**Introduction to Natural Language Processing: Curriculum for the ELA Classroom**

**(Student Version)**

**By Karen Schwarze**

**Introduction**

Welcome! You’re probably already familiar with the basic elements of an English sentence, like parts of speech and sentence structure (often called syntax). In this unit, you’ll think more deeply about how a sentence is structured. You’ll consider the **rules** that govern things like parts of speech and similes. Using those rules, you’ll use Python (a programming language) to build programs that identify words and generate literary devices.

**Why are we doing this?**

* Understanding how an English sentence is structured can help you write better. This is helpful when you’re writing an essay for school, an article in a newspaper, a business plan, or a manual about your newest invention.
* **Natural Language Processing** (NLP) is an area of computer science. It is a subfield within **Machine Learning** (ML), which is a subset within **Artificial Intelligence** (AI). All of these areas of computer science help us teach computers how to mimic human behavior. NLP is the study of how to help computers understand human language, whether written or spoken. If you have ever told Google to search for something, you have used NLP.

Many companies, like Google, Apple, and Amazon, are using NLP to build products. Think of things like Google Search, Siri, and Alexa, or text-checking software like Microsoft Word and Grammarly. By getting a basic introduction to NLP in your English class, you’ll be able to choose whether it’s something you would like to do in your career.

To effectively make NLP programs that respond to written or verbal English, you need to have a good understanding of how the English language works. This is why your English class is a great place to learn these concepts.

* In this unit you’ll practice skills like analyzing information, recognizing patterns, and making conclusions based on your findings. These skills can be very helpful in other school assignments and in job-related tasks.

**How to Use This Curriculum**

You’ll notice that most of the instructions are addressed to you the student, but some are for your teacher. When you see *Teachers:* , what follows is something for your teacher to think about. The list of resources at the beginning and end of this curriculum, the names of books and videos, and the citations throughout that look like this: (Good 44-45), are mostly so that your teacher knows where I got the information. Your teacher may use those sources in class or assign them to you for classwork or homework.

All the information you need to finish the activities is either here in this curriculum or linked to, if it’s an online source. Ask your teacher about timelines and due dates.

**How This Curriculum Is Organized**

This unit is divided into five parts. In Part 1: The English Sentence, you will:

* Review basic English sentence elements (like parts of speech)
* Discover and articulate generalizations, or rules, about sentence elements

In Part 2: Literary Devices, you will:

* Review definitions and examples of common literary devices
* Discover and articulate generalizations, or rules, about literary devices

In Part 3: Introduction to Python, you will:

* Learn basic elements of Python, a programming language often used in NLP programs

In Part 4: Natural Language Processing Projects, you will:

* Bring together your knowledge of English language structure and Python to make the following programs:
* Simile Generator
* Part of Speech Checker: Articles
* Student Choice Project

*Teachers:* Consider showing your students the following videos that further explain ML and NLP.

1. Hamedani, Mosh. “Python Tutorial for Beginners [Full Course] Learn Python for Web Development.” *Youtube*, uploaded by Programming with Mosh, 18 Feb 2019, <https://youtu.be/_uQrJ0TkZlc?t=15066>.

Start the video at 4:11:06. You should see a heading in the video that says “What is Machine Learning?” Stop the video at 4:12:59.

1. “Natural Language Processing: Crash Course Computer Science #36.”

*Youtube*, uploaded by CrashCourse, 22 Nov 2017, <https://www.youtube.com/watch?v=fOvTtapxa9c>*.*

**Key Terms:**

* Rules
* Natural Language Processing
* Artificial Intelligence
* Machine Learning

# PART 1: THE ENGLISH SENTENCE

A **sentence** is a group of words that conveys meaning. But there are different types of groups of words, and they are not all created equal. A sentence contains a **subject** and a **predicate**. A sentence is simply a **clause** with end punctuation, like a period, exclamation mark, or question mark. A clause is a group of words with a conjugated verb; a **phrase** is a group of words that does not have a conjugated verb.

A predicate includes the main verb and everything else that is not the subject. So, this is a sentence:

I slept.

[Subject] [predicate] (just a single verb—a past participle—in this case)

And so is this:

George likes to eat chocolate ice cream.

[Subject] [predicate] (in this case, a verb plus some other words)

To use C. Edward Good’s definition in *Who’s (Oops!) Whose Grammar Book Is This Anyway?* :

“The subject consists of a noun or pronoun (or a group of words acting like a noun) and other words modifying, or associated with, the noun or pronoun” (pg. 1). And the subject “stands as the main beacon in the sentence. Your reader’s attention should focus on that subject…Together the subject and the verb should form the main message of your sentence” (pg. 14).

To further borrow from Good:

“The predicate includes the verb and other words modifying, or associated with, the verb” (pg. 2).

* Article: “Main Verbs: Definition and Examples.” *Grammarly,* <https://www.grammarly.com/blog/main-verb/>.

Exercise 1. Let’s identify the subjects and predicates in these sentences:

1. The kitchen smelled like lavender and freshly made bread.
2. My laptop is 15 inches long.
3. Sarah’s books fell off the table.
4. A storm swept through the town.
5. His jacket is blue.

Notice that in each case, the predicate consists of a conjugated verb followed by other words. Of course, not every sentence follows this structure. For example, poets might like to put the verb at the end of the sentence, like this:

*Her heart he swiftly broke.*

A sentence like this is interesting poetically, but generally when writing something like a persuasive essay for school, it’s better to put our verbs upfront so that the reader doesn’t have to look for them.

Note that the predicate can include other things besides the main verb, such as any of the **parts of speech**, which are: verbs, adverbs, nouns, pronouns, adjectives, prepositions, conjunctions, and interjections.

**Verbs**

Action Verbs

Let’s look at our action verbs first. You may have heard of action verbs like run, jump, think, love, believe. These are verbs that describe an action that something or someone is taking. They can be **transitive**, meaning that the verb has an object, or **intransitive**, meaning that the verb doesn’t have any objects. For example:

*I ate*. – Intransitive: There is a subject and a verb, but no object; we don’t know what the subject *I* is eating.

*I ate pizza.* – Transitive: There is an object; we know what the subject *I* is eating (pizza).

A verb is transitive if the subject is acting on something or someone. That something or someone is a **direct object**. A direct object answers *What?* or *Who?.* In addition to direct objects, there are also **indirect objects**. Indirect objects answer *To whom?* or *For whom?* Consider this example:

*My mom gave me a present.*

Here, *me* is the indirect object because it answers the question *To Whom?.* The direct object is *a present*. The verb *gave* is transitive because it has objects: The indirect object *me* and the direct object *present*.

Exercise 2. Let’s practice identifying whether verbs are intransitive/transitive and whether they have direct or indirect objects. Write an *I* next to the sentence if the verb is intransitive. Write a *T* next to the sentence if the verb is transitive. Circle any indirect objects, and underline any direct objects.

1. China ate Turkey fried in Greece.
2. Elizabeth grinned.
3. Our teacher assigned us homework.
4. Mr. Cortez bought flowers.
5. The cat pounced.

Remember: A transitive verb does not need a preposition to attach to a noun, but intransitive verbs do. Only transitive verbs can have direct objects.

For example: I can say “I walked my dog” (*walked* is transitive and *my dog* is the direct object) but I have to say “I walked *to* the park” (*walked* is intransitive because the preposition *to* follows it). I can’t say “I walked park,” so *walked* in that sentence is intransitive.

To be

Conjugations of the verb *to be* refers to existence. For example: *I am from Columbia. The dog is brown. They are going to Cuba. She isn’t running for president. He was upset. They weren’t hungry, but now they want dinner.*

Exercise 3. Write your own examples here, with any subjects you choose:

Present tense:

Past tense:

Linking Verbs

To borrow from Mark Peters in *Grammar and Style*, linking verbs “express a state or condition, connecting the subject of a sentence with a subject complement—a noun, pronoun, or adjective in the form of a single word or a phrase that identifies or describes the subject” (pg. 37).

Let’s look at some examples, also from Peters (pg. 37):

1)You seem terribly curious.

Subject Linking verb Adjective phrase (subject complement)

2)Cecie is the girl with the crying-heart tattoo.

Subject Linking verb (*to be* conjugation) Noun phrase (subject complement)

3)Ted isn’t himself when he has a migraine headache.

Subject Linking verb Reflexive pronoun

Common linking verbs include all forms of *be* *(am, are, be, is, been, being, was, were)* and other verbs such as *appear, become, feel, seem, smell, sound, taste, look* (Peters pgs. 37-38).

Helping Verbs (Also Called Auxiliary Verbs)

From the venerable C. Edward Good: “The main verbs we use in the English language break down into the four major verb types: (1) action transitive verbs, (2) action intransitive verbs, (3) the verb *to be*, and (4) linking verbs. When we conjugate these verbs into the various tenses, we’ll need some help from the fifth category of verbs: (5) helping verbs” (pg. 43).

Here are the helping verbs: conjugations of *be (am, are, is, was, were)* to form progressive tenses and passive voice; *do (do, does)*, and *have (have, has, had, will have)* to form perfect tenses; *can, dare, may, might, must, need, ought (to), shall, should, used (to), would* (to express a condition), *could, will* (to form the future tense) (Good pgs. 44-45)*.* Consider these examples:

1. Nancy is preparing to be a firefighter.

*is* is the third-person singular, present tense conjugation of *to be* and *preparing* is a present participle. Pairing them gives us the present progressive tense.

1. The homework was completed by me.

*was* is the third-person singular, present tense conjugation of *to be* and *completed* is the past participle. Pairing them together and inverting the subject (i.e. adding it to the end of the sentence, rather than at the beginning) gives us passive voice.

1. Do you like salted caramel ice cream?
2. No, he does not like that kind of ice cream.
3. They will have left home by then.

*will* + the third-person conjugation of *have* (which is *have*), paired with the past participle *left*, gives us the future perfect tense.

1. Drew can read a whole book in one day!
2. I ought to clean the house before I go out for lunch.
3. We used to go to the beach, but then we saw a shark and we haven’t been back since!
4. Olaf will melt if he doesn’t get out of the sun.
5. Claire would go to prom if Josh asked her.

Exercise 4. Write your own examples!

11)

12)

13)

14)

15)

Infinitive Verbs

The infinitive of a verb is its unconjugated form. When a verb is conjugated, we know who is doing it and we know when. Consider: “I went to the park.” We know who went to the park (I did) and we know when (sometime in the past). Or: “They like classical music.” We know who likes classical music (they do) and we know when (right now, in the present).

