**Outline**

Secret messages

**Text Manipulations**

Introduction to Regex

Prof. Angela Chang

Lecture 7: Text I

Fall 2017. Sep 27

# CODE, CULTURE, AND PRACtICE

# Diagramming on Paper

Take a sheet of paper

Find a piece of code that has 4-15 lines which consists of::

- an iterator (for statement)
- some computation (+,/,*, etc)
- conditionals (if/then/switch)

e.g.. a poetry generator

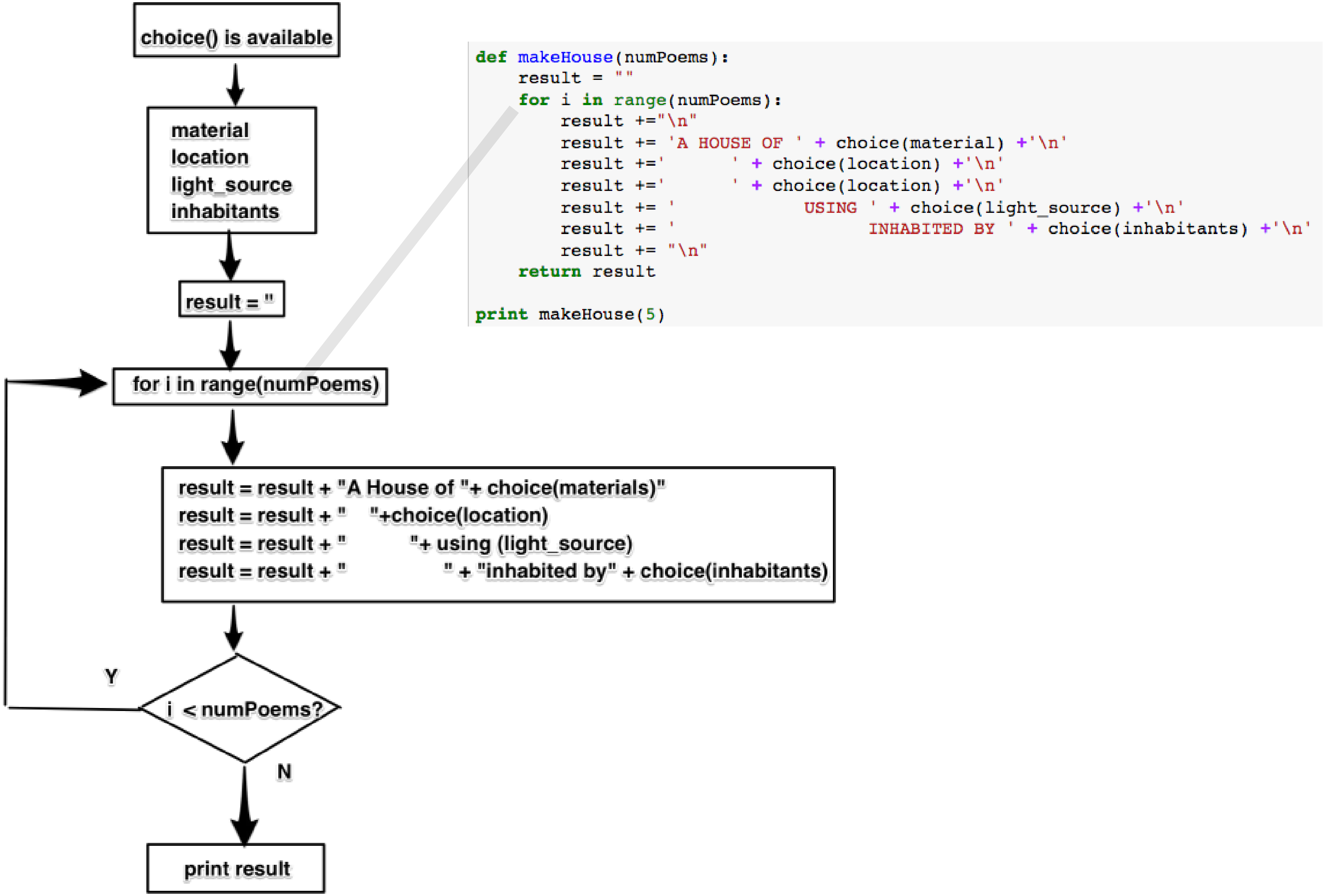
Diagram it out:

