String Manipulations

Thomas Schwarz, SJ

String Methods

- Strings are an example of a class
  - Classes have methods, and strings have a variety of useful methods
  - Python supplied methods tend to be very fast and are preferable to what we could code

String Methods

- There are a number of methods for strings. Most of them are self-explaining
  - `s.lower(), s.upper()`:
    - returns the lowercase or uppercase version of the string
  - `s.strip()`:
    - returns a string with whitespace removed from the start and end
  - `s.isalpha()/s.isdigit()/s.isspace()`:
    - tests if all the string chars are in the various character classes

String Methods

- There are a number of methods for strings. Most of them are self-explaining
- `s.startswith('other'), s.endswith('other')` :
  - tests if the string starts or ends with the given other string
- `s.find('other')`
  - searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found
- `s.replace('old', 'new')`
  - returns a string where all occurrences of 'old' have been replaced by 'new'

Strings and Characters

- Python does not have a special type for characters
  - Characters are just strings of length 1.

String Indexing

- Strings are *immutable:*
  - They cannot NOT be altered
  - But we can access individual parts of strings

String Indexing

- We use the bracket notation to gain access to the characters in a string
  - Numbering starts with 0 (for historical reasons)
  - `a_string[3]` is character number 3, i.e. the fourth character in the string

String Indexing

- Negative numbers are counted from the back,
  - starting with -1 for the last,
  - -2 for the penultimate letter
  - etc.

String Indexing

- Example:
  - Define a string:
    - `astring = 'hello world'`
    - `astring[-1]` is `'d'`
    - `astring[-2]` is `'l'`
    - `astring[0]` is `'h'`
    - `astring[1]` is `'e'`

String Indexing

- Slices:
  - We can create subsections of strings with slices
    - Notation uses the bracket and the colon
    - `a_string[a:b]` is a ***new*** consisting of the letters from `a` (start) to one before `b` (stop value)
    - A third parameter is the *stride*
    - Default values are beginning and end

String Indexing

- Slicing examples:
  - Define a string: `astring = 'hello world'`
  - Then:
    - `astring[0:5]` gives `'hello'`
    - `astring[3:10]` gives `'lo worl'`

String Indexing

- Examples:
  - `astring[:10]` gives `'hello worl'`
  - `astring[5:]` gives `' world'`
  - `astring[:]` makes a ***copy*** of the original string
    - This is a Python idiom, learn it

String Indexing

- Strides other than -1 are less frequently used
  - `astring[::2]` gives `'hlowrd'`
    - (Every other letter)

String Indexing

- A stride of -1 reverses:
  - `astring[::-1]` gives `'dlrow olleh'`
  - Since strings have an equality evaluator, this allows us to test for a palindrome quickly
  - A palindrome is a word or phrase that reads the same forward and backward
    - E.g. 'able was I ere I saw elba'

String Indexing

- First stab:

```
def is_a_palindrome(a_string):
    return a_string == a_string[::-1]
```

- This is faster than a “manual” method:

```
def palindrome(a_string):
    for i in range(len(astring)//2):
        if a_string[i]!= a_string[-i-1]:
            return False
    return True
```

Aside: timing

- We can check with the timeit module
  - Used to time snippets of python code
    - Define the two ways of palindrome calculation

```
def is_a_palindrome(a_string):
    return a_string == a_string[::-1]

def is_a_palindrome_manual(a_string):
    for i in range(len(a_string)//2):
        if not a_string[i] == a_string[-i-1]:
            return False
    return True
```

Aside: timing

- Set up three tests:

```
test = 'able was I ere I saw elba'
test1 = 'madamimadam'
test2 = 'voila'
```

- And run them

```
for t in [test, test1, test2]:
    print(t)
    for _ in range(10):
        print(timeit.timeit('is_a_palindrome(t)', globals=globals() ))
    for _ in range(10):
        print(timeit.timeit('is_a_palindrome_manual(t)', globals = globals()))
```

  - globals are needed in order to allow timeit to find the functions

Aside: timing

- Results: Number 1 is 2-8 times faster

```
madamimadam
0.08961983400513418
0.08896537500550039
0.08881758300412912
0.08956966700498015
0.08922679100942332
0.08946112499688752
0.08913508300611284
0.0890048329893034
0.08985433299676515
0.08941579100792296
0.3933403749979334
0.3930454159999499
0.39426012498734053
0.3996310420043301
0.39395316698937677
0.3976135419943603
0.39330237499962095
0.3946556250011781
0.3942780000070343
0.39377504200092517
```

```
able was I ere I saw elba
0.09834004098956939
0.09333004200016148
0.09327612500055693
0.09313091600779444
0.09320745800505392
0.09366233299078885
0.0931825830048183
0.09330437499738764
0.09343683300539851
0.09337479098758195
0.7937272920098621
0.8057669580011861
0.7939047079999
0.7947541250032373
0.79327695799293
0.7959979999868665
0.7983322079962818
0.7923826249898411
0.7974026669980958
0.7965114160033409
```

```
voila
0.08431341699906625
0.084524916994269
0.0841138329997193
0.08451995799259748
0.08391624999057967
0.08433470799354836
0.0848872089991346
0.08404904199414887
0.08421075000660494
0.08482595800887793
0.18094412500795443
0.18013808398973197
0.18068012500589248
0.18092833300761413
0.1799738330009859
0.1809990409965394
0.18096858300850727
0.18104429199593142
0.17901629199332092
0.182663457992021
```