With an unconjugated verb, we don’t know who is doing it or when. Consider these examples (infinitive verbs are italicized):

*To participate* in the contest, you must be at least 18 years of age or older.

Do we know who is participating in the contest? No. We know who is allowed to participate, but the infinitive verb doesn’t tell us who is, was, or will be actually participating in the contest.

Delila wants *to run* for class president and will give a speech at the assembly.

The infinitive verb *to run* doesn’t have anyone or any time attached to it. We could change the subject and main verb, and even the tense, and the infinitive wouldn’t change: *I want to run…They want to run…He wanted to run…* See? The infinitive is not conjugated, and it doesn’t change regardless of who the actor is or the tense of the preceding verb.

**Nouns**

A noun is a person, place, or thing. It can perform action and receive action. It can be the subject in a sentence. Generally, nouns can be grouped into four categories: **Common nouns** (nouns that don’t need to be capitalized—*blanket, salad, bike, didgeridoo*); **proper nouns** (names of specific people, places, and things—*the Eiffel Tower, Bilbo Baggins,* The Chosen); **compound nouns** (when one or more words are needed to represent something, like *sister-in-law, basketball, rain jacket*); and **collective nouns** (*herd, company,* or *flock*). Additionally, while most nouns can be made plural by adding -es, -s, or perhaps adjusting the word a bit and adding, -ies, there are others that need more modification. Take for instance *children* and *people.* (*Persons* is technically correct, but today we generally prefer *people* as the plural form).

Exercise 5. Circle the nouns in the sentences below. Identify each as a common noun (COM), proper noun (P), compound noun (COMP), or collective noun (COL).

1. Snow is my enemy.
2. Felicia met her fans at the concert.
3. Jose and I argued.
4. Mrs. Park brought her students pizza.
5. The students went to the dance.

There is one kind of noun to look out for—a gerund, which is an -ing verb (present participle) dressed like a noun. It’s usually paired with other words to form a gerund phrase. In these examples, the gerund is underlined and the entire gerund phrase is italicized:

*Going to parties* is my favorite pastime*.*

Our *leaving the play early* was perceived as rude.

Doesmy *playing piano late at night* bother you?

* Article: “What is a Gerund Phrase?” *Grammarly*, <https://www.grammarly.com/blog/gerund-phrase/>.
* Article: Simmons, Robin L. “The Gerund Phrase.” <https://www.chompchomp.com/terms/gerundphrase.htm>.

Exercise 6. Write five sentences containing a gerund phrase.



We can use nouns to modify other nouns, like in *Army base, pencil sharpener,* or *baking sheet*. The respective describing nouns (*Army, pencil,* and *baking*) take on the role of adjectives.

**Adjectives**

Adjectives describe or modify nouns or pronouns. Many are a single word and can come before or after the word they modify, like this (adjectives italicized):

The *red* wagon  
The *happy* child   
The computer is *old* (form of *to be* needed)  
The heir *apparent* (no form of *to be* needed)

Often, adjectives are more than one word and need a hyphen: *heavy-handed, dim-witted, light-footed, time-worn, ribbon-bedecked, sky-high.*

Adjectives can also be **comparative** or **superlative**, as in *hotter, meaner, kinder, larger* (comparative) and *hottest, meanest, kindest, and largest* (superlative).

Articles

*A, an,* and *the*—articles—are not exactly adjectives, but they do describe words, and so in that way they somewhat behave like adjectives. Remember that *a* comes before singular words that start with a consonant, as in *a beautiful day* or *she was a nice girl*. *An* comes before singular words that start with a vowel, as in *an extraordinary occurrence* or *an apple.*

*A* and *An* are called indefinite articles. Use these when you’re not referring to a particular noun. When your friend offers you a snack, you would say “I’d like an apple, please,” if there are multiple apples on the table and you don’t really care which one she gives you. But if there is only one apple on the table, or if there is a specific one you’d like, then you need to use *the*: “I’d like the green apple, please” (as opposed to all the other apples on the table, which are red).

Other types of words can also function as adjectives in a sentence. Consider for example demonstrative pronouns (*this, that, these, those*), possessive pronouns (*my, his, her, their,* etc.), and quantifying words like *many, much,* and *some* (Good pg. 98).

I’d like *that* green apple, please.  
*His* train was five minutes late.  
*Many* people shop at this store.

Exercise 7. Circle all the adjectives and articles in these sentences and underline the words they modify.

1. Lee waited for the tall man with the dark moustache.
2. Xander’s cell phone cover was colorful.
3. The brown dogs were tired after their long run.
4. Alejandra’s Jeep is green.
5. His favorite subject is music.

**Adverbs**

Adverbs modify verbs, adjectives, other adverbs, and can even modify an entire sentence or clause. They often help answer the following questions: *Where, Why, How or In What Way, When,* and *Under What Circumstances.* Many adverbs end in *-ly,* but there are those that do not.

Here are some examples. The adverbs are italicized, and the verb they’re modifying is underlined.

She *quickly* ran across the field.  
He thought *tirelessly* about how to solve the problem.  
I wear that shirt *often*.

* Article: “100 Adverbs.” *Yourdictionary.com,* <https://www.yourdictionary.com/index.php/pdf/articles/140.100adverbs.pdf>

Exercise 8. Circle all the adverbs in these sentences and underline the words they modify. (Sentences taken from Good’s chapter on adverbs, pages 104-5).  
 a. Emphatically, the mother denied the child’s request to ride without a seatbelt.  
 b. The teacher sometimes uses the dictionary.  
 c. I entirely understand the rules.   
 d. The manager will probably review the salary scales next month.  
 e. The policy will have become firmly entrenched in our tax law.

**Prepositions**

Prepositions describe relationships between nouns and other nouns, time, and space. Common prepositions are: *at, in, on, to, under, between, after, before*. Here are some examples:

He put the cup *on* the table.  
She felt better *after* a good night’s rest.  
The clue is hidden *between* the two books on the shelf.

Exercise 9. Circle all the prepositions in the sentences below.

1. The business owner ambled toward the post office.
2. Abhi gave presents to his friends.
3. Panda bears live at the zoo.
4. Allow the quiche to cool before serving it.
5. Boxes are stacked on the shelves.

**Pronouns**

We can use pronouns when we don’t want to keep repeating the original noun over and over. For example, take this sentence:

*I’m going to the store with Michael, Phyllis, and Creed.*

If I’m telling a story about what happened when we were at the store, I don’t have to keep saying things like “I was at the store with Michael, Phyllis, and Creed, when…” or “Michael, Phyllis, and Creed and I decided…” Instead, I can use a pronoun, in this case “We.” The word or words a pronoun replaces is called the **antecedent**. (FYI: *Ante-* means *before* in some words, like *antebellum* [before the Civil War] and **antediluvian** [before the biblical Flood].)

Exercise 10. Circle the pronouns in the sentences below and underline the word or words they replace (i.e. the antecedent).

a. Mark is my friend. He’s going with me to the party.  
b. Steve, Dante, and Rebecca are my cousins. They love Disneyland, like me.  
c. The cat has a brown tail. Its paws are white.  
d. Nancy is dating Peter. She’s going to meet him at the dance.  
e. My car and my dad’s car are both in the shop. They both need to be repaired.

**Conjunctions**

Coordinating conjunctions include: *for, and, nor, but, or, yet, so.* A good acronym to remember them is FANBOYS. There are also correlative conjunctions: *either/or, neither/nor, not/but, not only/but also, both/and.* And there are subordinating conjunctions, such as *after, although, how, if, in order that, as though, because,* and more.

Exercise 11. Circle the conjunctions in the sentences below.

1. I wanted to go, but I didn’t end up going.
2. My teacher wears striped shirts and cargo pants.
3. Use your laptop or cell phone to complete the assignment.
4. His words were not those of hate but of welcome.
5. Either they don’t like movies, or they had something else to do that night.

**Interjections**

Interjections are words that exclaim, such as

Pow!   
Oh, no!   
Rats.  
Hooray!  
Yes!

Exercise 12. Write interjections for each of these sentences.   
a. \_\_\_\_ I’ve just won the lottery!  
b. \_\_\_\_ Did you really flunk the exam?  
c. \_\_\_\_ That injury doesn’t look good.  
d. You can’t remember the password to unlocking this treasure chest of diamonds? \_\_\_  
e. The thing I’ve wanted most in all the world is finally on sale! \_\_\_

Exercise 13. In the space below, write a 5-sentence paragraph about a favorite hobby using the 8 parts of speech—verbs, nouns, adjectives, adverbs, prepositions, pronouns, conjunctions, interjections. Make sure you include each part of speech at least once. You’ll probably need several verbs in your paragraph, but you might only want to use one interjection. Circle and mark each part of speech each time you use it. For example, use N for nouns, V, for verbs. Extra kudos if you use additional terms you learned in Part 1, such as *direct object*, *transitive verb*, etc.

**English Usage Rules**

Exercise 14. You’ve just reviewed the basic structure of the English sentence and the 8 basic parts of speech. Now, come up with 10 rules about parts of speech or the English sentence, based on what you’ve learned. Here are two examples:

A sentence consists of a subject and a predicate.

A transitive verb can have direct objects and indirect objects.

Your 10 English Usage Rules

1.  
  
 2.  
  
 3.  
  
 4.  
  
 5.  
  
 6.  
  
 7.  
  
 8.  
  
 9.  
  
 10.

**Turning English Usage Rules into Conditional Statements**

Now you have a list of 10 rules, or generalizations, about how the English sentence works. You’ll use these rules to help you with two Python projects—the part of speech detector and your student choice project.

As we start preparing to write Python programs using rules of the English language, we’ll need to translate those rules into **conditional statements**. For the rest of this section, we’ll discuss what a conditional statement is and how to turn a rule into a conditional statement.

A conditional statement is a sentence that, at its most basic level, includes two parts: The condition (*If x happens…)* and what happens if the condition is met. A condition is something that has to be true in order to achieve the result described in the rest of the sentence. A conditional statement usually has the word *if*, like this:

*If* it rains today, I will wear a rain jacket.

A conditional statement is sometimes called an *if-then statement*, because the word *then* can be used:

If it rains today, then I will wear a rain jacket.

