For this project, you will write a program that interacts with a user at the Python console to tell "knock-knock" jokes.

What's a knock-knock joke?

A knock-knock joke is a highly structured dialog between a joke teller (let's call him Tom) and a willing participant (let's call him Jerry). The dialog consists of five lines. It starts with Tom saying "Knock-knock." and then Jerry saying "Who's there? The next two lines result in a surprise ending (line 5) that often reveals some sort of *pun or play-on-words*.

Here's an example …

Tom: **Knock-knock!**

Jerry: ***Who's there?***

Tom: **Cow says.**

Jerry: ***Cow says who?***

Tom: **No! Cows say moo. Owls say hoo!**

Maybe you find these humorous, maybe you don't. But it's often the first type of joke children learn how to tell in the USA. In your program, your computer will play the role of Tom and the user will play the role of Jerry. Let's analyze it …

The first two lines are always the same …

Computer: **Knock-knock!**

User: ***Who's there?***

Your program should check the user's response and should accept several variations such as "who's there", "who is there","whos there?", "who is there?", etc. and be case-insensitive and with or without punctuation. If the user does not respond appropriately, print a reminder of the ideal appropriate response and start over with "knock-knock".

Computer: **Knock-knock!**

User: ***What?***

Computer: **Sorry. The correct response is "Who's there?"**

**Try again.**

After the line 2 response has been validated, move on to line 3. The next line is called the "prompt". It is different for every knock-knock joke …

Computer: **Cow says.**

The prompt is a string you have stored in a Python list containing joke prompts.

In the next line, Jerry just adds the string, "who?", to the end as if to ask a question …

User: ***Cow says who?***

This user input must be validated in a case-insensitive way. A variety of responses with or without the punctuation should be acceptable. If the user does not respond appropriately, print a reminder of the appropriate response and start over with the prompt.

The next line is called the "punchline". It provides a surprise ending to the joke …

Computer: **No! Cows say moo. Owls say hoo!**

After the punchline has been delivered, the joke is finished.

If you are unfamiliar with the concept and want to hear some knock-knock jokes being told, watch this video … <https://youtu.be/jd4nTtycrN8>

**ANALYTICAL SUMMARY OF A KNOCK-KNOCK JOKE**

A complete knock-knock joke consists of five phrases:

p1: the opening (produced by the computer)

p2: the reply (from the user)

p3: the prompt (produced by the computer)

p4: the response (the user types: p3 + "who?")

p5: the punchline (produced by the computer)

The opening (p1) is always the same. The reply (p2) is always the same (or some acceptable variant). The prompt (p3) is taken from a list of prompts. Each prompt (p3) is associated with a unique punchline (p5) which gets delivered when the user produces an acceptable response (p4). An acceptable response is one where the user types the prompt (p3) followed by the word, "who?". Which should be case-insensitive and punctuation agnostic.

p1 and p2 are the same for every joke. Your program should be tolerant of slight errors in p2, respond to appropriate variants and send a gentle correction message if the user reply is not correct. p3 and p5 are the key parts of the joke. They go together and ought to be organized in that way. A correct response, p4, is constructed by concatenating "who?" to the end of p3.

WRITING THE PROGRAM.

As usual, you should plan carefully and proceed one small step at a time. Your program should be prepared to tell a minimum of 3 different knock-knock jokes in a randomized order. For every joke, you will need a prompt string and a punchline string. There are a variety of ways to organize this data. However, you are required to use Python list structures wherever possible. You should use lists to keep prompts and punchlines paired together to generate jokes and lists of response variants to validate user responses.

Your program must …

* + Provide an introduction / announcement that tells the user how many jokes are available.
  + Ask the user for a number that defines how many jokes to tell.
    - Receive and validate the response; handle exceptions.
  + Validate the user input string at each stage (p2 and p4) case-insensitive and punctuation agnostic.
  + Use a list to hold your jokes (pairs of prompts and punchlines).
  + Use a list to hold response variants that are also NOT case sensitive and be punctuation agnostic (any sentence ending or no sentence ending is ok). Here are some variants on "Who's there?" that your program should consider to be correct responses …
    - whos there
    - who is there
    - who there
    - WHO THERE
    - etc.

