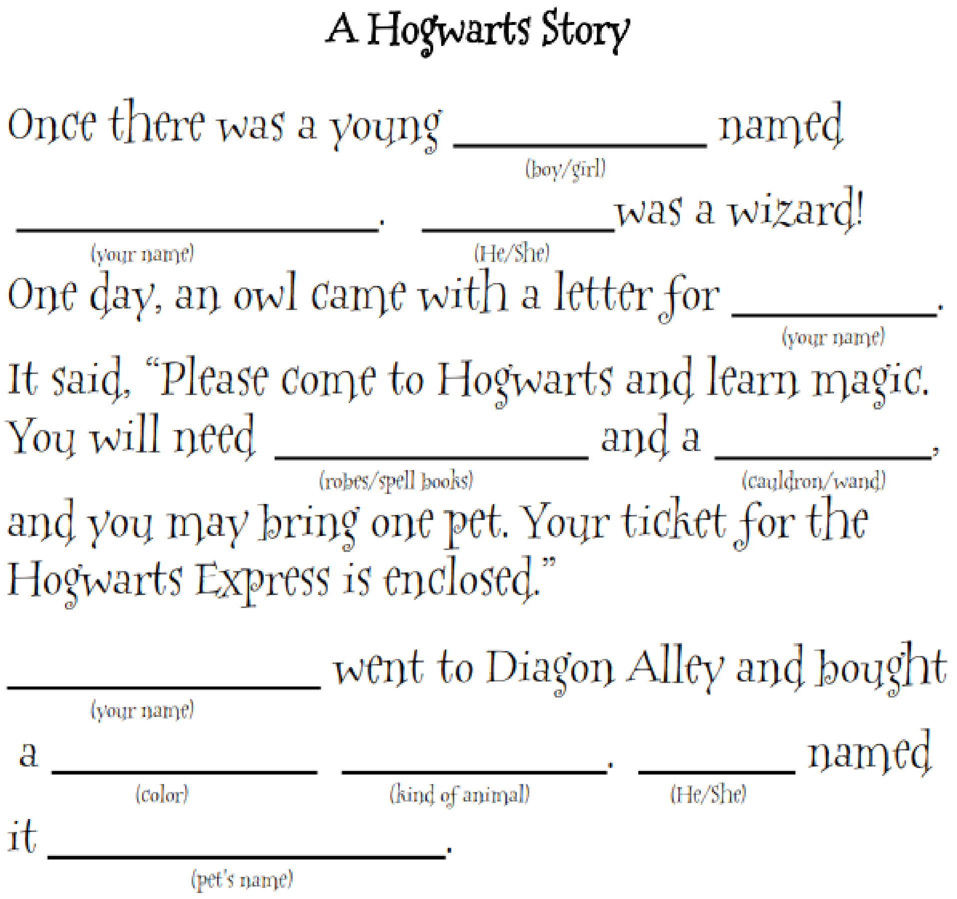
Ad-libs in Python

Cameron Johnston

June 15, 2020

1 Writing our stories

In this lesson, we are going to learn how to take in input, write it into a specific format, and save it so we can come back to it any time! To do that, we're going to make our own ad-lib stories and use Python to fill them in. First thing's first, we're going to need to make our story, it's time to get the creative juices flowing. Take twenty minutes to come up with a short ad-lib story, remember to leave plenty of blanks! Check out this example if you're stuck. Pay careful attention to how they've labelled all the blanks, this will be important later!



A Hogwarts Story

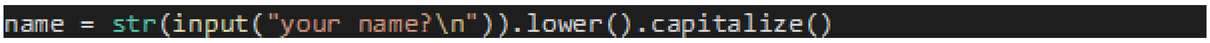
Once there was a young ____________ named
(boy/girl)
__________________. __________ was a wizard!
(your name) (He/She)
One day, an owl came with a letter for __________.
(your name)
It said, "Please come to Hogwarts and learn magic. You will need ________________ and a ___________,
(robes/spell books) (cauldron/wand)
and you may bring one pet. Your ticket for the Hogwarts Express is enclosed."

