Co-funded by the
Erasmus+ Programme
of the European Union

Computer Programming

For Language Students

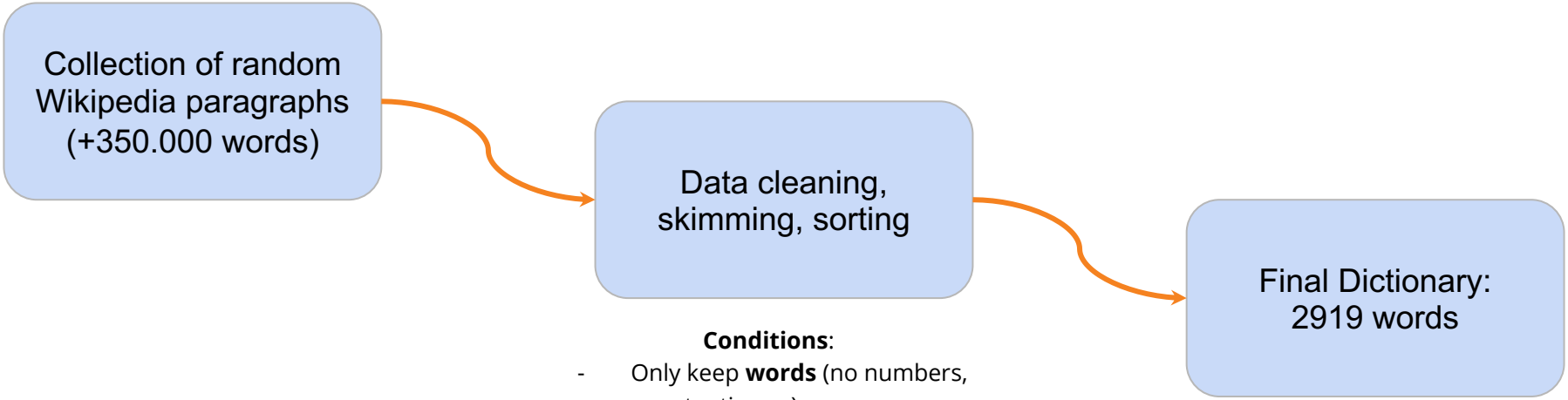
Alexandre Mazuir, Georgina Clarkson, Luca Giordano, Lucia Neuhold, Nikola Pušičić and Victor Chamot

Outline:

- Dictionary Creation
  - Python: Language as Building Material
- Functions
  - get_guess function
  - cover_letters function
- Different Code: Same Output
- Main Loop

Dictionary Creation

Where to get words to guess?



Conditions:

- Only keep **words** (no numbers, punctuations...)
- Length: **between 3 and 15 characters**
- Frequency: **more than 10 times** in the corpus

Only 10 lines of code!

```
with open('wiki_corpus.txt', 'r', encoding='utf-8') as f:
    text = f.read()

tokens = word_tokenize(text)

lower_tokens = [i.lower() for i in tokens]

token_frequencies = FreqDist(lower_tokens)

cleaned_words = []
for i in token_frequencies:
    if i.isalpha() and 3 < len(i) < 15 and token_frequencies[i] > 10:
        cleaned_words.append(i)
cleaned_words.sort()
```

Hey PC, I need you to:

1) Take a look at the original text

2) Throw every word, number, punctuation… in a bucket (they are all known as “tokens” → tokenization)

3) Lowercase whatever you can

4) Compute each token’s frequency in the corpus

5) Only keep words between 3 and 15 characters that appear more than 10 times

Python:
Language as Building Material

The dictionary creation step shows that:

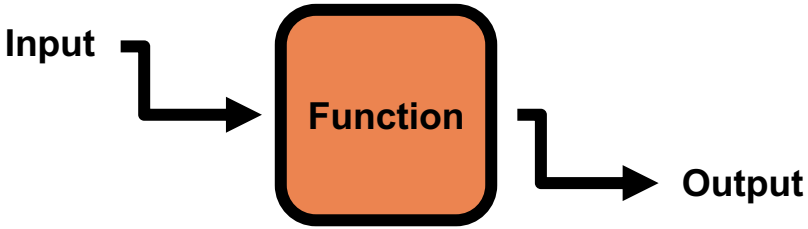
- Python is a tool used to **automate processes otherwise long, tedious and expensive** (e.g. lowercasing)

- Python makes it possible (and easy!) to treat language as building material: to reframe, reshape, mould, forge and **adapt language to one's needs** (Wikipedia entries → Hangman game dictionary)

- With Python it's easy to promote **gamification in linguistic research** (e.g. the Hangman game could easily be transformed in a spelling game for Second Language Acquisition research)

Functions

Functions are abstract, and useful through an application



Examples :

Applications :

x^2

2 → function → 4

no [+voice] consonant before a [-voice] one

[bs] → function → [ps]

Functions

But why use functions ?

