DOI 10.20535/2617-5339.2019.3.165687

UDK 81'33

Lesia Ivashkevych
PhD in Philology, Assistant Professor
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
ORCID ID 0000-0001-7166-5331
fliegekunst@gmail.com

TEACHING PROGRAMMING WITH PYTHON FOR LINGUISTICS STUDENTS: WHYS AND HOW-TOS

Abstract. The article shows why it is worth introducing the basics of computational linguistics in general and programming as a method of the natural language processing in particular to the linguistics students. Computational linguistics has become the basis for solving many practical tasks in the language industry. Providing the linguistics students with the basics and the methods of the computational linguistics we widen their views on linguistics and show a perspective field of their possible future engagement to them. To get acquainted with computational linguistics, students have to learn how to work with corpora and acquire the basics of programming. This article demonstrates why Python is a good choice for linguists to start learning to programme. It also suggests an approach to teaching the fundamentals of the programming in Python for such students and gives step by step the main structures which can be used for processing texts or corpora. To such structures belong strings, variables, lists, loops, print-function, split-method, incrementation, and control structure. Combining these elements one can, for instance, split text into sentences or words, count words or sentences in text or count only some concrete elements in the text which satisfy a special condition. The article also outlines how to start working with input files. The further structures of Python are named, which can be introduced to the students next so that they become able to do more operations with texts. It is stressed that teaching programming is impossible without trying out every structure, so it is important to encourage the students to write their own code experimenting with each new element of Python and offer them enough practical tasks. Some examples of such tasks are illustrated in the article.

Keywords: computational linguistics; linguistics students; teaching programming; methods of the computational linguistics; basic structures in Python.

1. INTRODUCTION

1.1. Why computational linguistics should be introduced to the linguistics students

Computer nowadays has changed almost every field of human activity, almost every science, giving people more possibilities in their work. Linguistics is not an exception. Computational linguistics started its development in the 1950s driven by the practical need to create the systems for machine translation (Mitkov, 2009). Since that time it has found the broad implementation in many spheres and nowadays it has become a separate, very vivid part of linguistics.

Among the main practical tasks which are being solved with the help of the means of the computational linguistics are machine translation, systems for automatic question answering, text retrieval on some subject, text summarization, error correction, analysis of texts or spoken language for some topic, sentiment or other psychological aspects, dialogue agents for accomplishing particular tasks (e.g. purchases, trip planning or medical advising), systems for better language acquisition and gaining knowledge from text (Schubert, 2014). It is the computational linguistics that is driving the developments which we conceive to be the "artificial intelligence". Without a doubt, the areas of computational linguistics' implementation will become vaster and vaster in the coming decades.

That is why we consider that there is a need to introduce computational linguistics to the students, who are majoring in linguistics. Their future work could with high probability be connected with one of the implementations of the computational linguistics and by presenting them

the basics of computational linguistics we substantially widen their outlooks.

Moreover, we also regard that computational linguistics itself needs more people coming from traditional linguistics. Following Grishman (1999) from Cambridge University, who writes in his book "Computational Linguistics: An Introduction" that "theoretical linguistics can provide valuable input to computational linguistics, an input which is too often ignored" (p. 7). We argue that traditional linguists coming to the computational linguistics could help it to develop in a more viable, efficient and humanistic way, by taking into consideration concrete linguistic dependencies in all their totality. This can direct the computation linguistics to the creation of instruments which can become better helpers of the human researchers and linguistic experts rather than aiming at the creation of the "perfect systems" that only base on the machine learning and strive to exclude the human participation. Thus, more cooperation with "traditional" linguists could bring fruitful results for the development of the strategies of the computational linguistics.

In addition, we have already seen how computational approaches influence each sector of linguistics nowadays. According to Johnson (2011), this impact will increase in the near future.

1.2. Why it is worth for linguists to learn programming

There are two basic fields that are necessary for linguists to understand the approaches of the computational linguistics. First, there are corpora, which are the basis for many practical implementations of the computational linguistics as, for example, machine translation, errors correction, search in web etc. Students should be taught the principle, how corpora are organized, what types of corpora are there and what preparation do the texts need before they can be included in a corpus. By learning what corpora are, they discover a lot of basic concepts of the computational linguistics like, e.g. segmentation, tokenization, and parsing. Practical search work with corpora, using special corpus managers and also search with regular expressions would be the best preparation for the further programming tasks. In our opinion, programming is the other important basis of computational linguistics. Namely, the programming gives us the possibility to directly dive into the processing of the natural language. Moreover, it allows us not to stay dependent on the possibilities of some concrete available tools but to develop free and flexible approaches to language processing.