________________ went to Diagon Alley and bought
(your name)
a _____________ ______________. ________ named
(color) (kind of animal) (He/She)
it ____________________.
(pet's name)

If you really can't think of anything, don't worry! Ask your tutor for help and they can give you a story pre-made.

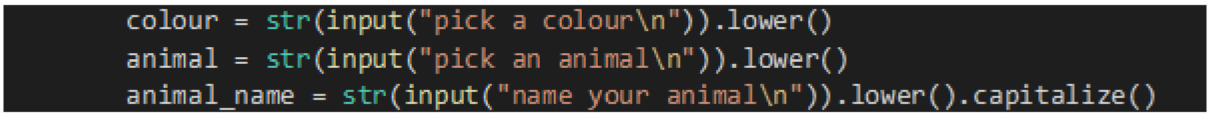
2 Getting our inputs

Now that we've got our stories, it's time to head into Python. Start by defining your function, call it whatever you think is best. Inside this function, we need to get inputs from our users. In order to do this, we can use the input() function. This function takes in a string as an argument which it will display and wait for the user to type something in response. An example of this is:



```
name = str(input("your name?\n")).lower().capitalize()
```

Let's break this down. The str() function is there to make sure that, whatever the user types in, Python will know to treat this like a string of text. The input function will display "your name?" to the console and pause the program until the user types in something and presses enter. That strange symbol at the end of the text ‘’ tells the console to move the cursor to the next line, meaning the user will be typing on the line below ‘your name?’. .lower() is going to make sure that, if our user types their name with AnY sTrAnGe capitals, they won't turn up in our story. And, to cap it all off, we use .capitalize() to make sure the first letter is capital, since this is a name and names always start with a capital letter.
To get all our user's choices, we need to call the input function for every different word, and make sure to save all of them with a variable name that tells you what it is you're saving. So if your story calls for a pet, with your user telling you their pet's name, colour, and what animal it is, you'd use something like this:



```
        colour = str(input("pick a colour\n")).lower()
        animal = str(input("pick an animal\n")).lower()
        animal_name = str(input("name your animal\n")).lower().capitalize()
```

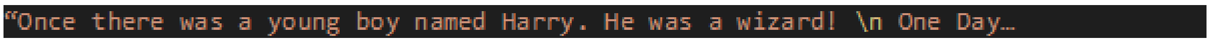
Take ten minutes to make sure you've got all your inputs. If you get stuck at any point, ask your tutor for help.

3 presenting our stories

Now that we have our story, and we have all our inputs from our user, it's time to bring it all together. I imagine your story is longer than just one line, and I'm sure you don't want it all to be all on the same line. There are a couple of ways we can achieve this in our code.

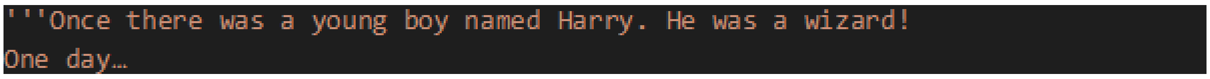
3.1 new lines

The first trick you've already met; remember that ‘’ we've been putting at the end of our strings? We can put that at any point in our string and it tells the console to start a new line. If we wanted to make the story we saw earlier, we'd need to start a new line after the word wizard. That would look something like this:



```
"Once there was a young boy named Harry. He was a wizard! \n One Day…
```

Another option is for us to use a multiline string, which we make by using three apostrophes, ”’. This way, instead of using to move to the next line, you just press enter like you would normally. Be careful with this though, any indentation after your first line will be read as part of the string! After you press enter to make a new line, make sure you remove any indents you have. Using the same example above, we would have:



```
'''Once there was a young boy named Harry. He was a wizard!
One day…
```

No one method is better than the other – both have their upsides and downsides; can you think of them? Choose whichever you prefer, and we can begin to get our story into Python with all the variables in the right place.

