Practical Python:

2: Data types, operators, methods

Dafydd Gibbon

First South African Workshop in Digital Humanities
North-Western University, Potchefstroom, SA
2015-04-04 to 2015-04-05

Overview of Python objects for modelling DH domains

- Simple data types
  - characters
    - 1-char strings
  - numbers:
    - integers
    - floats
- Complex data types
  - iterable:
    - strings
    - lists (mutable)
      - recursive
  - non-iterable:
    - sets (mutable)
      - recursive
    - dictionaries (mutable)
      - recursive
- Functions
  - built-in
    - pure
    - side-effects
  - home-made
- Local modules
  - components of applications
- Libraries
  - *numpy*
  - *matplotlib*
  - *sklearn*

Official Python Tutorial:

https://docs.python.org/2/tutorial/

Data types: strings

- Extending strings:

```
textstring = '"Good morning!"'
textstring += ' said John.'
textstring = textstring + ' She said.'

textstring
```

- Multi-line strings with triple-double-quotes:

```
storystring = textstring + """It was late in the
evening.
Next morning everything looked different.
The sky was blue. Birds were singing."""

storystring
```

Data types: strings

- Save to file:

  filename = “story.txt”

```
open(filename,'w').write(storystring)
```

- Read from file:

```
story = open(filename,'r').read(storystring)
story
```

- Assignment:
  - Repeat with a longer text, and calculate text statistics.

Data types: strings

- Define a reasonably long text string:

```
textstring = "This is a reasonably long text string."
```

- Measuring strings:

```
stringlength = len(textstring)
```

- Slicing strings (slicing lists works in the same way):

```
print textstring[0:5]
print textstring[10:15]
print textstring[:5]
print textstring[20:]
print textstring[-20:]
print textstring[::2]
print textstring[::-1]
```

- Discuss the properties of string slicing.

Data types: saving strings into a text file

- Define a string, a filename, and the end-of-line character:

```
textstring = "This is a reasonably long text string."
filename = "stringfile.txt"
eol = "\n"
```

- Open a file handle (file object) for writing, use it to write strings into the file, then close the file-handle.

```
fh = open("textfile.txt","w")
fh.write(textstring + eol)
fh.write(textstring[0:5] + eol)
fh.write(textstring[10:15] + eol)
fh.write(textstring[:5] + eol)
fh.write(textstring[20:] + eol)
fh.close()
```

Data types: reading strings into a text file

- Read a file into a single string:
  - define a file handle for reading, read, and close:

```
fh = open("textfile.txt","r")
text = fh.read()
fh.close()
```

- Read a file into a list of lines:
  - define a file handle for reading, read, and close:

```
fh = open("textfile.txt","r")
textlist = fh.readlines()
fh.close()

print text
print textlist
```

Data types: strings to lists and back again

```
textstring = "A long text string...\n...continued"
```

- Convenient abbreviations:
  ```
  eol = "\n" ; sp = " "
  ```

- Split a string into a list at linebreaks:
  ```
  linelist = textstring.split(eol)
  linelist
  ```

- Splitting lines into lists of words with no linefeeds, using a list comprehension (an operator for transforming lists):

  ```
  wordlinelist = [line.split(sp) for line in linelist]
  print wordlinelist
  ```

Data types: strings

- Normalising strings:

```
eol = "\n" ; sp = " " ; fs = "."

textstring = "This is a reasonably long text
string.\nAnd this is a continuation of it after a
linebreak."

textstring = textstring.lower()
textstring = textstring.replace(eol,sp)
textstring = textstring.replace(fs,sp + fs)

wordlist = textstring.split(sp)
wordset = set(wordlist)

ttr = float(len(wordset)) / len(wordlist)
print "TTR = ", ttr
```

Data types: strings

- Assignment:
  - import NLTK texts
  - use your knowledge so far
    - to normalise the texts
    - to create a corrected TTR table.

Data types: lists

- Creating and modifying lists (check each of these):

```
list01 = [1,2,3,4,3,2,1]
list01
list02 = ["a","b","c","d","e"]
list02

list03 = "Turn this string into a list.".split()
list03

list04 = "turn;this;csv;record;into;a;list".split(";")
list04

list04[1] = "that"
list04

list04[0] * 3
```

Data types: lists

- Creating and modifying lists (check eac

Official Python Tutorial:

https://docs.python.org/2/tutorial/

```
list01 = [1,2,3,4,3,2,1]

list02 = ["a","b","c","d","e"]

list03 = "Turn this string into a list.".split()

list04 = "turn;this;csv;record;into;a;list".split(";")

list04[1] = "that"

list04[0] * 3
```

Data types: lists

Official Python Tutorial:

https://docs.python.org/2/tutorial/

- Splicing lists:

```
list05=list04[1:4]
list06 = list04[-2:]
```

  - In principle this is the same syntax as for strings

- Give a list a new name (changes to one change the other):

```
list04
list07 = list04
list07[1] = "that"
list04
```

- Create a separate copy of a list:

```
list07 = list04[:]
list07[1] = "the"
list07
list04
```

List operations: iteration

- List comprehensions ('set-builder syntax'):

```
list02 = [ x for x in list01 ]
```

- This is equivalent to:

```
list02 = []
for x in list01:
    list02 += [ x ]
```

- A typical use:

```
floatlist = [float(i) for i in integerlist]
```

- Discuss these list comprehensions:

```
list03 = [ x for x in list01 if x != '' ]
list04 = [ 0 if x='' else 1 for x in list01 ]
```

List operations: iteration

- Test these range lists:

```
list01 = range(5)
list02 = range(1,5)
list03 = range(5,13,2)
```