1.3. Why we have chosen Python as a programming language for linguistics students

Among other programming languages, we have chosen Python for several reasons. As Panggabean and Tobing (2015) point out in their article "Computational Linguistics Application Using Python Programming" that Python is freely available and simple to learn, which is extremely important for our linguistics students who as a rule are very unsure about themselves when it comes to even think about programming (p. 19-20). Nevertheless, Python allows programming at a high level that means that really complicated tasks can be solved with its help. Besides, Python is widely used in science in general and in computational linguistics particularly, and there are quite a few learning resources that are aimed at linguists, e. g. by Dirk Hovy's ones (Hovy, 2012) or by Johann-Mattis List's (List, 2011) tutorials.

A great advantage for the linguistics students is also, that the syntax of Python is similar to the English and it is easy to understand the already written code (Hovy, 2012, p. 4). A detailed description of Python and its strong sides can be found in the book "Think Python: How to Think like a Computer Scientist" (Downey, 2002).

Having chosen Python, you will be soon able to solve such tasks as splitting text in sentences or splitting sentences in words, counting words, sentences, or finding the average number of words in a sentence across the text, or counting all words that have some special letter or have more than 10 characters, etc.

The purpose of the article is not only to show why computational linguistics should be introduced to the linguistics students, but also to demonstrate that programming is an important part of getting familiar with the approaches of the computational linguistics and to present the simple methodology how programming in Python, the most popular language for processing natural

languages, can be step by step taught to such students.

2. METHODS

The research paradigm was interpretive, which placed emphasis on the analysis of sources as well as research materials with the next synthesis of its results.

3. RESULTS AND DISCUSSION

3.1. How to teach Python to linguists: getting started with Python

There are two ways to start working with Python. One of them is to install Python and a special text editor which suits for Python (like Atom or Sublime) at your machine and run your programs in the shell. While Mac and Linux systems often have Python already installed, it must be done for Windows. Concrete steps on how to install Python are presented in the tutorials of Hovy (2012, p. 4–6) and Gorozhanov (2014, p. 5–7). The second way is to work online in some interactive environment like repl.it, where you can both write your code and run it. We consider the second way to be much easier.

Fig.1. The interacvite working surrounding

Python has several versions. Here we use Python 3.

3.2. First code in Python

Let us firstly discuss the topic of languages in Python. There will be things in your code which you can write with any languages (like German, French or Ukrainian), these are roughly said arbitrary names (strings or variables, see below), which you give to some entities. But there are also commands, which must be only written in English.

Now let us look at the concrete steps how the basics of Python can be introduced to the linguistics students. In many tutorials on Python, it is stressed, that in order to have successful experience with learning this programming language it is vital to immediately try out every new concept. So we highly recommend the teachers to encourage their students to experiment on each step writing their own code or solving simple relevant tasks.

3.2.1. Strings and the "print"-function

A string is any sequence of characters, which can be processed with some commands in Python. For example, a string can be some word, some sentence or text. Strings are marked with quotes, e.g. *"The weather is nice today"*. Strings can be added with a plus sign. For example, if we add two strings *"The weather is nice today. + I really enjoy it"*, we will have as output one string *"The weather is nice today. I really enjoy it"*. Another simple thing, what we can do with strings is multiplying them with *. So, "La"*3 will result in "LaLaLa". In order to see the result of the execution of your code, let us learn one simple function. That is the print function. After you are ready with your code, just put the command print in a new line and put in the parentheses after it the entity, which you would like to see on your screen like in figure 1. Then press the run-button and

look in the right field of the screen at the result of the execution of your code.

After (or even better while) presenting these simple concepts to students it is worth to give them some simple task, e. g. to write their own strings and operating with them adding and multiplying them.

3.2.2. Variables

Variables are some entities, with which we will operate in our code. Different types of content, e.g. strings, numerical variable, lists or dictionaries can be assigned to variables presented with some arbitrary name. It is better, however, to give variables some meaningful and logical name, so the code is understandable even after some time. Variables in Python are very similar to the variables in mathematics. In a figure 1 you can see an example of a variable of the type "string", which holds a sentence. As you can see, the content of a variable is assigned by the equal sign. There are some rules on how to choose a name for a variable so that the program has no problems with. They are presented below:

Variables CAN/SHOULD	Variables CAN'T
SHOULD begin with a letter (from a to z): *sentence, word, text etc.*	begin with a capital letter: ~~*Sentence_1*~~
CAN contain digits: *text_1, list_of_participants_3*	Contain spaces or dashes: ~~*number of words*~~, ~~*word-with-dash*~~
CAN contain underscore to separate the parts: *mini_text, first_sentence*	Consist of technical words, which represent commands like *if, for, print, include*

After having discussed that rules with students, it may be useful to do some small exercise (like given below) which would help the students to check themselves.