Now that you know what a conditional statement is, it’s time to turn some of our English Usage Rules into conditional statements so that you’ll be prepared for your Python projects. You may have already written some of your English Usage Rules as conditional statements. If so, that’s great! If not, don’t worry; we’re going to learn how to turn those rules into conditional statements now.

For example, let’s think about the rule that determines whether a group of words is a clause.

Rule: A group of words with a conjugated verb is a clause.

Now, ask yourself: How would I write that as a conditional statement? Take a moment to think about it before reading on to the answer. You’ll need to add the word *if*, and you’ll need to state a condition and a result that’s achieved when the condition is met.

Answer: If a group of words has a conjugated verb, it’s a clause.

Notice that in the example above, we have the word if + the condition (*a group of words has a conjugated verb*) + the result that’s achieved if the condition is true (*it’s a clause*).

Exercise 16. Using your rules, write 10 conditional statements.

1.  
  
 2.  
  
 3.  
  
 4.  
  
 5.  
  
 6.  
  
 7.  
  
 8.  
  
 9.  
  
 10.

Hold onto these rules, because you’ll use them when it’s time to make your Student Choice Project.

**Key Terms**

* Sentence
* Subject
* Predicate
* Clause
* Phrase
* Part of speech
* Transitive
* Intransitive
* Indirect object
* Direct object
* Action verb
* Linking verb
* Helping/auxiliary verb
* Infinitive verb
* Noun
* Adverb
* Adjective
* Pronoun
* Preposition
* Conjunction
* Interjection
* Conditional statement

# 

# PART 2: LITERARY DEVICES

In your English classes at school, you’ve probably learned about literary devices. **Literary devices**, or figurative language, can help someone better understand and remember our meaning. Which of the following is more memorable: “My girlfriend is nice” or “My girlfriend is like the stars: Even when life is as dark as night, she is there to brighten it”? In the second example, we have two similes (“My girlfriend is like the stars” and “when life is as dark as night”). These instances of figurative language make the message more interesting because they bring to mind visual images, like stars and a dark night.

In English, we have similes, metaphors, allegories, and many more literary devices. These kinds of devices are sometimes called *figurative imagery* or *figurative language* because the words paint pictures, or figures, in our minds that get the writer’s meaning across in a more powerful way. You’ll find them in poetry, fiction, and non-fiction. A literary device can be something as simple as a one-sentence metaphor, such as “You ain’t nothing but a hound dog,” or as complex as an allegorical play that symbolically represents historical events (such as *The Crucible* by Arthur Miller).

There are other literary devices that have to do with the way that words sound. Take for instance *alliteration*, the repetition of sounds at the beginning of several words in close proximity. Many tongue twisters use alliteration: Sally sells sea shells by the sea shore.

Let’s review some common literary devices in the English language. Then, just as we did after reviewing the parts of speech and elements of the English sentence, we’ll identify rules that can be transformed into conditional statements.

* Source for the following definition: “Literary Devices.” *Literary Devices: Literary Devices, Terms, and Elements,* <http://www.literarydevices.com/>.

**Allegory:** An allegory is a work of art, such as a story or painting, in which the characters, images, and/or events act as symbols.

**Metaphor:** A metaphor compares two subjects without the use of “like” or “as.” Metaphor is often confused with [simile](http://www.literarydevices.com/simile/), which compares two subjects by connecting them with “like” or “as” (for example: “She’s fit as a fiddle”). While a simile states that one thing is like another, a metaphor asserts that one thing ***is*** the other, or is a substitute for the other thing.

**Simile:** Simile is an explicit [comparison](http://www.literarydevices.com/comparison/) between two things through the use of connecting words, usually “like” or “as.”

**Motif:** A motif is a [narrative](http://www.literarydevices.com/narrative/) element with symbolic meaning that repeats throughout a work of literature. Motifs may come in the form of reoccurring [imagery](http://www.literarydevices.com/imagery/), language, structure, or contrasts.

**Theme:** As a literary device, theme is the central topic or idea explored in a text. Usually the theme of a work of literature can be stated in one word, such as “love” or “solitude.” A work of literature can, and often does, have more than one theme. The theme is expressed through the characters’ actions, words, and thoughts.

**Alliteration:** Alliteration is the [repetition](http://www.literarydevices.com/repetition/) of the same consonant sounds at the beginning of words that are in close proximity to each other.

**Dialogue:** Dialogue is a conversation between two or more people in a work of literature.

**Personification:** As a literary device, personification is the projection of characteristics that normally belong only to humans onto inanimate objects, animals, deities, or forces of nature. These characteristics can include verbs of actions that only humans do or adjectives that describe a human condition. The characteristics can also be emotions, feelings, or motives given to objects incapable of thought. For example, if someone said, “the trees whispered their discontent,” this would personify the trees both as able to whisper and of feeling unhappy.

**Metonymy:** Metonymy is a figure of speech in which something is called by a new name that is related in meaning to the original thing or concept. For example, it’s common practice to refer to celebrity life and culture in the United States as “Hollywood,” as in “Hollywood is obsessed with this new diet.” The meaning of this statement is not that the place itself has any obsession, of course, but instead refers to the celebrities and wannabe celebrities who reside there.

Exercise 1. Write 10 examples of literary devices. These could include examples of the literary devices listed above, or other devices (for example, ‘hyperbole’ is a common device not included in the list above). Think of your own examples or find them in books you’re reading for school.

1.  
  
 2.  
  
 3.  
  
 4.  
  
 5.  
  
 6.  
  
 7.  
  
 8.  
  
 9.  
  
 10.

Now that you’ve reviewed some literary devices, let’s write rules for them. Here are some questions to guide you as you write your rules: How do we identify a given literary device? What does it do? What specific words, if any, are usually or always present? What types of words are usually or always present?

Examples:

A simile always has ‘like’ or ‘as’.

A metaphor compares two things.

Exercise 2. Write 10 rules based on the literary devices you’ve studied. It’s ok if you have more than one rule for a device, as long as you have 10 distinct rules total. Write the corresponding literary device next to each rule in parentheses.

1.  
  
 2.  
  
 3.  
  
 4.  
  
 5.  
  
 6.  
  
 7.  
  
 8.  
  
 9.  
  
 10.

**Turning Literary Devices Rules Into Conditional Statements**

Now that you have some rules for literary devices, we want to turn them into conditional statements, just as we did with your sentence rules.

Exercise 3. Write 10 conditional statements based on the 10 rules you came up with in the previous exercise. Remember that a conditional statement always includes the word *if* and consists of a condition and a result that is achieved if the condition is true.

1.  
  
 2.  
  
 3.  
  
 4.  
  
 5.  
  
 6.  
  
 7.  
  
 8.  
  
 9.  
  
 10.

**Key Terms**

* Literary devices
* Allegory
* Metaphor
* Simile
* Metonymy
* Personification
* Motif
* Theme
* Dialogue
* Assonance
* Alliteration

# PART 3: INTRODUCTION TO PYTHON

Now you’re ready to learn how to use Python, a programming language often used in NLP programs.

**Syntax**

*Syntax* means *language structure.* Take the English sentence. As we’ve learned, there are rules that govern the English sentence. There are things you can and cannot do if you want to write a proper English sentence. For example, what if you encountered a group of words like this:

*Jim blanket turtle computer Tahiti built run;*

Hm. We’re not quite sure what to do with this. We have a subject, it appears: *Jim*. And there are two past tense verbs: *built* and *run*. But they are nowhere near the subject, so we can’t be certain they belong with Jim. And what are we to do with *blanket turtle computer Tahiti*? Notice that all four of those words are nouns, and while we can use nouns to modify other nouns, like in *Army base* or *pencil sharpener*, which nouns are acting as adjectives, and which are acting as the noun being modified? At the end of this group of words we have a semi-colon, but there are no words after it; they seem to have fled the scene.

The point is, we can’t just put words and punctuation wherever we feel like it when writing in English. We need to be aware of the relationships between words and punctuation, and place them accordingly.

Similarly, when writing in Python, we need to abide by the rules that govern the elements of Python syntax.

**How to Install Python**

We’re going to look at some of the basic rules when writing Python and write a few programs to get the hang of these rules. First, you’ll need to install Python on your computer. You can read the following instructions on your own, but **make sure to follow your teacher’s instructions before installing anything on a school computer or other device!**

*Teachers:* Consider different options for ensuring all students have access to a device with Python installed on it. Ask your school’s system administrator if Python can be installed on computers in your computer lab or on individual devices, if your students are one-to-one. If students have their own laptops that they can bring to school, that would be the ideal option.

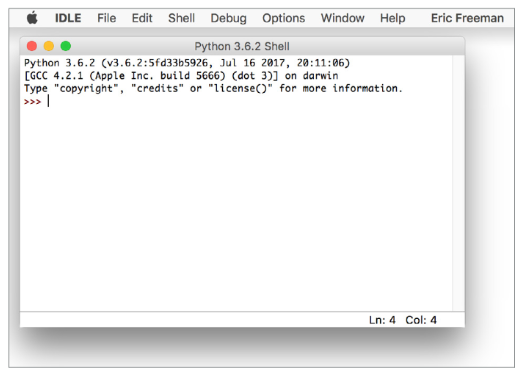
These instructions and accompanying graphics are taken from Eric Freeman’s *Head First Learn to Code: A Brain-Friendly Guide*, page xxxiv-xxxv. This is a good resource for getting started with Python, including a great introduction on how to think like a computer. I will give the disclaimer, however, that there are some minor errors in the text here and there.

You need to install Python version 3.7 or later. The process differs slightly depending on your operating system. An operating system (OS) is the software that manages your device. Most PCs have the Windows Operating System, while anything that is Apple brand has the Mac Operating System (macOS). There is another common operating system called Linux that you can choose to install on your machine. Otherwise, your computer uses the default OS that it came with.

* Source: The installation instructions and graphics below (through “To test”) are from: Freeman, Eric. *Head First Learn to Code: A Brain-Friendly Guide*. O’Reilly, 2018. Google Books preview available at <https://books.google.com/books?id=kytHDwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>.

