# PART 4: NATURAL LANGUAGE PROCESSING PROJECTS

# By Karen Schwarze

You’ve practiced formulating rules, and you’ve learned some Python syntax. Now you’re ready to dig into natural language processing by building three programs where the computer will perform actions with text you provide. Let’s get started!

**PART OF SPEECH CHECKER: ARTICLES**

For our second project, we are going to build a program that detects whether a part of speech is in a sentence the user provides.

*Teachers – Optional Unplugged Activity:* Have students revisit the parts of speech they reviewed at the beginning of the unit. Then pair up students for the following discussion. Have each partner pick one part of speech, explain it to their partner, and provide examples.

*Students:* For this project, we are going to use articles (a, an, the). Strictly speaking, articles are not their own part of speech category; they tend to fit more in the “adjectives” category. But unlike some parts of speech (POS), articles are fairly simple to identify—there are only three of them (*a, an,* *the*) and they don’t alternate POS categories like some other words do. This makes them a good candidate for our Part of Speech Checker Project.

Let’s sketch out some pseudocode to help us see what our program might look like. Our program will detect whether or not there is an article in the sentence (we’ll use *sentence* and *string* interchangeably in this section; *string* is the sentence’s value type) the user enters.

*Teachers***:** You might choose to pair or group students to brainstorm ideas for a program that identifies whether or not an article is in a string.

*Students:* Write out the steps you think we need to take to build our program. Give it a try before you check the answers on the next page.

My pseudocode:

Answers:

Have the program ask the user for a sentence, and make it possible for a user to enter a sentence.

Teach the program what articles are, so it knows what to look for in the sentence.

Have the program look for articles in the sentence.

Have the program tell the user whether or not there is an article in the sentence.

Now that you’ve got some pseudocode, it’s time to figure out how to turn it into skeleton code, which is code that is a little more advanced than pseudocode but not quite real code yet. To do this, we’ll need to apply what we already know about writing code. Even if you’re not sure what the exact code would be, based on what you know so far, take a guess as to what might be included in the code for each of our steps. Try it out yourself before looking at the answers. **Tip:** Check the code we used in our practice programs to see if anything there might be useful.

|  |  |
| --- | --- |
| Pseudocode | Skeleton Code |
| Have the program ask the user for a sentence, and make it possible for a user to enter a sentence. |  |
| Teach the program what articles are, so it knows what to look for in the sentence. |  |
| Have the program look for articles in the sentence. |  |
| Have the program tell the user whether or not there is an article in the sentence. |  |

Answers:

|  |  |
| --- | --- |
| Pseudocode | Skeleton Code |
| Have the program ask the user for a sentence, and make it possible for a user to enter a sentence. | Ask the user for input using the input function. We can have the program say “Please enter a sentence.” We’ll store their response as a value in a variable called user\_sentence. |
| Teach the program what articles are, so it knows what to look for in the sentence. | We’ll assign a list of values, in this case the list of articles. |
| Have the program look for articles in the sentence. | We’ll use an **if statement**, a for loop, and the .find() method to have the program iterate through all the words in the sentence and check them against the list of articles. |
| Have the program tell the user whether or not there is an article in the sentence. | We’ll use an if/else statement and the print function to have the program tell us whether or not there is an article in the sentence. |

Now that we’ve got our skeleton code, it’s time to look at how we’re going to turn it into real code (and learn some new Python concepts!)

Let’s look at that first step. Just as we did in Exercise 2 (the dog age calculator), we’re going to ask the user for input using the input function, like this:

user\_sentence = input(“Please type a sentence. ”)

*We’ve added a space here so that there is a space between the end of the sentence and the user’s input when the program runs.*

Next, we need to teach the program what an article is. We do this by defining another variable, this one called articles, and assigning to it our list of articles (*a, an, the*). Like this:

articles = (“ a ”, “ an ”, “ the ”)

*Extra spaces around each article*

The articles need to have spaces *on both sides* so that the program will understand that we’re looking for that article *by itself.* Otherwise, the program will think we are looking for ALL instances of *a, an,* or the, even if they are part of other words, like *cat*, *anthem*, or *these*. These articles are called **substrings** because they are strings we’re looking for within a larger string (the sentence the user entered).

Note that we’ve put our articles in a **list**, which is another value type.

**Index Numbers and For Loops**

Now that we have our first two lines of code figured out, we need to talk about **index numbers** and **for loops.**

Index Numbers

In Python, every character in a string (including spaces) has an **index number**. But you won’t see these numbers unless you ask your program to do something with them. They’re just always there in the background. For our program, these numbers are going to be important.