- at statements, use a rectangle
- at conditionals, use a diamond
- draw arrows as flow passes different parts of the diagram

Simulate it on paper:

- start at the beginning
- as you follow each step of your diagram
- write out each variable value and arithmetic performed

# House of Dust

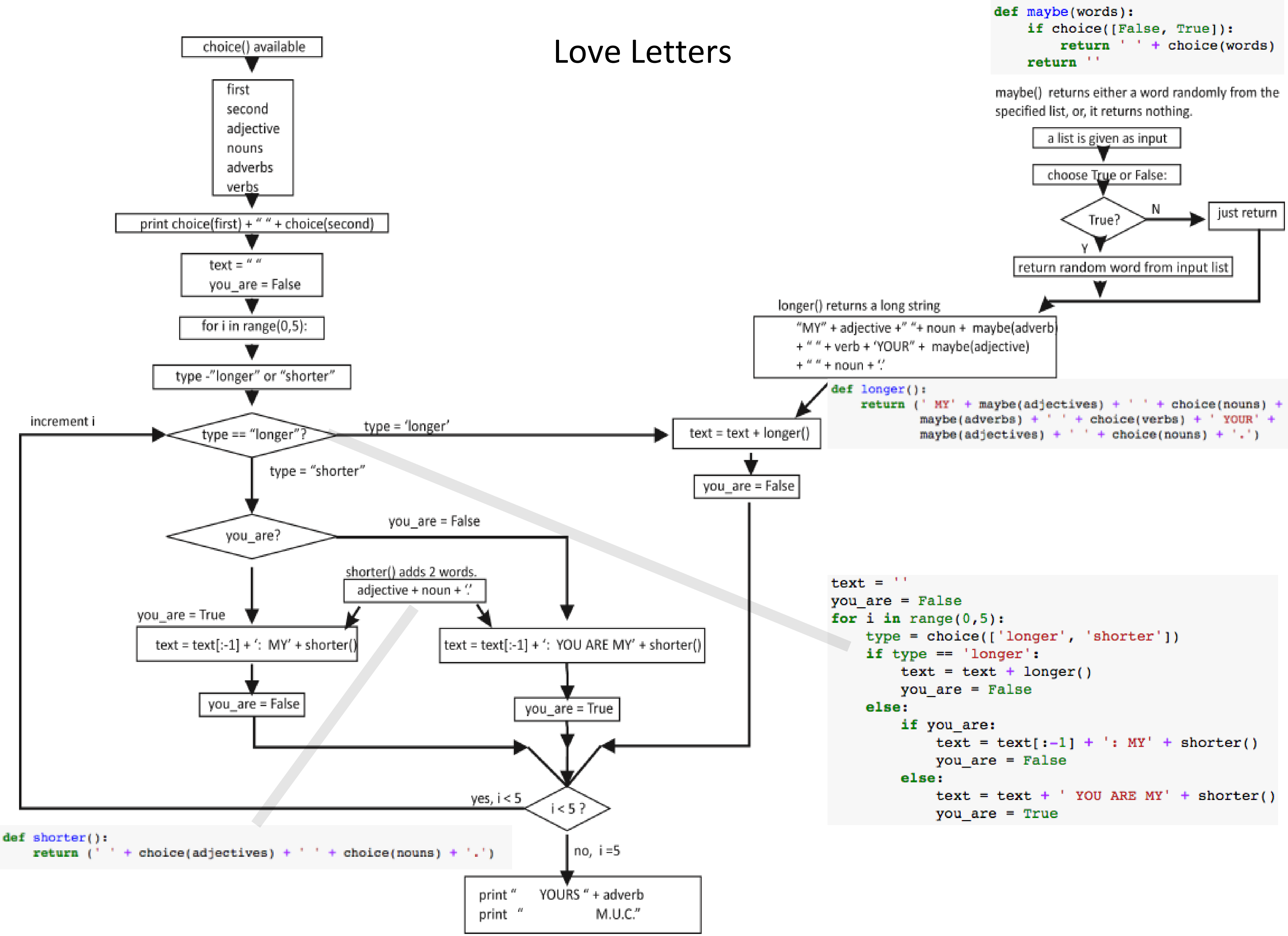


```
def makeHouse(numPoems):
    result = ""
    for i in range(numPoems):
        result +="\n"
        result += 'A HOUSE OF ' + choice(material) +'\n'
        result +='      ' + choice(location) +'\n'
        result +='      ' + choice(location) +'\n'
        result += '           USING ' + choice(light_source) +'\n'
        result += '                INHABITED BY ' + choice(inhabitants) +'\n'
        result += "\n"
    return result

print makeHouse(5)
```

