Lesson 6: Dictionary, User-Defined Functions

Fundamentals of Text Processing for Linguists

Na-Rae Han

Objectives

- Introducing `dict` (dictionary)
- User-defined functions

The 'tiger' script

- http://www.pitt.edu/~naraehan/ling1991/img/ex4.sC.png
- Output:

```
t
ti
tig
tige
tiger
i
ig
ige
iger
g
ge
ger
e
er
r
That's everything!
```

Non-empty **substrings** of 'tiger'

dict: a dictionary data type

```
{'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8, 'Maggie':1}
```

← A dictionary of the Simpson family members' age

```
{'go':'went', 'eat':'ate', 'see':'saw', 'say':'said'}
```

← A dictionary of verb past tense

- Dictionaries store a **mapping** between a set of **keys** and a set of **values**.
  - Keys can be any immutable type: string, integer, tuple
  - Values can be any type (can also be mixed types)
  - There is no inherent order (unlike lists and tuples)
  - You can define, modify, view, lookup, and delete the key-value pairs in the dictionary.

Looking up a dictionary

```
>>> en2es = {'cat':'gato', 'dog':'perro', 'tiger':'tigre'}
>>> en2es['cat']
'gato'
>>> en2es['dog']
'perro'

>>> en2es['gato']
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    en2es['gato']
KeyError: 'gato'
```

Dictionary is ***one way***.
Cannot look up based on the value.
Mapping can be many-to-one.

Checking if something's in a dictionary

```
>>> en2es
{'tiger': 'tigre', 'dog': 'perro', 'cat': 'gato'}

>>> 'fox' in en2es
False

>>> 'cat' in en2es
True

>>> 'gato' in en2es
False
```

