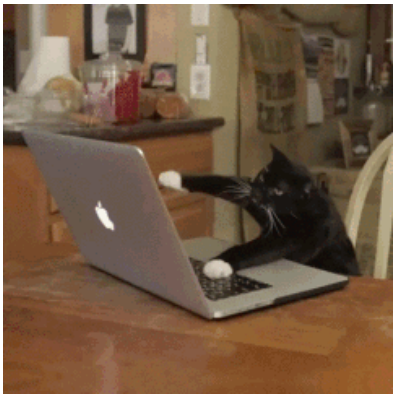
Project 2: CS 61A Autocorrected Typing Software

cats.zip (cats.zip)



Programmers dream of
Abstraction, recursion, and
Typing really fast.

Introduction

> **Important submission note:** For full credit:
>
> - Submit Phase 1 by **Monday, 9/30**.
> - Submit the whole project by **Thursday, Oct 3**.
> - You will receive an early submission bonus point for submitting the entire project by **Wednesday, Oct 2**.
>
> You may work with a partner for the entire project.

In this project, you will write a program that measures typing speed. Additionally, you will implement typing autocorrect, which is a feature that attempts to correct the spelling of a word after a user types it. This project is inspired by Type Racer (https://play.typeracer.com/).

Final Product

Our staff solution to the project can be interacted with at cats.cs61a.org (https://cats.cs61a.org) - if you'd like, try it out now! When you finish the project, you'll have implemented a significant part of this game yourself!

Download starter files

You can download all of the project code as a zip archive (cats.zip). This project includes several files, but your changes will be made to only `typing.py`. Here are the files included in the archive:

- `typing.py`: The typing test logic.
- `utils.py`: Functions for interacting with files and strings.
- `data/sample_paragraphs.txt`: A file containing text samples to be typed.
- `data/words.txt`: A file containing English words in order of frequency (https://github.com/first20hours/google-10000-english/blob/master/google-10000-english-usa-no-swears.txt).

- `gui.py` : A web server for the web-based graphical user interface (GUI).
- `gui` : A directory of files needed for the graphical user interface (GUI).
- `ucb.py` : Utility functions for CS 61A.
- `images` : A directory of images.
- `ok` , `proj02.ok` , `tests` : Testing files.

Logistics

The project is worth 20 points. 17 points are assigned for correctness of your final code, 1 point for submitting Phase 1 by the checkpoint deadline, and 2 points for composition.

You will turn in the following files:

- `typing.py`

You do not need to modify or turn in any other files to complete the project. To submit the project, run the following command:

```
python3 ok --submit
```

You will be able to view your submissions on the Ok dashboard (http://ok.cs61a.org).

For the functions that we ask you to complete, there may be some initial code that we provide. If you would rather not use that code, feel free to delete it and start from scratch. You may also add new function definitions as you see fit.

However, please do **not** modify any other functions. Doing so may result in your code failing our autograder tests. Also, please do not change any function signatures (names, argument order, or number of arguments).

Throughout this project, you should be testing the correctness of your code. It is good practice to test often, so that it is easy to isolate any problems. However, you should not be testing *too* often, to allow yourself time to think through problems.

We have provided an **autograder** called `ok` to help you with testing your code and tracking your progress. The first time you run the autograder, you will be asked to **log in with your Ok account using your web browser**. Please do so. Each time you run `ok` , it will back up your work and progress on our servers.

The primary purpose of `ok` is to test your implementations.

We recommend that you submit **after you finish each problem**. Only your last submission will be graded. It is also useful for us to have more backups of your code in case you run into a submission issue.

If you do not want us to record a backup of your work or information about your progress, you can run

```
python3 ok --local
```

With this option, no information will be sent to our course servers. If you want to test your code interactively, you can run

```
python3 ok -q [question number] -i
```

with the appropriate question number (e.g. `01`) inserted. This will run the tests for that question until the first one you failed, then give you a chance to test the functions you wrote interactively.

You can also use the debug printing feature in OK by writing

```
print("DEBUG:", x)
```

which will produce an output in your terminal without causing OK tests to fail with extra output.

Phase 1: Typing

Problem 1 (1 pt)

Implement `choose`, which selects which paragraph the user will type. It takes a list of `paragraphs` (strings), a `select` function that returns `True` for paragraphs that can be selected, and a non-negative index `k`. The `choose` function return's the `k` th paragraph for which `select` returns `True`. If no such paragraph exists (because `k` is too large), then `choose` returns the empty string.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 01 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 01
```

Problem 2 (2 pt)

Implement `about`, which takes a list of `topic` words. It returns a function that can be passed to `choose` as the `select` argument. The returned function takes a paragraph and returns whether that paragraph contains any of the words in `topic`.

To make this comparison accurately, you will need to ignore case (that is, assume that uppercase and lowercase letters don't change what word it is) and punctuation.

Assume that all words in the `topic` list are already lowercased and do not contain punctuation.

> **Hint**: You may use the string utiltiy functions in `utils.py`.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 02 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 02
```

Problem 3 (1 pt)

Implement `accuracy`, which takes a `typed` paragraph and a `reference` paragraph. It returns the percentage of words in `typed` that exactly match the corresponding words in `reference`. Case and punctuation must match as well.