* + Tell the jokes in a random sequence.
    - If the user runs the program twice and asks for one joke. The joke should be likely to be different the second time. (HINT: use **random.shuffle()**)
  + Exit with an appropriate, cheerful, and affirmative message.

Having difficulty finding knock-knock jokes for your program? Here are six …

----------------------------------------

**Knock knock.**

**Who’s there?**

**Etch.**

**Etch who?**

**Please sneeze into your elbow!**

----------------------------------------

**Knock knock.**

**Who’s there?**

**Cash.**

**Cash who?**

**No thanks, I prefer peanuts.**

----------------------------------------

**Knock knock.**

**Who’s there?**

**Tank.**

**Tank who?**

**You’re welcome.**

----------------------------------------

**Knock knock.**

**Who’s there?**

**Ya.**

**Ya who?**

**Yahoo! I’m happy to see you too!**

----------------------------------------

**Knock knock.**

**Who’s there?**

**Boo.**

**Boo who?**

**Don't cry, it’s only a joke.**

----------------------------------------

**Knock knock.**

**Who’s there?**

**Dwayne.**

**Dwayne who?**

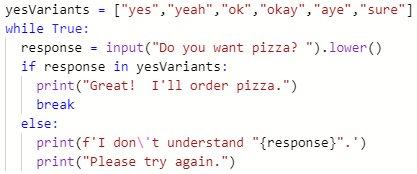
**Dwayne the tub before I dwown.**

----------------------------------------

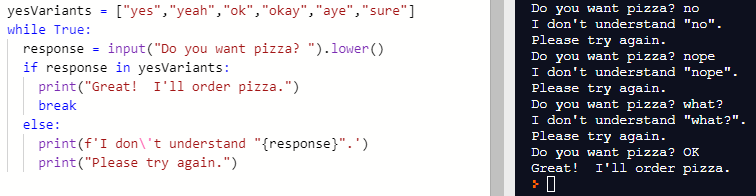
RESOURCES

This project is intended to solidify your problem-solving skills. Here are some resources that may help you: my [knock-knock problem-solving video](https://www.youtube.com/watch?v=cKUOHILgUlY) and the associated [Python problem solving / knock-knock jokes slide presentation](https://docs.google.com/presentation/d/1Pfq80_bnmy86e7XDkOhdMynquQUJ5OOEGYKohjq8VsQ/edit?usp=sharing).

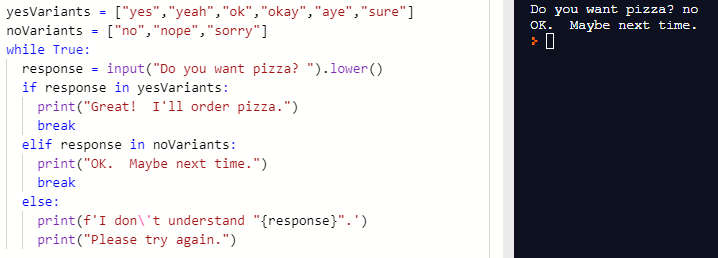
DEALING WITH RESPONSE VARIANTS

Suppose you ask a “yes or no” question and want to be able to handle a variety of affirmative user responses. The first thing you ought to do to handle response variants is to convert the response to lowercase … essentially eliminating a large number of cases making the response case-insensitive. There are many ways people colloquially express an affirmative response (see [42 ways to say “Yes” in English](https://www.inklyo.com/ways-to-say-yes-in-english/) if you’re interested). For this demonstration, we’ll stick with these common words and spellings: yes, yeah, ok, okay, aye and sure. To program this, make a list of the response variants and see if the lowercase version of the user’s response is in the list. Put the process in a while loop and don’t exit until you get the correct response. 