choice() is available
material
location
light_source
inhabitants
result = ''
for i in range(numPoems)
result = result + "A House of "+ choice(materials)"
result = result + "     "+choice(location)
result = result + "          "+ using (light_source)
result = result + "               " + "inhabited by" + choice(inhabitants)
Y
i < numPoems?
N
print result

# Love Letters



```
def maybe(words):
    if choice([False, True]):
        return ' ' + choice(words)
    return ''
```

maybe() returns either a word randomly from the specified list, or, it returns nothing.

longer() returns a long string

“MY” + adjective +” “+ noun + maybe(adverb)
+ “ “ + verb + ‘YOUR” + maybe(adjective)
+ “ “ + noun + ‘.’

```
def longer():
    return (' MY' + maybe(adjectives) + ' ' + choice(nouns) +
            maybe(adverbs) + ' ' + choice(verbs) + ' YOUR' +
            maybe(adjectives) + ' ' + choice(nouns) + '.')
```

shorter() adds 2 words.

adjective + noun + ‘.’

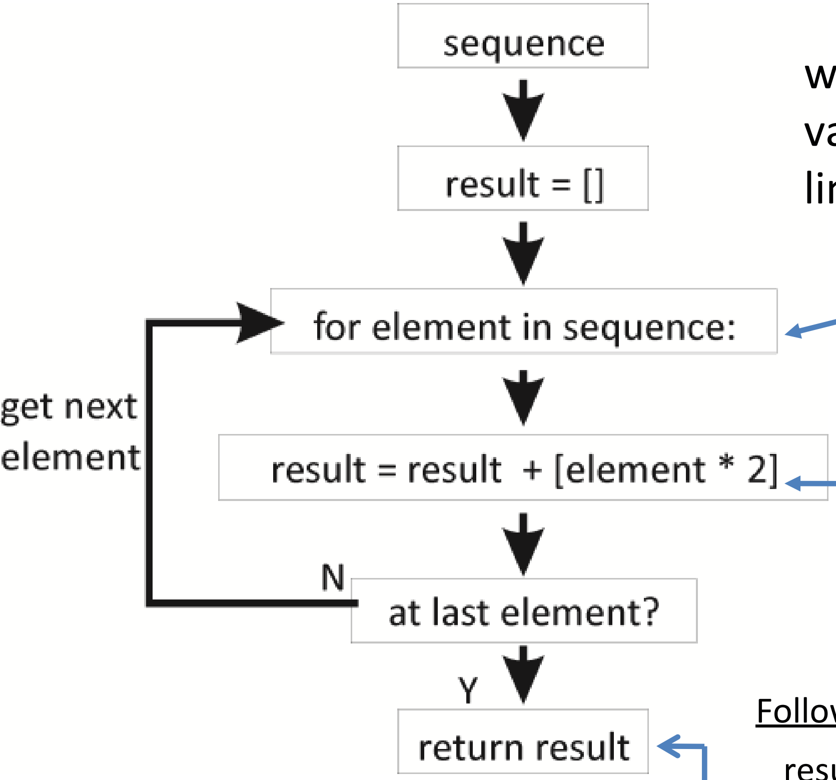
print “ YOURS “ + adverb
print “ M.U.C.”

```
text = ''
you_are = False
for i in range(0,5):
    type = choice(['longer', 'shorter'])
    if type == 'longer':
        text = text + longer()
        you_are = False
    else:
        if you_are:
            text = text[:-1] + ': MY' + shorter()
            you_are = False
        else:
            text = text + ' YOU ARE MY' + shorter()
            you_are = True
```

```
def shorter():
    return (' ' + choice(adjectives) + ' ' + choice(nouns) + '.')
```

# Double()

we can also use the diagram to follow how the variables are updated as the computer processes each line.



sequence

result = []

for element in sequence:

result = result + [element * 2]

at last element?

