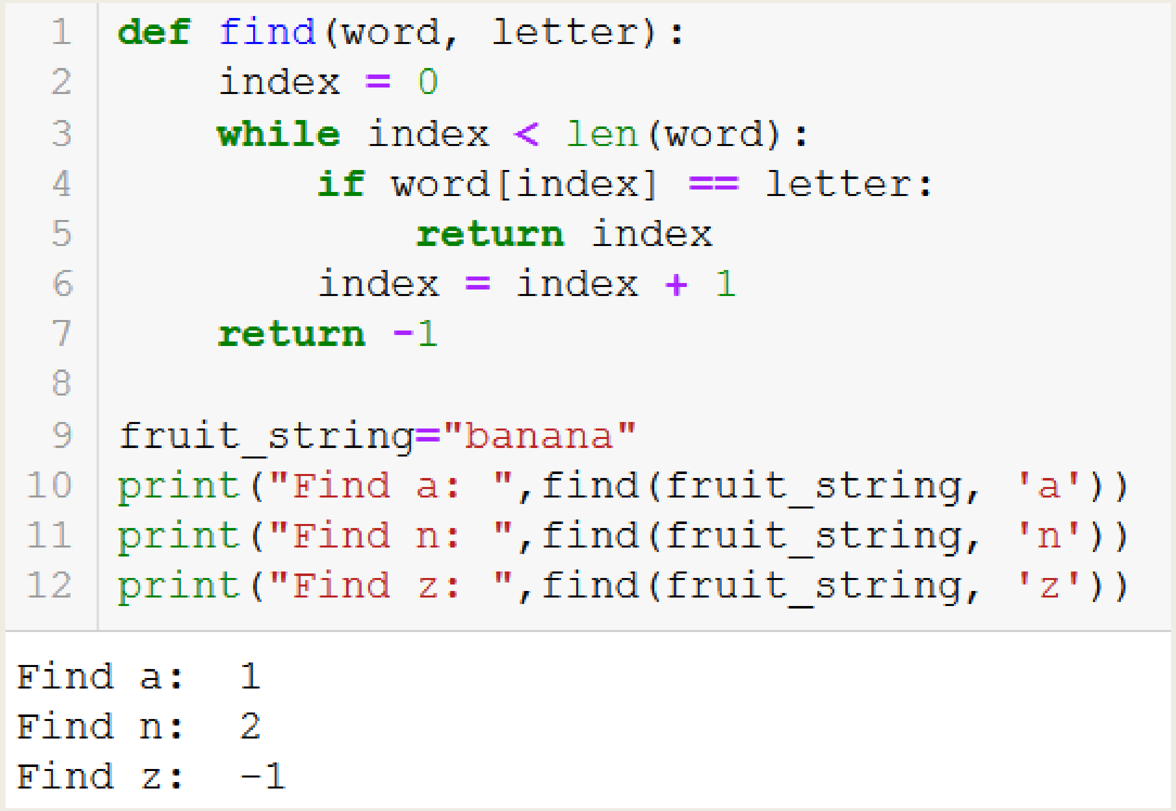
```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```



