Python Programming

Day 3: *String Manipulation*

String Manipulation

- Traversal
- Slices
- Comparison
- Formatting

String

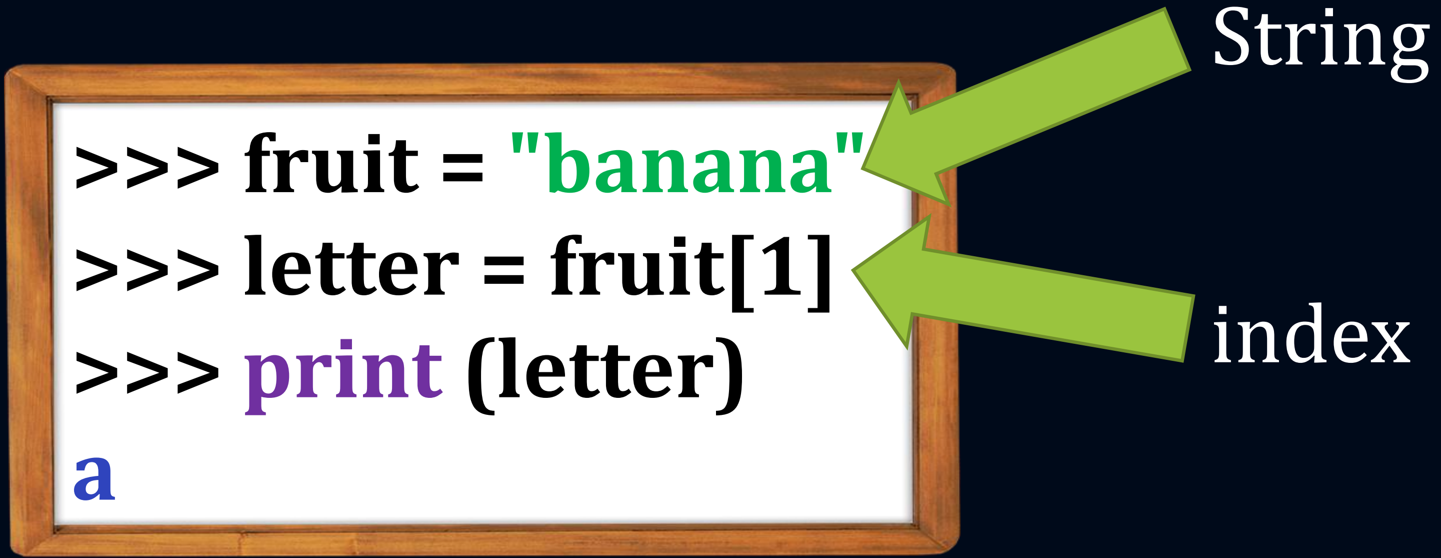
A string is traditionally a sequence of characters, either as a literal constant or as some kind of variable.

"the quick brown fox"

"0 1 2 3 4 5 6 7 8 9"

String - bracket operator

The bracket operator selects a single character from a string

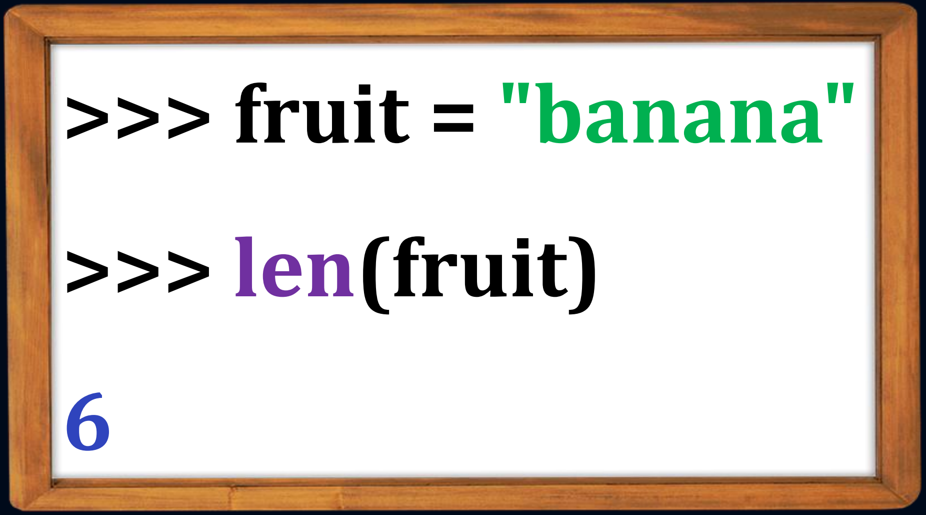


```
>>> fruit = "banana"
>>> letter = fruit[1]
>>> print (letter)
a
```

