RICE SOCIAL SCIENCES
Department of Linguistics

functions & files

LING 430 Computational Linguistics Fall 2026
lecture 3b
Sep 10 2026

Siyu Liang
siyu.liang@rice.edu

last time

- python on language data
- Zipf’s law
- nltk

agenda

- function
- filestream

today's notebook

class03b.ipynb

- course website → week 3 → activity
- **File → Save a copy in Drive**

functions

function

why use functions

- imagine you write something useful
  - you want to use it again on similar data
  - you end up copying and pasting into cell after cell
  - there's a mistake!
  - you have to fix it for every version
- a *function* wraps a code and lets you reuse it on similar data

function

def

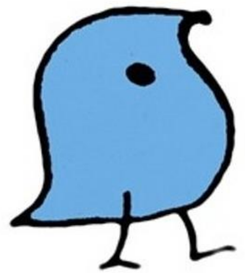
- key components: `def` and `return`

```
def plural(noun):
    return noun + "s"
plural("cat")
```

→ 'cats'

```
plural("wug")
```

→ 'wugs'



function

parameter vs argument

- `def plural(noun)`
  - has a **parameter** which is the `noun` variable
  - anything you pass onto the function will be `noun` inside the function
- `plural("cat")`
  - passes an **argument** which is the value that goes into the slot
  - inside the function, `noun` is `"cat"` for as long as that call runs

function

- the regular past tense

```
def past(verb):
    return verb + "ed"
past("rick")
```

→ 'ricked'

Jean Berko

- *rick* is one of Jean Berko's nonce verbs
  - "this man knows how to rick; yesterday he ___"
  - also the person who invented 

function

parameters

- you can have as many parameters as the function, **in order**

```
def affix(stem, suffix):
    return stem + suffix
affix("duck", "ling")
```

→ 'duckling'

```
affix("dog", "s")
```

→ 'dogs'

function

default arguments

- a parameter can carry a default value

```
def shout(word, mark="!"):
    return word.upper() + mark
```

- you can omit it or override it, by **position** or by **name**
- remember nltk's `concordance("monstrous", lines=3)`

```
shout("rice")
```

→ 'RICE!'

```
shout("rice", mark="?!")
```

→ 'RICE?!'

function

nested return

- any `return` will exit the function immediately

```
def article(word):
    if word[0] in "aeiou":
        return "an"
    return "a"
```

```
article("owl")
```

→ 'an'

```
article("hour")
```

→ 'a'

return vs print

- `print` displays a value, then throws it away
- `return` passes it on to the rest of the program

```
def plural_p(noun):
    print(noun + "s")
```

```
form = plural_p("cat")
```

→ cats

```
print(form)
```

→ None

scope

- names created inside a function live only **inside** it
- unless you `return` it to the rest of the program

```
def plural(noun):
    form = noun + "s"
    return form
```

```
plural("cat")
```

→ 'cats'

```
print(form)
```

→ NameError: name 'form' is not defined

docstring

- a **docstring** says what the function does
  - `help()` shows it
  - one sentence, in triple quotes, first thing in the function

```
def plural(noun):
    """Return the English plural of noun."""
    return noun + "s"
help(plural)
```

→ Help on function plural in module __main__:
→ plural(noun)
→ Return the English plural of noun.

Turkish vowel harmony

- Turkish has eight vowels in two groups, governed by vowel harmony
  - back: **a ı o u**
  - front: **e i ö ü**
- a suffix vowel agrees with the backness of the word's last vowel
- plural marking, *-lAr*, is either *-lar* or *-ler*
  - *kebap* → *kebaplar*, *ev* → *evler*, *okul* → *okullar*, *otobüs* → *otobüsler*

the locative

- the locative is a case that indicates location, meaning “in, on, at, …”
- in Turkish it is the suffix *-DA*
  - *ev* → *evde*, *okul* → *okulda*
- its consonant agrees with the voicing of the word’s last consonant
  - after a voiceless consonant (ç f h k p s ş t), *d* becomes *t*
  - *kebap* → *kebapta*, *yoğurt* → *yoğurtta*, *otobüs* → *otobüste*
- in summary, four forms: *-da -de -ta -te*

