Lecture 5 – Strings – Part 1 Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A string is a **sequence of characters**. You can access the characters one at a time with the bracket operator:

PROGRAM

fruit = 'banana'

letter = fruit[1]

print letter

So b is the 0th letter (“zero-eth”) of 'banana', a is the 1th letter (“one-eth”), and n is the 2th (“two-eth”) letter.



len is a built-in function that **returns the number of characters** in a string:

PROGRAM

>>>**fruit = 'banana'**

>>>**fruit[:3]**

**'ban'**

>>>**fruit[3:]**

**'ana'**

What is the error message? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Does this fix it?\_\_\_\_\_ Change to last = fruit[length-1]

PROGRAMin PyCharm

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

**Do this - Exercise 1**  Write a*while*loop that starts at the last character in the string and works its way backwards to the first character in the string, printing each letter on a separate line, except backwards.

**Try this –**

fruit = **'banana'
for** char **in** fruit:
 print (char)

**PROGRAM**

s = **'Monty Python'**print (s[0:5])

Change the values a few times and check the output

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**:

**fruit = 'banana'**

**fruit[:3]**

**'ban'**

**fruit[3:]**

**'ana'**

**Exercise 2**  Given that*fruit*is a string, what does*fruit[:]*mean? (Try it)

READ

## 6.5  Strings are immutable

It is tempting to use the [] operator on the left side of an assignment, with the intention of changing a character in a string. For example:

>>> greeting = 'Hello, world!'

>>> greeting[0] = 'J'

TypeError: object does not support item assignment

The “object” in this case is the string and the “item” is the character you tried to assign. For now, an **object** is the same thing as a value, but we will refine that definition later. An **item** is one of the values in a sequence.

The reason for the error is that strings are **immutable**, which means you can’t change an existing string. The best you can do is create a new string that is a variation on the original:

**PROGRAM**

greeting = **'Hello, world!'**new\_greeting = **'J'** + greeting[1:]
print (new\_greeting)

This example concatenates a new first letter onto a slice of greeting. It has no effect on the original string.

word = **'banana'**count = 0
**for** letter **in** word:
 **if** letter == **'a'**:
 count = count + 1
print (count)

This program demonstrates another pattern of computation called a **counter**.

**Exercise 3**

*Encapsulate this code in a function named count, and generalize it so that it accepts the string and the letter as arguments.*

(This is a bit of a challenge)

Boolean operator in – try in Python editor (Chevron prompt)

>>> 'a' in 'banana'

>>> 'T' in 'banana'

PROGRAM

word = input(**'enter a word'**)
**if** word < **'banana'**:
 print (**'Your word,'** + word + **', comes before banana.'**)
**elif** word > **'banana'**:
 print (**'Your word,'** + word + **', comes after banana.'**)
**else**:
 print (**'All right, bananas.'**)

Python does not handle uppercase and lowercase letters the same way that people do. All the **uppercase letters** come before all the **lowercase letters**.

A common way to address this problem is to convert strings to a standard format, such as all lowercase, before performing the comparison.

PROGRAM in PyCharm

stuff = **'banana'**print(stuff)
print(str.capitalize(stuff))
print(stuff)

TRY THIS IN THE EDITOR (Chevron Prompt)

>>> stuff = 'Hello world'

>>> type(stuff)

>>> dir(stuff)

>>> help(str.capitalize)

PROGRAM

stuff = **'banana'**print(stuff)
print(str.capitalize(stuff))
print(stuff)
print(stuff.upper)

word = **'pomegranate'**print(word.upper())

This form of dot notation specifies the name of the method, upper, and the name of the string to apply the method to, word. The empty parentheses indicate that this method takes no argument.

A method call is called an **invocation**; in this case, we would say that we are invoking upper on the word.

For example, there is a string method named find that searches for the position of one string within another:

>>> word = 'banana'

>>> index = word.find('a')

>>> print index

TRY IN EDITOR

>>> line = ' Here we go '

>>> line.strip()

TRY IN EDITOR

>>> line = 'Please have a nice day'

>>> line.startswith('p')

False

>>> line.lower()

'please have a nice day'

>>> line.lower().startswith('p')

True

Do Exercise 4 – See below and play around with it.

str = **"this is string example....wow!!!"**sub = **"i"**print (**"str.count(sub, 4, 40) : "**, str.count(sub, 0, 40))
sub = **"wow"**print (**"str.count(sub) : "**, str.count(sub))