Applied Natural Language Processing

Dr Jeff Mitchell, University of Sussex

Autumn 2025

What is Applied Natural Language Processing?

- Natural languages are …
  - Languages invented by humans
  - In order to communicate with humans
- Natural language processing is …
  - getting computers to handle natural language inputs and outputs
- Applied Natural Language Processing is
  - using computer tools and applications to do interesting things with natural language

What applications of Natural Language Processing can you think of?

?

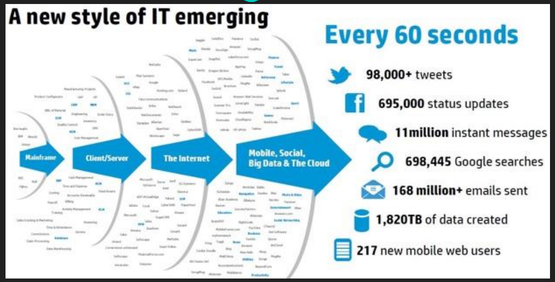
Some applications I thought of….

- Information retrieval
  - Retrieve and rank documents on the web that are relevant to a query
- Question answering
  - Provide answers to questions
- Machine translation
  - Translate documents (or speech) from one natural language to another
- Text simplification
  - Simplify a document (for a child or non-native speaker)
- Text summarization
  - E.g., summarize today's important news in 500 words

Some more applications ….

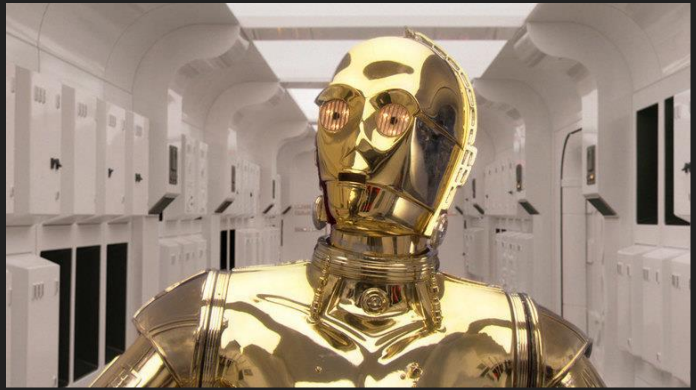
- Opinion monitoring
  - E.g., Find products/companies with good or bad reviews
- User-content moderation
  - Filter inappropriate content
  - Automatic anonymization
- Automatic recommendation
  - Products, jobs, people ….
  - Targeted advertising
- Chatbots and virtual assistants
  - Siri, Alexa, ChatGPT ….
- And many more …

Why study AppNLP?



Core field of Computer Science, Data Science and Artificial Intelligence

Lots of applications
⇒ Lots of jobs
⇒ Lots of money



Difficult, unsolved research problems
⇒ Interesting
⇒ Lots of money

Why is NLP difficult?

Surely it is easy … the vast majority of humans can process and generate speech and text, right?

- **Variation**: many different ways of expressing the same thing
  - Different languages
  - Different styles / genres
  - Messiness … ungrammatical, fragmentary, non-standard
- **Ambiguity**: same bit of language can have many different interpretations
  - So much depends on linguistic (and other) context
- **Creativity and evolution**: language change
  - New terms, new meanings, non-literal interpretations
- **Sparsity**: so many different words, so many different combinations, so many possible contexts ….
- **Grounding and world knowledge**: language does not exist in isolation
  - Humans do not learn language by purely observing endless stream of language
  - Ultimately an AI-complete problem

What will you do in AppNLP?

Guide to teaching and assessment

What will you do in AppNLP?

You will

- Learn about NLP problems and their potential solutions
- Learn the Python programming language
- Develop an appreciation of the challenges of NLP

What should I know already?

The module

- Does **not** assume knowledge of Python
    - And will teach the Python you need to know to apply NLP
- Does **not** assume knowledge of machine learning
    - And will explain any machine learning techniques used
- Does **not** assume any knowledge of linguistics
    - Beyond basic familiarity with the English language, vocabulary and grammar

How is AppNLP taught?

- 1x 2hr **F2F** lecture
  - get the theory
  - questions and answers with module tutor
  - discussion of lab exercises
  - recordings provided for online study/revision/catch-up
- 1 x 2h **F2F** lab
  - programming exercises
  - discussion
- Self-study
  - Completion of programming exercises
  - **Extra** course content provided on canvas videos
  - Background reading

- Taught by:
  - Dr Jeff Mitchell
- Assisted by:
  - postgraduates and postdoctorates, all actively engaged in research in this field
- All teaching materials available on Canvas, but….
- Regular attendance is critical to ensure success

What content is covered in NLE?

week 1	Intro to NLE and Python
week 2	Text Documents and Pre-processing
week 3	Document classification
week 4	Further document classification
week 5	Document Similarity and Clustering
week 6	Lexical semantics and word senses
week 7	Distributional semantics
week 8	Part-of-Speech (POS) Tagging and Hidden Markov Models
week 9	Named entity recognition (NER) and Information Extraction (IE)
week 10	Question answering (QA)
week 11	Lab catch-up

How is AppNLP assessed?

- Two contributory assessments
  - Report (30%), due week 7, assessing material from weeks 1-5
  - Computer-based Exam (70%), Jan assessment period, assessing material from weeks 1-12
- Non-contributory quizzes also available on Canvas

More on assessment

Coursework Report

- The report will consist of 2 questions (with 3-4 parts) and will assess a mixture of:
  - Programming
  - Analysis (requiring experimental results)
  - Evaluation (requiring written answers)
  - **Parts of questions in each assessed coursework will be based directly on programming exercises from the labs**

Computer-based Examination

- Assess knowledge and understanding of key theoretical concepts
- Assess ability to evaluate a scenario and make recommendations as to the probable effectiveness of different NLE technologies
- Assess python programming

How can I learn more?