Let’s look at a string and see what index numbers it has. Python uses **zero-based indexing**, which means that we start at zero when counting the index numbers. So:

name = ‘My name is Mary’

Here’s what each character in that string would have as its index number (I’ve spaced out the characters in the string so you can better see how they correspond to their respective index numbers):

M y n a m e i s M a r y   
0 1 2 3 4 5 6 7 89 10 11 12 13 14

Do you see how each character in the string, including the spaces, corresponds to a number? By pairing this numbering system with for loops and .find() (more on these in a bit), we can have the program check for articles in the user’s sentence.

***\*One very important note:***

In our case, the number that the program gives us will not be the location of the first *letter* of the article: It’ll be one number *less* than where the first letter actually is. This is because we have embedded extra spaces around each article, and the program interprets that first extra space as being the starting character of the substring it’s looking for. So, our index numbers will correspond to characters in our string like this:

I h a v e a c a t

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Because we embedded extra spaces around *a* when we defined our variable articles, if the user enters a sentence that includes it, the program will think there are two spaces around the word. That’s why *a* would have the index number of 8 in the above example instead of 7.

For Loops

If we want a program to complete the same action multiple times, we don’t have to write the exact same line of code multiple times. Instead, by using just a few lines of code and a loop, we can tell the program to repeat an action.

There are different kinds of loops in Python. For our Part of Speech Checker, we’re going to use a **for loop**. A for loop allows us to **iterate** over items in a list or in a string. To iterate basically just means that each time the program “looks” at a character in the string, it’s going to perform an action, then it’s going to look at the next item and perform the same action, and so on. In our case, we want the program to iterate over, or look at, each character in the sentence that the user enters. We then want the program to figure out whether there is an article in that sentence.

To include a for loop in our program, we’ll need to have a starting point where the program can start looking at each index number in our string and identifying whether that number is the first character of an article from our list. We’ll use t = 0 as our starting place, with 0 as the index number of the first character in our string (remember that the first character in a string always has 0 as its index number). ‘t’ does not stand for anything in itself; it’s just a variable we’re using as an **iterator**. We’ll also add the necessary ; at the end of this line of code.

So far, the code for our article checker program looks like this:

user\_sentence = input(“Please type a sentence. ”)  
articles = (“ a ”, “ an ”, “ the ”)  
t = 0;

Now we need to actually write our for loop. To do this, we’re going to tell the program to check our list of articles against the words in the string. We need to tell our program, “For each item in our articles list, do the following” and then have the rest of the code for our for loop. So, our fourth line of code will look like this:

for article in articles:

Next, we’re going to learn about the .find() method and how it will help us with our program.

The .find() Method

Our program will use the **.find() method** to check whether there is an article in the string. It will make more sense if I show you the code for this first and then explain it:

t = user.sentence.find()

With this code, we are telling the program to apply the .find() method to user.sentence. We do this by adding it to the end of user.sentence, as you see above. Then we need to tell the program what to look for, or what we’re trying to *find,* in user.sentence. What are we trying to find? That’s right—an article. We’ll put article in the parentheses:

t = user.sentence.find(article)

Remember how earlier we said that we needed t as an iterator, and that we would start it at index number 0? Now we’re telling the program to make t cycle through every character in the string (that is, in user.sentence) until it finds the first character of an article (and its corresponding index number).

To recap, here is our code so far:

user\_sentence = input(“Please type a sentence. ”)  
articles = (“ a ”, “ an ”, “ the ”)  
t = 0;

for article in articles:  
 t = user.sentence.find(article)

*Note the indent here; this indent shows where our for loop starts*

If we were to translate these last two lines of code into English, here’s what that translation would be:

“For every article in our list of articles, iterate through the user’s sentence to see if the article is there. If you find an article, you’ll also find the index number of the first character in that article.”

If the program does not find any articles in the string, t will **return**, or be equal to, -1. *The .find() method always returns -1 if it cannot find what it’s looking for.* Think of it this way: Because of the for loop, the program is looking for the articles and cycling through the string’s index numbers in the background. If it cannot find an article (and therefore cannot find an index number that corresponds to the first character of an article), the .find() method will say to us, “Sorry, I can’t find any articles, so I’m going to return -1.” If .find() does find an article, t will return the index number of whatever the first character of that article is. You’ll get to see exactly what that number is when we finish our program.

