Strings.

Chapter 8

Two strings try to sneak past a checkpoint…

Today's Outline

- Data types
- Working with strings (length, indexing, string slicing). Why might this be important?
  - Concatenating elements (e.g. web scraping)
  - Renaming Files, outputs, values
  - Anytime you want to manipulate a string value!
- Another form of iteration!
  - for loop (with range function)
- More fun with strings (as time allows)

Revisit Values & Types

- What are common value types?
  - `type(4)`
    - `type 'int'`
  - `type(3.14159)`
    - `type 'float'`
  - `type('Hello World!')`
    - `type 'str'`
- What's a compound data type?
  - Types that comprise smaller pieces.
  - `int` and `float` are simple data types
  - a string consists of smaller pieces: it is a string of characters!

Length Function

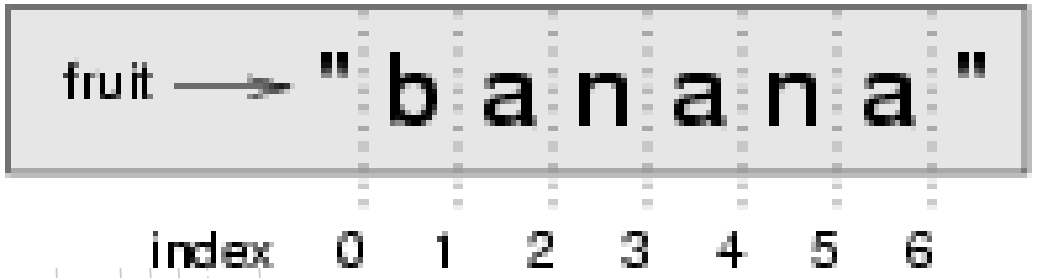
- `len` is a built in function that returns the numbers of characters in a string.

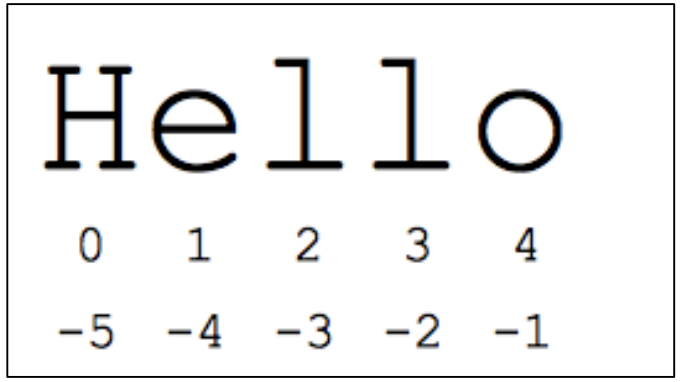
```
fruit = "apple"
len(fruit) # This gets the length
# What's the output

length = len(fruit)
last = fruit[length-1]
last
# What's the output
```

Index of a slice

- We need to understand how strings are *indexed* or "how Python counts the spaces."



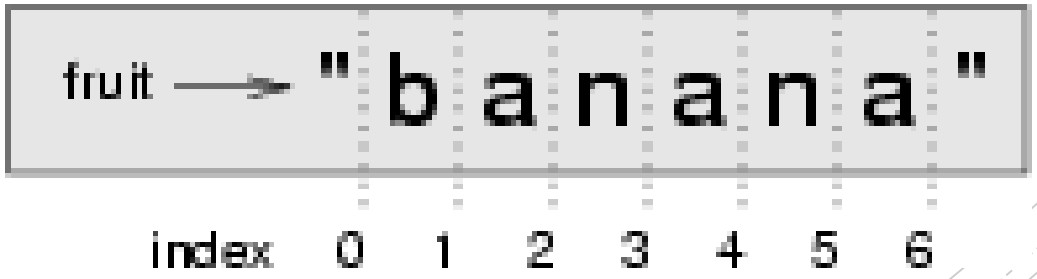


- This is important for "slicing."
- A *slice* is a segment of a string, which we get with the operator `[x:y]` defines start and end points.

Slicing

- If you **omit the first index** (before the colon) slice starts at the beginning of the string (at the "zero place").