- See Canvas for all teaching materials and links to more resources
- Recommended reading (core text for this module)
  - Jurafsky & Martin, Speech and Language Processing, 3rd ed., forthcoming
  - Available online: https://web.stanford.edu/~jurafsky/slp3/
- Ask questions
  - In labs
  - On Discord
  - On the Canvas forum
  - In my office hours
- Experiment with code
  - Make sure you complete and/or review labs outside of class
  - Try out your own ideas / extensions
  - Take part in peer-assessment
- Take Advanced Natural Language Processing option next semester

Python

Applying NLP Through Python Programming

Why Python?

- Flexible and powerful programming language
- Easy-to-learn
- Extremely popular in natural language processing and machine learning communities
- Extensive libraries making it possible for novices to experiment with complex technology

Which python?

- Python > 3.5
- Google CoLab
  - https://colab.research.google.com/notebooks/intro.ipynb
- Anaconda distribution available
  - On lab machines
  - From https://www.anaconda.com/
- Jupyter notebooks
  - Provide integrated environment for code, output and text

Making progress in the lab

- This week you should complete **all** of the exercises in **both notebooks** of the Introduction to Python course:

  - □ Part 1: NLE2023_Lab_1_1.ipynb
  - □ Part 2: NLE2023_Lab_1_2.ipynb

- There is also an extension notebook (NLE2023_Lab_1_3.ipynb) for students already familiar with Python programming.

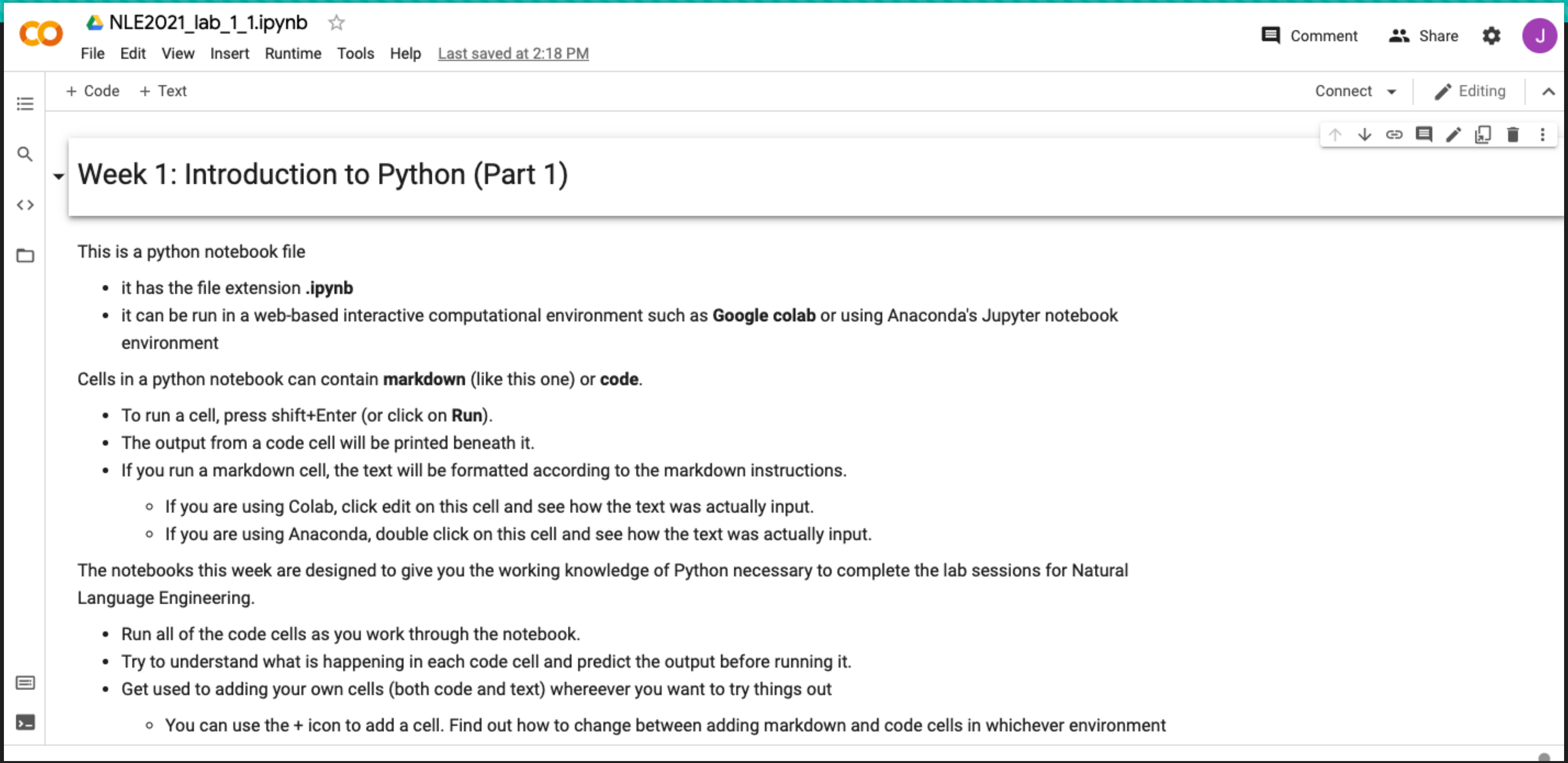
Getting started (on Google Colab)

- Go to Google Drive
  - https://drive.google.com/drive/u/0/my-drive
- Make a new directory e.g., AppNLP_Notebooks
- Download resources for week 1(Week1Labs.zip) from Canvas and store in your AppNLP_Notebooks directory on Google Drive
- Uncompress week1Labs.zip
- Navigate to Week1Labs folder
- Right click 1st notebook: NLE2023_Lab_1_1.ipynb
  - Choose open with Google CoLaboratory
  - You might need to connect this app (using Connect more apps option) first.