activity

- write a function for Turkish locative derivation
  - see the notebook for the starter cell
  - `locative(word)` should return suffixed forms with one of *–da, -de, -ta, -te*
  - then print `locative(w)` for every word in the list
- tips
  - last vowel back → *a*, front → *e*
  - last letter voiceless → *t*, otherwise *d*
  - one decision inside the other: nested `if`s, each ending in a `return`
- extra challenge: `plural(word)` with *-lar / -ler*, then `locative(plural(word))`: *ev* → *evlerde*, *otobüs* → *otobüslerde*

files

discussion

- how do you know what sounds there are in a language you don’t know?
  - Wikipedia?
  - reference grammars?
  - textbooks?

the North Wind and the Sun

- Journal of the IPA (JIPA) publishes illustrations, the phonetic description for a language
- every illlustration uses the same Aesop fable, “the North Wind and the Sun”, in the target language with a phonetic transcription and a recording

file

getting a file into Colab

- download northwind.txt from Canvas
- click the folder icon on the left, then drag `northwind.txt` into the file list
- the file is in `/content`, which belongs to this runtime only
  - it disappears when a fresh runtime starts empty
- for files you want to keep, mount your Google Drive instead
  - you can click on Files → the Google Drive icon
  - or run the following

```
from google.colab import drive
drive.mount("/content/drive")
```

file

with keyword

- `with` runs the indented block with the file open, then closes the file
- `open("northwind.txt")` opens the file and hands back a file object
- `as f` names that object so the block can use it

```
with open("northwind.txt") as f:
    text = f.read()
```

file

open & read

- `"r"` is read mode, and the default
- `.read()` gives the whole file as one string

```
with open("northwind.txt", "r") as f:
    text = f.read()
len(text)
```

→ 594

file

- we could look inside the file by calling on the variable or using print
- although we see slightly different things

```
text[110:145]
```

→ ' cloak.\nThey agreed that the one wh'

```
print(text[110:145])
```

→ cloak.

→ They agreed that the one wh

file

escape characters

- a reminder

written as	stands for
`\n`	a line break
`\t`	a tab
`\'`	a single quote
`\"`	a double quote
`\\`	a backslash itself

file

reading line by line

- `.readlines()` reads the text into a **list** with one string for **each line** ending in `\n`

```
with open("northwind.txt") as f:
    lines = f.readlines()
```

```
len(lines)
```

→ 5

```
lines[3][-29:]
```

→ 'traveler took off his cloak.\n'

file

f-string

- f-string is a great way to format output
  - an `f` before the quote, and `{…}` slots filled with values
  - anything can go in a slot, including a whole expression
  - cleaner than gluing with `+` and `str()`

```
print(f"{len(lines)} lines, {len(text)} characters")
```

→ 5 lines, 594 characters

file

open & write

- `"w"` is write mode: it **creates** the file, or **empties** it if it already exists
  - you can lose the original file!

```
with open("new_file.txt", "w") as out:
    out.write("new file")
```

file

open & append

- `"a"` is append mode
  - it adds to the end and keeps what is already there
  - you won’t lose the original file this way

```
with open("new_file.txt", "a") as out:
    out.write(f"northwind.txt has {len(lines)} lines\n")
```

activity

- write to a text file
- read the text file and append additional text to it
- print a line that contains things you'd like to report using f-string
  - e.g. length of the entire text, the average length of a word, number of lines, etc.
- write over it

HW2

- released today
- due Thursday, Sep 17, 11:59 pm
- submit on Canvas as an .ipynb file

next time

- **Tuesday**: how do computers break sentences into words and how to look for the words you want
  - tokenization & regular expressions
- **Thursday**: how do LLMs process words from different languages?
  - Unicode & subword tokenization
- **HW 1 due tonight, 11:59 pm**