```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello, world!'
```

Searching in a string

- Search
  - *Find a character in a string*
  - *Return the index of the first matched element*

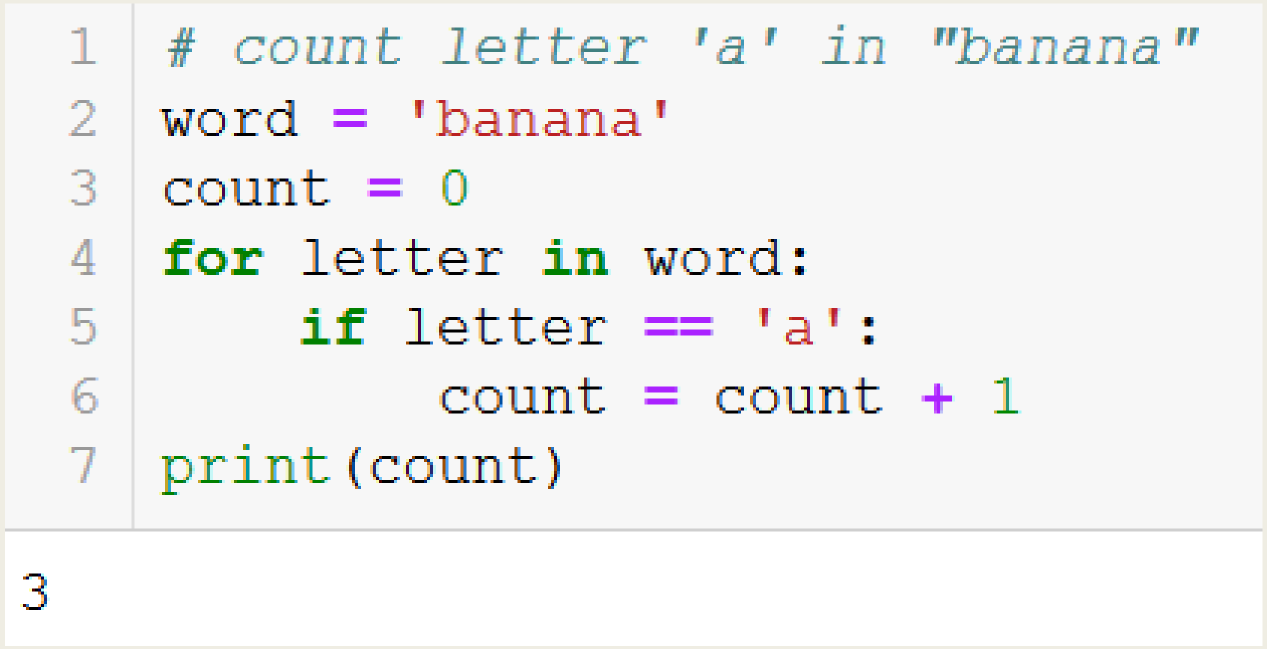


```python
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1

fruit_string="banana"
print("Find a:  ",find(fruit_string, 'a'))
print("Find n:  ",find(fruit_string, 'n'))
print("Find z:  ",find(fruit_string, 'z'))
```

```
Find a:   1
Find n:   2
Find z:   -1
```

Example: counting a letter in a string

- Count the number of times the letter appears in a string

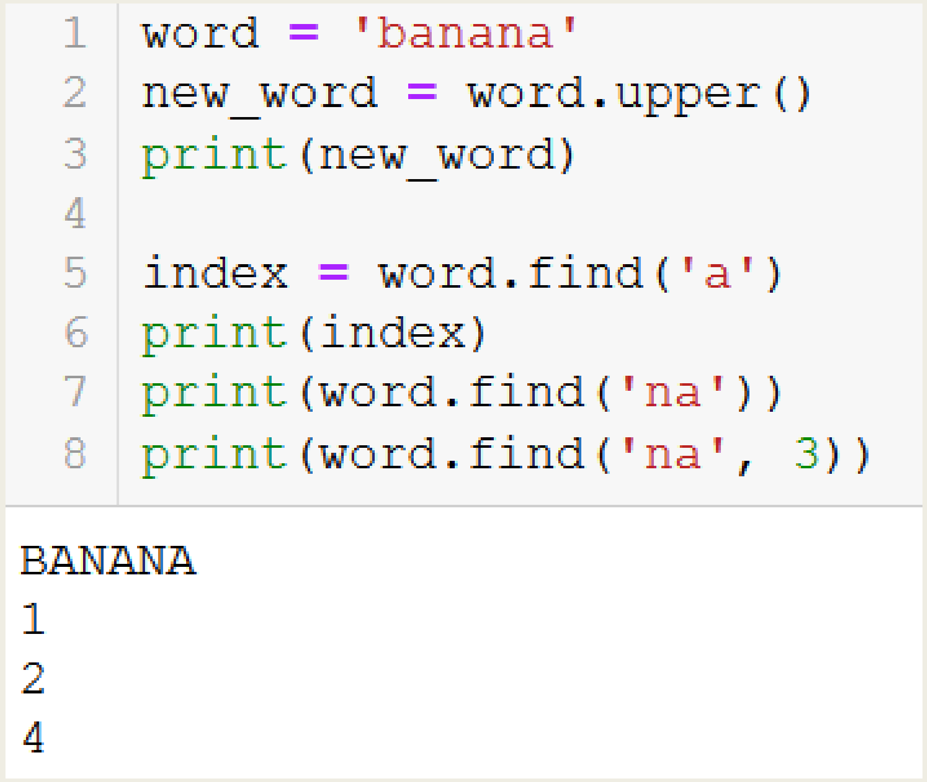