N

Y

return result

get next element

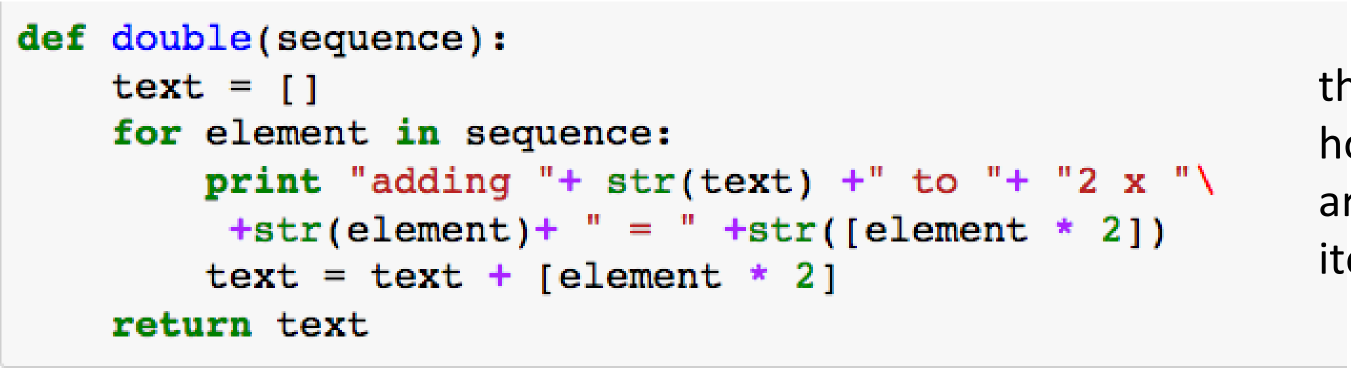
sequence = ‘python’
result is an empty list to start

loop through the characters
compute right side of the equation
then use the result on the next iteration

Follow the variables through loop, list the variables and computation:

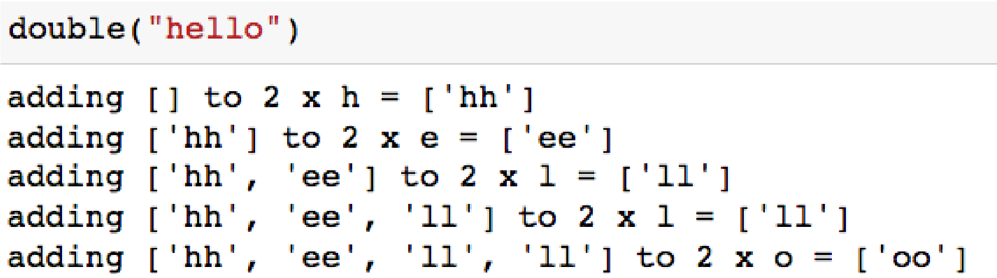
| result | element | [element *2] | result +[element *2] |
|---|---|---|---|
| [ ] | ‘p’ | [‘pp’] | [ ] + [‘pp’] = [‘pp’] |
| [‘pp’] | ‘y’ | [‘yy’] | [‘pp’] + [‘yy’] = |
| [‘’ppyy’] | ‘t’ | [‘t’t’] | [‘ppyy’] + [‘tt’] = |
| [‘ppyytt’] | ‘h’ | [‘hh’] | [‘ppyytt’] + [‘hh’] |
| [‘ppyytthh’] | ‘o’ | [‘oo’] | [‘ppyytthh’] + [‘oo’] |
| [‘[ppyytthhoo’] | ‘n’ | [‘nn’] | [‘ppyytthhoo’] + [‘nn’] |
| [‘ppyytthhoonn’] | | | |

# printing out what’s happening



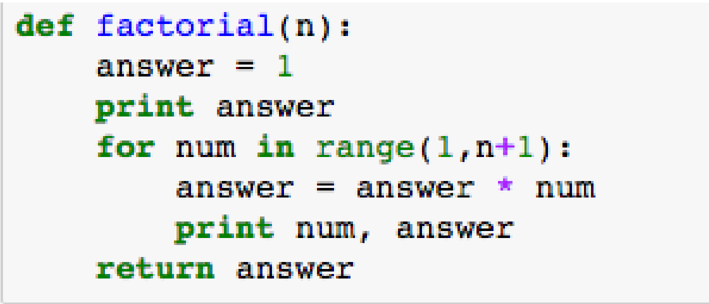
```
def double(sequence):
    text = []
    for element in sequence:
        print "adding "+ str(text) +" to "+ "2 x "\
         +str(element)+ " = " +str([element * 2])
        text = text + [element * 2]
    return text
```