- Enumerate objects (cardinal - list element pairs):

```
list04 = list(enumerate(list01))
list04
```

- Discuss **`range`** with **`zip`** and **`enumerate`** objects:

```
list05 = zip(range(len(list01)),list01)
list05
a, b = zip(*list02)
a
b
```

List operations: iteration

- Assignment:
  - discuss and experiment with
    - **range**
    - **enumerate**
    - **`for`**-loops
    - list comprehensions

Data types: tuples

- Tuples:

```
tuple01 = (10, 9, 8, 15)
type(tuple01)

for item in tuple01:
    print(item)
```

- Tuple packing and unpacking tuples:

```
tuple01 = 10, 9, 8, 15
tuple01
a, b, c, d = tuple01
(a, b, c, d) = tuple01

list01 = [a, b, c, d]
```

Data types: sets

- Sets differ from lists in being unordered and having no duplicate elements:

```
list01 = [ 'a', 'b', 'c' ] * 5
len(list01)
list01 = sorted(list01)

set01 = set(list01)
len(set01)
ordereduniquelist01 = sorted(list(set01))
ordereduniquelist01
```

- Empty set:

```
set01 = set()
```

  - and **NOT** `set01 = {}`, which defines an empty ‘dictionary’

Data types: dictionaries (hashes, association lists)

Dictionaries are special sets (unordered):

```
phone = {'Jane': 1066, 'Joan': 1989}

phone

phone['Jane']

phone['June'] = 1812

phone.keys()

phone = dict([('Mary', 1492), ('Mona', 1688)])
```

Data types: dictionaries Python style

- Dictionaries (aka associative lists, hash lists, ...) are
  - sets of pairs of items with properties
  - properties are accessed with names of items

- Toy examples:

```
lex = { 'dog' : 'tame', 'wolf' : 'wild', 'pony' : 'monochromezebra'}
lex
lex['cat'] = 'feline'
lex
lex['wolf']
lex.keys()
```

- Human readable output:

```
['a '+key+' is '+lex[key] for key in lex.keys()]
```

Data types: dictionaries Python style

- The properties can themselves be dictionaries (or any other object):

```
lexicon = {
    'dog'  : {
             'pos' : 'noun',
             'num' : 'sing',
             'sem' : 'canine'
             },
    'bark' : {'pos' : 'noun',
              'num' : 'sing',
              'sem' : 'treewrapping'
             }
            }
```

- For discussion:
  - How can homonyms (ambiguous items) be accommodated?

Summary: data types: dictionaries Python style

- Assignment:
  - Make a lexicon as a module which can be imported into any Python script and queried, extended, formatted, printed, etc.
  - Use this as a model, but replace with different information:

```
lexicon = {
    'dog'   : {
                'pos' : 'noun',
                'num' : 'sing',
                'sem' : 'canine'
                },
    'bark' : {'pos' : 'noun',
                'num' : 'sing',
                'sem' : 'treewrapping'
                }
            }
```

- Hint:
  - check Unit 1, about modules.

Summary: control structures: conditions and Loops

```
list01 = ['one', 'two', 'three', 'four', 'five']
```

- Conditions and two kinds of loop:

```
for item in list01:
    if len(item) < 4:
        print item, 'is short'
    else:
        print item, 'is long'
while list01:
    print list01
    list01 = list01[1:]
list01
```

- Assignment:

  Design tasks in which loops (and list comprehensions) must be used.

Summary: math and logic operators

- Arithmetic Operators
  + - * / % ** //
- Comparison (Relational) Operators
  == != <> < > <= >=
- Assignment Operators
  = += -= *= /= %= **= //=
- Logical Operators
  and or not
- Bitwise Operators
  & (AND) | (OR) ^ (XOR) ~ (bit flip) << l-shift >> r-shift
- Membership Operators
  in not in
- Identity Operators
  is is not

Summary: string operators

- String Operators (also lists)
  s1 + s2 (concatenate **s1** and **s2**) s * **n** (repetition **n** times)
- Slice, Range Slice (also lists)
  [ ] [:]
- Assignment Operators (also lists)
  = +=
- Membership Operators (also lists, tuples, sets)
  in not in
- String type Operators
  - **' ' " " """ """** (3 double quotes - multi-line)
  - **r" "** (raw, no escaping) **u" "** (unicode)
- Formatting Operator examples
  - Strings: **'%s %s' % ('one', 'two')**
  - Integers: **'%d %d' % (1, 2)**
  - Floats: **'(%f), (%10.3f)' % (2.34, 3.1459)**

Assignment: experiment with string methods

```
mystr = "Joe met Mr. Brown."
mystr.upper()
mystr.title()
```

capitalize()
center(width, fillchar)
count(str, beg=0,end=len(string))
decode(encoding='UTF-8',errors='strict')
encode(encoding='UTF-8',errors='strict')
endswith(suffix, beg=0, end=len(string))
expandtabs(tabsize=8)
find(str, beg=0 end=len(string))
index(str, beg=0, end=len(string))
isalnum()
isalpha()
isdigit()
islower()
isnumeric()
isspace()
istitle()
isupper()
en(string)
ljust(width[, fillchar])
lower()
lstrip()
maketrans()
max(str)
min(str)
replace(old, new [, max])
rfind(str, beg=0,end=len(string))
rjust(width,[, fillchar])
rstrip()
split(str="", num=string.count(str))
splitlines(num=string.count('\n'))
startswith(str, beg=0,end=len(string))
strip()
swapcase()
title()
translate(table, deletechars="")
upper()
zfill (width)
isdecimal()

Google the Python documentation wiki for details.

End of Unit 2