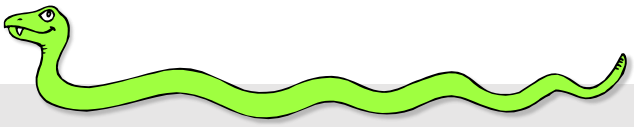
Agenda

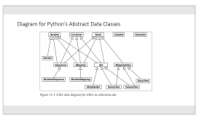
- Recap String (Text Sequences) and String operations
- Recap Regular Expressions
- Lab Reinforcement

# Quick Recap: Python Text Sequences



## Python's String Data Type

- A string is a **sequence** of characters
- A string literal uses quotes 'Hello' or "Hello"
- For strings, + means "concatenate"
- When a string contains only numeric digits, it is still a string!
- We can convert a string composed entirely of numeric digits into a number using int() or float()



```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
```

## Looking Inside Strings

- We can get at any single character in a string using an index specified in square brackets (aka., accessor or index operator)
- The index must be an integer value and indices start at zero
- The index value can also be any valid expression resulting in an integer
- NOTE: applies to ALL sequences

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

| b | a | n | a | n | a |
|---|---|---|---|---|---|

## Strings Have Length – len() Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

## Looping Through Strings

- Indefinite loop
- Using a while statement, an iteration variable, and the len function, we can construct a loop to look at each of the letters in a string individually

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

```
0 b
1 a
2 n
3 a
4 n
5 a
```

## Looping Through Strings

- A definite loop using a for statement is more “pythonic”
- The iteration (*letter* in this case) variable is completely taken care of by the compound statement ‘for’

```
fruit = 'banana'
for letter in fruit :
    print(letter)
index = 0
```

```
b
a
n
a
n
a
```

## Looping Through Strings

- A definite loop using a for statement is more “pythonic”
- The iteration variable (*letter* in this case) is completely taken care of by the compound statement ‘for’

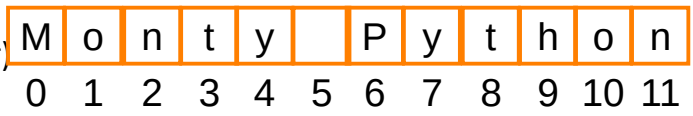
```
fruit = 'banana'
for letter in fruit :
    print(letter)
index = 0

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

```
b
a
n
a
n
a
```

## Slicing *Sequences* and Thus Strings

- We can also look at any continuous section of a string using a colon (slice) operator
- The second number is one beyond the end of the slice - "**up to but not including**"
  - slice [a:b] , results in (b-a) items
- If the second number is beyond the end of the string, it stops at the end (not an error)

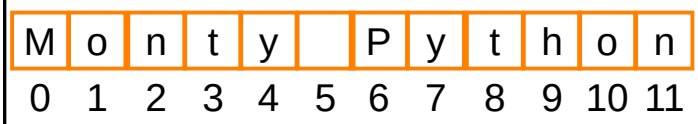
```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```




## Slicing Strings

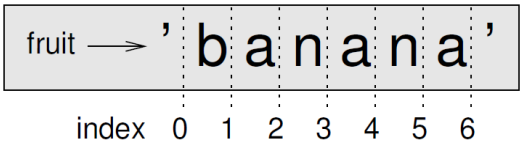
- If we leave off the first number of the slice (before the : ) it is assumed to be the beginning -ie, 0
- If we leave off the last number of the slice (after the : ), it is assumed to be the end of the string

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```



## Indexing and Slicing Strings from the Right

- Negative index values pull from the end of the sequence
- Negative slice values slice from the end of the sequence




```
>>> s = 'banana'
>>> print(s[-1])
a
>>> print(s[-2])
n
>>> print(s[-2:])
na
>>> print(s[-3:])
ana
```

## Using in as a Logical Operator

- The in keyword can also be used to check to see if one string is "in" another string
- The in is a logical expression that returns **True** or **False** and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

https://docs.python.org/3/library/stdtypes.html#string-methods

## Summary

- String built-in data type
  - str Class
- Looping through strings with for and while
  - for - strings are iterable
  - while - strings are sequences and can be indexed into
- Indexing strings via index operator – []
- Slicing strings via slice operator – [2:4], [:]

## Try It…

- Complete the text sequence lab in the Canvas module *Unit - Manipulating Text Sequence Data*

## Diagram for Python's Abstract Data Classes

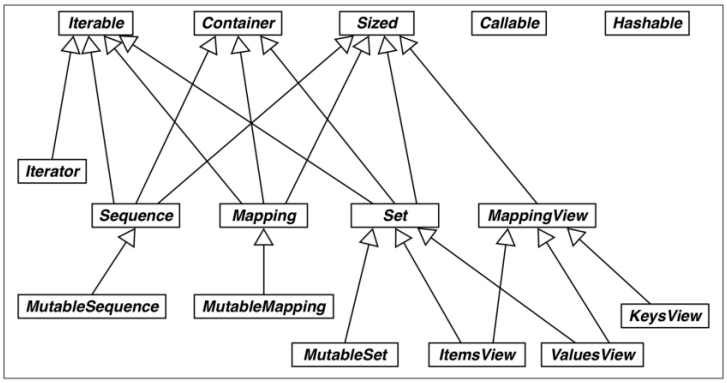



*Figure 11-3. UML class diagram for ABCs in collections.abc*