3.2 the f-string

There are quite a few ways to put a variable into a string and print it out, you may have even seen some before! They all have their pros and cons but for our stories, our best bet is to use f-strings. F-strings are relatively new to Python – only being added in 2016 – and they make formatting big blocks of text much easier. To make an f-string out of a regular string, all you need to do is to write the letter 'f' in front of your quotations or apostrophes, f". Now that we have an f-string, we tell it where to put our variables by writing the name of our variable inside curly brackets wherever we want them to be written. In our example, we'd write it like:

```
f'''Once there was a young {gender} named {name}.
```

And when we print it, assuming gender=boy and name=Harry, we get:

```
Once there was a young boy named Harry.
```

Now you know how to put all your variables into a string, and how to make your strings more than one line long, make a new variable called story and, in fifteen minutes, write out your story in Python, making sure to put all the variables and the new lines in the right place. If you get stuck, make sure to ask your tutors for help!

4 showing and saving our stories

Now we have our stories in Python, and a way for our users to tell us all their choices, we need to show our users their complete story and save it for them to keep.

4.1 printing

Showing our user their story is easy enough since we saved our story as the variable story. All we need to do is use the print function on our variable.

4.2 opening a file

To save our story, we need to make a place to save it to. Luckily, Python has us covered for that. Usually, the open function is used to open a file that already exists, but, if Python can't find that file, it'll make it for us. The open function requires two pieces of information from us to work, two parameters. The first is the full name of the file we're trying to open or make – this is including its extension (.txt, for example). The second is the mode, what Python is going to let us do with the file once it's open. There are four different options for the mode – read, write, append, and create. If you want to save every story, writing the new one after the old ones, you need to use append mode, but if you just want to save the most recent story, you need write mode. Each mode is given a letter as a shorthand, which is what you type in as the second parameter. For append mode, you pass in a. For write mode, it's w. Both the filename and the mode are passed in as strings. For example:

```
open("Stories.txt","a")
```

This will open, or make, a text file called Stories in append mode. We want to assign this function to a variable, giving it a sensible name, as it refers to the file that we are saving in.

4.3 writing

Now that we have our file open, we need to write our story to it. For that, it's as easy as calling the .write() function on our open file and passing in our story variable. If you're using append mode, think about how you would go about making sure that each new story starts on its own line – and how you can label which story belongs to who.

4.4 closing

When we're done with the file, we need to tell Python that. Leaving a file open for longer than we need it for leaves the file inaccessible and, in a bigger project with more files, could slow your computer down by a lot. So, when you're done writing to the file, make sure to call the .close() function on it.

Now we know how to print our stories while the program is running, and how to save them even after we've closed Python, it's time to put it all to use. Take ten minutes to add these features to your program.

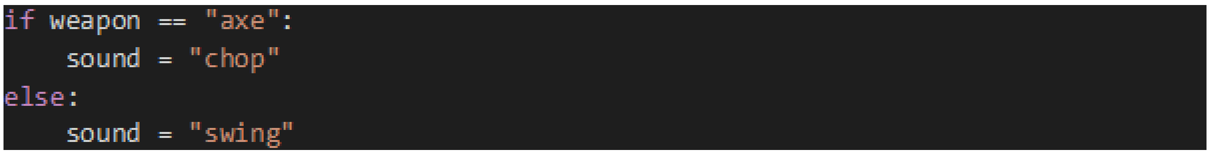
5 Finishing up

The last thing for us to do is to call our function and run the program. If everything went well you should have a working ad-lib story on your hands. If you're looking to stretch yourself, check out the bonus challenges below to add a few more features to your program.