Which of the variables are written correctly?

1) *Number_of_charachters* = 0
2) *initial_width* = int(sys.argv[1]) * 4
3) *fist-10-letters* = sorted_items[:10]
4) *set names* = set()
5) *set_surnames* = set()
6) *2nd_sentence* = "We are tired."

The correct answer is in this case: variables under numbers 2 and 5 are written without errors. All the rest contain a mistake.

Then the students might be asked to experiment with their own variables of the type "string" and to print them with the print-function.

3.2.3. Lists

A list is an object, which contains some elements. It can be for example some strings (letters, words, sentences) or numbers. Lists are introduced with square brackets [ ] and can be assigned to some variables. The elements inside of a list are separated with commas. Here are some examples of the lists:

1. list = ["a", "b", "c"].

2. list_of_even_numbers = [2, 4, 6, 8, 10]

3. small_text = ["The weather is fine today.", "I really enjoy it.", "Let us go for a walk."]

Similar to strings, lists can be added, too. The example you can see in figure 2 below.

```
1  list_of_numbers_1 = [1,2]
2  list_of_numbers_2 = [3,4]
3  print(list_of_numbers_1+list_of_numbers_2)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
[1, 2, 3, 4]
```

Fig. 2. Adding two lists

3.2.4. Loops

One more basic structure in Python is a loop. It is a form to organize your code in a way that you can repeat some operation for each element of a list. A loop is represented with the structure "for ... in ..." and looks like this:

```
1  sentence_1 = "How are you?"
2  sentence_2 = "Thanks, I am fine."
3  small_text = [sentence_1, sentence_2]
4  for sentence in small_text:
5      print(sentence)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
How are you?
Thanks, I am fine.
```

Fig. 3. Example of a loop (rows 4 and 5)

As you can see, in a loop you already operate with the previously defined entities (in this case variable of the type "string" (*sentence*) and variable of the type "list" (*small_text*)).

It is important to understand that when you are dealing with some object consisting of several elements of the same type (like the list *small_text* in our case, which consists of two strings (*sentence_1* and *sentence_2*)), for the program it does not play any role how you will name the element of this object, no matter what will be the name, the program will in any case process it in the same way. The reason is that it "sees" the element independent of the name, basing on the structure of the object itself. In our example, the element is called *sentence*, but if we arbitrary change that name (e. g. to *element*), the program will accomplish the same operation:

```
1  sentence_1 = "How are you?"
2  sentence_2 = "Thanks, I am fine."
3  small_text = [sentence_1, sentence_2]
4  for element in small_text:
5      print(element)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
How are you?
Thanks, I am fine.
```

Fig. 4. Elements in the loop are defined by the structure of an object, not by the name we give to them

In the example above we have just printed the elements. But in the same way we can do some

other action with them. For instance:

list_numbers_1 = [1, 2]
list_numbers_2 = [3, 4]
list_together = list_numbers_1+list_numbers_2
for number in list_together:
*print(number*2)*

As you can guess, the results of the code above are numbers 2, 4, 6, 8 printed one per line.

One more important aspect: you have probably noticed that the space in line 5 in the both figures is larger than it was in the code before. In the loops, we use colon and additional spaces to tell the computer that both parts of the program – the line with the colon at the end and the lines with additional space before code – are linked.

3.2.5. Split method

We already know one function. That is the print function. As you have noticed, we always use brackets after it, defining there the argument of the function, namely, what do we want to print. Now we will learn another kind of function – a split method. This method is very useful to work with natural language texts. Methods work very similar to functions, the difference is, that they are added to the object we want to process via dot. We can split strings, e.g. a word, a sentence or a text. Like in the print function, we will use brackets after it to introduce the argument, but here we will define in brackets, by what character we want to split our string:

(1) text.split (".")
(2) sentence.split (" ")

In the example (1) above we want to split the text in sentences by the dot. In the example (2) we want to split the sentence in words by the space. Of course, in real texts dots and spaces are not enough to really get sentences and words, because there are also such punctuation marks like commas, colons, semicolons, question marks, exclamation marks etc., which separate the words and sentences from each other. It is possible to take all that into consideration, but for now let us try out the simplified version of such split.