Getting started (on Anaconda)

- Go to your OneDrive or home directory
- Make a new directory e.g., AppNLP_Notebooks
- Download resources for week 1 (Week1Labs.zip) from Canvas and store in your AppNLP_Notebooks directory
- Uncompress week1Labs.zip
- Open a terminal window and navigate to Week1Labs folder e.g.:
  - `cd AppNLP_Notebooks/week1Labs`
- Launch jupyter notebook from the terminal window by typing
  - `jupyter notebook`
- A browser window should open saying that you are being redirected to Jupyter Notebook. Then a list of files in your current directory should appear.
- Select and open 1st notebook: NLE2023_Lab_1_1.ipynb

Open the first notebook


NLE2021_lab_1_1.ipynb
File Edit View Insert Runtime Tools Help Last saved at 2:18 PM
Comment Share
+ Code + Text
Connect Editing

Week 1: Introduction to Python (Part 1)

This is a python notebook file

- it has the file extension **.ipynb**
- it can be run in a web-based interactive computational environment such as **Google colab** or using Anaconda's Jupyter notebook environment

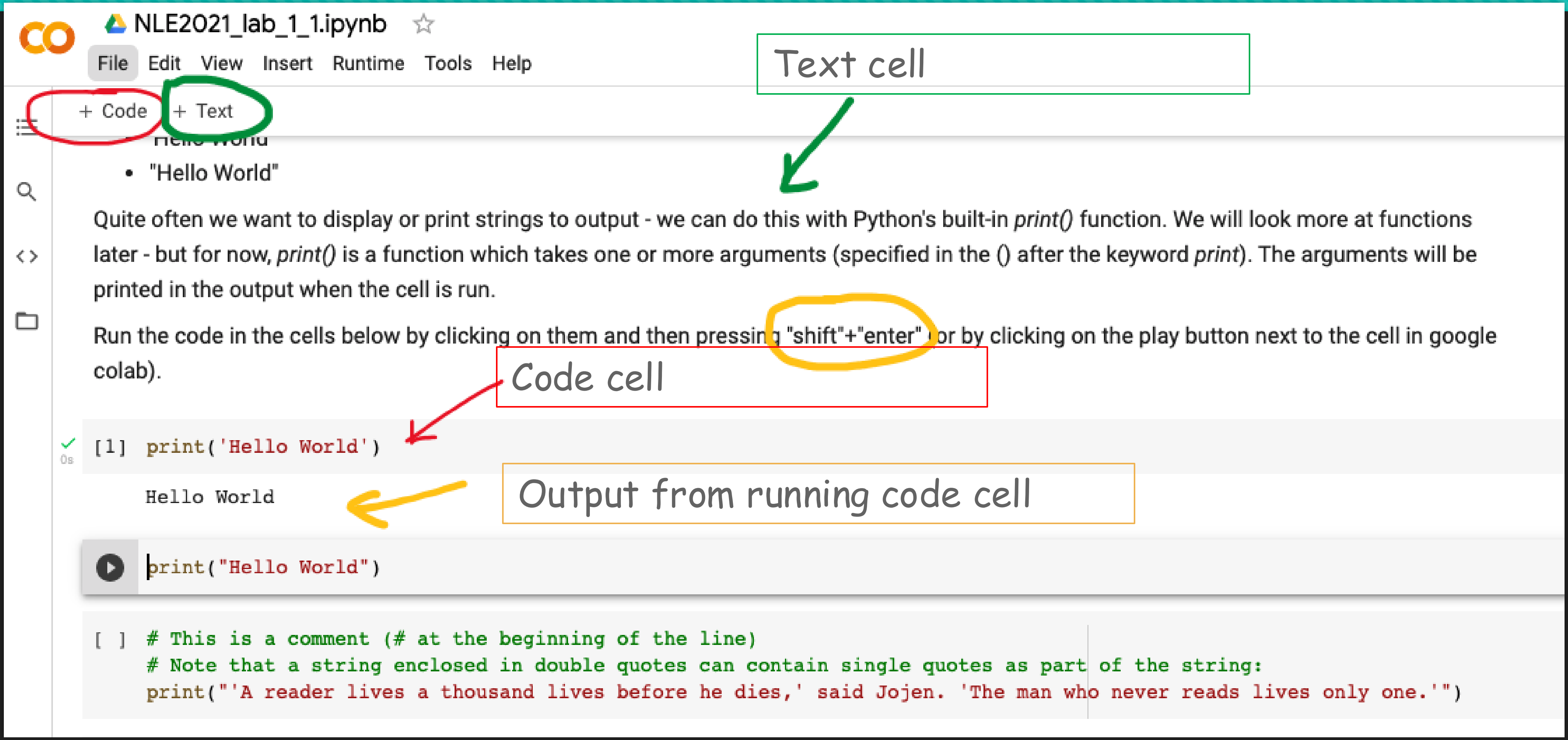
Cells in a python notebook can contain **markdown** (like this one) or **code**.