the print statement shows how the variables are updated at each iteration

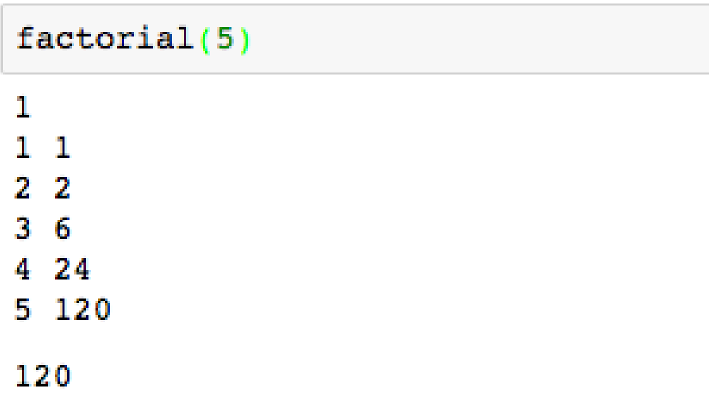


```
double("hello")
```

```
adding [] to 2 x h = ['hh']
adding ['hh'] to 2 x e = ['ee']
adding ['hh', 'ee'] to 2 x l = ['ll']
adding ['hh', 'ee', 'll'] to 2 x l = ['ll']
adding ['hh', 'ee', 'll', 'll'] to 2 x o = ['oo']
```



```
def factorial(n):
    answer = 1
    print answer
    for num in range(1,n+1):
        answer = answer * num
        print num, answer
    return answer
```



```
factorial(5)
```

```
1
1 1
2 2
3 6
4 24
5 120

120
```

# PYTHON _ Text Olympics

**Counting Spaces**

Write count_spaces(), a function that accepts a string as an argument and returns the number of spaces in the string. Use iteration to determine this.

If you can think of more than one way to accomplish this, write count_spaces2() and go on to write count_spaces_3() and beyond if you like, showing the alternatives. To accomplish the basic, initial count_spaces() function, no special knowledge of Python is needed beyond what has already been covered.

**Counting Non-spaces**

Write a function count_nonspaces() that returns the number of characters in a string that are not spaces. Try figuring this out using iteration, with reference to the problem just solved. Once you have solved the problem this way, see if you can you determine how do this in a single line (not counting the line beginning with def) by having count_nonspaces() call count_spaces() from before.

**Same Last Character**

Write same_last(), a function that accepts two strings as arguments and returns True if they have the same last letter, False otherwise. For this exercise, you can assume that both of the strings are at least one letter long—it does not matter what happens (the program could crash, etc.) if one or both of the strings is the null string.

After you write a function that works, see if you have more than one line in the function body—that is, if you have any code besides the def line and one line after it. If your function is more than two lines long, refactor it so that it is only two lines long.

**Determining Initials**

Write a function initials() that takes a string containing any name (a personal or business name, for instance) and returns the initials. For instance, the values returned by the following function calls will be:

Initials(“International Business Machines”) → IBM

Initials(“M. Lee Pelton”) → MLP

You should be able to tell what type the return value (that is, your result: the initials) should be. The function should work properly on names with any length string. Do not worry about special handling for cases where punctuation makes up its own “word,” or where a word begins with a punctuation mark, or where you know that a compound word. Just return the first character of each part of the string separated by whitespace.

**Remove Vowels**

Write **devowel()**, a function that accepts a string as an argument and returns the string without the vowels. For instance, given ‘hello world’ it will return ‘hll wrld’. Just consider the five standard, full vowels for this exercise, neglecting poor y and w.

**Tautonyms**

Write a function **tautonym()** that accepts a string and returns **True** if the string consists of some sequence of characters (call it A) followed by the same sequence of characters, A, The function should return **False** otherwise. For instance, given ”hello world” it should return **False** but given ’worldworld’ it should returns **True**. Of course, for ‘worldworldbaby’ the answer **False**.

# Text challenges

**Counting Spaces**

Write count_spaces(), a function that accepts a string as an argument and returns the number of spaces in the string. Use iteration to determine this.