Python is zero index based

String - Length

The len function returns the number of characters in a string.



```
>>> fruit = "banana"
>>> len(fruit)
6
```

String

To get the last letter of a string, you might be tempted to try something like this:

```
>>> length = len(fruit)

>>> last = fruit[length]
```

String

To get the last letter of a string, you might be tempted to try something like this:

```
>>> length = len(fruit)
>>> last = fruit[length]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in
<module>
    last = fruit[length]
IndexError: string index out of
range
```

String

```
>>> length = len(fruit)
>>> last = fruit[length -1]
>>> print(last)
a
```

String

Alternatively, we can use negative indices, which count backward from the end of the string.

Try

fruit[-1]
Fruit[-2]

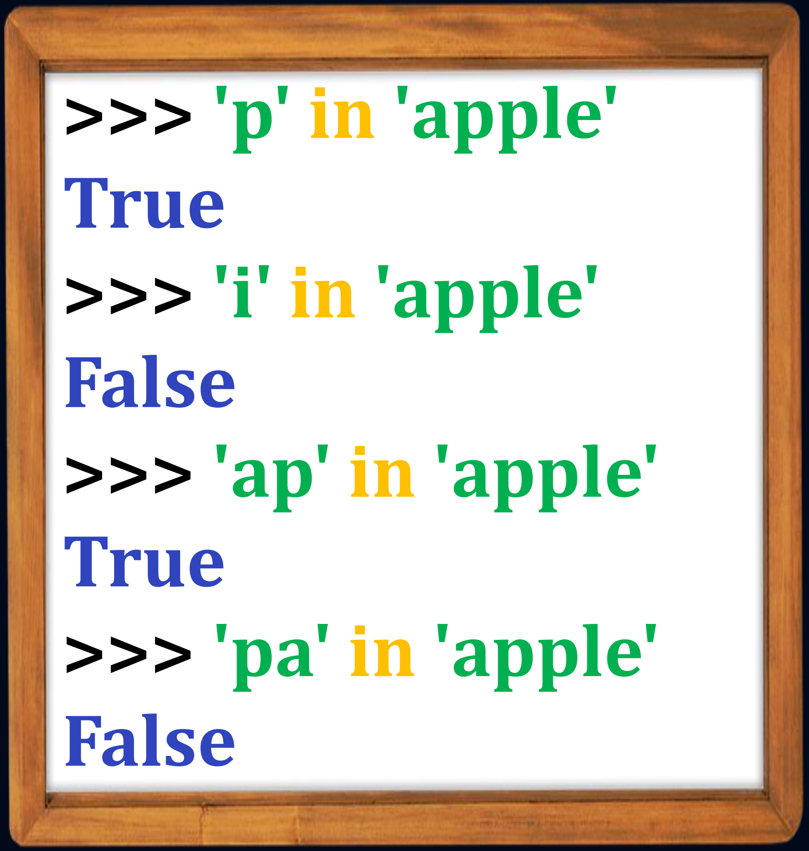
String

Alternatively, we can use negative indices, which count backward from the end of the string.

```
>>> fruit
'banana'
>>> fruit[-1]
'a'
>>> fruit[-2]
'n'
```

String - in operator

The in operator tests if one string is a substring of another



```
>>> 'p' in 'apple'
True
>>> 'i' in 'apple'
False
>>> 'ap' in 'apple'
True
>>> 'pa' in 'apple'
False
```

A string is a substring of itself
E.g. 'p' in 'apple'
True

String - in operator

Combining the *in* operator with string concatenation we can write a function that removes all the vowels from a string.

```
def remove_vowels(s):
    vowels = "aeiouAEIOU"
    s_without_vowels = ""
    for letter in s:
        if letter not in vowels:
            s_without_vowels += letter
    return s_without_vowels
```

String Traversal – while loop

String traversal is a process of visiting each item in a string. One way to encode a traversal is with a while statement:

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

String Traversal – for loop

Each time through the loop, the next character in the string is assigned to the variable char. The loop continues until no characters are left.

```
fruit = “banana”
for char in fruit:
    print (char)

prefixes = "JKLMNP"
suffix = "ack"
for letter in prefixes:
    print (letter + suffix)
```

Try this!

