

自然言語処理プログラミング勉強会 0 - プログラミング入門

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

本チュートリアルについて

- 7 部構成、比較的簡単なトピックから
- 各回：
 - チュートリアルで：新しい内容
 - 宿題：プログラミング演習
 - 次の週：結果について発表、もしくは話し合いをする
- プログラミング言語：任意
 - スライドは Python で
 - Python, C++, Java, Perl についての質問い答えられる
- 2 人で組んで作業をするのもおすすめ

環境設定

端末を開く

- Linux, Mac
 - プログラムメニューから「端末」を選択
- Windows
 - cygwin を利用
 - もしくは Linux マシンに ssh で接続

ソフトのインストール

- 3 種類のソフト :
 - `python`: プログラミング言語のインタープリター
 - `gvim`: テキスト編集ソフト
 - `git`: バージョン管理ソフト
- **Linux**:
 - `sudo apt-get install git vim-gnome python`
- **Windows**:
 - cygwin の `setup.exe` を実行
 - 「プログラム」で「`git`」「`gvim`」「`python`」を選択

チュートリアルファイルを github からダウンロード

- 「git clone」を使ってチュートリアルファイルをダウンロード

```
$ git clone https://github.com/neubig/nlptutorial.git
```

- このファイルは nlptutorial ディレクトリにあるはず

```
$ cd nlptutorial
```

```
$ ls download/00-intro/nlp-programming-en-00-intro.pdf
```

gvim の使い方

- どのテキストエディタでも良いが、vim を使う場合：
- 初めてなら、vim の設定を記述する vimrc をコピーすると使いやすくなるかも：

```
$ cp misc/vimrc ~/.vimrc
```

- vim で「test.txt」というファイルを作る：

```
$ gvim test.txt
```

- 「i」を押すと入力開始、「test」を書く
- エスケープを押して、「:wq」でファイルを保存して終了（:w は保存、:q は終了）

git の使い方

- git を使って書いたコードの履歴管理することが可能
- まず、追加したファイルを add

```
$ git add test.txt
```

- 「commit」で変更を保存

```
$ git commit
```

(「テストファイルを追加」などのメッセージを入力)

- 他の昨日は最後の commit への巻き戻し (git reset)、
サーバーに置いてあるコードの変更の反映 (git pull)、
サーバーへのコードのアップロード (git push)

プログラミングの基礎

Hello World!

1) my-program.py をエディタで開く (gvim, emacs, gedit)

```
$ gvim my-program.py
```

2) 下記のプログラムを入力

```
#!/usr/bin/python  
print "Hello World!"
```

3) プログラムを実行可能に

```
$ chmod 755 my-program.py
```

4) プログラムを実行

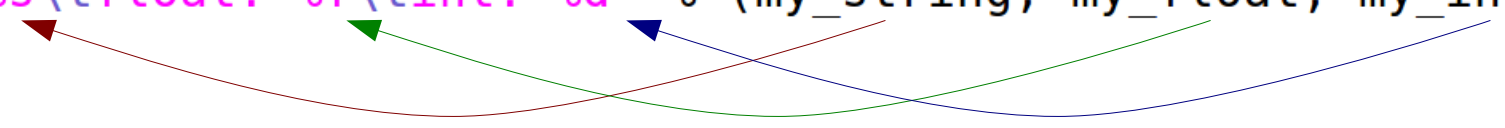
```
$ ./my-program.py  
Hello World!
```

データタイプ

- 文字列: “hello”, “goodbye”
- 整数: -1, 0, 1, 3
- 浮動小数点: -4.2, 0.0, 3.14

```
my_int = 4
my_float = 2.5
my_string = "hello"

print "string: %s\tfloat: %f\tint: %d" % (my_string, my_float, my_int)
```



```
$ ./my-program.py
string: hello    float: 2.500000 int: 4
```

if/else, for

```
my_variable = 5
```

```
if my_variable == 4:
    print "my_variable is 4"
else:
    print "my_variable is not 4"
```

← 条件が満たされれば
← これをする
← そうでなければ
← これをする

```
for i in range(1, my_variable):
    print "i == %d" % (i)
```

← 各要素に対して
← これをする

```
$ ./my-program.py
my_variable is not 4
i == 1
i == 2
i == 3
i == 4
```

← 注意！
range(1, 5) == (1, 2, 3, 4)¹²

複数のデータ点の格納

密行列

キー	値
0	20
1	94
2	10
3	2
4	0
5	19
6	3

疎行列

キー	値
49	20
81	94
96	10
104	2

or

キー	値
apple	20
banana	94
cherry	10
date	2

配列 (Python で「リスト」)

- 密なデータの格納に適している
- キーは整数で、0 から始まる

```
my_list = [1, 2, 4, 8, 16]
```

5 要素のリストを作成

```
my_list.append(32)
```

リストの最後尾に要素を追加

```
print len(my_list)
```

リストの長さを表示

```
print my_list[3]
```

4 番目の要素を表示

```
print ""
```

```
for value in my_list:
    print value
```

リストの各要素を表示

マップ (Python で「辞書」)