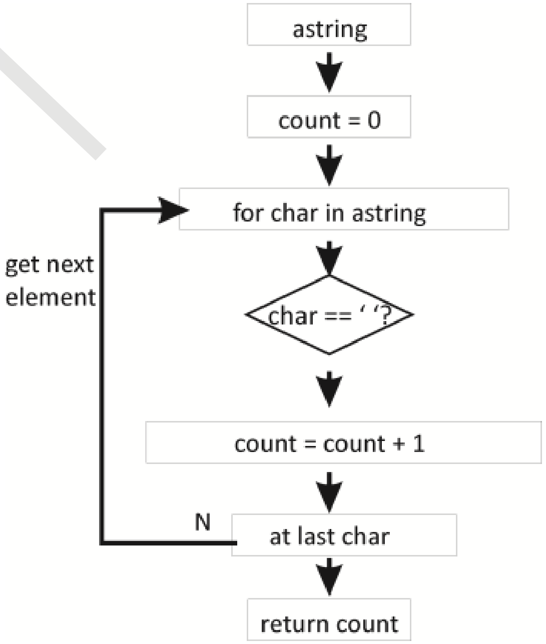
If you can think of more than one way to accomplish this, write count_spaces2() and go on to write count_spaces_3() and beyond if you like, showing the alternatives. To accomplish the basic, initial count_spaces() function, no special knowledge of Python is needed beyond what has already been covered.

Read and break it down:

1. function that accepts a string
2. returns a number
3. iterate on each character
4. test if each character is a space/or nonspace
5. counts the spaces/nonspaces

**Counting Non-spaces**

Write a function count_nonspaces() that returns the number of characters in a string that are not spaces. Try figuring this out using iteration, with reference to the problem just solved. Once you have solved the problem this way, see if you can you determine how do this in a single line (not counting the line beginning with def) by having count_nonspaces() call count_spaces() from before.



*ps. use stackoverflow or google to find syntax and sample code
e.g    !=   inequality in python

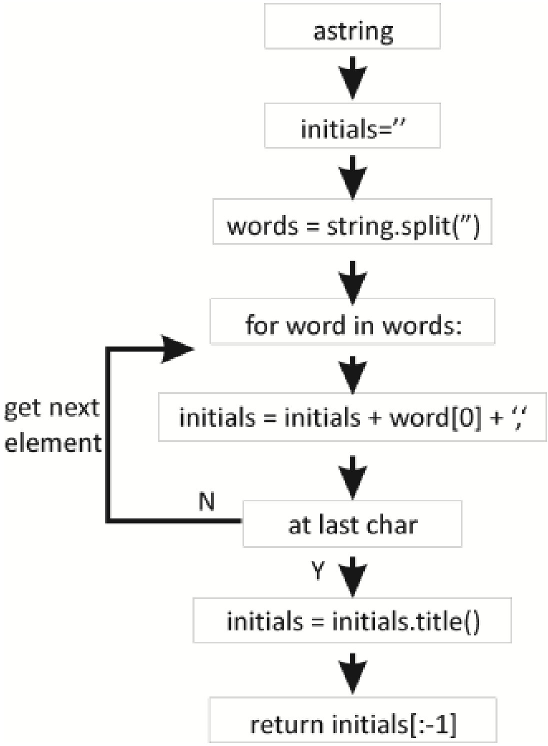
### Determining Initials

Write a function initials() that takes a string containing any name (a personal or business name, for instance) and returns the initials. For instance, the values returned by the following function calls will be:

Initials(“International Business Machines”) → IBM

Initials(“M. Lee Pelton”) → MLP

You should be able to tell what type the return value (that is, your result: the initials) should be. The function should work properly on names with any length string. Do not worry about special handling for cases where punctuation makes up its own “word,” or where a word begins with a punctuation mark, or where you know that a compound word. Just return the first character of each part of the string separated by whitespace.



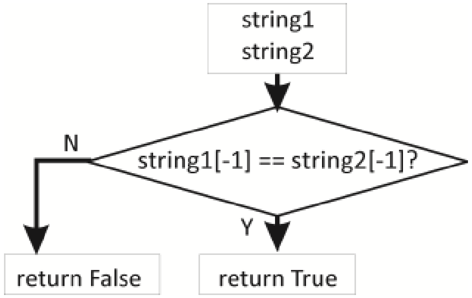
1. function that accepts a string
2. returns a string
3. iterate on each word
4. finds each first initial
5. joins together all the initials
6. returns initials

### Same Last Character

Write same_last(), a function that accepts two strings as arguments and returns True if they have the same last letter, False otherwise. For this exercise, you can assume that both of the strings are at least one letter long—it does not matter what happens (the program could crash, etc.) if one or both of the strings is the null string.