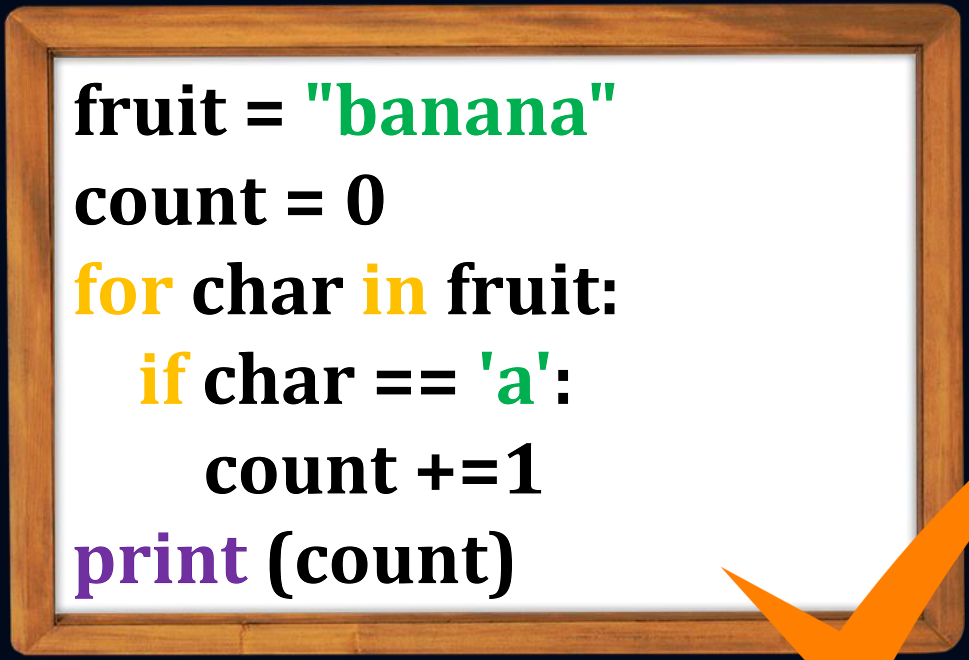
String concatenation

String Traversal – for loop

Write a program to count how many times “a” appears in the string “banana”

Hint: Use for loop with if statement

String Traversal – for loop



```
fruit = "banana"
count = 0
for char in fruit:
    if char == 'a':
        count +=1
print (count)
```

Hint: Use for loop with if statement

String - Slices

A substring of a string is called a slice. Selecting a slice is similar to selecting a character

```
>>> s = "Peter, Paul, and Mary"
>>> print (s[0:5])
Peter
>>> print (s[7:11])
Paul
>>> print (s[17:21])
Mary
```

The operator [n:m] returns the part of the string from the n-eth character to the m-eth character, including the first but excluding the last.

String - Slices

fruit →	b	a	n	a	n	a
Index	0	1	2	3	4	6

```
>>> fruit = "banana"
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

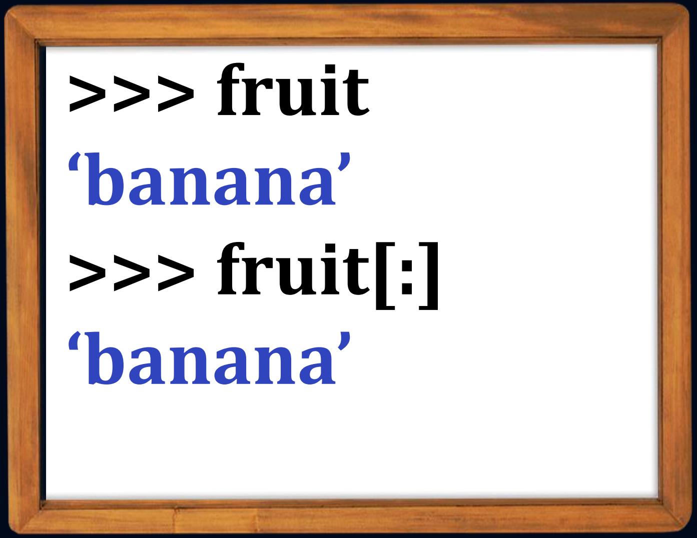
If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string.

What do you think fruit[:] means?

String Comparison

What do you think fruit[:] means?

It simply duplicate the whole string



