

# **Functions and File Input/Output**

Linguistics 445/515

Autumn 2008

# Functions

Let's say we have input text line-by-line and for each line, we do the same operation. It's extremely useful to use a function in this case.

In one part of code (traditionally, near the beginning):

```
def get_word(x):  
    # x is in the form 'word tag'  
  
    # split takes a string and turns it into a list  
    text = x.split()  
    return text[0]
```

In another part of code, where needed, we put:

```
word = get_word(line)
```

Even though it takes two steps within the function, it's one conceptual idea, and now it takes up less space in the main part of the code, where we need it.

## Functions, piece by piece

Let's re-look at that function we just saw:

```
def get_word(x):
```

Two things are happening here:

- `def` is being used to indicate that a function is being defined. You give the name to the function, in this case, `get_word`
- `x` is an *argument*, which means that it's a variable and can take on any value
  - When we call this function with `get_word(line)`, the value of the variable `line` in the main part of the program gets assigned to `x`
  - In other words, `x` is *local* only to the function, but gets passed values from other parts of the program
- `return` is used to send a value (or values) back to the part that called the function.

```
    return text[0]
```

## Default arguments

Here's a program that calculates  $x^n$ , but by default it calculates  $x^2$

```
def power(x,n=2):  
    result = 1  
    for i in range(n):  
        result *= x  
    return result  
  
x = power(3,4)    # x = 81  
y = power(3,2)    # y = 9  
z = power(3)      # z = 9
```

## Function exercise

In pairs, write a program where at the top you have a function, and the rest of the program consists only of a call to that function.

- The function takes a string and reverses it.

```
def my_reverse(...):  
    ...
```

```
print my_reverse("manitoba")    # should print "abotinam"
```

If you finish before everyone else, start writing a function that returns 1 if the string is a palindrome and 0 otherwise.

# Input

Most likely, when inputting a file, you'll want to use open

```
f = open('text.txt', 'r')    # 'text.txt' is file name
                             # 'r' means for reading
```

And then we have to make the file contents readable, i.e., convert them into types of things python understands

```
# Different ways to read in a file
text = f.read()              # reads file as a string
textlist = f.readlines()    # reads file as a list of lines
line = f.readline()         # reads one line at a time
```

## Inputting line-by-line

So one way to input every line is the following:

```
f = open('text.txt', 'r')
line = f.readline()
while line:
    line = line.rstrip() # because every line ends in a newline
    ...
    # process line
    ...
    line = f.readline()
```

This is another example of an iteration procedure, and here we perform actions for each line

# Output

We already saw how to output single lines of text using `print`, but we can also write to files directly

```
f2 = open('output.txt','w')    # 'w' option means for writing
f2.write('add some text')
f2.write(' and then\nsome more text\n')
num = 2
f2.write(str(num))
```

The file `output.txt` then looks like this:

```
add some text and then
some more text
2
```

## In-class exercise (if time)

In groups, grab some text off the internet (e.g., from Project Gutenberg) and store it into a file

1. Read in this file in your python program
2. Write a function for each input line which does the following:
  - (a) Break down each line, using `split`, into a list of its component words.
  - (b) Alphabetize (sort) the list
  - (c) Outputs (into a file) the word from each line which is the first alphabetically