After you write a function that works, see if you have more than one line in the function body—that is, if you have any code besides the def line and one line after it. If your function is more than two lines long, refactor it so that it is only two lines long.

1. function that accepts 2 strings
2. returns True or False
3. finds each last character
4. tests if they are the same



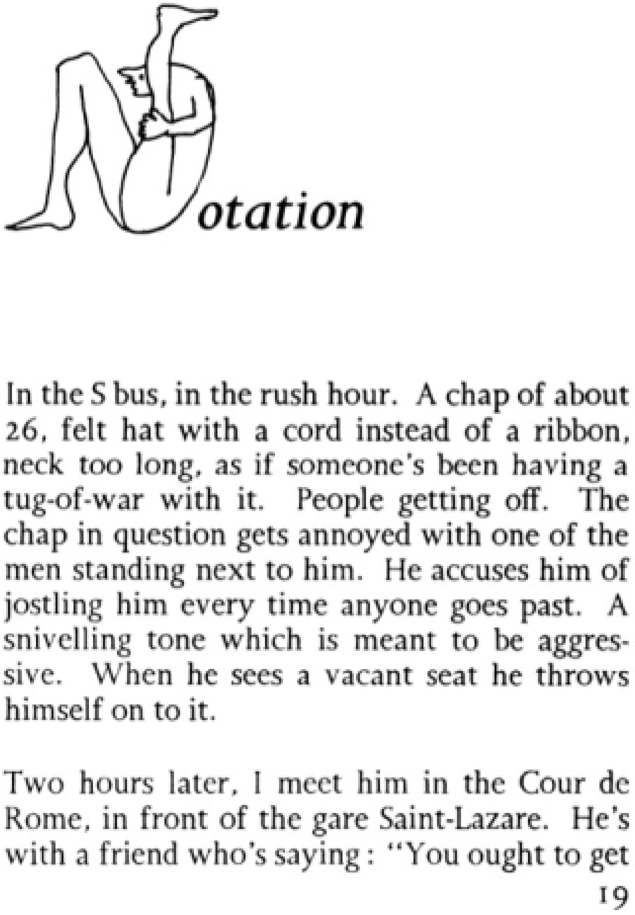
**Remove Vowels**
Write **devowel()**, a function that accepts a string as an argument and returns the string without the vowels. For instance, given ‘hello world’ it will return ‘hll wrld’. Just consider the five standard, full vowels for this exercise, neglecting poor y and w.

1. function that accepts a string
2. returns a string
3. iterate on each letter
4. saves only consonants
5. returns string without vowels

**Tautonyms**
Write a function **tautonym()** that accepts a string and returns **True** if the string consists of some sequence of characters (call it A) followed by the same sequence of characters, A, The function should return **False** otherwise. For instance, given ”hello world” it should return **False** but given ’worldworld’ it should returns **True**. Of course, for ‘worldworldbaby’ the answer **False**.

1. function that accepts a string
2. tests if the first half of the string is the same as the second half
3. returns True if they’re equal
4. otherwise returns False

# Homework Reading



## Notation

In the S bus, in the rush hour. A chap of about 26, felt hat with a cord instead of a ribbon, neck too long, as if someone's been having a tug-of-war with it. People getting off. The chap in question gets annoyed with one of the men standing next to him. He accuses him of jostling him every time anyone goes past. A snivelling tone which is meant to be aggressive. When he sees a vacant seat he throws himself on to it.