```python
# count letter 'a' in "banana"
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```

```
3
```

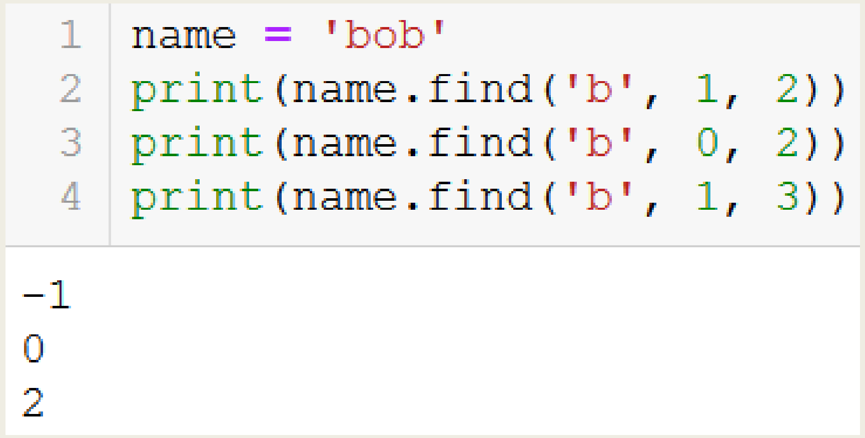
String methods: upper, find

- Convert to upper-case
  - *myString.upper()*
- Find a character/sub-string
  - *myString.find('a')*
  - *myString.find('na')*
  - *myString.find('na', start_position)*
  - *myString.find('na', start_pos, stop_pos)*



```python
word = 'banana'
new_word = word.upper()
print(new_word)

index = word.find('a')
print(index)
print(word.find('na'))
print(word.find('na', 3))
```

```
BANANA
1
2
4
```



```python
name = 'bob'
print(name.find('b', 1, 2))
print(name.find('b', 0, 2))
print(name.find('b', 1, 3))
```