- To run a cell, press shift+Enter (or click on **Run**).
- The output from a code cell will be printed beneath it.
- If you run a markdown cell, the text will be formatted according to the markdown instructions.
    - If you are using Colab, click edit on this cell and see how the text was actually input.
    - If you are using Anaconda, double click on this cell and see how the text was actually input.

The notebooks this week are designed to give you the working knowledge of Python necessary to complete the lab sessions for Natural Language Engineering.

- Run all of the code cells as you work through the notebook.
- Try to understand what is happening in each code cell and predict the output before running it.
- Get used to adding your own cells (both code and text) whereever you want to try things out
    - You can use the + icon to add a cell. Find out how to change between adding markdown and code cells in whichever environment

Important Notebook Functionality



Text cell

- "Hello World"

Quite often we want to display or print strings to output - we can do this with Python's built-in *print()* function. We will look more at functions later - but for now, *print()* is a function which takes one or more arguments (specified in the () after the keyword *print*). The arguments will be printed in the output when the cell is run.

Run the code in the cells below by clicking on them and then pressing "shift"+"enter" (or by clicking on the play button next to the cell in google colab).

Code cell

```
[1] print('Hello World')
```

```
Hello World
```

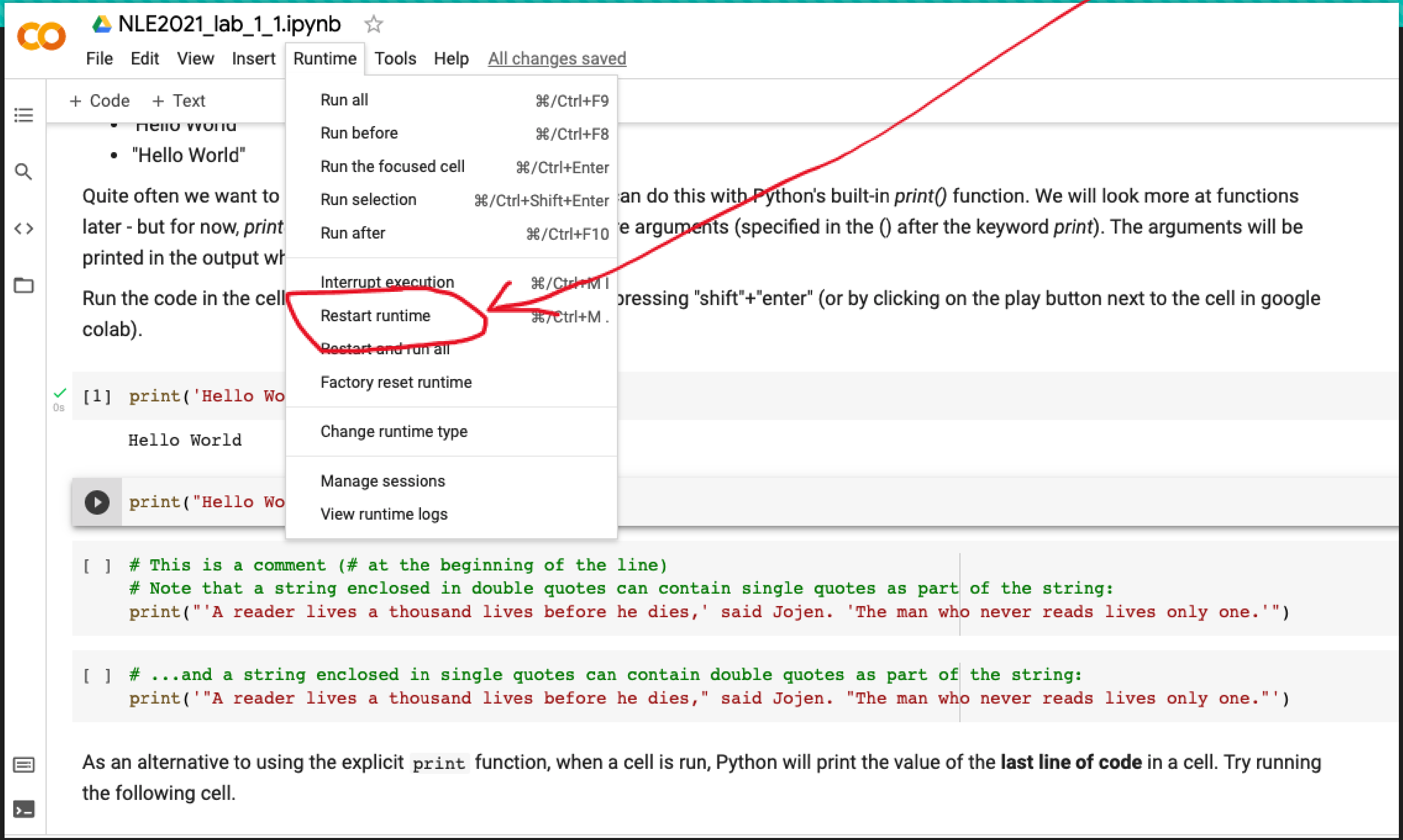
Output from running code cell

```
print("Hello World")
```

```
[ ] # This is a comment (# at the beginning of the line)
    # Note that a string enclosed in double quotes can contain single quotes as part of the string:
    print("'A reader lives a thousand lives before he dies,' said Jojen. 'The man who never reads lives only one.'")
```

More Notebook Functionality

This is your "turn it off and on again" – found on the "Kernel" menu on Anaconda



- Hello World
- "Hello World"