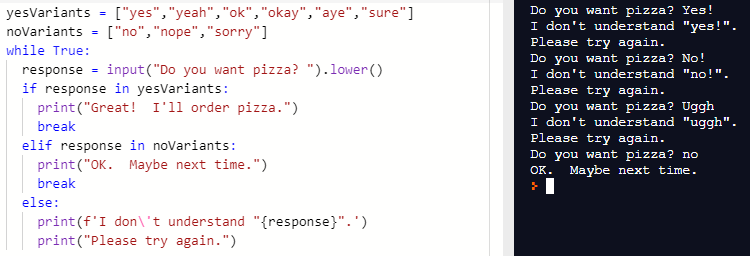
But wait. If we don’t also include negative response variants, this program will never end. This program doesn’t take “no” for an answer!

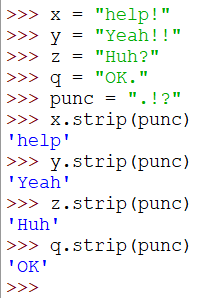


So, we need negative response variants so we can determine if the answer is negative. Common ways to say “no” are: no, nope, and sorry. So, lets add some negative responses exactly the same way …

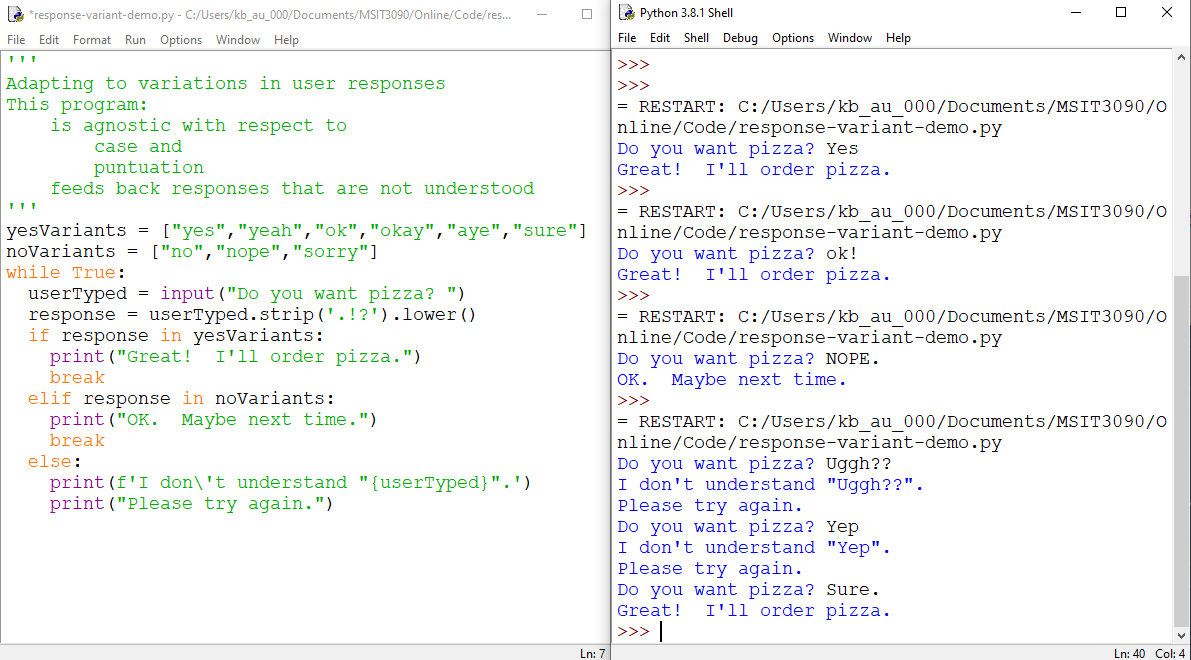


But there’s still a problem. What about punctuation? Things will not go well if the user includes a period or exclamation point at the end of the response. The punctuation will make the response unrecognizable.