Bonus Challenge 1

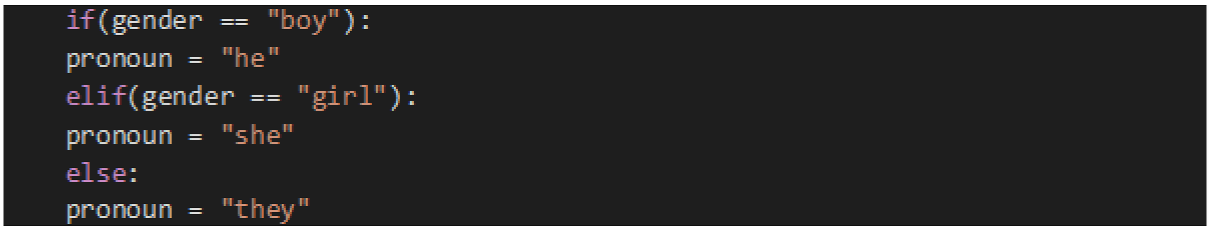
Sometimes, you want a word in your story that depends on what answers the user gave you, but you don't want to use the exact words they gave you. For example, if you ask the user if they have a sword or an axe in the story, you want them to slash with the sword and chop with the axe. In that case, you need a conditional statement.

If you've met conditional statements, skip on over to the next paragraph for an extra challenge. If you're new to these, keep reading on. A conditional statement, or an 'if' statement, is a piece of programming that tells Python if I have one thing, do this, I have another thing, do something else. In the case of our axe sword conundrum we would say if our weapon is an axe, make a chop sound, else make a swing sound. Here we are assuming that the weapon will only ever be a sword or an axe. This is okay if you know that's true, but if the weapon could ever be something else, like a bow, then you'll want to be more careful. In Python, that would look something like this:



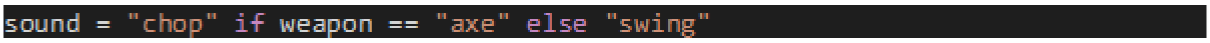
```
if weapon == "axe":
    sound = "chop"
else:
    sound = "swing"
```

Notice the two equals signs between weapon and axe. That is telling Python check to see if these two things are the same, while one equals means set this to that. What do we do if we have more than two cases? That's where the elif statement comes in. Elif is short for 'else-if' and means exactly that. You use it after the first if to say if that condition isn't the case, try this one. For example, say we ask the user for their gender, we don't want to then also ask for their pronouns. In that case, we'd need to use something like this:



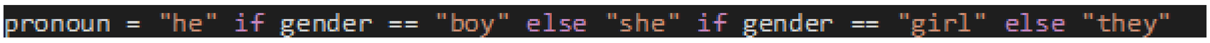
```
    if(gender == "boy"):
    pronoun = "he"
    elif(gender == "girl"):
    pronoun = "she"
    else:
    pronoun = "they"
```

If you skipped over the introduction to conditionals, welcome back. Conditionals like this can be really powerful, but they also take up quite a few lines. If we want to reduce the number of lines our program takes up, we can use something called a ternary operator. A ternary operator works the same as a conditional, it even uses the same statements, but it only takes up one line. In the case of the axe and the sword, you'd write:



```
sound = "chop" if weapon == "axe" else "swing"
```

Just like the conditional, we're using the double equals to check if the weapon variable is an axe. The key differences here are the order in which you write the ternary operator, and that you don't write sound = before swing. That's because ternary operators are telling Python how to work in a completely different way to how conditionals work. In a conditional, you're saying if this first statement is true, do this, else do that. In a ternary operator, you're saying assign this variable to one value if this is true, else assign it to some other value. It is possible to make ternary operators that use more than one condition, though generally it's best to avoid them as they can get harder to read with more conditions. In ternary operators, there is no elif, instead you write assign this value if the first condition is met, else assign that value if the second condition is met, otherwise assign a different value. In the case of the pronouns, that would look like:



```
pronoun = "he" if gender == "boy" else "she" if gender == "girl" else "they"
```

Now that you understand conditionals and ternaries, go back through your variables, and see if there's any you could infer from inputs you got before. If there are any, replace the input with a conditional statement, or a ternary operator if you're up to it.

Bonus Challenge 2

You may have noticed that this app only works once, you have to restart it to use it again. While this approach is okay for some python programs, it's usually not the best for games. To remedy this, we need to make what's called a game loop, a piece of code that is repeated until the user tells it to stop. Remembering that Python has two types of loop, the for loop and the while loop, think about how you would go about implementing a game loop into your program and how you would ask the user if they want to play again.