Quite often we want to … an do this with Python's built-in *print()* function. We will look more at functions later - but for now, *print* … e arguments (specified in the () after the keyword *print*). The arguments will be printed in the output wh…

Run the code in the cell … pressing "shift"+"enter" (or by clicking on the play button next to the cell in google colab).

```
[1] print('Hello Wo

    Hello World
```

```
print("Hello Wo
```

```
[ ] # This is a comment (# at the beginning of the line)
    # Note that a string enclosed in double quotes can contain single quotes as part of the string:
    print("'A reader lives a thousand lives before he dies,' said Jojen. 'The man who never reads lives only one.'")
```

```
[ ] # ...and a string enclosed in single quotes can contain double quotes as part of the string:
    print('"A reader lives a thousand lives before he dies," said Jojen. "The man who never reads lives only one."')
```

As an alternative to using the explicit `print` function, when a cell is run, Python will print the value of the **last line of code** in a cell. Try running the following cell.

Atomic Data Types

Make sure you know what all of these are and how to do basic operations with them

- **int**
- **float**
- **bool**
- **str** *(well maybe its atomic!)*

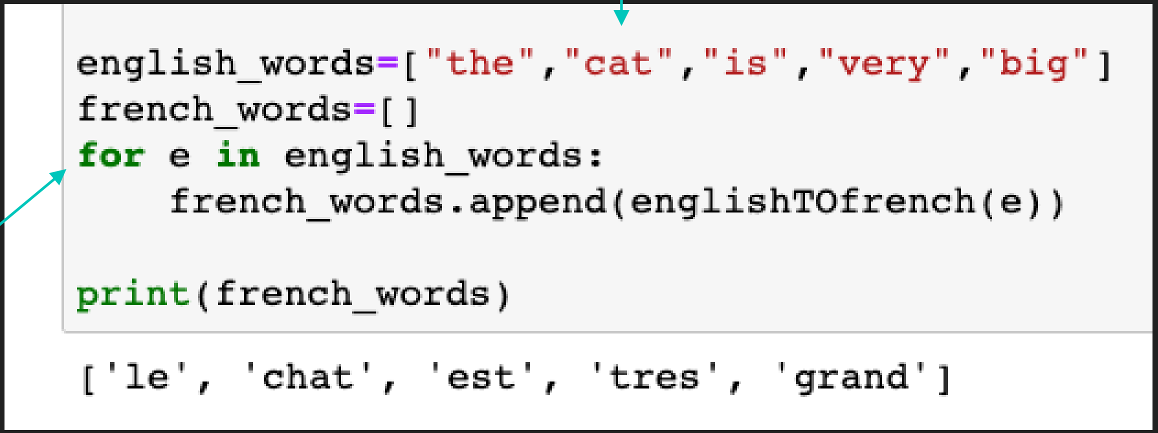
Some Really Useful Complex Data Structures

- Lists
- Tuples
- Sets
- Dictionaries

Lists

- A list is an **ordered** collection of other data types
- Can vary in length
- They are really useful in NLE for storing sequences of textual objects (characters, words, sentences, paragraphs, pages, documents!)
  - especially if we might have a lot and want to do the same thing to each object
  - in this case we will **iterate over the list**

Order matters! Not the same as ["very", "big", "is", "the", "cat"]



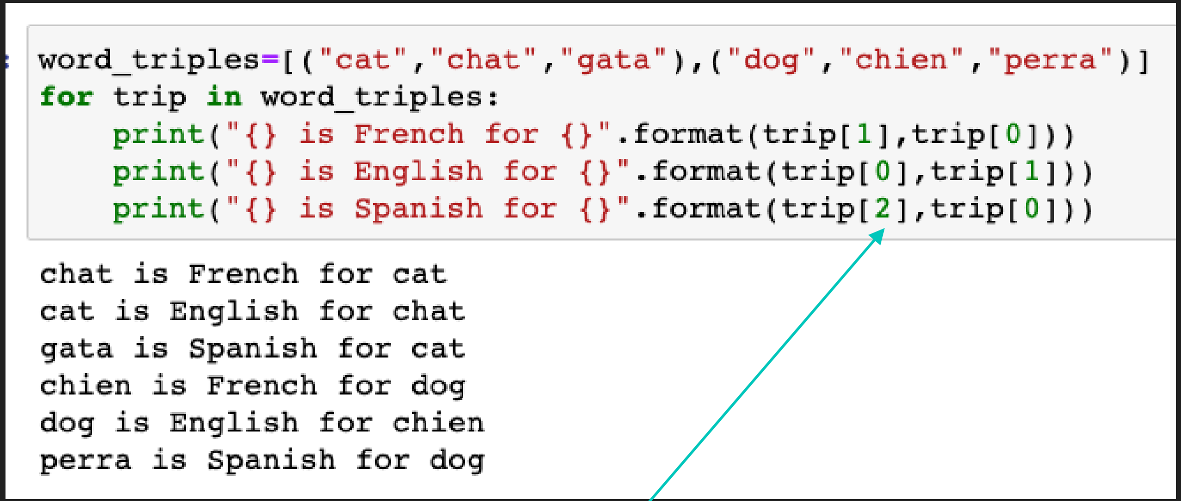
```
english_words=["the","cat","is","very","big"]
french_words=[]
for e in english_words:
    french_words.append(englishTOfrench(e))

print(french_words)
```

```
['le', 'chat', 'est', 'tres', 'grand']
```

Question for you: why is this word based approach unlikely to yield good translations?