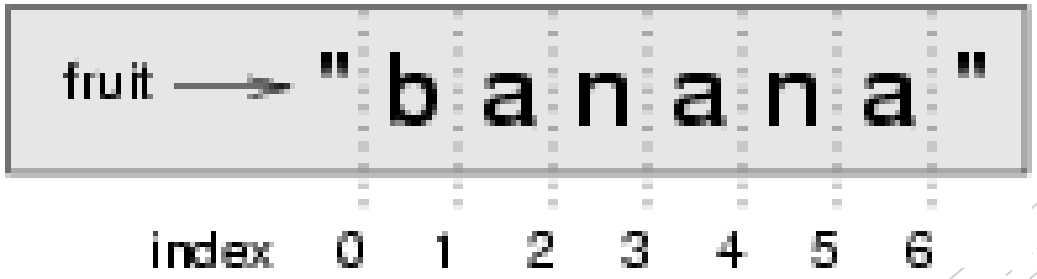
```
greeting = "hello"
letter = greeting[:3]
print(letter)
#What's the output?
```



Slicing

- If you **omit the second index** (after the colon) slice goes to the end of the string.

```
greeting = "hello"
letter = greeting[3:]
print(letter)
#What's the output?
```



Traversal using while loop

- Print each letter of a string. Use a while loop to traverse each character.

```
fruit = "banana"
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
# What's the output?
```

Traversal using while loop

- Let's go through what's happening here:

```
fruit = "banana"
index = 0
while index < len(fruit): #loop condition!
    letter = fruit[index]
    print(letter)
    index = index + 1
```

for Loop: more iteration!

- for loops are used when you have a piece of code which you want to repeat some number of times.
- General format for a for loop:

```
for item in object:
    statements to do stuff
```

- Add letters to suffixes

```
letter = "LB" #Watch out for spaces!
suffix = "ike"
for characters in letter:
    print(characters + suffix)
#What's the output?
```

for Loop

- You don't have to specify "characters" in the example below: it automatically stands in as a variable to represent items in the list (or object).
- I often see code with things like, `for x in...` or `for i in...`

```
letter = "LB" #Watch out for spaces!
suffix = "ike"
for characters in letter:
    print(characters + suffix)
```

for Loops using range()

- In loops, the `range()` function controls how many times the loop repeats.
- You can pass three arguments to it: `range(0,100,1)`
  - start states the integer value at which the sequence begins, if this is not included then start begins at 0
  - Endpoint of the sequence. **This item will not be included in the sequence**.
  - step is the integer that is counted up to but not included

for Loops using range()

- Example

```
for i in range(0,5):
    print(i)
```

- What is the output? Why?
  - 0,1,2,3,4
- What if I replace ‘i’ with “jelly”?

```
for jelly in range(0,5):
    print(jelly)
```

String Comparison

- Used to check if strings are equal.
- Why is this important?
  - Putting names, things, in order.
- However.
  - If strings aren't integers, how is this possible?
- Python compares string lexicographically i.e using ASCII value of the characters.

String Comparison

- ASCII stands for American Standard Code for Information Interchange.
- Computers can only understand numbers
- ASCII code is the numerical representation of a character ('a' or '@‘)
- Comparing strings are actually comparing the corresponding ASCII codes of their characters.

String Comparison Ex.

Suppose you have

```
str1 = "Mary"
str2 = "Mac"
```

The first two characters from `str1` and `str2` (M) are compared. As they are equal, the second two characters are compared. Because they are also equal, the third two characters (r and c) are compared. Because 'r' has greater ASCII value than 'c' , `str1` is greater than `str2`.

String Comparison Ex.

- Suppose you have

```
str1 = "Mary"
str2 = "Mac"

print(str1 < str2)
```

- What's the output? Now what if you make a change…

```
str1 = "Mary"
str2 = "mac" # change this to lower case
print(str1 < str2)

#What's the output?
```

String Comparison Ex.

- Fix this by converting all strings to a standard format.
- Here we use `.lower()` method to make everything lower case at the time of comparison.

```
str1 = 'Mary'
str2 = 'mac'

Print(str1.lower() < str2.lower())
```

- Is there another method we could use?

  `.upper()`

More on Strings

- They are immutable! You cannot modify them! Try making “Hello World” into ”Jello World”.

```
greeting = 'Hello, world!’
greeting[0] = 'J’
```

What’s the output? What’s the fix?
You need to make a new string!

```
greeting = 'Hello, world!’
newgreeting = 'J' + greeting[1:]
print(newgreeting)
```

count Function

- Talk me through this, line by line:

```
1. word = 'Zanzibar'
2. count = 0
3. for letter in word:
4.             if letter =='z':
5.                 count = count + 1
6. print(count)
```

What's the output? What's the fix?

```
for letter in word.lower():
```

String methods

- Which is the proper syntax to make a the variable 'country' all upper case?

```
country = "Tanzania"
country_uppercase = upper(country)
```

...or

```
country = "Tanzania"
country_uppercase = country.upper()
```

Summary

- Data types
- Working with strings (length, indexing, string slicing). Another form of iteration!
- for loop (with range function)
- More fun with strings and things to look out for: always standardize your data!