"`in`" tests if a **key** is in a dictionary

"`in`" does *not* work with **value**

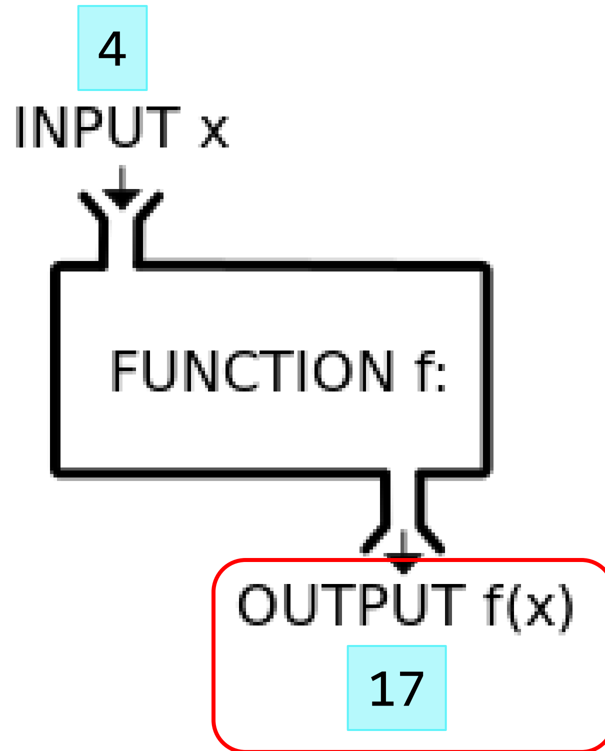
Practice

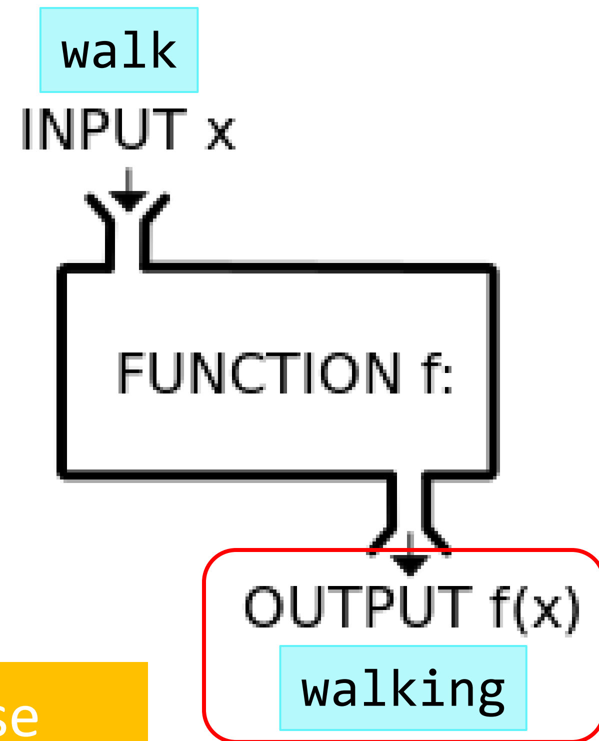
- A dictionary of irregular plural nouns:

```
>>> irrpl = {'man':'men', 'goose':'geese', 'mouse':'mice',
'child':'children'}
>>> 'cat' in irrpl
False
>>> 'mouse' in irrpl
True
>>> irrpl['mouse']
'mice'
>>> mouse_pl = irrpl['mouse']
>>> mouse_pl
'mice'
```

Functions

- f(x): $x^2 + 1$
- f(x): x + "ing"





These values are RETURNED

Functions in python

- f(x): $x^2 + 1$

```
def f(x) :
    return x**2 + 1
```

```
>>> f(4)
17
>>> f(12)
145
```

- f(x): x + "ing"

```
def f(x) :
    return x + "ing"
```

```
>>> f('walk')
'walking'
>>> f('bake')
'bakeing'
```

Functions in python

- Python functions we have seen so far:

```
raw_input()   int()
len()         float()
range()       list()
type()        str()
```

- `print` is not a function (WHY?); called a *statement*.
- `s.upper()`, `s.split()`, `l.reverse()`, etc. are technically *methods*, defined on a specific object type only.
- For common routines, one can:
  - Find ready-made functions (packaged in 'modules'), or
  - Write his/her own.

Use def to make your own function

- A function for counting "vowels" in a string:

```
>>> def vowelCount(wd):
        wd = wd.lower()
        cnt = wd.count('a') + wd.count('e') + wd.count('i') \
              + wd.count('o') + wd.count('u')
        return cnt
```

```
>>> vowelCount('queue')
4
>>> vowelCount('sly')
0
>>> vowelCount('Hello, world!')
3
>>> vowelCount('Animal')
3
```

Why write our own functions?

- Reduce redundancy
  - Routine procedures may be used repeatedly: implement as a single function
- Functions make your programs easier to:
  - read, write, test, fix, improve, add to, develop
- Structured programming
  - Your Python programs can become a library of utilities, more than a bunch of single-use individual scripts (more on this later)

- … and because we're tired of `raw_input()`. ☺

Defining your own function

- Palindrome test as a function:

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

```
>>> isPalindrome('level')
True
>>> isPalindrome('elephant')
False
```

Declaring a function

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

define a function called ...

name of function

input (function argument)

colon

Body of function

- Content of a function is marked by indentation.

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

"Body" of function

indentation

Defining your own function

- A "docstring" (documentation string) at the top of function def

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

"**docstring**" (optional) provides a short description of the function. It prints out when `help(functionname)` is called.

Function variables

- Variables in a function are *local*.

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

These variables exist only within the function's body.

Function returns a value

- Function output is *returned*

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

Value of *verdict* is **returned**.
Output type of this function is Boolean.

Practice

- Write a function that:
  - takes two strings as input: a noun and a verb
  - returns a single string as a sentence, with number agreement

```
def makeSent(noun, verb) :
    ??
```

```
>>> makeSent('Mary', 'walk')
'Mary walks'
>>> makeSent('Cats', 'sleep')
'Cats sleep'
```

Returning vs. printing

- This function **returns** a string:

```
def makeSent(noun, verb) :
    "Takes a noun and a verb, and returns a sentence"
    if noun.endswith('s') :
        return noun+' '+verb          # returns a single str
    else :
        return noun+' '+verb+'s'
```

```
>>> makeSent('Mary', 'walk')
'Mary walks'
>>> foo = makeSent('Cats', 'sleep')
>>> foo
'Cats sleep'
```