Tuples

- Tuples are an **ordered fixed length** collection of pieces of data types
  - typically **pairs(2-tuples)** and **triples (3-tuples)**
- *For example*, a pair of words which have the same meaning 1) in English and 2) in French
- Of course, we might have a triple (or more of languages)
  - the point is that it is a fixed number (and order) for our application



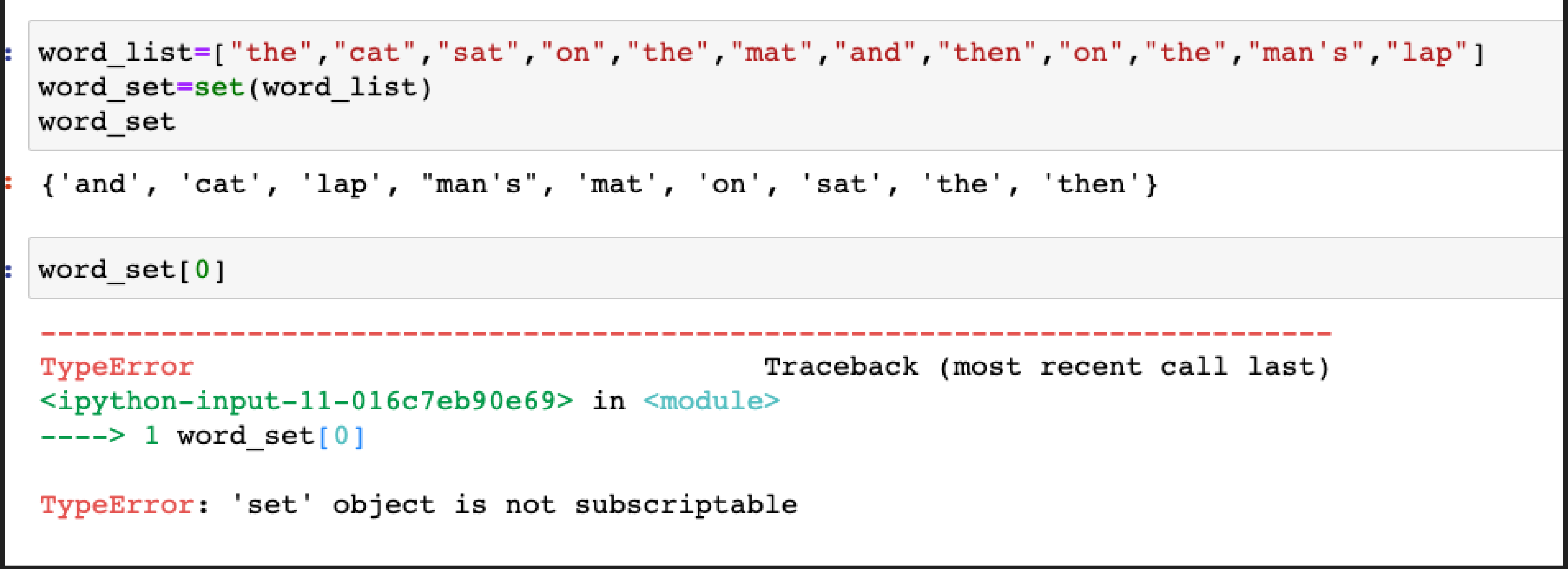
```
word_triples=[("cat","chat","gata"),("dog","chien","perra")]
for trip in word_triples:
    print("{} is French for {}".format(trip[1],trip[0]))
    print("{} is English for {}".format(trip[0],trip[1]))
    print("{} is Spanish for {}".format(trip[2],trip[0]))
```

```
chat is French for cat
cat is English for chat
gata is Spanish for cat
chien is French for dog
dog is English for chien
perra is Spanish for dog
```

we can use subscripts or indices to reference items in a tuple (or list) by their position

Sets

- Sets are unordered collections of other data types
- Each item in a set must be distinct i.e., unique
- We can iterate over sets but not reference items by index (as they are unordered)



```
word_list=["the","cat","sat","on","the","mat","and","then","on","the","man's","lap"]
word_set=set(word_list)
word_set
```