So, the whole code can look like this:

```
c/Project_Class_2018    run ▶    share
main.py    saved
1  text = "Programming is not that hard as it might seem to be.Everybody can learn the basics of programming.There are already special books in programming even for children."
2  for sentence in text.split("."):
3      print(sentence)
```

```
https://ProjectClass2018.lesiaivashkevyc.repl.run
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Programming is not that hard as it might seem to be
Everybody can learn the basics of programming
There are already special books in programming even for children
```

Fig. 5. Split method helps to devide strings into some elements

You might have noticed two things here. Firslty, the dot in the end of the sentences has disappeared. This is always so: the character, which is used as a separator, is being "eaten" by the program. Secondly, as you can see in the figure 5, it is possible to use the split metod on some object just in the loop instead of creating additional variable. It is very convinient and helps to save time.

Using the split method and the loops, you already can do some interesting things with texts like creating the list of words or the list of sentences in your corpus. More possibilities you can get by learning some other basic structures in Python, which we are going to describe below.

3.2.6. Incrementation

With this structure you can count some elements, f.e. words or sentences, in the text you are processing. To do that, we use a variable which can "grow". Firstly, we create a numeric variable with value 0, then we go through the elements of the string each time increasing that variable.

So, f.e. the code, in which we count the words in the sentence, will look like this:

```
sentence = "Programming is not that hard"
number_words = 0
for word in sentence.split(" "):
  number_words = number_words+1
print(number_words)
```

In a similar way you can count words or sentences in a big corpus and also count the number of words for each sentence separately. With the next structure, introduced in paragraph 2.2.7, you can also count some concrete elements which satisfy your condition.

3.2.7. Control structure

This structure allows you to complete some action not with all elements of your processed text, but only with some elements, which fit the condition you need. That might be, e.g. only all words that contain the letter "z" or the letter combination "th" or which contain more than 10 letters etc. If you are working with an annotated corpus, you can search for some combinations of particular parts of speech. Control structures really essentially widen our possibilities in Python. Here you can see an example of code, with which we count all words, containing the letter "a":

```
text = "Control structures are very important in Python, because they can give us much more possibilities while processing texts"
number_words_a = 0
for word in text.split(" "):
if "a" in word:
number_words_a = number_words_a+1
print (number_words_a)
```

If you add the continuation of the structure with the command *else*, you can do two parallel things with it, e.g. count all words with letter "a" and without it, like in the code below:

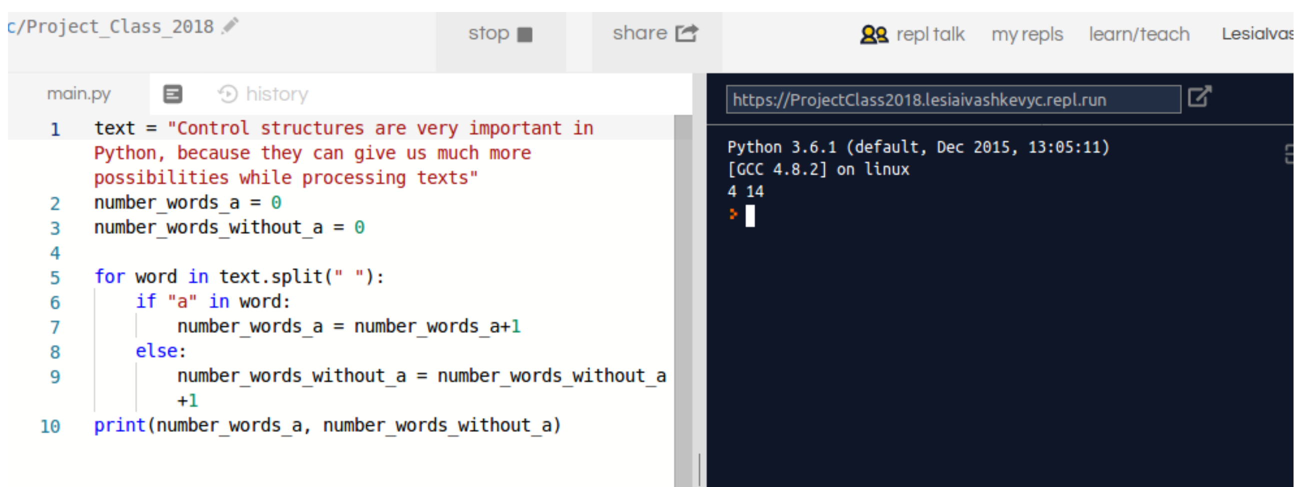

Fig. 6. Control structure

3.2.8. Processing files

You can work with input files by loading them in repl.it and adding them to your code as follows:

(1) with open ("corpus.txt") as text:

(2) text = text.read ( )

In the first line, you open the file for reading; in the second line you read the content of the file into a string. You can do with it anything you do with a regular string.