* Video: You can watch this video for another explanation of how .find() works:

“The Python find() String method.” *Youtube*, uploaded by John Philip Jones, 24 Apr 2015, <https://www.youtube.com/watch?v=NXaO7eiKIXw>.

* Website: Check out this webpage to learn more about the .find() method:

“Python String find() Method.” *W3schools.com*, <https://www.w3schools.com/python/ref_string_find.asp>.

**Relational Operators and If Statements**

Relational Operators

Before we continue writing our code, let’s pause and talk about **relational operators** (also called **comparison operators**). A relational operator is a symbol that tells us about the relationship between the items on either side of it. Here are some examples:

t == -4

== means “equals”, so the above statement would read in English as “t equals -4.”

name != ‘Susan’

!= means “does not equal,” so the above statement would read in English as “name does not equal ‘Susan’”

age <= 32

<= means “less than or equal to,” so the above statement would read in English as “age is less than or equal to 32”

bank\_account >= 1000

>= means ‘greater than or equal to,’ so the above statement would read in English as ‘bank\_account is greater than or equal to 1000’

There are other operators in Python, but for our current project, we’ll just need our == and != operators. Hold onto this information as we look at our next few lines of code.

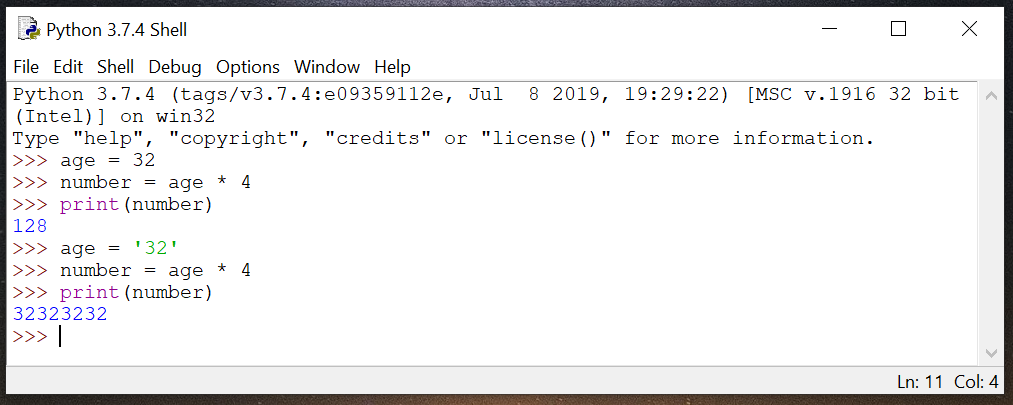
**\*Note:** You’ll notice that the above numbers do not have quotation marks around them. That’s because we’d probably want to treat them as integers, not as strings. To see the difference, type the following into the IDLE Shell (no need to make a new file) and hit Enter after each line:

age = 32 Here, 32 is an integer value type  
number = age \* 4 The symbol \* is the multiplication symbol  
print(number)

After you’ve seen the result, type the following into the shell and hit Enter after each line:

age = ‘32’ Here, 32 is a string value type  
number = age \* 4  
print(number)

Here’s what you should see in your shell:



*Teachers:* Consider requiring students to pair up and explain to each other why they got these results.

If Statements

Remember way back in Part 1 and Part 2 where you identified rules and then turned those rules into conditional statements? You’re going to see how conditional statements work in Python, right now! In Python we call conditional statements **if statements** because they start with the word **if.** Take a look at this if statement, which happens to be the next line of code in our program and lives inside our **for loop**:

if t != -1:  
 break  
   
 *Note the indent*

This **if statement** is saying that if t does not equal -1, the program should break out of the loop. That is, if the program identifies an article and therefore t equals the index number of the first character of that article (an index number which will never be -1), the loop should stop. We just need to find *one* article; we don’t need the program to find all the articles in the sentence, so the first time the loop cycles through and t doesn’t equal -1, we want the loop to stop.

If/Else

We’re almost done with our program! Let’s recap: So far, we’ve declared three variables and assigned them values. Then, we initiated a for loop and used the .find() function and a relational operator to have our program check for articles. Now, we need to tell the program what to print when t != -1 (when it finds an article) and when t == -1 (when it doesn’t find an article).

Let’s use an **if statement** and the print function to assign some output to the result of t == -1.

if t == -1:  
 print(“There is not an article in the sentence.”)