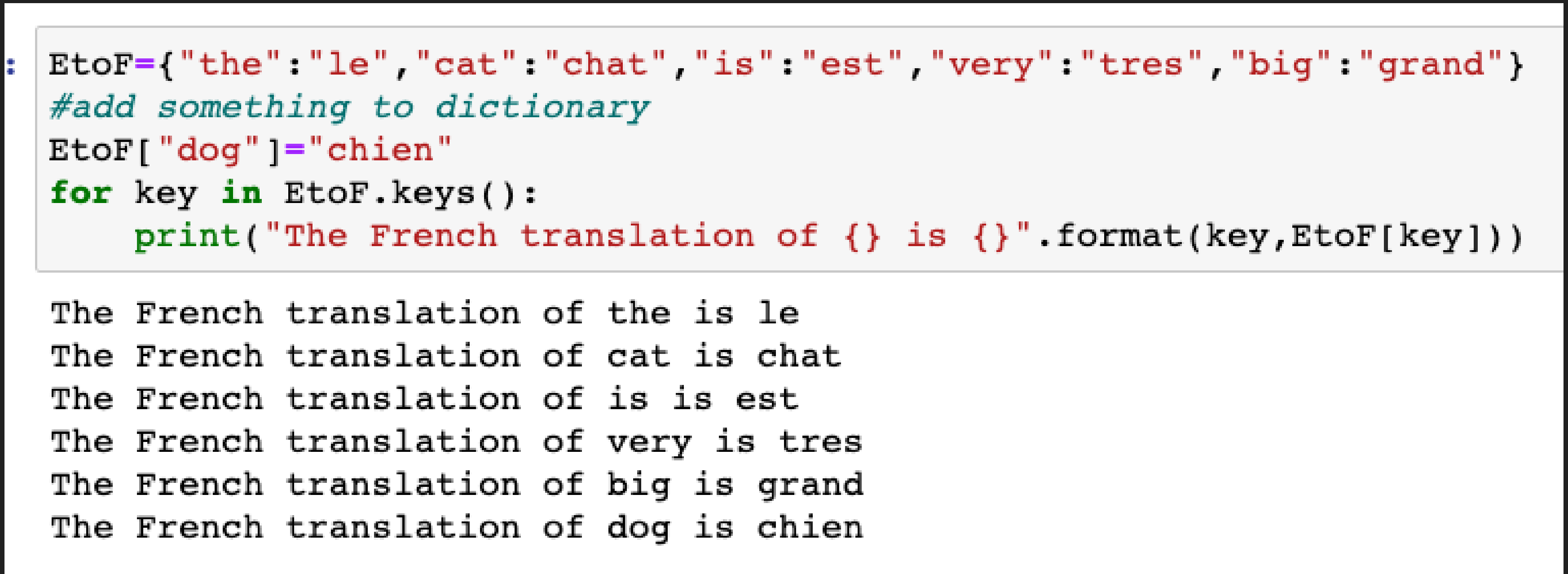
```
{'and', 'cat', 'lap', "man's", 'mat', 'on', 'sat', 'the', 'then'}
```

```
word_set[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-016c7eb90e69> in <module>
----> 1 word_set[0]

TypeError: 'set' object is not subscriptable
```

Dictionaries

- A mapping from keys to values
- An unordered set of (key,value) pairs - with fast look-up via hashing



```
EtoF={"the":"le","cat":"chat","is":"est","very":"tres","big":"grand"}
#add something to dictionary
EtoF["dog"]="chien"
for key in EtoF.keys():
    print("The French translation of {} is {}".format(key,EtoF[key]))
```

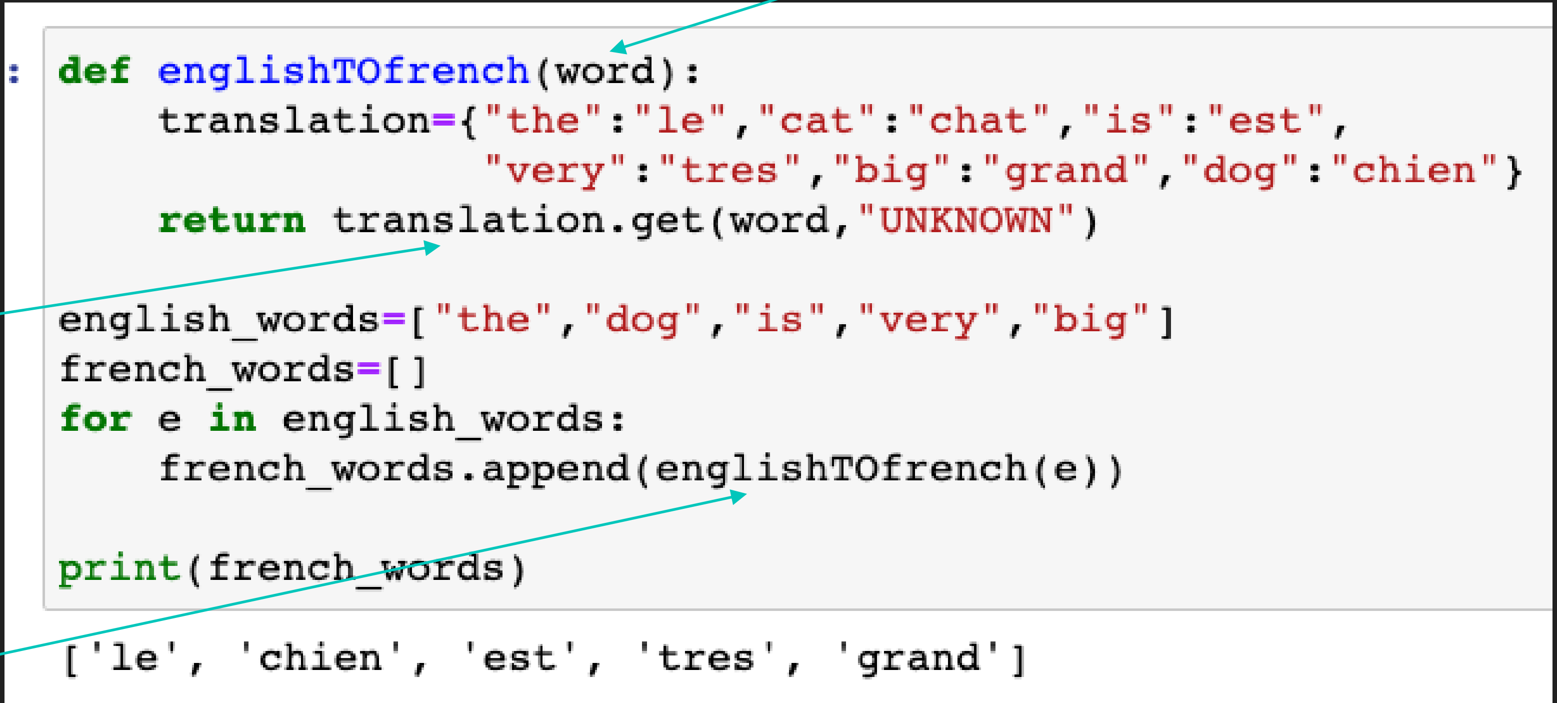
```
The French translation of the is le
The French translation of cat is chat
The French translation of is is est
The French translation of very is tres
The French translation of big is grand
The French translation of dog is chien
```

This python **dict** is storing an English:French bilingual dictionary but a **dict** can store any mapping. We will see lots of dictionaries throughout the course!