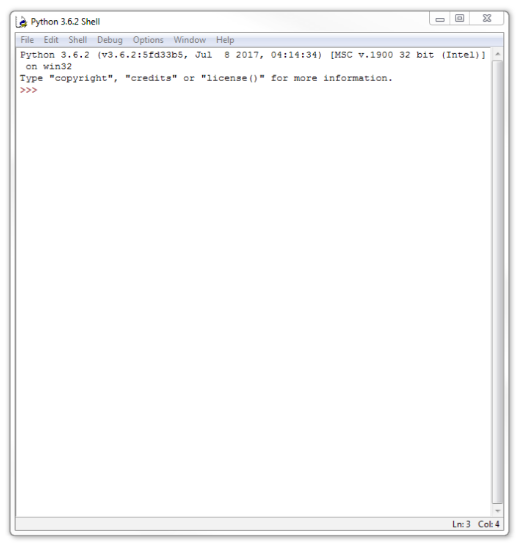
For macOS: Go to <https://www.python.org/downloads>. You should see download links for macOS. If you don’t, look under the Downloads menu on the page.

* Click the Download button for Python 3.x (where x is the latest version).
* Once the installer is downloaded, open the installation package in your downloads folder and follow the installation instructions.
* After you’ve completed the install, navigate to your Applications folder, under which you’ll find the Python 3.x folder. To test your installation, double click the IDLE application in the Python 3.x folder:



For Windows: Go to <https://www.python.org/downloads>

* Click the Download button for Python 3.x (where x is the latest version).
* Choose to either save or run the executable installer. If the latter, click to run the installer after you’ve downloaded it.
* When you see the installer window appear on your screen, make sure the “Add Python to PATH” checkbox is checked at the bottom of the installer, then click “Install Now.”
* After you’ve completed the install, navigate using the Start button to All Programs, and in your list of apps you should see a menu option for Python 3.x (with your version number in place of the x). Under the Python menu you’ll see choices for Python 3.x, documentation, and IDLE, which is the Python Integrated Development Learning Environment. It is usually bundled with (comes with) Python when you install it.
* To test, click the IDLE menu item; when the IDLE application appears on your screen, you should see something similar to the screenshot below.



* Video: If you need more help with installing Python, watch the video below that shows you how to do this.

Hamedani, Mosh. “Python Tutorial for Beginners [Full Course] Learn Python for Web Development.” *Youtube*, uploaded by Programming with Mosh, 18 Feb 2019, <https://youtu.be/_uQrJ0TkZlc?t=15066>. (Watch from 1:50 to 3:19)

At 3:20, Mosh shows you how to install PyCharm. PyCharm is an Integrated Development Environment, which has special features for programming with Python. For the purposes of this course, you don’t need to install PyCharm. Students can do all of the projects within IDLE. They will need to be able to save files in a folder on their computer.

That said, if you are able to install an IDE, such as PyCharm or Visual Studio Code, on one or all of the computers, this will give students the ability to see what a real programming environment looks like.

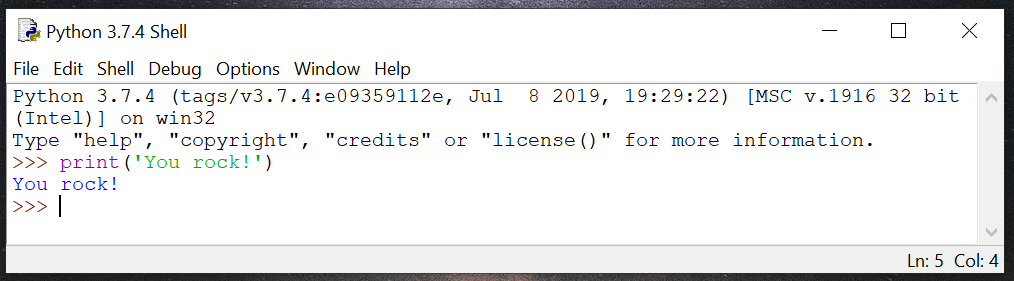
*Students:* Follow your teacher’s instructions to create a folder on your computer that will contain all the Python files you will make during this unit. Title this folder with your name and “NLP”, like this:

FirstName\_LastName\_NLP

Check with your teacher to see if she has any other guidelines for you as you’re naming and saving your files.

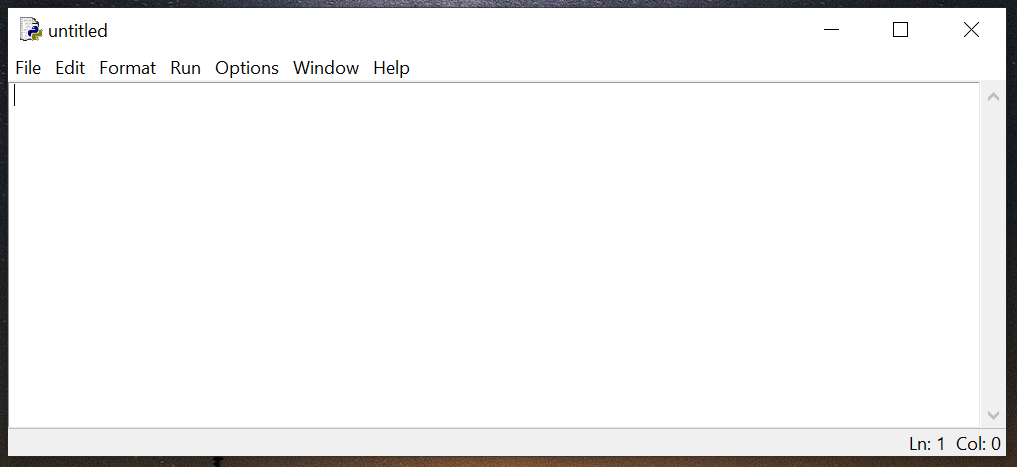
Let’s do a test run to make sure Python is working on your computer.

* First, you’ll need to open IDLE. MacOS: Find IDLE under the Applications > Python 3.x folder. Windows: Search for IDLE in the search field at the bottom of your computer, or use the Start button to go to All Programs and find IDLE under the Python 3.x menu option.
* You’ll see “Python 3.7.4 Shell” at the top. This space is often called the Python shell. To do a test run, after the >>>, type: print(‘You rock!’). Then hit Enter. Your results should look like this (the text above the >>> might look different if this is the first time you’re using IDLE):

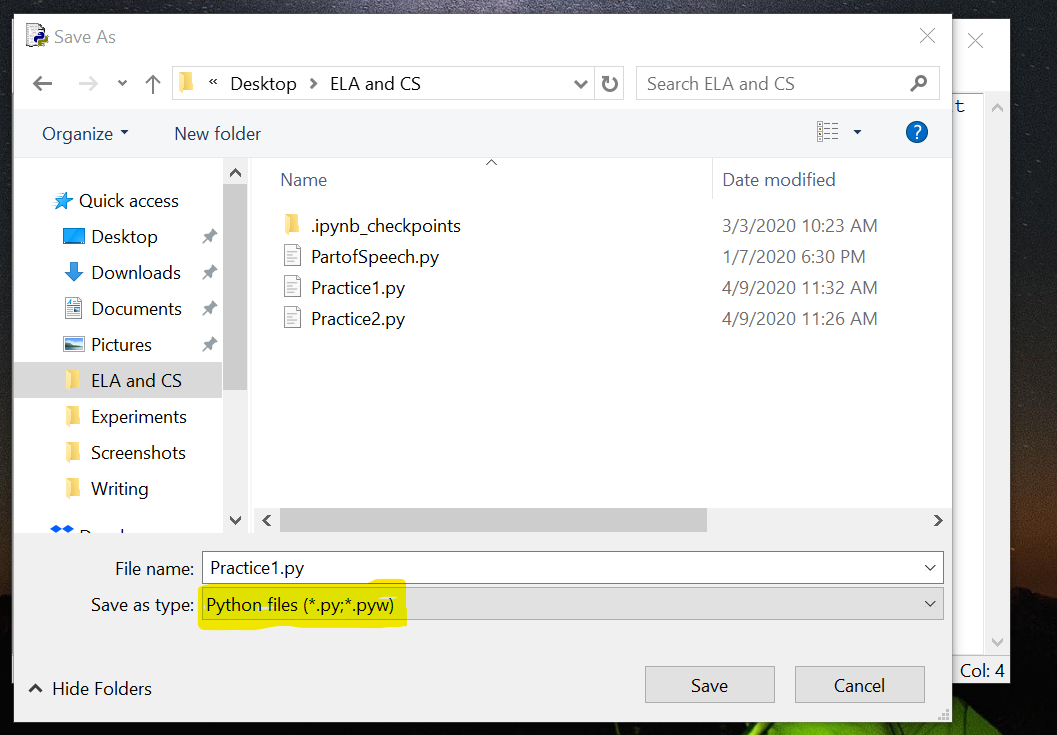


You’ve used the print **function** (think of function as an “action”) to tell the computer to display some text on the screen. The print(‘You rock!’) is Python code that you have written, or your **input**. The blue text, You rock!, is the **output**, or result, of your code.

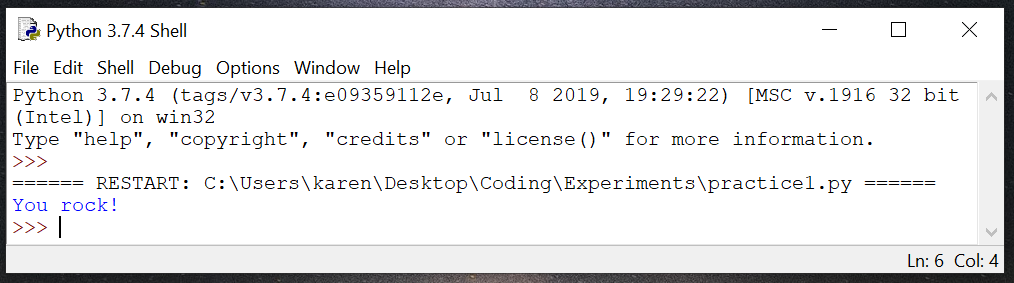
Before we move on, let’s do one more thing: Save and “run” a Python module, also called a file or program. With your shell still open, go to File > New File. You should get a blank file that looks like this:



Now, type the same thing you did in the shell a moment ago: print(‘You rock!’). Go to File > Save. Open the folder you made earlier, where you are keeping all of your Python files. Then choose a name for this practice file. Make sure that the “Save as type” field says “Python files” as highlighted below. **All of your Python files will end in .py.**



Once you’ve saved the file, hit Run > Run Module. Your module that you just wrote will run in the Python shell:



**Basic Python Syntax**

Now that you’ve installed Python and have experimented with the Python shell, it’s time to learn more about how to properly write Python code, just as we have to learn how to properly write English.

