Lecture 2: Encoding Language, Palindromes

Ling 1330/2330 Intro to Computational Linguistics
Na-Rae Han, 8/31/2023

Objectives

- Course organization
  - "Starting Each Class" checklist https://sites.pitt.edu/~naraehan/ling1330/checklists.html
  - Your Python setup
  - MS Teams Forum, office hours
  - Policies: https://sites.pitt.edu/~naraehan/ling1330/policies.html

- Structured programming
  - Palindrome: four scripts

- L&C ch.1: Understand the fundamentals of how language is encoded on a computer
  - Text encoding systems

for loop to build a new string

2 minutes

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...       new = new + x + x
...
>>> print(new)
    ❶
```

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...       new = ❷
...
>>> print(new)
    niugneppenguin
```

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...       new = ❸
...
>>> print(new)
    niugnep
```

for loop to build a new string

2 minutes

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...     new = new + x + x
...
>>> print(new)
    ppeenngguuiinn
```

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...     new = x + new + x
...
>>> print(new)
    niugneppenguin
```

```
>>> wd = 'penguin'
>>> new = ''
>>> for x in wd :
...     new = x + new
...
>>> print(new)
    niugnep
```

Works as a **string-reversing** routine!

Practice: palindrome

45 minutes

Let's practice writing Python scripts.