Two hours later, I meet him in the Cour de Rome, in front of the gare Saint-Lazare. He's with a friend who's saying: "You ought to get

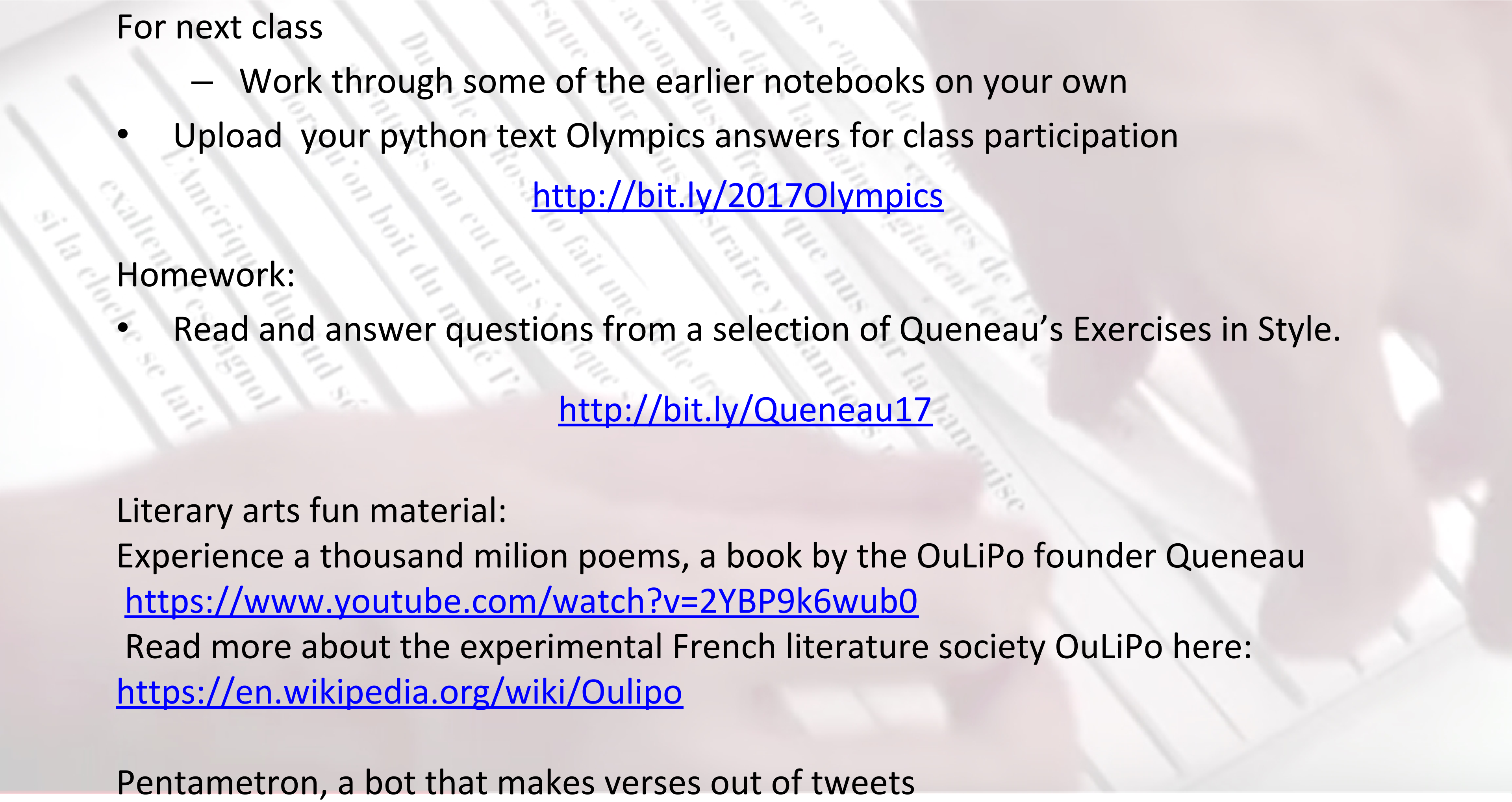
19

EXERCISES IN STYLE

RAYMOND QUENEAU

Read and answer questions from an excerpt of *Queneau's Exercises in Style.*

# Summary



For next class

- Work through some of the earlier notebooks on your own

- Upload  your python text Olympics answers for class participation

http://bit.ly/2017Olympics

Homework:

- Read and answer questions from a selection of Queneau's Exercises in Style.

http://bit.ly/Queneau17

Literary arts fun material:
Experience a thousand milion poems, a book by the OuLiPo founder Queneau
https://www.youtube.com/watch?v=2YBP9k6wub0
Read more about the experimental French literature society OuLiPo here:
https://en.wikipedia.org/wiki/Oulipo

Pentametron, a bot that makes verses out of tweets
https://twitter.com/pentametron?lang=en