

# Codelet 0: Python Refresher

The due date for this codelet is **Monday, Jan 27 at 11:59PM**.

## Introduction

The following reintroduces you to using python via some concrete exercises. At the conclusion of this codelet, you should have demonstrated (re)familiarity with key aspects of Python.

## Outline

- ☐ Grading
- ☐ Doctests
- ☐ Exercises
  - ☐ *if* statements
  - ☐ *for* loops
  - ☐ basic text processing
  - ☐ basic class structure
  - ☐ challenge problem

## Your assignment

Your task is to:

1. Download `codelet0.zip` from the course website and open it. You will find these instructions, some relevant data files, and `codelet0.py` which has scaffolding for you.
2. Complete each of the exercises using the scaffolding provided in `codelet0.py`
3. Provide at least 2 doctest examples for the first two exercises

## Grading

When assessing your code, satisfactory achievement is demonstrated, in part, by:

- Simple and clear code
- Code passes relevant tests
- Provided test cases are appropriate to the function

## Doctests

Doctests are a useful testing module provided with python. The full documentation can be found [here](#). What's relevant is that it allows the specification of test cases in docstrings. Consider the below example,

```
import doctest

def add(a: int, b: int) -> int:
    """ Returns the sum

    Args:
        a (int): First number
        b (int): Second number
    Returns:
        int: Sum of a and b

    >>> add(3, 4)
    7
    >>> add(0, 1)
```

```

1
"""
return a + b

if __name__ == '__main__':
    doctest.testmod(verbose=True)

```

Here, we add to the `add` function, two simple test cases. Notice that the desired solution is under the `>>>` with no extra indentation. If our function outputted the wrong thing, running `doctest.testmod(verbose=True)` would tell us what case failed.

## Exercises

For the exercises not mentioned below, you should find sufficient detail in their docstrings.

### Basic classes

The final exercise is meant to introduce you to python classes, which you may have only seen in Java. To show you how to format a basic class, I've provided an example of a class called `Word` below:

```

class Word:
    """
    A class word manipulating words.

    Attributes:
        vowels (set): (class attribute) A set of all vowels
        text (str): Text of the word
    """

    #A variable shared by all instances of the class
    vowels = set(['a', 'e', 'i', 'o', 'u'])

    def __init__(self, text):
        #self identifies the instance of class and can be used
        #to reference the variables of a specific instance of
        #a class. So for an instance W of Word, W.text will
        #return the text of that W. Within the class methods
        #self.text can be used to pick out the text of the instance.
        self.text = text

    def whereAreTheVowels(self):
        """
        A function that returns the position of the vowels in the word.

        Returns:
            list: A list of the positions of the vowels in the word.

        """
        where = []
        for position, letter in enumerate(self.text):
            #Checks if the letter is contained in the set
            #vowels, if so, we append the position to where
            if letter in Word.vowels:
                where.append(position)

```

```
return where
```

There are some special methods specific to classes. The `__init__` method is one such method, and it serves as the class constructor. Moreover, the arguments, if there are any beyond `self` (we return to `self` in a moment), in this function are required for instantiating the class. In the case of `Word`, that means we need to pass in a variable filling the slot of `text`.

```
t = 'Fig'
w = Word(t)
```

Above, we create an instance of `Word` that has, as its `text` attribute “Fig”. Finally, in creating a class we can create methods which can be called by class instances. In the example of `w`, we can call the function `whereAreTheVowels` by

```
w.whereAreTheVowels()
```

Notice that we don’t pass any arguments in, despite `self` being in the function declaration.

For this exercise, follow the description provided and create a class called `Linguist` with two attributes, `name` and `advisor`, and a function `addUniversity`, which takes one argument and adds it to the class instance (via `self`).

### Challenge problem

As a final refresher on python, `conllu2sents.py` lays out a task of reading in a file and changing the data in some way. To assist you in understanding the goals of the function, example input/output pairs are given. As per the typing hints in the function instantiation, `conllu2sent`, expects the name of a file and returns a list. The `conllu` in `conllu2sent` refers to a file format used for dependency parsing, among other things (see here). It consists of tab separated columns, with each column corresponding to a type of linguistic annotation. For example, the second column holds the form of a word (as it appears in the sentence), and the third column holds the lemma or stem of the word. For example, above you should see the pair “Is” and “be”. Blank lines mark sentence boundaries, and lines beginning with `#` are comments (similarly, `#` are comments in python). Your task is to read in a `conllu` file and output a list containing each sentence in the file. To do this, you’ll want to loop over the lines in the file, paying particular attention to the sentence boundaries and to specific columns in the file. There is one final thing to keep in mind. Namely, the sentence should maintain normal spacing. That is, punctuation should directly follow the relevant word. One of the hints in the file should help.

My recommendation would be to approach this in two parts. The first, try to get the sentences from the file `fr_easy.conllu`. Once you can do that, move onto `fr_hard.conllu` which has punctuation.