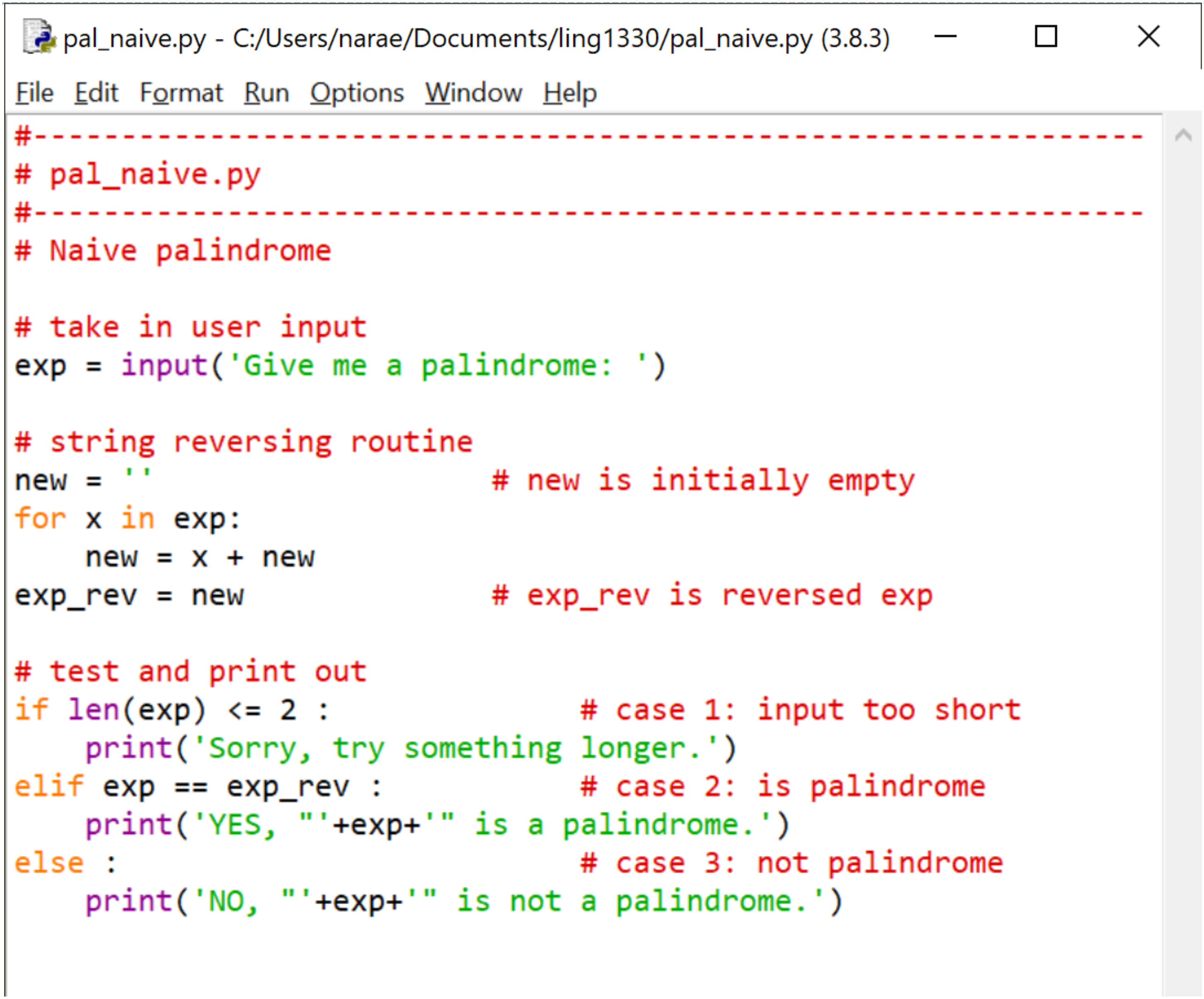
You will learn:

- How to develop a structured program, from simple to complex
- How to clean and manipulate strings
- How to modularize your program through the use of custom functions

- Head to:
  - https://sites.pitt.edu/~naraehan/ling1330/palindrome.html

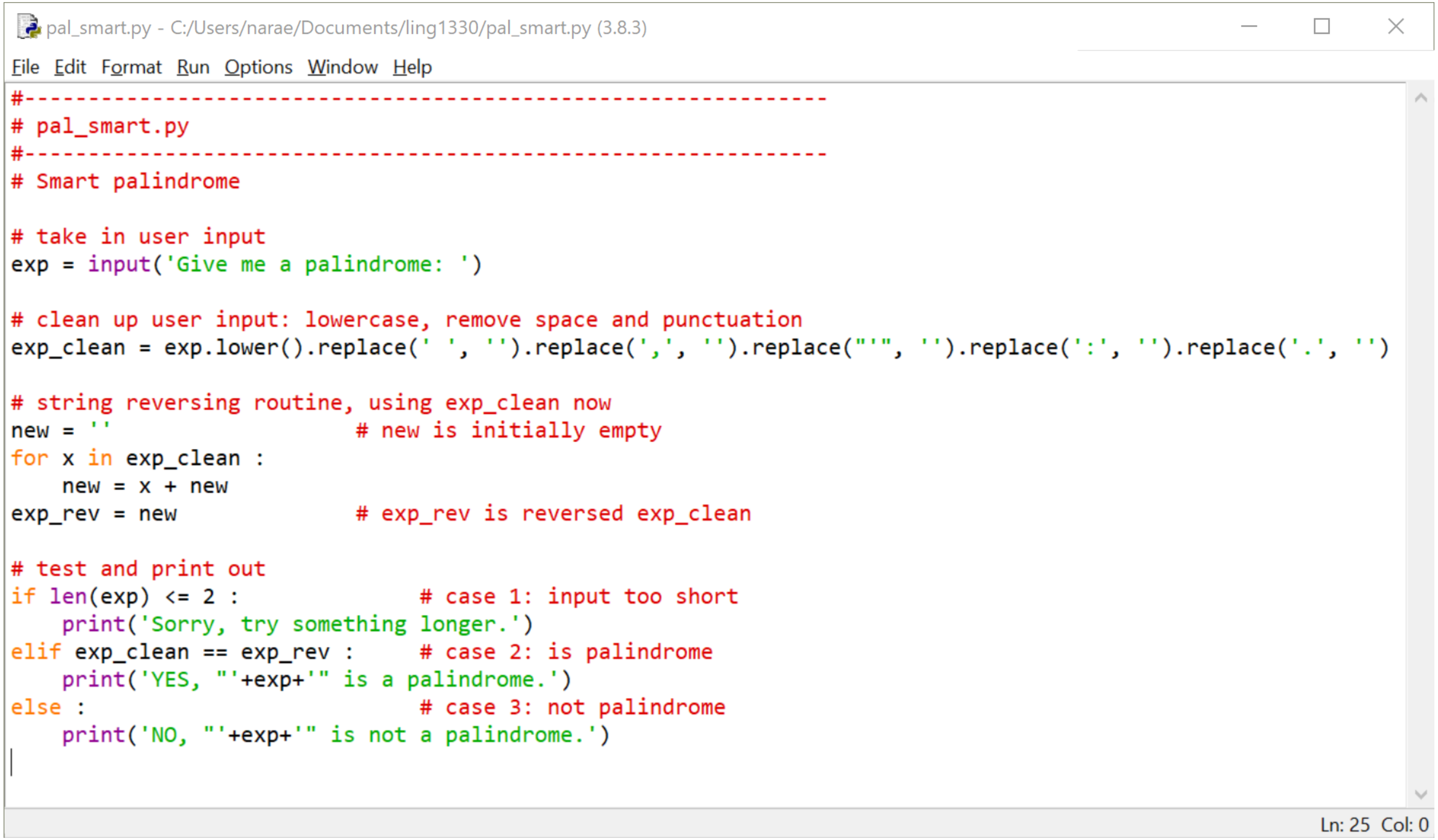
  (link on Schedule page)

#1: Naïve palindrome


pal_naive.py - C:/Users/narae/Documents/ling1330/pal_naive.py (3.8.3)
File
Edit
Format
Run
Options
Window
Help

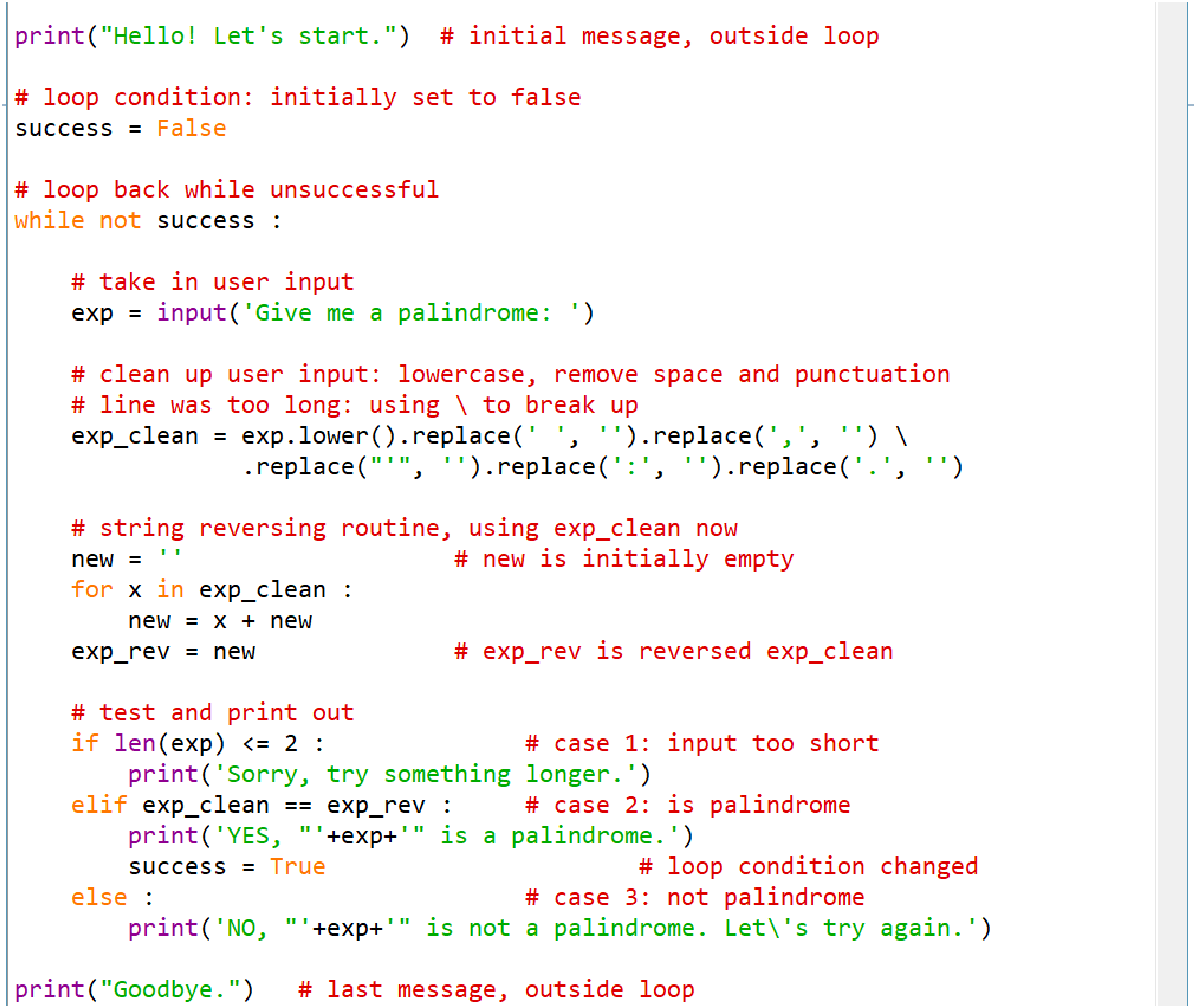
```python
#---------------------------------------------------------------
# pal_naive.py
#---------------------------------------------------------------
# Naive palindrome