Functions

- We use a function when we want to apply the same code (or functionality) to different inputs
- We **call** a function on some **arguments** (or parameters)
- Here it **returns** a value to the calling code

This function takes a single argument or parameter



```
: def englishTOfrench(word):
      translation={"the":"le","cat":"chat","is":"est",
                   "very":"tres","big":"grand","dog":"chien"}
      return translation.get(word,"UNKNOWN")

  english_words=["the","dog","is","very","big"]
  french_words=[]
  for e in english_words:
      french_words.append(englishTOfrench(e))

  print(french_words)
```

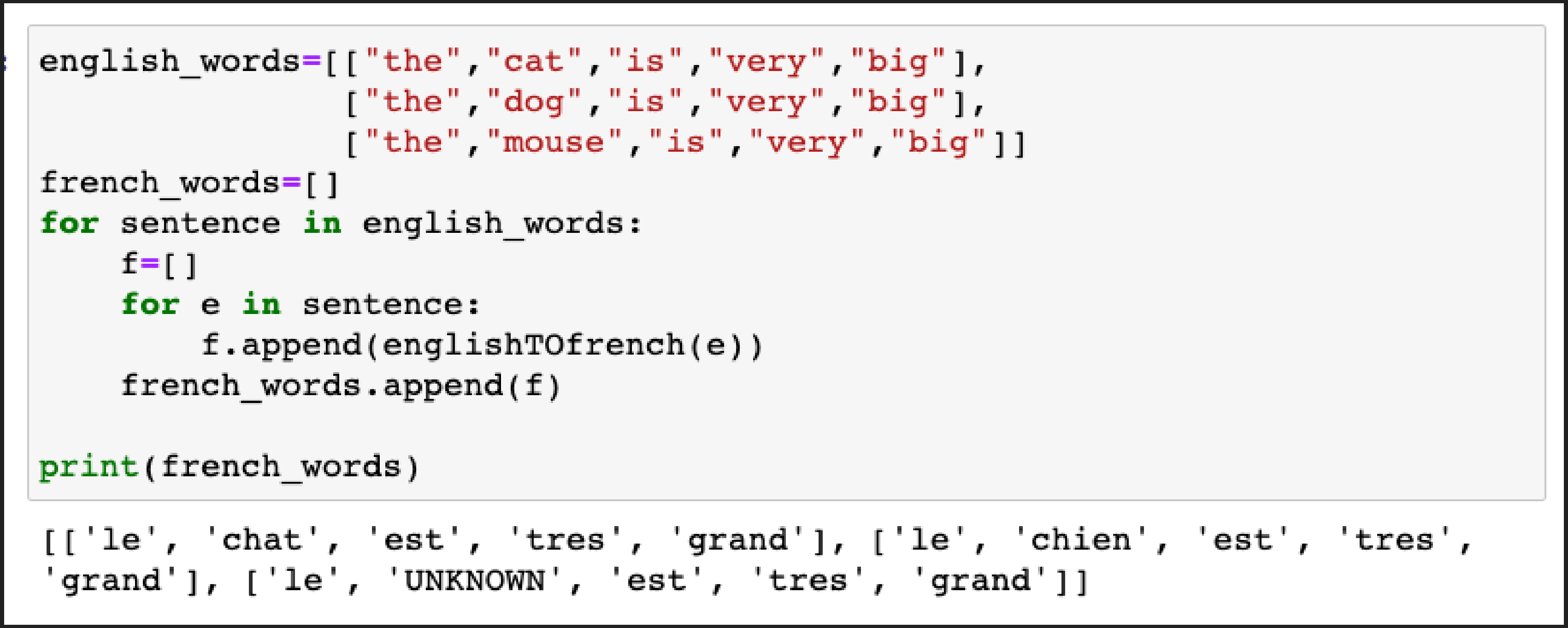
```
['le', 'chien', 'est', 'tres', 'grand']
```

call to function with argument supplied

L1

Nested Structures

- Complex data structures such as lists and dictionaries can be nested
- Arbitrary levels of nesting possible
- We can do nested iterations as follows



```
english_words=[["the","cat","is","very","big"],
               ["the","dog","is","very","big"],
               ["the","mouse","is","very","big"]]
french_words=[]
for sentence in english_words:
    f=[]
    for e in sentence:
        f.append(englishTOfrench(e))
    french_words.append(f)

print(french_words)
```

```
[['le', 'chat', 'est', 'tres', 'grand'], ['le', 'chien', 'est', 'tres',
'grand'], ['le', 'UNKNOWN', 'est', 'tres', 'grand']]
```

Lab work

- Really important to complete all of labs 1_1 and 1_2 this week
- Ask questions in labs if there is anything you don’t understand or want to know more about
- Or on the Canvas forum
- Extension material (lab 1_3) is for those who are already proficient Python programmers … we will come back to this material as required

Keywords Check

list		int	
set		boolean	
function		string	
tuple		kernel	
pair		nested list	
triple		function call	
dictionary		return value	
argument		natural language	
iterate		variation	
float		ambiguity	