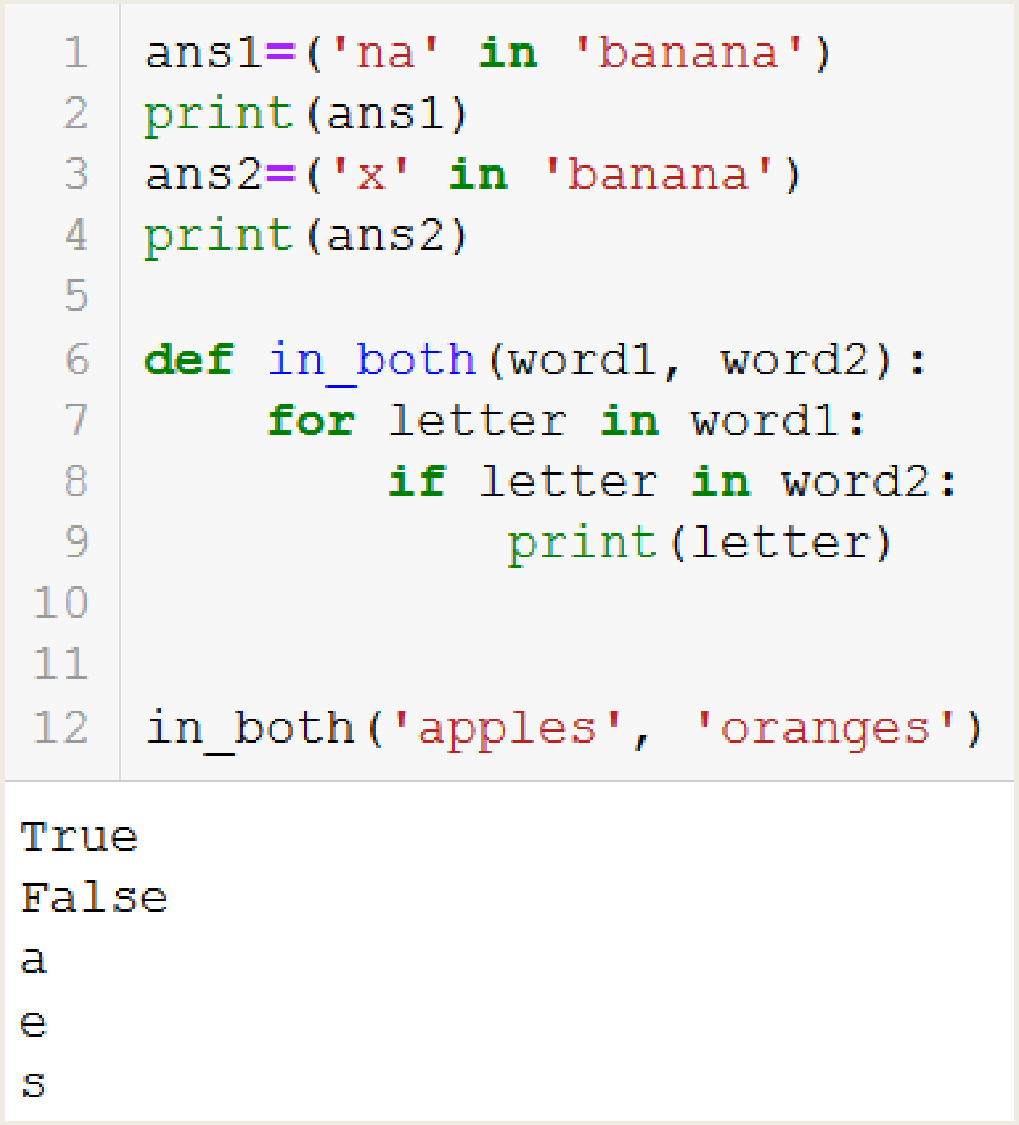
```
-1
0
2
```

in operator

- Boolean (True/False)
  - *Whether a sub-string or character in a string*

- Syntax

  'sub-string' ***in*** *'string'*

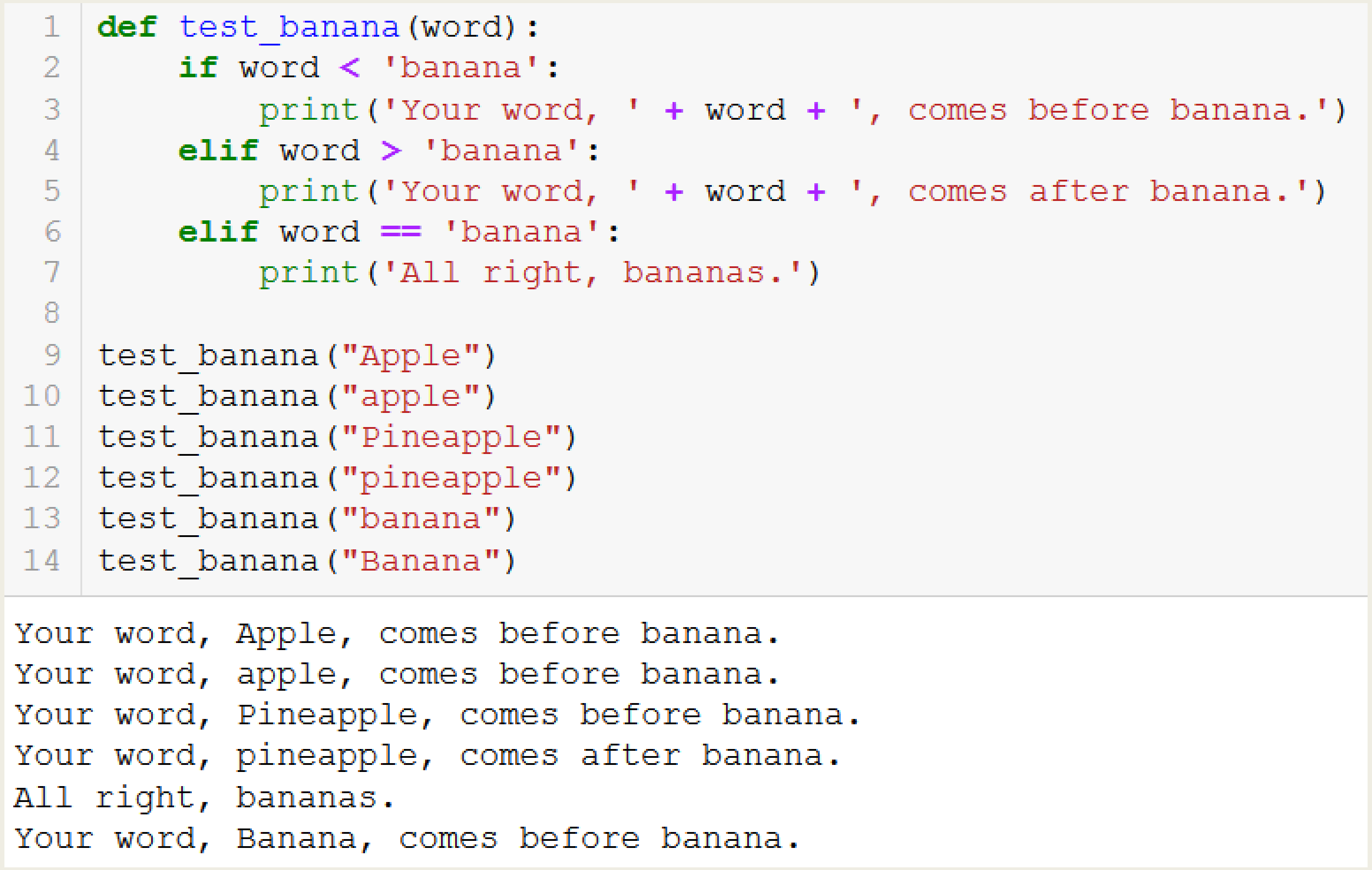


```python
ans1=('na' in 'banana')
print(ans1)
ans2=('x' in 'banana')
print(ans2)

def in_both(word1, word2):
    for letter in word1:
        if letter in word2:
            print(letter)


in_both('apples', 'oranges')
```

```
True
False
a
e
s
```