# take in user input
exp = input('Give me a palindrome: ')

# string reversing routine
new = ''                           # new is initially empty
for x in exp:
    new = x + new
exp_rev = new                      # exp_rev is reversed exp

# test and print out
if len(exp) <= 2 :                      # case 1: input too short
    print('Sorry, try something longer.')
elif exp == exp_rev :                   # case 2: is palindrome
    print('YES, "'+exp+'" is a palindrome.')
else :                                  # case 3: not palindrome
    print('NO, "'+exp+'" is not a palindrome.')
```

#2: Smart palindrome


pal_smart.py - C:/Users/narae/Documents/ling1330/pal_smart.py (3.8.3)
File Edit Format Run Options Window Help
Ln: 25 Col: 0

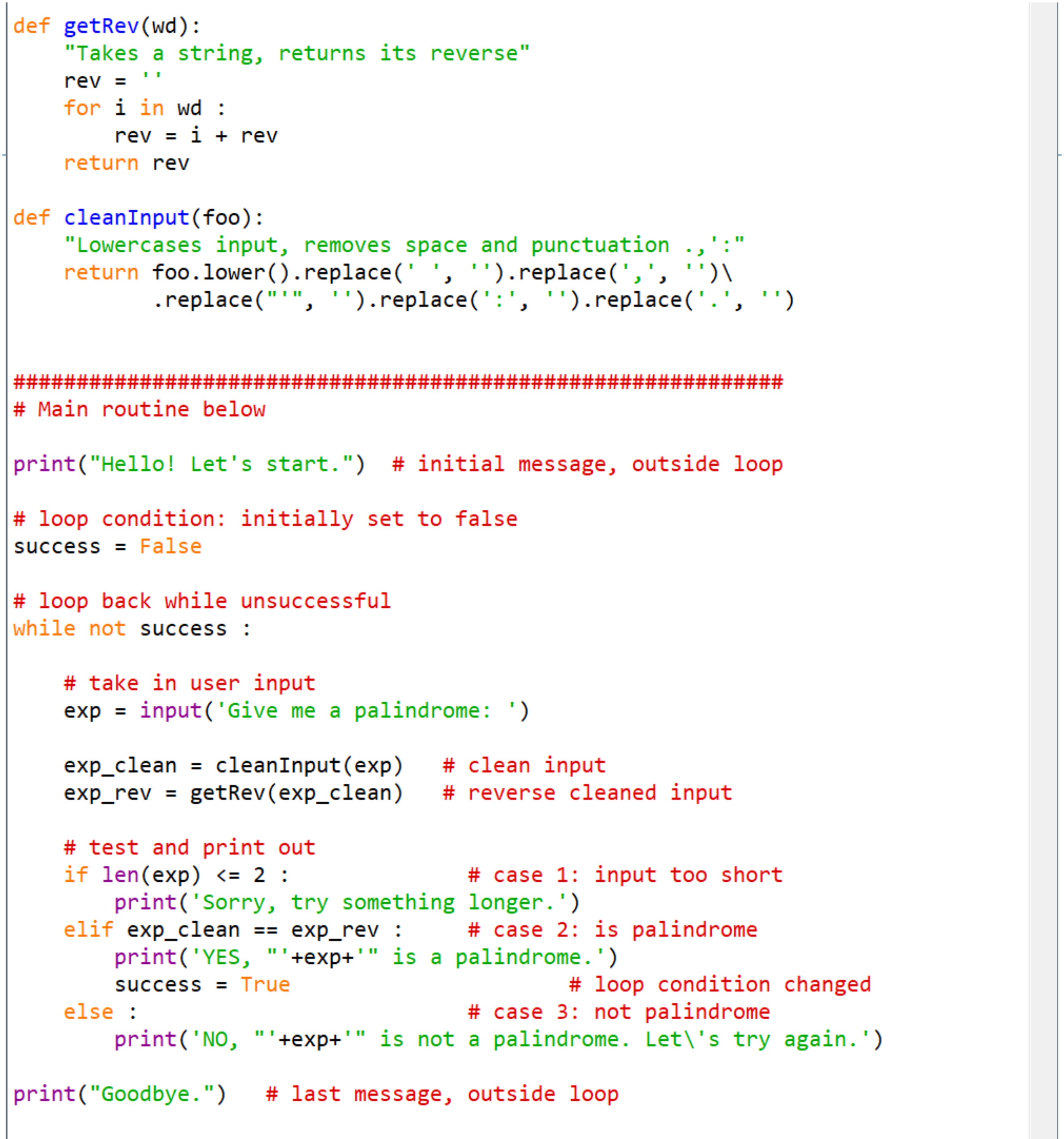
```python
#---------------------------------------------------------------
# pal_smart.py
#---------------------------------------------------------------
# Smart palindrome