- 疎行列に適している。引数は何でも OK。

キー (「alan」) と値 (「22」) からなる辞書を作成

```
my_dict = {"alan": 22, "bill": 45, "chris": 17, "dan": 27}
```

```
my_dict["eric"] = 33
```

新しい要素を追加

```
print len(my_dict)
print my_dict["chris"]
```

サイズを表示
1つの要素を表示

```
if "dan" in my_dict:
    print "dan exists in my_dict"
```

キーが辞書内に
存在するかどうか

```
for foo, bar in sorted(my_dict.items()):
    print "%s --> %r" % (foo, bar)
```

キー・値の各組を
表示 (キー順で)

defaultdict

- デフォルトの値を定義する辞書の拡張

```
from collections import defaultdict
```

ライブラリ読み込み

```
my_dict = defaultdict(lambda: 0)
```

```
my_dict["eric"] = 33
```

デフォルトを 0 に設定

```
print my_dict["eric"]
```

```
print my_dict["fred"]
```

存在するキーをプリント
存在しないキーをプリン
(dict ならエラー終了)

文字列の分割、連結

- NLP で文を単語に分割することはしばしばある

```
import string
```

```
sentence = "this is a pen"
words = sentence.split(" ")
```

文を空白区切りで単語の
配列に分割

```
for word in words:
    print word
```

```
print string.join(words, " ||| ")
```

配列を“ ||| ”を区切りと
して文字列に連結

```
$ ./my-program.py
```

```
...
```

```
this ||| is ||| a ||| pen
```

文字列の分割、連結

- NLP で文を単語に分割することはしばしばある

```
import string
```

```
sentence = "this is a pen"
words = sentence.split(" ")
```

文を空白区切りで単語の
配列に分割

```
for word in words:
    print word
```

```
print string.join(words, " ||| ")
```

配列を“ ||| ”を区切りと
して文字列に連結

```
$ ./my-program.py
```

```
...
```

```
this ||| is ||| a ||| pen
```

関数

- 関数は入力を受け取り、入力を変換し、戻り値を返す

```
def add_and_abs(x, y):
    z = x + y
    if z >= 0:
        return z
    else:
        return z * -1

print add_and_abs(-4, 1)
```

← add_and_abs の入力は「x」と「y」

x と y を足し、絶対値を返す

← add_and_abs を x=-4 と y=1 として呼ぶ

コマンドライン引数

```
import sys
my_file = open(sys.argv[1], "r")
for line in my_file:
    line = line.strip()
    if len(line) != 0:
        print line
```

最初の引数

ファイルを読み込み「r」で開く
1行ずつファイルを読み込む

行終端記号「\n」を削除

行が空でなければ表示

```
$ ./my-program.py test.txt
```

コードのテスト

入力・出力の簡単なテスト

例：

プログラム word-count.py はファイルの中の単語を数える

1) 小さな入力ファイルを作成

2) 人手で単語を数え、出力の正解ファイルを作成

test-word-count-in.txt

a	b	c
b	c	d

test-word-count-out.txt

a	1
b	2
c	2
d	1

3) プログラムを実行

```
$ ./word-count.py test-word-count-in.txt > word-count-out.txt
```

4) 結果を比較

```
$ diff test-word-count-out.txt word-count-out.txt
```

単体テスト

- 各関数をテストするコードを書く
- 様々なテストを行い、不正解の場合はエラーを表示
- 全てのテストが通った場合のみ 1 を返す

```
def test_add_and_abs():
    if add_and_abs(3, 1) != 4:
        print "add_and_abs(3, 1) != 4 (== %d)" % add_and_abs(3, 1)
        return 0
    if add_and_abs(-4, 1) != 3:
        print "add_and_abs(-4, 1) != 3 (== %d)" % add_and_abs(-4, 1)
        return 0
    return 1
```

コードのテストは必要不可欠！

- テストを作ること：
 - コードを書く前に解きたい問題の意識をはっきり
 - デバッグに使う時間が激減
 - 時間を置いてコードを読み返す時に分かりやすい

演習問題

演習問題

- ファイルの中の単語の頻度を数えるプログラムを作成

this is a pen
this pen is my pen

a 1
is 2
my 1
pen 3
this 2

- テスト入力 = test/00-input.txt, 正解 = test/00-answer.txt
- 実行 : data/wiki-en-train.word に対して
- 報告 :
 - 単語の異なり数
 - 「the」「a」「in」などの単語の頻度

擬似コード

create a map *counts*

単語と頻度を格納するために

open a file

for each *line* **in** the file
 split *line* **into** *words*

for *w* **in** *words*
 if *w* exists in *counts*, **add** 1 to *counts*[*w*]
 else set *counts*[*w*] = 1

print *counts*["in"], *counts*["the"] ... etc