Functions

A **function** is an action that you want your computer to take. In the code you wrote earlier, you used the print function to tell the computer to display some text in the Python shell. There are many functions in Python and in other programming languages like JavaScript. Some of these functions are commonly used in Python, like Print, and others you will have to write yourself.

Variables and Values

In Python, we define **variables** and assign them **values**. Then, you can do things with those variables. Think of a variable like a box, and the value (or values) as the contents in that box. When we write variables and their values, we use a = sign to show that we’re assigning whatever’s on the right side (the value) to whatever is on the left (the variable).

For example, let’s say you wanted to assign the data (information) of your name to the variable name.

In my case, I would write it as: name = ‘Karen Schwarze’

You should name your variable something that is clearly identifiable. You may need to use two words to name your variable. In that case, use an underscore between the words and keep everything in lowercase, like this:

person\_name = ‘Karen Schwarze’

Note that Python takes variable names—and all other syntax—at face value. So, person\_name is a different variable from person\_names. If you accidentally add an extra letter to a variable name, or capitalize it once and lowercase it later, Python will not assume you’re referring to the same variable.

More on Values

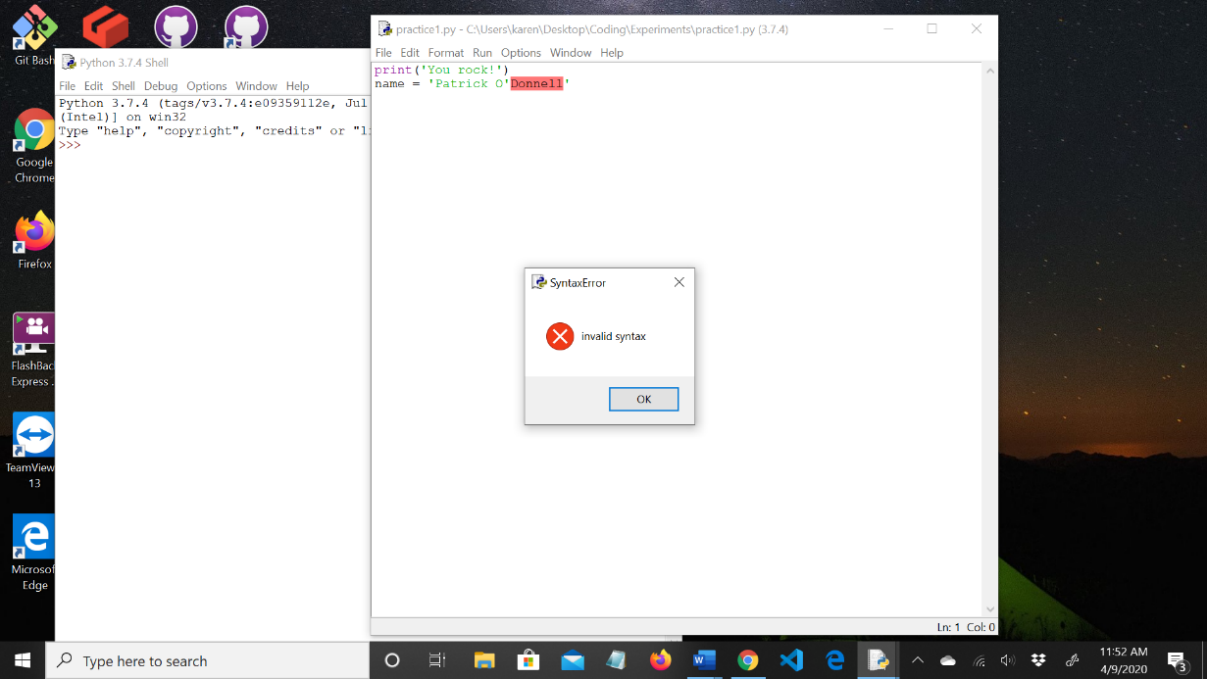
There are many data types, or categories of values. Among other things, values can be strings, integers, floats, Booleans, or lists (a collection of one of the other data types, like a list of strings). A **string** is a group of characters like words, numbers, spaces, or symbols, like the ‘You rock!’ you wrote earlier. An **integer** is a whole number, and a **float** is a decimal number. A **Boolean** is a ‘true’ or ‘false’ value.

Quotation Marks

In English, we use quotation marks for several purposes, one of which is to set off text or show when someone in speaking. In Python, we also use quotation marks. You’ve probably noticed that we need to use quotation marks when we’re using strings. You can use either single or double quotes when you’re writing strings, but generally, you should be consistent and use either single or double quotes for all of the strings in your code. Note that in my variable example above, I used single quotation marks around my name, which is a string:

name = ‘Karen Schwarze’

However, there are times when you will *have* to use single or double quotes. For example, let’s say your name had an apostrophe in it, like in the last name O’Donnell. You couldn’t write your code like this: name = ‘Patrick O’Donnell’ because Python would see the second apostrophe (the one in O’Donnell) and think it was the end of your string. It would get confused because there are still letters after that apostrophe, and you would get an error like this:



Math Functions

We can perform a variety of math functions on numbers with Python.

* Multiplication: Use the \* symbol
* Division: Use the / symbol
* Addition: Use the + symbol (?)
* Subtraction: Use the – symbol
* Modulo: You can perform a special form of division, where the output is the remainder. So, 5 % 2 would be 1, because 5 divided by 2 is 2 with 1 as the remainder.

**Python Practice**

*Teachers - Optional Unplugged Activity:* If you want to help students think about the idea of random word generation, consider playing the game Mad Libs before working through this first Python exercise.

*Students:* Now that you know some basics about Python, let’s write a couple of practice programs.

Exercise 1.

1. Go into your Python shell and open a new file. It’s a good habit to save it with a name right away. For this program, call it Practice1.
2. Now, type in all of the following code, exactly as you see it below.

import random

verbs = [ ]

adjectives = [ ]

nouns = [ ]

verb = random.choice(verbs)

adjective = random.choice(adjectives)

noun = random.choice(nouns)

phrase = verb + ‘ ’ + adjective + ‘ ’ + noun

print(phrase)

Let’s break down what’s happening here. Remember that earlier we said a function is an action you want the computer to take? We’re using a function at the very beginning of this program—import. We are importing the random module, a module or file that is available to you because you downloaded Python. Next, we have three lists (which we will add to shortly): verbs, adjectives, and nouns. Note that we have words on the left side, a = sign, and data on the right side. That means that we are dealing with variables and values. verbs, adjectives, and nouns are all variables, and we are going to add their values within the brackets shortly.

Next, we need to assign values to three new variables—verb, adjective, and noun. Note that the plural forms of these words are our lists, while the singular form of these words are going to give us single items from those lists. Because we are importing the random module, we can use a function called random.choice. This function is going to randomly pull from the lists we made earlier to give us the values for the variables verb, adjective, noun.

Then, we are going to define the variable phrase. The computer is going to take whatever was generated earlier for verb, adjective, and noun and stick them in the value for phrase. Note that we added + signs. When we are combining, or concatenating, strings, we need to use + signs. And the ‘ ’ spaces we have tell the computer to put a space between the words. Otherwise, or verb, adjective, and noun values will be mashed up against each other.

Finally, we tell the computer to print phrase to the screen using the print function so we can see what phrase is.

1. Now that you see what is happening, go back and add values to each of the three lists at the top—our verbs, adjectives, and nouns. For each list, add 5 words. For the first list, you should write 5 verbs. For the second list, write 5 adjectives, and for the third list, write 5 nouns. Put quotation marks around each item in your lists, and separate them with commas.

Example:

verbs = [‘Run’, ‘study’, ‘chew’, play’, ‘read’]

adjectives = [‘purple’, ‘scary’, ‘happy’, ‘tired’, ‘energetic’ ]

nouns = [‘puppy’, ‘bottle’, ‘student’, ‘baseball’, ‘computer’]

1. Your entire program should now look like this, with your own words in the verbs, adjectives, and nouns lists.

import random

verbs = [‘Run’, ‘study’, ‘chew’, play’, ‘read’]

adjectives = [‘purple’, ‘scary’, ‘happy’, ‘tired’, ‘energetic’ ]

nouns = [‘puppy’, ‘bottle’, ‘student’, ‘baseball’, ‘computer’]

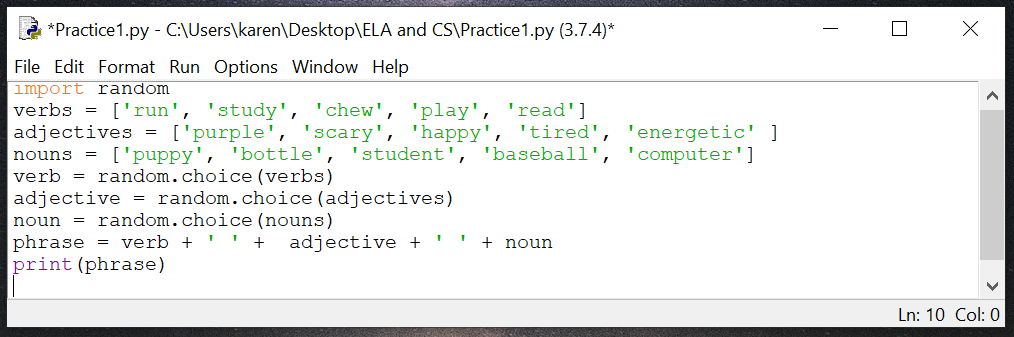
verb = random.choice(verbs)

adjective = random.choice(adjectives)

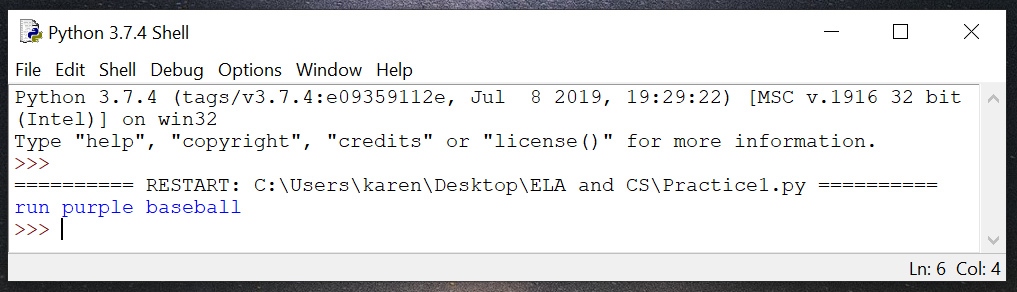
noun = random.choice(nouns)

phrase = verb + ‘ ’ + adjective + ‘ ’ + noun

print(phrase)



1. Save your file again, and hit Run > Run Module.
2. phrase has shown up on your screen, containing the random values generated by your program.