We know that t is -1, but just to confirm this, let’s also tell the program to show us t.   
  
 print(t)

To write our next block of code, we’re going to start with **else**, which means “otherwise.” That is, if t does not equal -1, we want the program to execute the following commands. This time, we want the program to display the words “There is an article in the sentence.”

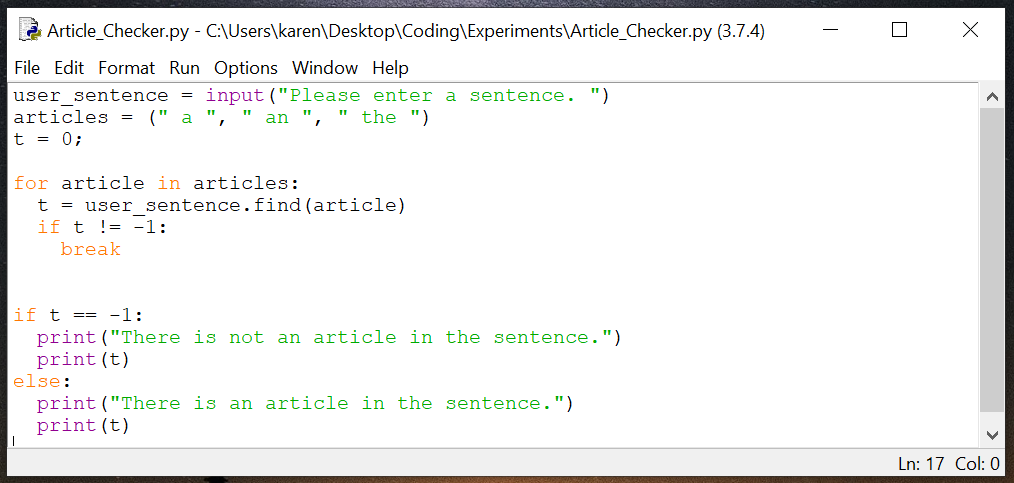
else:  
 print(“There is an article in the sentence.”)

We also want to see index number of the first character of the article.  
 print(t)

This section of code could be translated as:

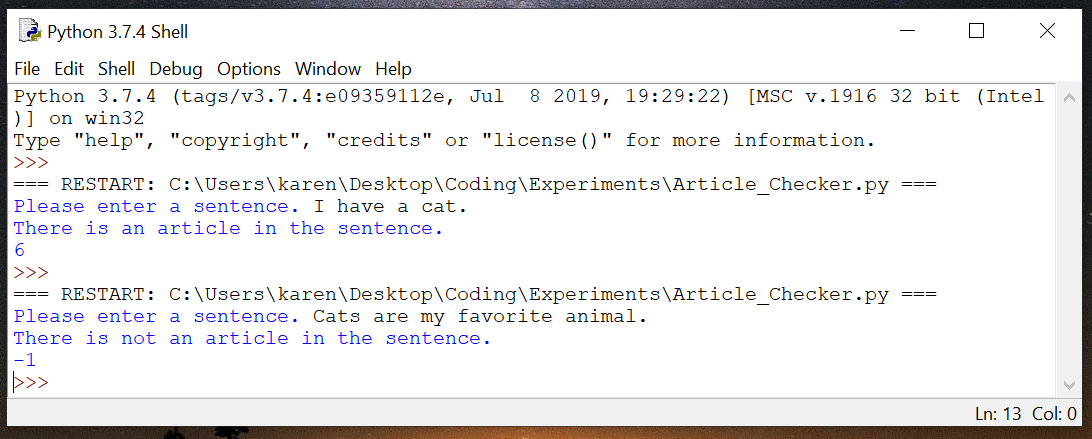
“If t equals -1, display the words ‘There is not an article in the sentence.’ Also, show what number **t** is. Else, display the words ‘There is an article in the sentence’ and show what number **t** is.”

Let’s take a look at what all of our code looks like in IDLE:



Enter your code into IDLE, save it, and the hit Run 🡪 Run Module. You should be prompted to enter a sentence. Type in a sentence and hit Enter. The program should tell you whether or not there is an article in the sentence and what **t** is.

Here are some examples of what your output could look like. I ran my program twice—the first time I entered a sentence with an article; the second time I entered a sentence without an article.



**Project Extension:** You may have noticed that there is a flaw in our program. What if the user starts the sentence with an article? We haven’t accounted for that. How would you go about making sure that the program can also check for articles at the beginning of a sentence?