```
>>> fruit
'banana'
>>> fruit[:]
'banana'
```

String Comparison

Write the Python interpreter's evaluation to each of the following expressions:

```
>>> 'Python'[1]
>>> "Strings are sequences of characters."[5]
>>> len("wonderful")
>>> 'Mystery'[:4]
>>> 'p' in 'Pinapple'
>>> 'apple' in 'Pinapple'
>>> 'pear' in 'Pinapple'
>>> 'apple' > 'pinapple'
>>> 'pinapple' < 'Peach'
```

String Comparison

```
>>> 'Python'[1] → y
>>> "Strings are sequences of characters."[5] → g
>>> len("wonderful") → 9
>>> 'Mystery'[:4] → ‘Myst
>>> 'p' in 'Pinapple' → True
>>> 'apple' in 'Pinapple' → True
>>> 'pear' in 'Pinapple' → False
>>> 'apple' > 'pinapple' → False
>>> 'pinapple' < 'Peach' → False
```

String Comparison

The comparison operators work on strings. To see if two strings are equal

```
if word == "banana":
   print ("Yes, we have banana")
```

String Comparison

Other comparison operations are useful for putting words in lexigraphical order

```
if word < "banana":
   print ("Your word, " + word + ", comes before banana")
elif word > "banana":
   print ("Your word, " + word + ", comes after banana")
else:
   print ("Yes, we have no banana")
```

String Comparison

- What will be the output of the code below (True or False)
- How can you get otherwise without altering line 2

Uppercase letters come before all the lowercase letters

```
>>> word = "Zebra"

>>>  word > "banana"
```

String Comparison

```
>>> word = "Zebra"
>>>  word > "banana"
False
```

Because the uppercase letters come before all the lowercase letters

Use string method to convert the string object before comparison

String methods	Description
s.lower()	Converts s to lowercase
s.upper()	Converts s to uppercase
s.capitalize()	Capitalize the first character

String Methods

String methods	Description
s.find(sub [,start [,end]])	Finds the first occurrence of the specified substring sub or returns -1
s.isalnum()	Checks whether all characters are alphanumeric and return True or False
s.isalpha()	Checks whether all characters are alphabets
s.isdigit()	Checks whether all characters are digits
s.islower()	Checks whether all characters are lowercase
s.strip()	Removes leading and trailing whitespace in s
s.split([sep])	Splits a string using sep as a delimiter and returns a list object

String - Immutable

Strings are immutable, which means you can’t change an existing string.

The best you can do is to create a new string that is a variation on the original.

```
>>> greeting = "Hello, world!"
>>> greeting[0] = "y"
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    greeting[0] = "y"
TypeError: 'str' object does not support item assignment
```

String - Immutable

The solution here is to concatenate a new first letter onto a slice of greeting. This operation has no effect on the original string.

```
>>> greeting = "Hello, world!"
>>> new_greeting = 'Y' + greeting[1:]
>>> print(new_greeting)
Yello, world!
```

String formatting

The % operator

String formatting enable us to combine strings and other data types efficiently.

The syntax for the string formatting operation looks like this:

"<FORMAT>" % (<VALUES>)

String formatting

The % operator

```
>>> age = 20
>>> "I am age " + age
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in
<module>
    "I am age " + age
TypeError: Can't convert 'int' object to
str implicitly
```

String formatting

The % operator

```
>>> "I am age %d" %age
'I am age 20'

>>> "The %s arrived and counted
%d %s" %("principal", 10,
"students")

'The principal arrived and counted
10 students'
```

Operator	Description
%s	Object is a string
%d	Object is an integer
%f	Object is a floating number

String formatting

The string.format method

```
>>> "I am age {0}".format(age)
'I am age 20'

>>> "The {0} arrived and counted
{1} {2}".format("principal", 10,
"students")

