(lab) Knock Knock Joke

Start Assignment

- Due Friday by 11:59pm
- Points 100
- Submitting a file upload

Overview

Your second project will be an interactive knock knock joke. The computer will have a 'conversation' with the player which will involve a knock-knock joke. There will be some twists, though, as the joke will be optional, and we'll check to see if the player is playing along correctly. A run of the program might look like this:

```
What is your name? Andy
Hi, Andy, do you want to hear a joke? (Y/N) Yes
Knock knock.
Who's there?
Boo.
Boo Who?
No need to cry.
```

Although this is quite a simple program, there is a fair amount of sophistication built in that you can't see from this run. Watch what happens if the user doesn't play along:

```
What is your name? Evil Andy
Hi, Evil Andy, do you want to hear a joke? (Y/N) No
It was going to be really funny.
```

Somehow the computer will decide whether or not to tell the joke. Also, the 'do you want to hear a joke' question is more sophisticated than it looks. Any input that begins with a 'y' or 'Y' will play the joke. If the response begins with any other letter, we will get the "It was going to be funny" comment from the now dejected artificial agent. Great. You've made the computer sad.

Also, the game will not continue of the user doesn't respond correctly:

```
What is your name? Evil Andy
Hi, Evil Andy, do you want to hear a joke? (Y/N) you betcha
Knock knock.
I refuse to conform to institutional norms
You were supposed to say 'Who's there?'
```

There's a couple of interesting things that happened here.

- 'you betcha' begins with a 'y' so it counts as 'yes'
- My noble attempt at individualism did not count as a proper response
- The capitalization of "who's there?" doesn't matter, but the spelling and punctuation does

- The computer gave a response and quit

The program also checks to see that the user answered the joke prompt correctly. Here's another run:

```
What is your name? Evil Andy
Hi, Evil Andy, do you want to hear a joke? (Y/N) you can't make me
Knock knock.
who's there?
Boo.
knock knock jokes are a social construct
You were supposed to say 'Boo who?'
```

For a program that's about telling jokes, this thing has no imagination. Here's a couple of thoughts about this run

- "You can't make me" means "no" but because the computer is looking for a leading "y" it reads it as "yes"
- We aren't going to worry about that now. You can get a graduate degree in natural language processing that tries to solve this kind of problem, but not today, Zurg.
- If I don't say "Boo who?" it politely tells me what to do.

Process

This is our first non-linear program. That means the general behavior of the program can change on the fly. It's a key concept in programming, and it's also something that can go truly wrong if you don't know how to think about it. This week we're going to spend even more time thinking about pseudo-code. We need to know exactly what we want the computer to do **before** we focus on how to do it. Many beginners jump straight to code. Many beginners never become experts. Think about it.

New Ideas

Like every program this semester, this project builds on the previous ideas, and adds some new ones. Here's the highlights:

- How to think through more complex logic in pseudo-code
- How to construct a condition, and how they work
- How to build an if statement
- How indentation works in Python
- How and when to use the : character in Python
- How to use the 'else' clause for more sophisticated conditional behavior
- How to use the input function without a prompt
- How to ignore case on input
- How to check for the first character of an variable
- How to nest if statements for sophisticated behavior
- How to test program flow

- What string methods are and how to use them to test and manipulate text

Planning the Joke

The first thing you'll need is the joke. Here's a few guidelines about picking a joke:

- There are hundreds of resources online for finding knock knock jokes.
- I like this one: **https://parade.com/944054/parade/knock-knock-jokes/** **(https://parade.com/944054/parade/knock-knock-jokes/)**
- Please keep the jokes family-friendly. I have delicate sensibilities.
- Stay with jokes that follow the standard pattern. 'Impatient Cow' is a classic, but it won't work here.
- The joke itself doesn't matter much, but if I groan out loud or see a new one, that's a bonus.
- Only knock-knock jokes today. We're going to take advantage of the formulaic pattern of this type of joke as we practice structuring our logic.

Copy your joke down and write it in your documentation. You don't want to be thinking about the joke when you're writing or debugging your code. These are separate problems to be solved in their own time.

Be sure you've gone over how to use the 'if' structure. It seems very simple, but there are a lot of details you'll need to understand to get it to work correctly:

- What is a condition?
- What type of value does the if statement generate?
- How do you tell the program what to do if the condition is true?
- What's the special usage of the : character?
- What's special about the next line after an if statement?
- How do you do multiple lines of code if a condition is true?
- How do you manage code that should happen if a condition is false?
- Should you be using words like 'and' and 'or' in your conditions yet? (spoiler alert: No.)
- Is an if statement used for repeating behavior?
- How does 'elif' fit into the 'if - else' structure
- Can you have 'elif' or 'else' without 'if?'