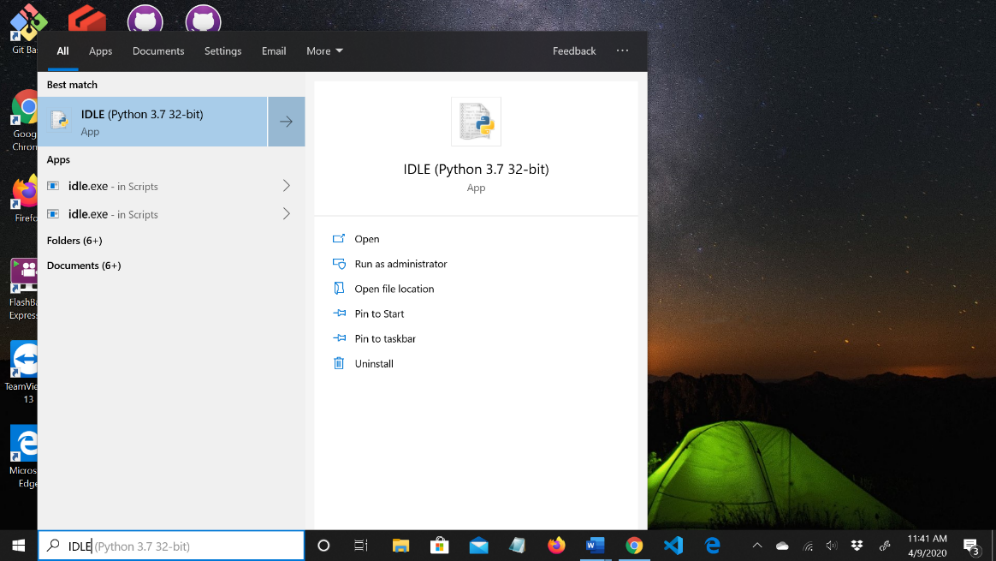


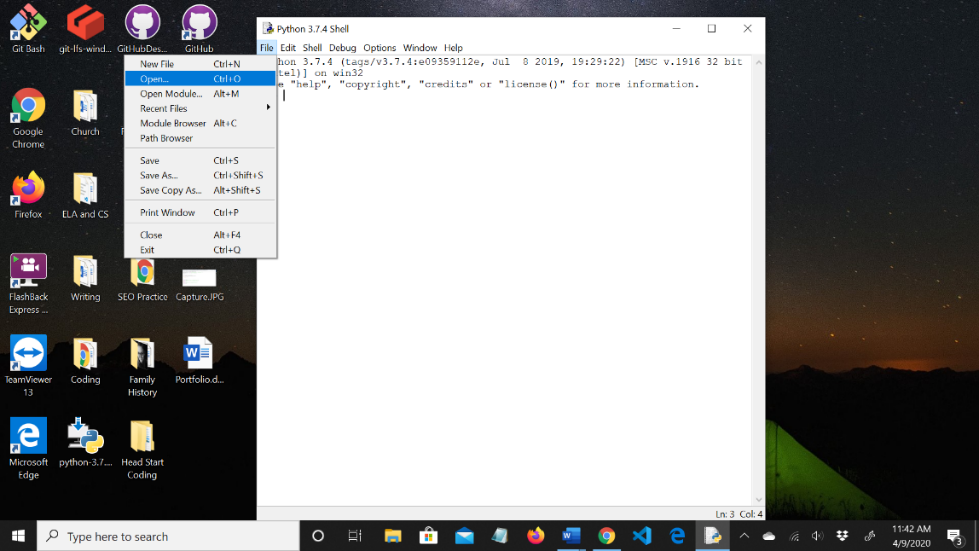
If you got any errors, you will need to do some debugging. If your quotation marks are incorrect, for instance, the program won’t run correctly. In fact, there are even categories of errors. **Syntax errors** occur when we mistype something or make a mistake like omitting a quotation mark. **Semantic errors** are also known as logic errors. Eric Freeman, author of *Head First Learn to Code*, gives a great definition of semantic and **runtime errors** (pg. 54):

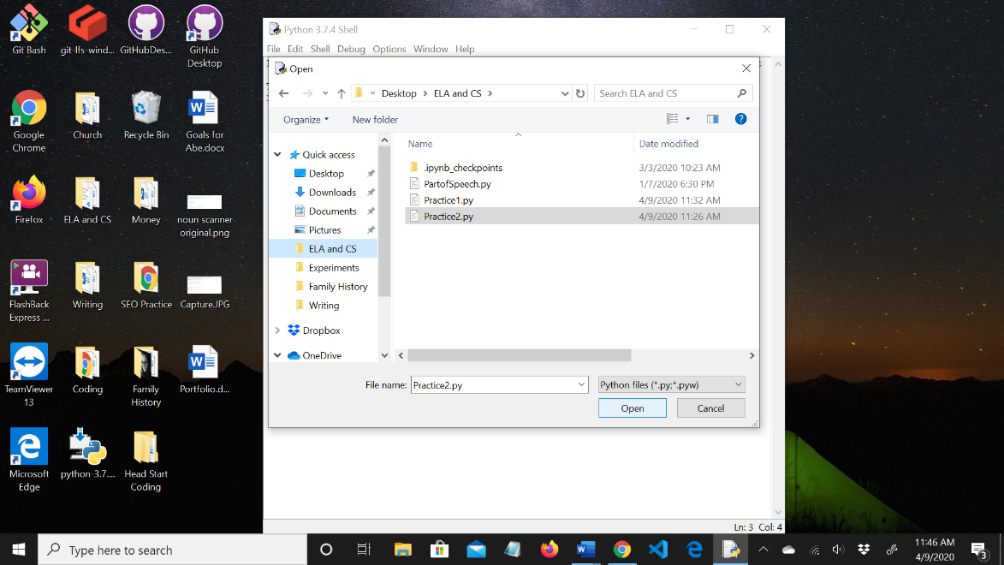
“With a semantic error program will appear to operate normally—the interpreter won’t complain that you’ve made a syntax error at runtime you won’t encounter any issues, but your program won’t give you the results you expected. These always occur because what you think you’ve told your program to do isn’t actually what you’re telling it to do.

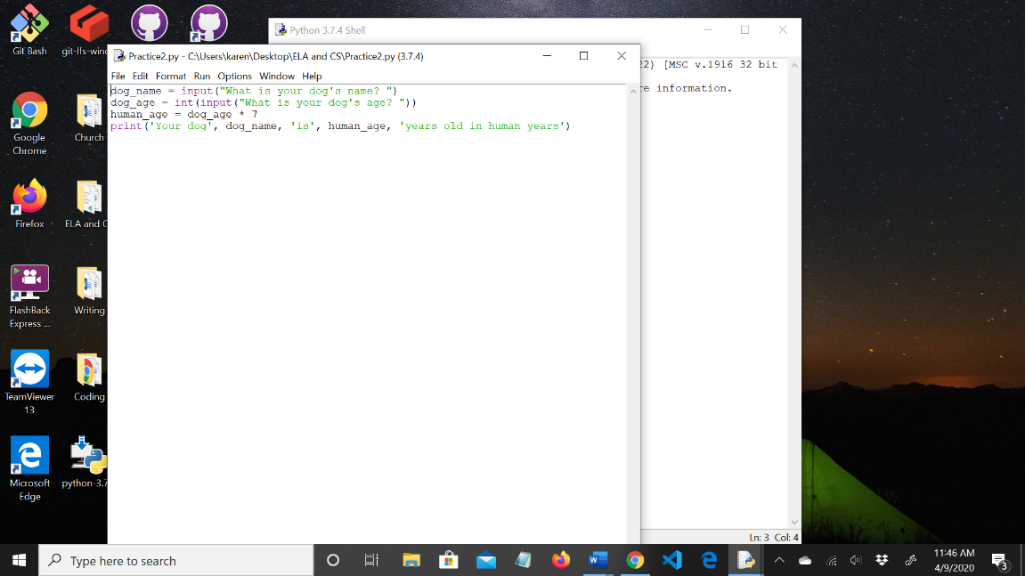
Runtime errors occur when you’ve written a syntactically correct program, but Python encounters a problem running your program. An example of a runtime error would be if at some point in your code you accidentally divided a number by zero (an invalid mathematical operation in any language).”

**\*Note:** After you have run your program, you can close it. To open it again: 1) Open IDLE on your machine. Usually you can do this by typing ‘IDLE’ in the search box on your taskbar (depending on your machine’s operating system). If your teacher told you to put IDLE on your machine’s desktop, you can simply click on it there. 2) Go to File 🡪 Open. 3) Navigate to the program file on your computer. 4) Click on the file and hit Open. 5) The program will appear.

1) 

2) 

3) 

4) 

*Teachers:* Consider taking time to discuss parallels between needing to use proper Python syntax and proper English syntax. What happens when we don’t write our Python syntax correctly? What happens when we make mistakes with English syntax? (Computers don’t understand what we mean and can’t ask for clarification or make inferences based on context/People may not understand what we mean but can ask for clarification or make inferences based on context)

Exercise 2. The code for this practice exercise is from *Head First Learn to Code: A Brain-Friendly Guide* by Eric Freeman, page 24.

*Teachers – Optional Unplugged Activity:* Have students simulate what they are about to do computationally. Have students pair up and give them index cards. Give one student a piece of paper where you’ve written the name and age in human years of an imaginary dog. Have the student without the dog information ask the student who has the dog information: “What is your dog’s name?”, get the response, and then write the name on the index card. Have this student then ask: “What is your dog’s age in human years?”, get the response, and then write this number on the card. Then this student should perform the multiplication formula to get the dog’s age in dog years (multiply the dog’s age in human years by seven). Then this student should tell the other student, “Your dog (dog’s name) is (calculated age) in dog years.” If you like, you can solicit the needed steps from students before doing this process so that they have to think through the steps.

*Students:* With Python, we can get a response from the **user** and perform some actions on that information. Let’s create a dog age calculator to do just that!