# take in user input
exp = input('Give me a palindrome: ')

# clean up user input: lowercase, remove space and punctuation
exp_clean = exp.lower().replace(' ', '').replace(',', '').replace("'", '').replace(':', '').replace('.', '')

# string reversing routine, using exp_clean now
new = ''                   # new is initially empty
for x in exp_clean :
    new = x + new
exp_rev = new              # exp_rev is reversed exp_clean

# test and print out
if len(exp) <= 2 :              # case 1: input too short
    print('Sorry, try something longer.')
elif exp_clean == exp_rev :     # case 2: is palindrome
    print('YES, "'+exp+'" is a palindrome.')
else :                          # case 3: not palindrome
    print('NO, "'+exp+'" is not a palindrome.')
```

#3: Insistent palindrome



```
print("Hello! Let's start.")  # initial message, outside loop

# loop condition: initially set to false
success = False

# loop back while unsuccessful
while not success :

    # take in user input
    exp = input('Give me a palindrome: ')

    # clean up user input: lowercase, remove space and punctuation
    # line was too long: using \ to break up
    exp_clean = exp.lower().replace(' ', '').replace(',', '') \
                .replace("'", '').replace(':', '').replace('.', '')

    # string reversing routine, using exp_clean now
    new = ''                   # new is initially empty
    for x in exp_clean :
        new = x + new
    exp_rev = new              # exp_rev is reversed exp_clean

    # test and print out
    if len(exp) <= 2 :              # case 1: input too short
        print('Sorry, try something longer.')
    elif exp_clean == exp_rev :     # case 2: is palindrome
        print('YES, "'+exp+'" is a palindrome.')
        success = True                      # loop condition changed
    else :                          # case 3: not palindrome
        print('NO, "'+exp+'" is not a palindrome. Let\'s try again.')

print("Goodbye.")    # last message, outside loop
```

#4: Modular palindrome



```
def getRev(wd):
    "Takes a string, returns its reverse"
    rev = ''
    for i in wd :
        rev = i + rev
    return rev

def cleanInput(foo):
    "Lowercases input, removes space and punctuation .,':"
    return foo.lower().replace(' ', '').replace(',', '')\
           .replace("'", '').replace(':', '').replace('.', '')


#############################################################
# Main routine below

print("Hello! Let's start.")  # initial message, outside loop

# loop condition: initially set to false
success = False

# loop back while unsuccessful
while not success :

    # take in user input
    exp = input('Give me a palindrome: ')

    exp_clean = cleanInput(exp)   # clean input
    exp_rev = getRev(exp_clean)   # reverse cleaned input

    # test and print out
    if len(exp) <= 2 :              # case 1: input too short
        print('Sorry, try something longer.')
    elif exp_clean == exp_rev :     # case 2: is palindrome
        print('YES, "'+exp+'" is a palindrome.')
        success = True                      # loop condition changed
    else :                          # case 3: not palindrome
        print('NO, "'+exp+'" is not a palindrome. Let\'s try again.')