Also review methods of the str (string) object in Python. Make sure you know how to:

- Convert the case of a string
- Look at the first letter of a string
- See if a string is inside another string
- be familiar with the other string methods in case they become handy
- look at the documentation of a Python object to see what it can do

Now you need to design your logic. Do this on paper or whiteboard. Be sure to think through your logic before you write your code. This is really important. We will show you a couple of tools including flowcharting and pseudocode. You can think through the logic without understanding every nuance of syntax, and it's a really good idea to do so. Once you've got the logic working, it's a separate step to turn that logic into actual code.

Your documentation should include pseudocode of your logic. You can also produce a flowchart if you wish (**http://mermaid.live** **(http://mermaid.live)** is a good online editor.) However, you **must** include pseudo-code, as it's quite a bit easier to produce and actually much more flexible and informative.

Here's some helpful clues about this week's base project:

- You will need input and print statements
- You will need if statements with conditions
- You will need else statements
- You will not need elif (although you can use them on the black belt)
- You will not need logic operators like 'and' and 'or'
- You will need nested if statements
- You'll have to carefully plan how the various if statements fit inside each other
- You will need some string manipulation methods
- The code this week is ridiculously easy. It is the logic that is tricky. Focus on that first
- Indentation is absolutely critical this week. The logic will change if you do not indent correctly.
- Figure out your indentation in the pseudocode
- Work with your partner on pseudocode. Test each other's ideas before writing code

Writing the Code

You can actually test your pseudocode before you write a single line of code, and that's a great idea. "Be the computer" and follow the steps of your pseudo-code line by line. Check to see if your logic makes sense. Once you feel like you know what you're doing, you can begin to convert each line of pseudo-code into a line of code. This part should be very easy.

Save and test your code frequently. It's easier to find mistakes in two lines of new code than in twenty.

Be sure to work with your partner as you build code. You each need to write your own program, but help each other with implementation and debugging.

Testing

This code is non-linear, so you'll need to test it multiple times. First, try it with obvious answers and make sure the logic works if the user is not an idiot. Then try all the variations. See if you can break the program by putting in wrong inputs (It's possible, but pretty difficult with this project) Be sure that you get the intended behavior you are looking for in each run-through of the program. Be certain you check all the ways things can go wrong.

Note, it's really tempting to have the program repeat if the user gets something wrong, but that is **not** required this week. In fact, you'll find it quite difficult to get the looping behavior to act correctly, and you'll see why when we start talking about loops in a few weeks. Feel free to try looping behavior in a black belt version, but do not be upset if it doesn't work exactly right.

Blackbelt Extension

You have many choices this week.

- You can try to do multiple jokes.
- You can try to implement looping
- You can look at a data structure (list or dictionary) for storing joke data
- You can experiment with other types of jokes
- You can try to improve the error handling

If you try one of these things and fail, that's perfectly OK. The idea here is once you've got the basic version going, see what you can do with it to push the envelope.

Turning it in

Please save your main program as <username>_knock.py. So my project would be ajharris8_knock.py

Save your documentation as <username>_knock_docs.txt (or whatever extension you want)

If you have a blackbelt, save it as <username>_knock_blackbelt.py

Upload all files to canvas

Have fun!

General Assignment rubric

Criteria	Ratings		Pts
Algorithm and pseudocode Describe clearly the steps needed to solve the problem. Best solution is in a separate file, with each concept broken into small enough steps they can be implemented in a single line of code. Later in the semester, functions and classes should have their own algorithms. Can be in a text or readme file. For some projects, a diagram may also be useful or necessary.	**50 pts Full Marks**	**0 pts No Marks**	50 pts
Follows the guidelines Every assignment has specific guidelines listed in canvas. Full points will be awarded for following these guidelines. You will earn fewer points if the guidelines are not completely met.	**20 pts Full Marks**	**0 pts No Marks**	20 pts
Code runs as expected The best solution has all code working without modification. This involves eliminating syntax and logic errors, as well as being thoughtful on file naming conventions, documentation, and including all required dependencies.	**20 pts Full Marks**	**0 pts No Marks**	20 pts
Personal Style A great program shows creativity and flair. Did you make some efforts on personalizing the project to make it uniquely your own?	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Pushing the envelope Once you've got everything working, what did you do to push beyond the basics? This can be a blackbelt extension, or it can be some kind of extra effort implementing a new feature. Make sure the effort you are making here is clearly marked in your documentation so we can see it.	**5 pts Full Marks**	**0 pts No Marks**	5 pts
			Total Points: 100