A string is RETURNED

Returned string value can then be assigned to the variable foo

Returning vs. printing

- This function **prints** strings:

```
def printSent(noun, verb) :
    "Takes a noun and a verb, and prints out a sentence"
    if noun.endswith('s') :
        print noun, verb
    else :
        print noun, verb+'s'
```

No return statement is given.
A special value **'None'**
is returned implicitly.

```
>>> printSent('Mary', 'walk')
Mary walks
>>> foo = printSent('Cats', 'sleep')
Cats sleep
>>> foo
>>>
```

Printing happens independently from the variable assignment

foo's value is 'None'

Calling a function

- Within the same script where the function is:

```
def isPalindrome(wd) :
    # ... (function body clipped)
    if wd == rwd : verdict = True
    else : verdict = False
    return verdict


if isPalindrome(x) :
    print 'YES', x, 'is a palindrome.'
else :
    print 'NO', x, 'is not a palindrome.'
```

Function is called by its name.

Functions can call another function

- Palindrome test as a single function:

```
def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."

    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)

    if wd == rwd : verdict = True
    else : verdict = False

    return verdict
```

Functions can call another function

- One function for reversing the word, another for palindrome test:

```
def getRev(wd) :
    "Takes a string, returns reverse"
    wdl = list(wd)
    wdl.reverse()
    rwd = ''.join(wdl)
    return rwd

def isPalindrome(wd) :
    "Palindrome test. Takes a string, returns True/False."
    rwd = getRev(wd)
    if wd == rwd : verdict = True
    else : verdict = False
    return verdict
```

Practice

- Write a function that:
  - outputs whether or not a given string contains a vowel.

```
>>> def vowelCount(wd):
        wd = wd.lower()
        cnt = wd.count('a') + wd.count('e') + wd.count('i') \
              + wd.count('o') + wd.count('u')
        return cnt
```

```
def hasVowel(wd) :
```

Use the ready-made vowelCount function!

```
>>> hasVowel('colorless')
True
>>> hasVowel('sly')
False
```

Practice

- Write a function that:
  - outputs whether or not a given string contains a vowel.

```
>>> def vowelCount(wd):
        wd = wd.lower()
        cnt = wd.count('a') + wd.count('e') + wd.count('i') \
              + wd.count('o') + wd.count('u')
        return cnt
```

```
def hasVowel(wd) :
    return vowelCount(wd) == 0
```

```
>>> hasVowel('colorless')
True
>>> hasVowel('sly')
False
```

Practice

- Write a function that:
  - tests whether or not a word starts with a consonant cluster.

```
def startsWithCC(wd) :
    ??
```

```
>>> startsWithCC('star')
True
>>> startsWithCC('pay')
False
>>> startsWithCC('b')
False
```

Practice

- Write a function that:
  - tests whether or not a word starts with a consonant cluster.

```
def startsWithCC(wd) :
    if len(wd) < 2 :
        return False
    elif wd[0] not in 'aeiou' and wd[1] not in 'aeiou':
        return True
    else:
        return False
```

Without this length check, short strings like 'b' will generate error

```
>>> startsWithCC('star')
True
>>> startsWithCC('pay')
False
>>> startsWithCC('b')
False
```

Practice

- Write a function that:
  - outputs the plural form of a noun, including irregular nouns:

```
>>> irrpl = {'man':'men', 'goose':'geese', 'mouse':'mice',
'child':'children'}
```

```
def getPlural(wd) :
        ??
```

```
>>> getPlural('cat')
'cats'
>>> getPlural('child')
'children'
```

Practice

- Write a function that:
  - outputs the plural form of a noun, including irregular nouns:

```
>>> irrpl = {'man':'men', 'goose':'geese', 'mouse':'mice',
'child':'children'}
```

```
def getPlural(wd) :
    if wd in irrpl :
        return irrpl[wd]
    else :
        return wd+'s'
```

If wd is in the irregular pl noun dictionary, return the dict value

```
>>> getPlural('cat')
'cats'
>>> getPlural('child')
'children'
```

Wrap-up

- **Next class**
  - list methods, dictionary methods, sorting

- **Homework #2**
  - http://www.pitt.edu/~naraehan/ling1901/HW2.pdf
  - Past-tense generator script
  - Due Tuesday midnight
  - This one is fairly complex. START EARLY and GET HELP.