'The principal arrived and counted
10 students'
```

String formatting

The string.format method

String.format (key=value)

This provides more flexibility to specify interjections by use of name/key

```
>>> "The {x} arrived and counted {y} {z}".format(x = "principal", y = 10, z ="students")

'The principal arrived and counted 10 students'
```

Python Programming

Tutorials

Exercise 1:

Try each of the following formatted string operations in a Python shell and record the results:

a. “%s %d %f” % (5, 5, 5)

b. “%-.2f” % 3

c. “%-10.2f%-10.2f” % (7, 1.0/2)

d. print (“$%5.2fn $%5.2fn $%5.2f” % (3, 4.5, 11.2))

Exercise 2:

The following formatted strings have errors. Fix them:

a. “%s %s %s %s” % (‘this’, ‘that’, ‘something’)

b. “%s %s %s” % (‘yes’, ‘no’, ‘up’, ‘down’)

c. “%d %f %f” % (3, 3, ‘three’)

Solution 2:

a. “%s %s %s” % (‘this’, ‘that’, ‘something’)

b. “%s %s %s %s” % (‘yes’, ‘no’, ‘up’, ‘down’)

c. “%d %d %s” % (3, 3, ‘three’)

Exercise 3:

Given the sequences
dna = ’gcatgattacgact’ and dnasuite = ’ ccgttcctggcctcg’
a. Find the concatenation of the two strings to form a new string.
b. Find the length of the dna
c. Replace ‘a’ with ‘A’ in the dna sequence.
d. Find the occurrences of ‘a’ and ‘g’ in the dna sequence.

Solution 3:

```
dna = "gcatgattacgact"
dnasuite = "ccgttcctggcctcg"
(a)
#concatenation of the two strings
new_string = dna + dnasuite
print(new_string)
(b)
# length of dna
print(len(dna))
```

Solution 3 *cont.*:

```
(c)
#Replace ‘a’ with ‘A’ in the dna sequence.
new_dna = ''
for char in dna:
    if char == 'a':
        char = 'A'
    new_dna += char
print(new_dna)
(d)
# Find the occurrences of ‘a’ and ‘g’ in the dna sequence
count_a = 0
count_g = 0
for char in dna:
    if char == 'a':
        count_a += 1
    elif char == 'g':
        count_g += 1
print ("'a' occurred %d times" %count_a)
print ("'g' occurred %d times" %count_g)
```

Exercise 4:

Create a function named subCount() having both local and global variable count. It increments local variable count while global variable count remain constant.

Solution 3:

```
global_count = 1
def subCount():

    local_count = 1
    for i in range(10):
        local_count += 1
    print(global_count)
    print(local_count)

subCount()
```

Exercise 5:

Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.

Sample String : 'w3resource'
Expected Result : 'w3ce'
Sample String : 'w3'
Expected Result : 'w3w3'
Sample String : ' w'
Expected Result : Empty String

Solution 5:

```
def string_both_ends(str):
  if len(str) < 2:
    return ''

  return str[0:2] + str[-2:]

print(string_both_ends('w3resource'))
print(string_both_ends('w3'))
print(string_both_ends('w'))
```

Exercise 6:

Write a Python function that takes a list of words and returns the length of the longest one.

Solution 6:

```
def find_longest_word(words_list):
    word_len = [ ]
    for n in words_list:
        word_len.append((len(n), n))
    word_len.sort()
    return word_len[-1][1]

print(find_longest_word(["PHP", "Exercises", "Backend"]))
```

Exercise 7:

Write a Python program to change a given string to a new string where the first and last chars have been exchanged.

Solution 7:

```
def change_sring(str1):
    return str1[-1:] + str1[1:-1] + str1[:1]


print(change_sring('abcd'))
print(change_sring('12345'))
```

Exercise 8:

Write a Python script that takes input from the user and displays that input back in upper and lower cases.

Solution 8:

```
user_input = input("What's your favourite language? ")
print("My favourite language is ", user_input.upper())
print("My favourite language is ", user_input.lower())
```

Exercise 9:

Write a Python program to remove a newline in Python

Solution 9:

```
str1='Python Exercises\n'
print(str1)
print(str1.rstrip())
```

Next Lecture ...

Python Programming

Day 4: Sequence Data Types