print("Goodbye.")   # last message, outside loop
```

Solutions

- Found here:
  - https://sites.pitt.edu/~naraehan/ling1330/pal_SOLUTIONS.txt
  - (copy and paste each of the four scripts)

- Palindrome speed coding video:
  - https://sites.pitt.edu/~naraehan/ling1330/pal-speed-coding.mp4

- Lessons learned?

How is language represented on a computer?

- Natural ("Human") languages:
  - Spoken form
  - Written form

  *Also: sign languages

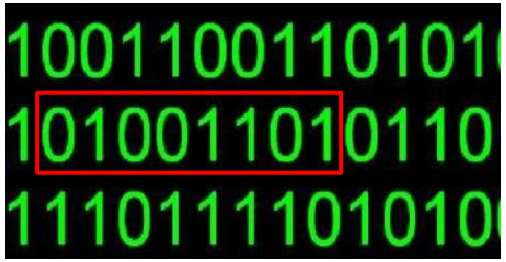
- The language of computers:

The language of computers

- At the lowest level, computer language is *binary*:

  Information on a computer is stored in **bits**
  - A bit is either: ON (=1, =yes) or OFF (=0, =no)
  - This language essentially contains <u>two</u> alphabetic characters

- Next level up: **byte**
  - A byte is made up of a sequence of **8 bits**
    - ex. `01001101` →
  - Historically, a byte was the number of bits used to <u>encode a single character</u> of text in a computer
  - Byte is a basic addressable unit in most computer architecture



Encoding a written language

- How to represent a *text* with 0s and 1s?
  - **`Hello world!`**
  - **010010000110010101101100011011000110111100100000011101110110111101110010 0**

    10011011000110010000100001
  - Each character is mapped to a ***code point*** (=*character code*), e.g., a unique integer.
    - H → 72_{dec}
    - e → 101_{dec}
  - Each code point is represented as a binary number, using a fixed number of bits.
    - **8 bits** == **1 byte** in the example above
    - H → 72_{dec} → **01001000** ($2^6+2^3 = 64 + 8 = 72$)
    - e → 101_{dec} → **01100101** ($2^6+ 2^5 + 2^2 + 2^0 = 64 + 32 + 4 + 1 = 101$)
  - One byte can represent 256 (=2^8) different characters
    - 00000000 → 0_{dec} 11111111 → 255_{dec}

ASCII encoding for English

- How many bits are needed to encode *English*?
  - 26 lowercase letters: a, b, c, d, e, …
  - 26 uppercase letters: A, B, C, D, E, …
  - 10 Arabic digits: 0, 1, 2, 3, 4, …
  - Punctuation: . , : ; ? ! ' "
  - Symbols: ( ) < > & % * $ + -
- ← We are already up to 80
- ← 6 bits (2^6 = 64) is not enough; we will need at least 7 (2^7 = 128)
- ← **ASCII** (the American Standard Code for Information Interchange) did just that, back in 1963
  - Uses 7-bit code (= 128 characters) for storing English text
  - Range 0 to 127

The ASCII chart

- https://en.wikipedia.org/wiki/ASCII

Decimal	Binary (7-bit)	Character
0	000 0000	(NULL)
…	…	…
35	010 0011	#
36	010 0100	&
…	…	…
48	011 0000	0
49	011 0001	1
50	011 0010	2
…	…	…

Decimal	Binary (7-bit)	Character
65	100 0001	A
66	100 0010	B
67	100 0011	C
…	…	…
97	110 0001	a
98	110 0010	b
99	110 0011	c
…	…	…
127	111 1111	(DEL)

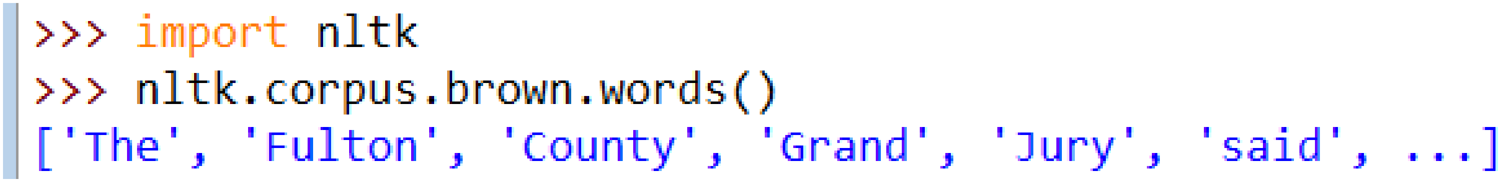
Wrap-up

- Exercise #2 out
  - Due Tuesday 10:45am, on Canvas
  - Pig Latin script
- Monday is Labor Day; no office hours
  - Need help? Utilize MS Teams
- Next class:
  - More encoding systems, Unicode
  - Text processing with NLTK
- **Install NLTK!!** → DETAILS NEXT PAGE
- Get started with the NLTK Book, chapters 1 through 3.

NLTK installation!

- Instructions on the "Checklists" page
  - https://sites.pitt.edu/~naraehan/ling1330/checklists.html#setup-nltk

- After successful install + data download, you can:



```
>>> import nltk
>>> nltk.corpus.brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

- **Anaconda** users: nltk is already installed, but you still need to download language data packs.
- **Python.org** users: You need to install nltk through **pip**, in command line. If you're new to command line, chances are you will need help. (Plan ahead!!)