- Modular approach
- Get rid of repetitions
- Easy identification of mistakes

get_guess function

- Asks for a letter (line 4)
- Verifies if the input is an actual single letter (line 5-6)
- Checks if this letter belong to the argument of the function (i.e. valid_letters) (line 7-8)
- If the letter is not valid the loop start again (cf. While loop line 3)
- If the input letter respects the conditions, function returns it : the valid letter can be reused as an input in the final program (line 9)

```
def get_guess(valid_letters):
    x = ""
    while len(x) != 1 or x not in valid_letters :
        x = input("Input a letter")
        if len(x) != 1 :
            print("Not a letter")
        if x not in valid_letters :
            print("Not an available letter")
    return x

letter = get_guess('abc')
print('letter entered was', letter)
```

```
Input a letter d
Not an available letter
Input a letter ab
Not a letter
Input a letter 1
Not an available letter
Input a letter a
letter entered was a
```

cover_letters function

-Replaces the letters of the *target_word* with an asterix (*****)
-It checks up on changes in the *unguessed_letters*
-Makes the letter show up instead of the (*****) for each correct answer

```
def cover_letters(target_word, unguessed_letters):
    for x in target_word:
        if x in unguessed_letters:
            target_word=target_word.replace(x,'*')
    return target_word

print('Should print "****":', cover_letters('beer', 'abcdefghijklmnopqrstuvwxyz'))
print('Should print "*ee*":', cover_letters('beer', 'abcdfghijklmnopqrstuvwxyz'))
```

```
target_word=random.choice(cleaned_words)
unused_correct_letters=target_word
unused_letters='abcdefghijklmnopqrstuvwxyz'
num_fails=0

while unused_correct_letters != '':
    letter=get_guess(unused_letters)
    if letter not in target_word:
        num_fails=num_fails+1
    unused_letters=unused_letters.replace(letter, '*')
    unused_correct_letters=unused_correct_letters.replace(letter,'')


    print('the secret word:',cover_letters(target_word, unused_letters))
    print('number of fails: ', num_fails)
    print('available letters:', unused_letters)
```

```
target_word=random.choice(cleaned_words)
unused_correct_letters=target_word
unused_letters='abcdefghijklmnopqrstuvwxyz'
num_fails=0

while unused_correct_letters!='':
    letter=get_guess(unused_letters)
    print('Number of fails:',num_fails)
    print('Letters left:',unused_letters)
    print('Current word:',cover_letters(target_word,unused_letters))
    if letter not in unused_correct_letters:
        num_fails=num_fails+1
        unused_letters=unused_letters.replace(letter,'*')
    else:
        unused_correct_letters=unused_correct_letters.replace(letter,'')
        unused_letters=unused_letters.replace(letter,'*')
```

Different code, same output

```
enter a letter:k
the secret word: ********
number of fails:  1
available letters: abcdefghij*lmnopqrstuvwxyz
enter a letter:e
the secret word: e******e*
number of fails:  1
available letters: abcd*fghij*lmnopqrstuvwxyz
enter a letter:a
the secret word: e******e*
number of fails:  2
available letters: *bcd*fghij*lmnopqrstuvwxyz
enter a letter:s
the secret word: e******e*
number of fails:  3
available letters: *bcd*fghij*lmnopqr*tuvwxyz
```

The Main Loop

```
1  target_word = random.sample(cleaned_words, 1)[0]
   unused_correct_letters = target_word
   unused_letters = "abcdefghijklmnopqrstuvwxyz"
   num_fails = 0

2  while num_fails <11:
       print('Number of Fails: ', num_fails)
       print('Available Letters: ', unused_letters)
       print('Current Word: ', cover_letters(target_word, unused_letters))
3      letter = get_guess(unused_letters)
4      if letter not in unused_correct_letters:
           num_fails = num_fails+1
       unused_letters = unused_letters.replace(letter, '')
       unused_correct_letters = unused_correct_letters.replace(letter, '')
5      if unused_correct_letters == (''):
           print('''Yaaaay, you got it! The word was: ''', target_word)
           break
       if num_fails == 10:
           print('''Bad luck!''')
```

1 - A random word is chosen from the set. Available letters and fails are put in a variable.

2 - The loop begins and prints text with the parameters. The cover-letters function is used here.

3 - The letter variable is set using the get_guess function.

4 - If loops: to accumulate the number of fails, remove the letters guessed and replace the * for correctly guessed letters.

5 - Parameters are set for the game to conclude.

Co-funded by the Erasmus+ Programme of the European Union

The End!

Thanks for your attention

Alexandre Mazuir, Georgina Clarkson, Luca Giordano, Lucia Neuhold, Nikola Pušičić and Victor Chamot