Relational operator for string

- ==
- >
- <



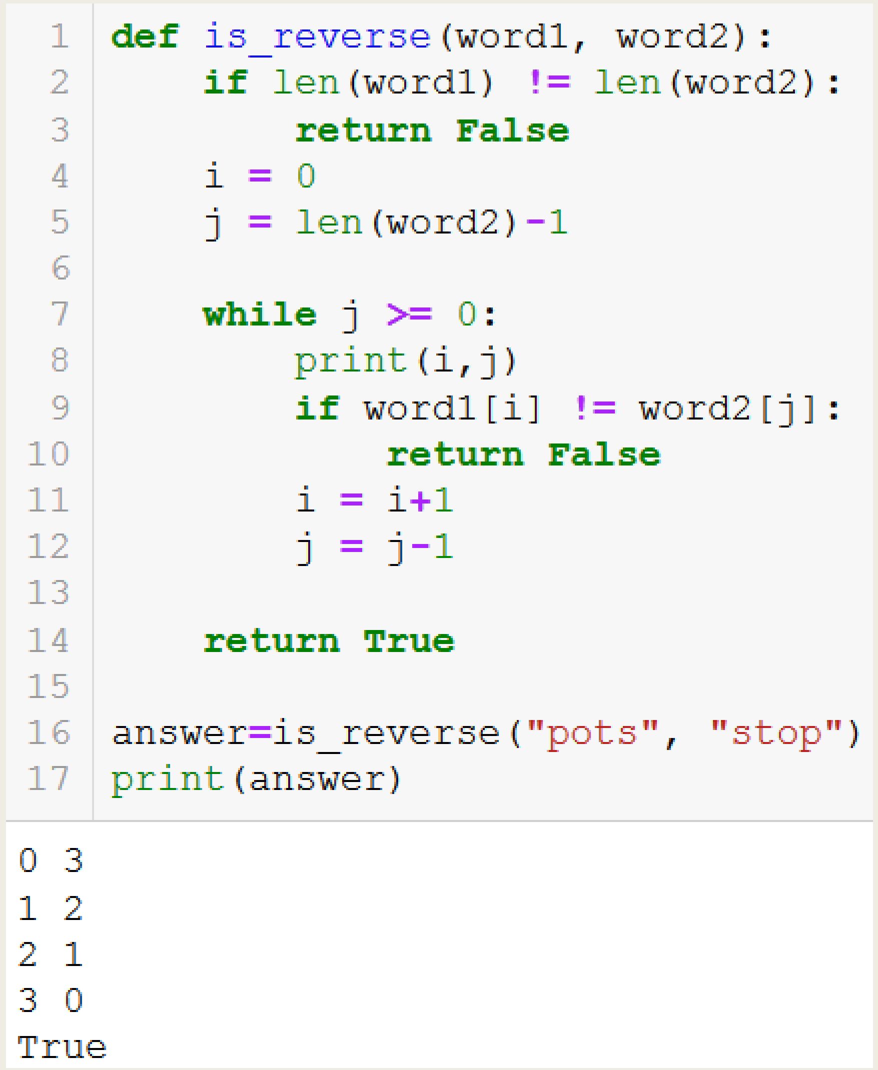
```python
def test_banana(word):
    if word < 'banana':
        print('Your word, ' + word + ', comes before banana.')
    elif word > 'banana':
        print('Your word, ' + word + ', comes after banana.')
    elif word == 'banana':
        print('All right, bananas.')

test_banana("Apple")
test_banana("apple")
test_banana("Pineapple")
test_banana("pineapple")
test_banana("banana")
test_banana("Banana")
```

```
Your word, Apple, comes before banana.
Your word, apple, comes before banana.
Your word, Pineapple, comes before banana.
Your word, pineapple, comes after banana.
All right, bananas.
Your word, Banana, comes before banana.
```

- UPPER case comes before lower case

Example: is_reverse

- Is a string the reverse version of the other string?
  - *Compare element-by-element*



```
def is_reverse(word1, word2):
    if len(word1) != len(word2):
        return False
    i = 0
    j = len(word2)-1

    while j >= 0:
        print(i,j)
        if word1[i] != word2[j]:
            return False
        i = i+1
        j = j-1

    return True

answer=is_reverse("pots", "stop")
print(answer)
```

```
0 3
1 2
2 1
3 0
True
```

Reading

- Chapter 8 in textbook “Think Python”