If you did the unplugged activity and you were the dog-owner, you’ve already experienced the process of giving information. If you were the computer, you practiced getting information from a user and doing a calculation with that information. You’re going to learn how to tell a computer what to do with information from a user. In this case, we want to create a program to calculate a dog’s age in human years.

1. Open a new file from the Python shell. Title it Practice2 and save it.
2. Copy the following code into your new file and save it.

dog\_name = input(“What is your dog’s name?”)

dog\_age = int(input(“What is your dog’s age?”))

human\_age = dog\_age \* 7

print(‘Your dog,’ dog\_name, ‘is’, human\_age, ‘years old in human years’)

1. Let’s break this down. Here we are defining three variables: dog\_name, dog\_age, and human\_age. To get a value for the first variable, we need some information from the user. We get this information by calling the input function. When the program starts, the user will see the question “What is your dog’s name?” and can type their answer. The computer then stores this input (the user’s answer) as the value for the variable dog\_name. That way, the computer can use this variable/value pair later. Note that the user’s response (the name of the dog) will be a string.

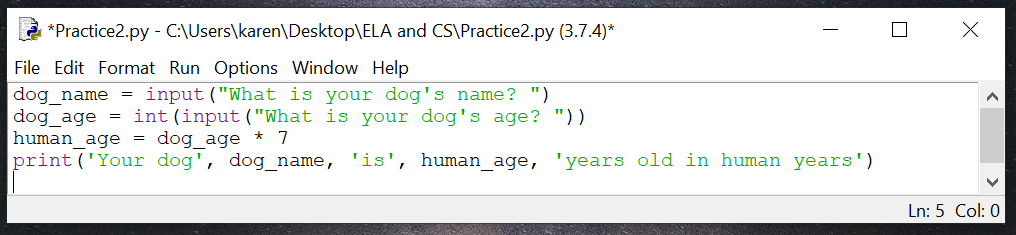
Next, we need to get the dog’s age. We’re going to get this information in the same way as we got the dog’s name, but with a key addition. This time, we need to get a number. We need to transform that number, which the computer would otherwise compute as it would a string, into an integer (a whole number, and one of our other data types). To make this transformation happen, we need to put int in front of our input function. When we do this, the computer knows that the input from the user should be treated as an integer.

Then, we need to calculate the dog’s age in human years. That means that we need to multiply that number we got from the user by 7. This will give us a value for the variable human\_age.

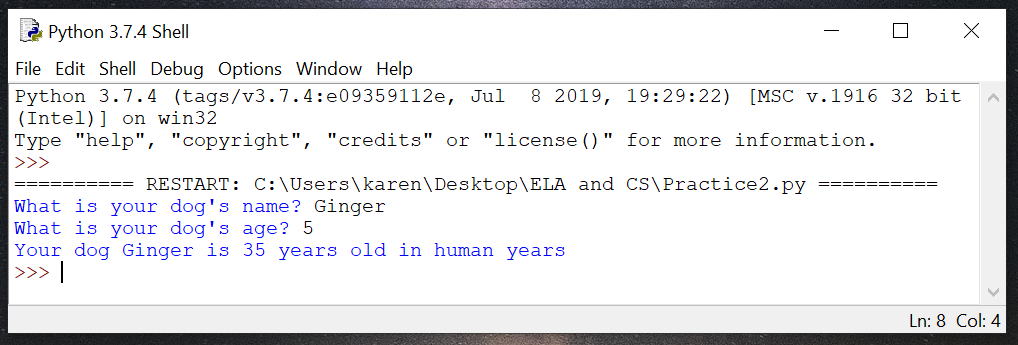
Finally, we want to print some output for our user to see. This output will include the dog’s name and their age in human years, all in a complete sentence.

1. Run your program and enter in a name and an age for your imaginary (or real) dog. Hit enter after each of your responses.

Here’s what your program should look like in IDLE before you run it:



Here’s what your program should look like when you run it (though your answers will be different):



**Key Terms**

* Syntax
* Function
* Input
* Output
* Variables
* Values
* Strings
* Integers
* Floats
* Booleans
* Lists
* Data types
* Syntax errors
* Semantic errors
* Runtime errors
* User

# PART 4: NATURAL LANGUAGE PROCESSING PROJECTS

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!

**SIMILE GENERATOR**

Remember those rules that you wrote earlier about different literary devices? Now we’re going to use those rules to make a simile generator. When you’ve finished writing this program, you’ll be able to run it and get a different simile every time!

First, let’s revisit the rules you wrote earlier about a simile. They are as follows (your wording may be slightly different, but should contain these basic principles):

*A simile compares two things  
A simile always contains some form of ‘to be’, such as ‘is,’ ‘am’, ‘are’, or their past/future forms  
A simile always contains the word ‘like’ or ‘as’*

For the purposes of our simile generator, we’re going to parse down these requirements a bit. First off, our similes will compare two specific categories of things—abstract nouns and concrete nouns (more on those below). Also, our similes only use the singular, present tense of *to be (is).* And finally, our similes will only contain *like,* not *as.*

Abstract Nouns and Concrete Nouns

An **abstract noun** is an idea or feeling that can’t be seen, touched, felt, or otherwise detected with our five senses (smell, sound, touch, taste, sight). A **concrete noun** can be detected with our senses in some way—just like how concrete, the material used to make sidewalks, can be seen and touched. Abstract nouns include things like love, happiness, anger, disappointment, and victory. Concrete nouns include things like water, the sun, ice cream, puppies…you get the idea.

Before we start coding, we need to map out some ideas on paper. Just as you write an outline before writing an essay, we can write **pseudocode** before we code our program. Our simile generator is going to be very similar to the Exercise 1 you did at the beginning of the section, where you wrote a program that randomly generated a group of words containing a verb, an adjective, and a noun. We’re going to use the import random module again so that our program can randomly select from two lists (our abstract nouns and our concrete nouns). Then, we’re going to tell the program to assign values from those respective lists to two variables. Next, the program will create a simile using a randomly generated abstract noun and concrete noun, joined by the phrase ‘is like’. Finally, we’ll tell the program to display the simile using the print function.

Let’s write out the steps above using some pseudocode. Then we’ll translate it into Python.

Use the import random module.

Create two variables, each containing a list of values.

Randomly assign a value from each list to a variable.

Create a simile using a randomly generated abstract noun and concrete noun, separated by the phrase ‘is like’.

Display the simile using the print function.

Note that all we did to write our pseudocode was to break down the steps. Now, go back and look at the code we wrote for Exercise 1. See if you can use it to help you get close to writing the actual code for your simile generator. Your code will probably be mostly **skeleton code**—code that is more fleshed out than pseudocode but not quite real code yet. Use the space below to write your skeleton code.

(If you what the real code is, you can write that instead of the skeleton code. The goal here is for you to do your best to write the closest thing you can to what the real code will look like before moving on to the answer). Remember that you’ll need to write two lists—one for abstract nouns and one for concrete nouns. Choose any abstract nouns and concrete nouns you want to use—these are your similes!

Write your skeleton code here

After you’ve written some skeleton code, check it against the answers below. As you read the answer, make sure you understand why I wrote the code this way. If you’re not sure, ask for help!

I’ve written out each step followed by the Python code. Remember that your abstract and concrete nouns will probably be different from mine.

1. Use the import random module  
   import random
2. Create two variables, each containing a list of values   
   abstract\_nouns = [‘Love’, ‘Forgiveness’, ‘Impatience’, ‘Penitence’]  
   concrete\_nouns = [‘wind’, ‘light’, ‘the sea’, ‘darkness’]
3. Randomly assign a value from each list to a variable  
   abstract\_noun = random.choice(abstract\_nouns)  
   concrete\_noun = random.choice(concrete\_nouns)
4. Create a simile using a randomly generated abstract noun and concrete noun, separated by the phrase ‘is like’.   
   simile = abstract\_noun + ‘is like’ + concrete\_noun
5. Display the simile using the print function.  
   print(simile)

Ready to see your simile generator in action? Put all of the code above into IDLE, save it as a new file, and select Run > Run Module. Here’s the code again for you:

import random

abstract\_nouns = [‘Love’, ‘Forgiveness’, ‘Impatience’, ‘Penitence’]  
concrete\_nouns = [‘wind’, ‘light’, ‘the sea’, ‘darkness’]

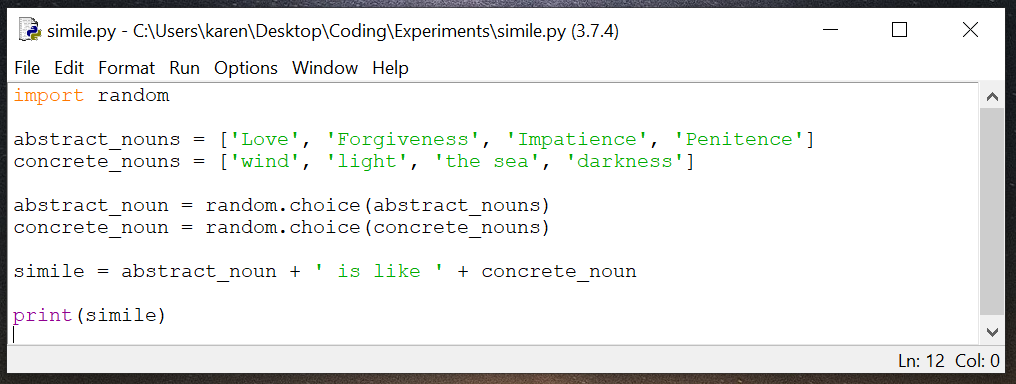
abstract\_noun = random.choice(abstract\_nouns)  
concrete\_noun = random.choice(concrete\_nouns)

simile = abstract\_noun + ‘is like’ + concrete\_noun

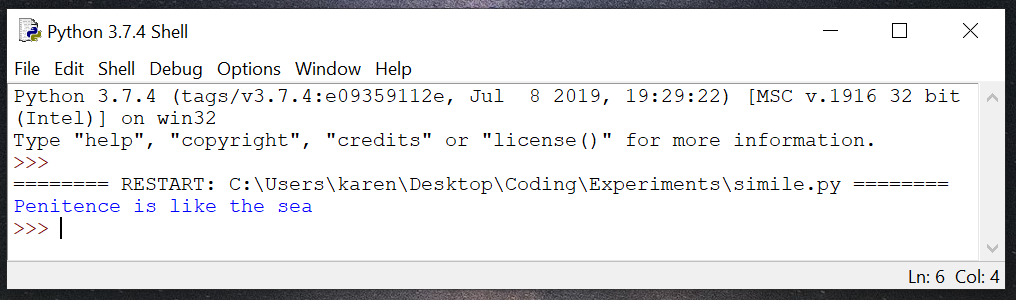
print(simile)

Keep selecting Run Module to generate new similes. Show your classmates your similes! What kind of abstract nouns and concrete nouns did they choose? What kind of similes are their programs generating?