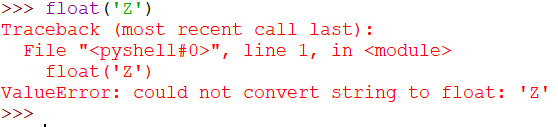


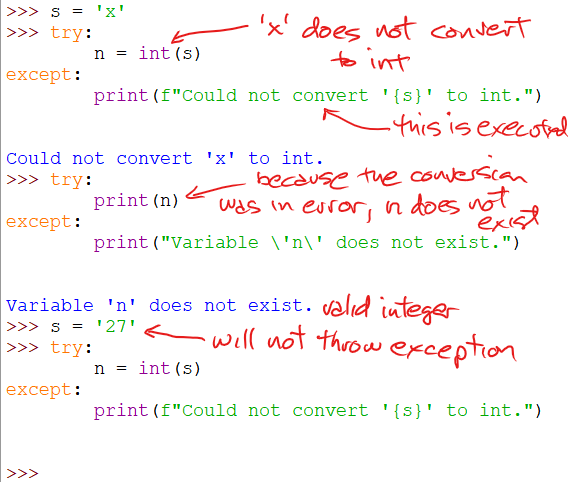
If you’re paying attention to detail, the exchanges demonstrated above reveal two things. We must evaluate the response without the punctuation and also, we are feeding the lowercase version of the response to the user. We ought to be feeding back the actual user response. So, in the next version, let’s take care of both things. We can make some assumptions that will make handling punctuation easier. To simplify the task, let’s assume that the punctuation will always be at the end of the response and that it will always be either a period or an exclamation point. There is a string method called **strip()** that can help. The **strip()** method removes whitespace or any other characters you specify from the beginning or end of a string. Take a look at the shell demo on the right. Try it yourself! 

As you can see below, all of the discussed issues are now fixed. There are, of course, other ways to handle them. Below,, the resulting code is quite readable.



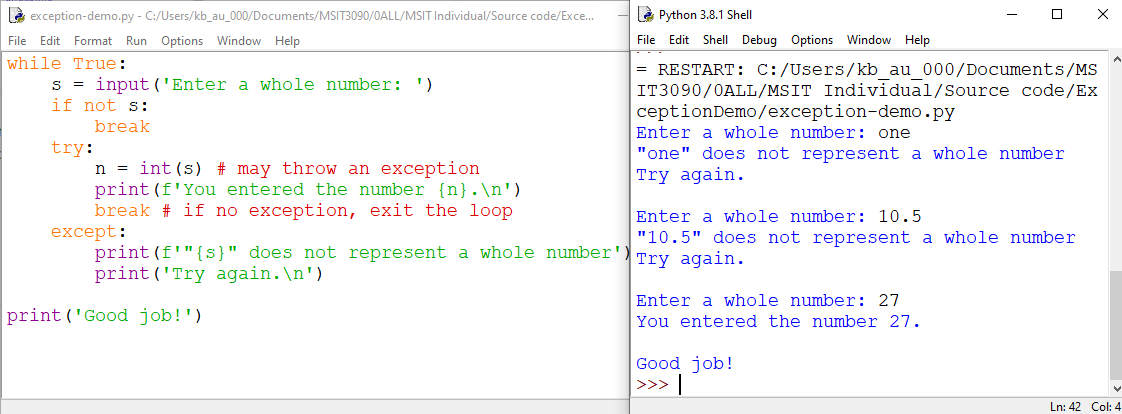
If you now understand how to make your code respond appropriately to a variety of user responses and apply this to the knock-knock jokes problem … go get some pizza! You deserve it!

DEALING WITH EXCEPTIONS

A runtime error is called an exception. If you try to convert the letter “Z” to a float, an exception will be raised called a “ValueError”. We say that a statement *raises* an error. To prevent the program from crashing, the program must *catch* the exception and *handle* it. In Python, we do this using a construct called a try / except block. First we *try* and if trying raises an exception we catch it using *except*. Here’s an example where I demonstrate try / except in the shell …

Two runtime errors are generated. The first is a *ValueError* because s does not contain a string that can be converted to an integer. The second is a *NameError* since **n** was not assigned a value due to the *ValueError*.

Here’s a short program that demonstrates try / except:



In the example, the program loops continuously until the user enters a value that can be converted to an integer or enters nothing at all (challenge: make the program shorter with an expression using the walrus operator).

When you have the power to handle exceptions, you can prevent program crashes at runtime. The description above just covers the very minimum. Like most aspects of a programming language, exception handling can be a very deep topic. Python even allows you to define your own exceptions … way beyond the context of this course.