CS 145 Programming Task 0: Agreement A.Y. 2023-2024, 2nd Semester

1 Introduction

Alice wanted to talk to Bob, but they didn't agree on the time!

2 Details

This is a simulation of the way communication works through a channel (such as a physical wire) and some common challenges when dealing with them.

Write a program that has two functions, `sender` and `receiver`, representing Alice and Bob, respectively.

Your program will executed twice, across *two different processes*. In one process, the `sender` function will be called, and in the other, the `receiver` function will be called.

The `sender` function will be given a string argument named `sentence` which consists of five English words. The `receiver` doesn't have the sentence (yet)—the goal is for the `receiver` to receive the sentence from the `sender`. However, the only means of communication between them is through a certain channel which only supports a single stream of bits.

Both functions will be given an object of type `Channel`, which is the interface for the said channel. The following methods are available:

- `channel.send(bit: int) -> None`. Calling this method with argument `0` or `1` will send that bit through the channel. This method is only available to the sender.
- `channel.get() -> int`. Calling this method will return the next bit through the stream. This method is only available to the receiver.

Note that the `sender` and the `receiver` shouldn't communicate in any way other than through the channel.

Your score will depend on the number of sentences the receiver received, as well as the total number of bits that the sender sent.

3 The channel

Recall that Alice and Bob didn't agree on the time. So, we will simulate the fact that the receiver doesn't necessarily know the beginning and end of the sender's data as follows:

- An unspecified number of `0` bits will be *prepended* to the stream. The number of prepended bits will vary from 100 to 1000, inclusive, and will be chosen randomly. This means that the first actual bit from the sender that the receiver reads may actually be 101st or the 1001st bit in the most extreme cases.
- After the sender ends its output, the channel doesn't "close down"; the receiver may still read bits from it! All bits after the end of the sender's stream will be `0`s.

The goal is for the sender to send the whole sentence exactly, character by character (including spaces, capitalization, and punctuation marks) to the receiver *despite* the fact that you don't know where the stream starts or ends!

4 The sentence

The sentence is formed by choosing 3 to 5 words from a corpus of English words. All the words are available at `corpus.txt`. All words in this corpus are lowercase letters and dashes. These words are then joined by spaces and formatted like an English sentence.

Here are some examples of sentences:

- `Heal the world.`
- `The more the merrier.`
- `Parcel fell to the ocean.`

The exact procedure is given in Algorithm 1.

Algorithm 1 The algorithm to generate a sentence.

function MAKESENTENCE
  Choose the number of words w randomly between 3 and 5.
  repeat w **times**
    Choose a word randomly from `corpus.txt`.
  end
  Join the w words with single spaces.
  Capitalize the first letter.
  Append the dot character '`.`'.
end function

5 Implementation

Your task is to modify `task0.py` by implementing the `sender` and `receiver` functions to achieve the goal stated above.

You may create other functions and global variables, although keep in mind that since the sender and receiver are in separate processes, they don't necessarily share the same variables, so you can't communicate "through global variables".

Important: Do not read from stdin and print to stdout! The standard I/O streams will be used to simulate communication. If you wish to print for debugging purposes, use stderr. (Simply pass `file=stderr` to your `print` calls.)

6 Testing

The sender process can be run on its own with the following command:

```
python3 task0.py sender
```

while the receiver process can be run on its own with the following command:

```
python3 task0.py receiver
```

When running the sender process on its own, the sentence will be taken from the standard input. (So you may enter the sentence manually.) For example, to use the sentence "`The world wonders.`" you could run

```
echo The world wonders. | python3 task0.py sender
```

When running the receiver process on its own, the bits will also be taken from the standard input. For example,

```
# sends 00101
printf "00101" | python3 task0.py receiver

# sends "00101" repeated 1000 times
printf "00101%.0s" {1..1000} | python3 task0.py receiver
```

Note that whitespace will be ignored, but anything other than `0` or `1` will be treated as an error:

```
# this raises an error (assuming the receiver
#     tries to read enough bits from the stream)
printf "00101 We are two. 00101" | python3 task0.py receiver
```

To test both of your sender and receiver simultaneously through the channel, run

```
python3 -m cs145lib.task0.test  [COMMAND]
```

where `[COMMAND]` stands for the command to run the sender or receiver. For example,

```
python3 -m cs145lib.task0.test  python3 task0.py
```

The sentence that will be given to the sender will be taken from the standard input, so for example, to use the sentence "`We are one.`" you could run

```
echo We are one. | python3 -m cs145lib.task0.test python3 task0.py
```

To generate a random sentence, you can run

```
python3 -m cs145lib.task0.make_sentence
```

This will print a random sentence.

As usual, you can combine this with the testing script above via pipe, e.g.,

```
python3 -m cs145lib.task0.make_sentence \
    | python3 -m cs145lib.task0.test  python3 task0.py
```

These commands and scripts accept a number of arguments to help with testing. You can run the following commands to see the available options and their documentation:

```
# to see the options available when running the sender/receiver
python3 task0.py --help

# to see the options available when running the testing script
python3 -m cs145lib.task0.test --help

# to see the options available when running the sentence generator
python3 -m cs145lib.task0.make_sentence --help
```

For example, there are options to choose a random seed, choose a different number of `0`s to prepend (instead of between 100 and 1000), etc. For example, here are some ways to use the script:

```
# the number of prepended zeroes is between 10 and 20
echo The world wonders. | \
    python3 -m cs145lib.task0.test --zero-pad 10 20 \
    python3 task0.py

# use the random seed 314159
echo The world wonders. | \
    python3 -m cs145lib.task0.test --seed 314159 \
    python3 task0.py

# verbose output
echo The world wonders. | \
    python3 -m cs145lib.task0.test --verbose \
    python3 task0.py

# to make a random sentence with the random seed 314159
python3 -m cs145lib.task0.make_sentence --seed 314159

# to test on a fixed random sentence
# and a fixed number of prepended bits
python3 -m cs145lib.task0.make_sentence --seed 314159 | \
    python3 -m cs145lib.task0.test --seed 314159 \
    python3 task0.py
```

Note that `--zero-pad`, `--seed`, and `--verbose` are equivalent to `-z`, `-s` and `-v`, respectively.

7 Example

The file `sample.py` contains an example implementation. Note that it doesn't actually attempt to send the sentence; the sender just sends a bunch of random bits and the receiver just constructs an arbitrary string from the stream (not even a valid sentence). Its only purpose is to show you how to use the `channel` object passed to the sender and receiver, as well as how to print to `stderr` (e.g., for debugging).

8 Scoring

Your solution will be tested against 100 randomly-generated sentences of the above form.

Let b be the *total* number of bits sent by the sender across all 100 cases, and let $x := \lg b$ (base-2 logarithm). Then your score is computed as follows:

$$\text{score} = \begin{cases} 0 & \text{if } x > 18.8 \\ 100 & \text{if } x < 12.8 \\ 154 - 5x & \text{otherwise} \end{cases}$$

Thus, the fewer the number of bits, the higher your score.

Minus Wrong: In addition, for every sentence that the receiver fails to receive, 1.5 **points is subtracted off your score**.

Time Limit: You also get 0 points if your solution took at least 10 seconds on at least 3 sentences.

If the score ends up negative (due to the Minus Wrong rule), it will be considered 0. Thus, your score is between 0 and 100, inclusive. A score of 60 or above will be considered passing.

9 Submission

Since you're only modifying `task0.py`, you need only submit that file. Don't include anything else.

10 Notes

The corpus is taken from `https://github.com/sibosop/speclib/blob/master/corncob_lowercase.txt`