What my program looks like in IDLE before I run it:



What my program looks like after I hit Run > Run Module:



What kind of similes does your program generate?

**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.

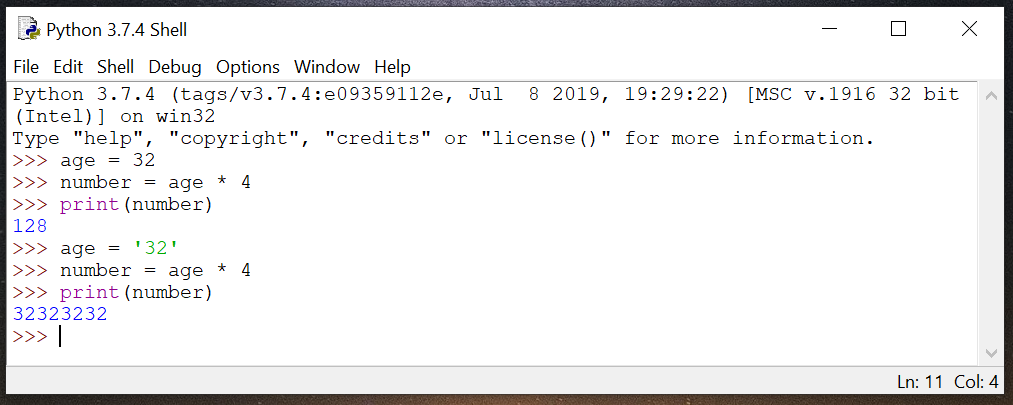
**\*Note:** You’ll notice that the above numbers do not have quotation marks around them. That’s because we’d probably want to treat them as integers, not as strings. To see the difference, type the following into the IDLE Shell (no need to make a new file) and hit Enter after each line:

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.”

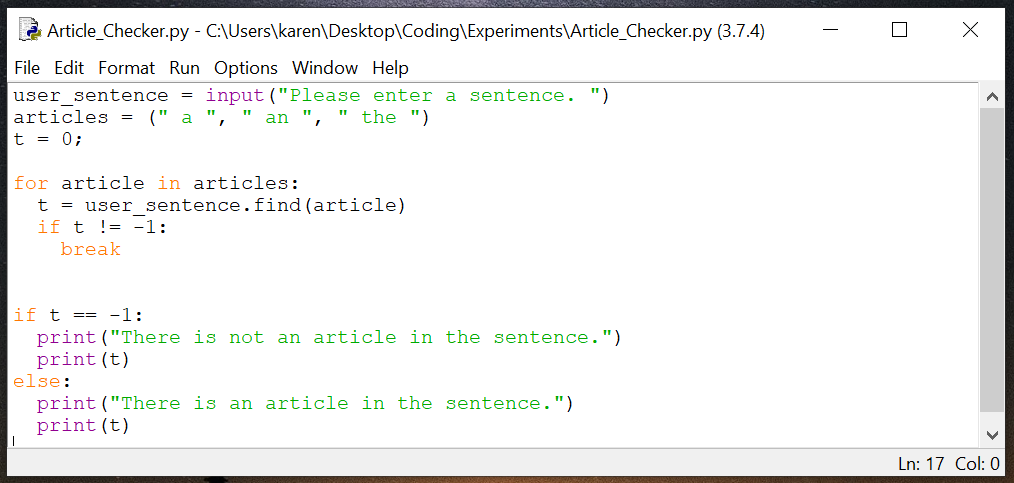
else:  
 print(“There is an article in the sentence.”)

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

This section of code could be translated as:

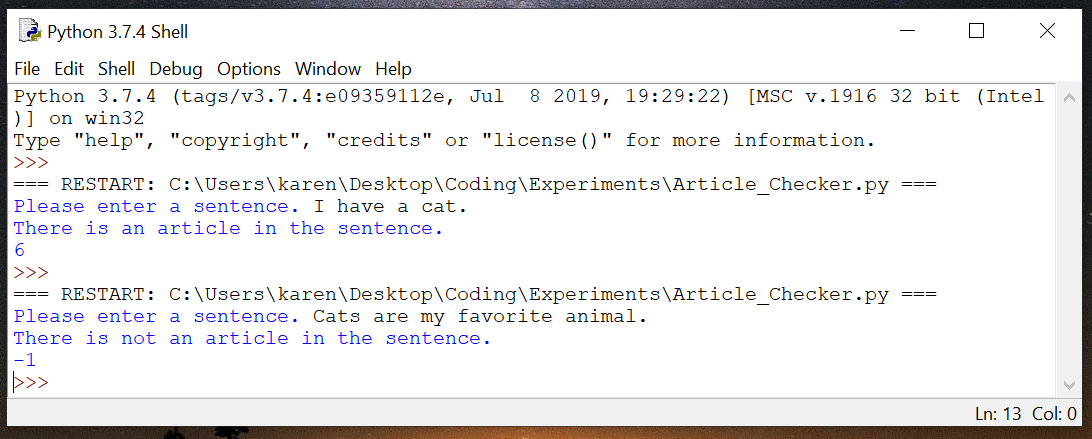
“If t equals -1, display the words ‘There is not an article in the sentence.’ Also, show what number **t** is. Else, display the words ‘There is an article in the sentence’ and show what number **t** is.”

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?

**STUDENT CHOICE: YOUR OWN PROJECT!**

Now that you have made several of your own projects, you understand some basic Python fundamentals. You also have an understanding of how to translate rules of English syntax into code, so that a computer can interact with these rules computationally.

Now it’s time to get creative and put your knowledge of Python and English to the test. You’re going to create your very own program! If you’re not sure how to start, just think about the rules you studied earlier about the English language. Will you write a program involving parts or speech or a literary device? Consider how you could use those rules to make a program.

Review the other programs you’ve written so far and the explanations about Python syntax. Do additional research about Python to learn additional necessary syntax. Use the spaces below to help you map out your program before you write it using code in IDLE.

My favorite/best idea

My ideas

Pseudocode for my favorite/best idea

Skeleton code for my favorite/best idea

Final code for my favorite/best idea

Ask for help as you’re working on your project if you get stuck. Share your program with your classmates when you’re finished!

*Teachers:* As part of their final project, consider requiring students to write a reflection about their program. What is the purpose of their project? What English concepts do they use? What Python code do they use? What new Python or English concepts did they have to research?

**Key Terms**

* Abstract noun
* Concrete noun
* Pseudocode
* Skeleton code
* Substring
* List
* Index numbers
* Zero-based indexing
* For loop
* Iterate
* Iterator
* .find() method
* return
* relational operators
* If statements
* if/else

\*\*\*

Are your students ready for more advanced NLP concepts? If so, check out these videos to show your class:

* “A.I. Experiments: Visualizing High-Dimensional Space.” *Youtube*, uploaded by Google Developers, 15 Nov 2016, <https://www.youtube.com/watch?v=wvsE8jm1GzE>
* “Stanford CS224NL NLP with Deep Learning | Winter 2019 | Lecture 1 – Introduction and Word Vectors.” Youtube, uploaded by stanfordonline, 11 March 2019, <https://www.youtube.com/watch?v=8rXD5-xhemo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z>

Start video at about 10 minutes in. Previous to that Professor Manning talks about class-specific items.

**Bibliography**

“100 Adverbs.” *Yourdictionary.com,* <https://www.yourdictionary.com/index.php/pdf/articles/140.100adverbs.pdf>

“A.I. Experiments: Visualizing High-Dimensional Space.” *Youtube*, uploaded by Google Developers, 15 Nov 2016, <https://www.youtube.com/watch?v=wvsE8jm1GzE>

Bradley, Laura. “Coding in English Class? Yes! And In Your Class, Too!” *KQED.org,* 22 Jun 2017, <https://www.kqed.org/education/523829/coding-in-english-class-yes-and-in-your-class-too>.

Common Core State Standards Initiative. “Common Core Standards for English Language Arts & Literacy in History/Social Studies, Science, and Technical Subjects.” <http://www.corestandards.org/ELA-Literacy/>.

Freeman, Eric. *Head First Learn to Code: A Brain-Friendly Guide*. O’Reilly, 2018. Google Books preview available at <https://books.google.com/books?id=kytHDwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>.

Good, Edward C. *Who’s (Oops!) Whose Grammar Book is This Anyway?* MJF Books, 2002.

“Grammar: Direct and Indirect Object.” *MyEnglishPages.com*, <https://www.myenglishpages.com/site_php_files/grammar-lesson-direct-indirect-object.php>.

Hamedani, Mosh. “Python Tutorial for Beginners [Full Course] Learn Python for Web Development.” *Youtube*, uploaded by Programming with Mosh, 18 Feb 2019, <https://youtu.be/_uQrJ0TkZlc?t=15066>.

“Literary Devices.” *Literary Devices: Literary Devices, Terms, and Elements,* <http://www.literarydevices.com/>.

“Main Verbs: Definition and Examples.” *Grammarly,* <https://www.grammarly.com/blog/main-verb/>.

“Natural Language Processing: Crash Course Computer Science #36.” *Youtube*, uploaded by CrashCourse, 22 Nov 2017, <https://www.youtube.com/watch?v=fOvTtapxa9c>*.*

Peters, Mark. *Idiot’s Guides:* *Grammar and Style.* Penguin, 2014.

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

“Stanford CS224NL NLP with Deep Learning | Winter 2019 | Lecture 1 – Introduction and Word Vectors.” *Youtube*, uploaded by stanfordonline, 11 March 2019, <https://www.youtube.com/watch?v=8rXD5-xhemo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z>.

Simmons, Robin L. “The Gerund Phrase.” <https://www.chompchomp.com/terms/gerundphrase.htm>.

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

“What is a Gerund Phrase?” *Grammarly*, <https://www.grammarly.com/blog/gerund-phrase/>.