4. CONCLUSIONS AND SCOPE FOR FURTHER RESEARCH

After students have learned the presented above simple operations in Python, they can already do many interesting and useful actions with texts or with corpus. They can find the average word length in sentences or average sentences length in texts. After learning some further structures like dictionaries and tuples they are able to get the frequency list of the words in texts.

Also, to widen the text processing possibilities of Python, it is very useful to use regular expressions while writing a code. Next important step in learning programming with Python could be writing functions.

All in all, the basics of programming in Python are not difficult to acquire, but they help the linguists to understand, how one can handle the natural language texts with computer and to accomplish many operations which can be used both in linguistic research and in the practical language-processing tasks.

In the context of the problem under study, the scope of further research envisages studing the types of practical tasks of programming in Python.

REFERENCES

Downey, A., Elkner, J., & Meyers, Ch. (2002). *Think Python: How to Think Like a Computer Scientist Learning with Python.* Wellesley, Massachusetts: Green Tea Press.

Gorozhanov, A. (2014). PyQt 5 dlia lingvistov: professional'no orientirovannoie programmirovaniie. Elektronnoe uchebnoe posobiie dlia studentov lingvisticheskih vuzov i fakul'tetov (bakalavriat i magistratura) [PyQt 5 for linguists: professionally oriented programming. Electronical textbook for students of linguistics (bachelors and masters)]. Retrieved from http://pyqtforlinguists.appspot.com/book.pdf [in Russian]

Grishman, R. (1999). *Computational Linguistics: An Introduction.* Cambridge University Press.

Hovy, D. (2012). *Programming in Python for Linguists. A Gentle Introduction.* Retrieved from http://www.dirkhovy.com/portfolio/papers/download/pfl_handout.pdf

Johnson, M. (2011). How relevant is linguistics to computational linguistics? *Linguistic Issues in Language Technology, 6*(7), 1–23.

List, J.-M. (2011). *Python für Linguisten.* Retrieved 22 Feb.2019 from http://lingulist.de/documents/lectures/list-2011-lecture-ss-python-for-linguists.pdf [in German]

Mitkov, R. (2009). *The Oxford Handbook of Computational Linguistics.* Oxford: Oxford University Press.

Panggabean, H., & Tobing, A. (2015). Computational Linguistics Application Using Python Programming. *IOSR Journal of Humanities and Social Science (IOSR-JHSS), 20*(7), 18-30.

Schubert, L. (2014). Computational Linguistics. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. Stanford, CA: Stanford University Press. Retrieved from https://plato.stanford.edu/entries/computational-linguistics/

Леся Івашкевич. Навчання студентів-лінгвістів програмування мовою Python: для чого та яким чином. Статтю присвячено основам навчання майбутніх лінгвістів програмування мовою Python. Розкрито питання про необхідність ознайомлення студентів-лінгвістів з основами комп'ютерної лінгвістики, а саме з корпусною лінгвістикою та програмуванням для обробки природної мови. Зазначено, що комп'ютерна лінгвістика знаходить все ширше застосування як у лінгвістичних дослідженнях, так і в лінгвістичній індустрії, де дозволяє вирішувати багато практичних завдань, серед яких: машинний переклад, створення досконалих пошукових систем, розпізнавання мови тощо. Знайомлячи студентів-лінгвістів з основами комп'ютерної лінгвістики, ми розширюємо їхні горизонти та можливості їхнього працевлаштування. У статті висвітлено, чому при навчанні майбутніх лінгвістів програмування варто зупинитися саме на мові Python. Представлено покроковий підхід до ознайомлення студентів з основними поняттями та структурами Python, які дозволяють опрацьовувати природну мову у вигляді текстів чи корпусів. До таких структур належать стрічки, змінні, списки, цикли, функція *print*, метод *split*, інкрементація та контрольні структури. Поєднуючи ці елементи, лінгвіст може, приміром, розбити величезні масиви текстів або ж корпуси на речення чи слова, порахувати кількість слів чи речень у тексті або ж порахувати лише конкретні елементи, які задовільняють певні умови. Також у статті описано, як почати працювати з зовнішніми файлами, та представлено структури Python для подальшого вивчення, які дозволять розширити можливості обробки природної мови. У статті

підкреслено, що навчання програмування потребує постійного практичного випробування кожного нового елементу, тому важливо заохочувати студентів експериментувати з власним кодом та пропонувати їм вправи, приклади яких подано у статті.

Ключові слова: комп'ютерна лінгвістика; студенти-лінгвісти; навчання програмування; методи комп'ютерної лінгвістики; основні структури мови програмування Python.

Received: April 09, 2019
Accepted: April 25, 2019