A *word* in this context is any sequence of characters separated from other words by whitespace, so treat "dog;" as all one word.

If a typed word has no corresponding word in the reference because `typed` is longer than `reference`, then the extra words in `typed` are all incorrect.

If `typed` is empty, then the accuracy is zero.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 03 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 03
```

Problem 4 (1 pt)

Implement `wpm`, which computes the *words per minute*, a measure of typing speed, given a string `typed` and the amount of `elapsed` time in seconds. Despite its name, *words per minute* is not based on the number of words typed, but instead the number of characters, so that a typing test is not biased by the length of words. The formula for *words per minute* is the ratio of the number of characters typed divided by 5 (a typical word length) to the elapsed time in minutes.

For example, the string `"I am glad!"` contains three words and ten characters (not including the quotation marks). The words per minute calculation uses 2 as the number of words typed (because 10 / 5 = 2). If someone typed this string in 30 seconds (half a minute), their speed would be 4 words per minute.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 04 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 04
```

Time to test your typing speed! You can use the command line to test your typing speed on paragraphs about a particular topic. For example, the command below will load paragraphs about cats or kittens. See the `run_typing_test` function for the implementation if you're curious (but it is defined for you).

```
python3 typing.py -t cats kittens
```

You can try out the web-based graphical user interface (GUI) using the following command. This interface picks random paragraphs instead of choosing by topic.

```
python3 gui.py
```

To submit your Phase 1 checkpoint type:

```
python3 ok --submit
```

You can submit again once you've finished the whole project, and we will score only your latest submission, but please submit at least once before the checkpoint deadline (after finishing at least the Phase 1 questions) to receive credit for the checkpoint.

Phase 2: Autocorrect

In the web-based GUI, there is an autocorrect button, but right now it doesn't do anything. Let's implement automatic correction of typos. Whenever the user presses the space bar, if the last word they typed doesn't match a word in the dictionary but is close to one, then that similar word will be substituted for what they typed.

Problem 5 (2 pt)

Implement `autocorrect`, which takes a `user_word`, a list of all `valid_words`, a `diff_function`, and a `limit`.

If the `user_word` is contained inside the `valid_words` list, `autocorrect` returns that word. Otherwise, `autocorrect` returns the word from `valid_words` that has the lowest difference from the provided `user_word` based on the `diff_function`. However, if the difference is greater than `limit`, then `user_word` is returned instead.

A diff function takes in three arguments, which are the two strings to be compared (first the `user_word` and then a word from `valid_words`), as well as the `limit`. The output of the diff function, which is a number, represents the amount of difference between the two strings.

Assume that `user_word` and all elements of `valid_words` are lowercase and have no punctuation.

Important: if multiple strings have the same lowest difference according to the `diff_function`, `autocorrect` should return the string that appears first in `valid_words`.

> **Hint**: Try using `max` or `min` with the optional `key` argument.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 05 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 05
```

Problem 6 (2 pts)

Implement `swap_diff`, which is a diff function that takes two strings. It returns the minimum number of characters that must be changed in the `start` word in order to transform it into the `goal` word. If the strings are not of equal length, the difference in lengths is added to the total.

Here are some examples:

```
>>> big_limit = 10
>>> swap_diff("nice", "rice", big_limit)    # Substitute: n -> r
1
>>> swap_diff("range", "rungs", big_limit)  # Substitute: a -> u, e -> s
2
>>> swap_diff("pill", "pillage", big_limit) # Don't substitute anything, length difference of 3.
3
>>> swap_diff("roses", "arose", big_limit)  # Substitute: r -> a, o -> r, s -> o, e -> s, s -> e
5
```

If the number of characters that must change is greater than `limit`, then `swap_diff` should return any number larger than `limit` and should minimize the amount of computation needed to do so.

These two calls to `swap_diff` should take about the same amount of time to evaluate:

```
>>> limit = 4
>>> swap_diff("roses", "arose", limit) > limit
True
>>> swap_diff("rosesabcdefghijklm", "arosenopqrstuvwxyz", limit) > limit
True
```

Important: You may not use `while` or `for` statements in your implementation. Use recursion.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 06 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 06
```

Try turning on autocorrect in the GUI. Does it help you type faster? Are the corrections accurate? You should notice that inserting a letter or leaving one out near the beginning of a word is not handled well by this diff function. Let's fix that!

Problem 7 (3 pt)

Implement `edit_diff`, which is a diff function that returns the minimum number of edit operations needed to transform the `start` word into the `goal` word.

There are three kinds of edit operations:

1. Add a letter to `start`,
2. Remove a letter from `start`,
3. Substitute a letter in `start` for another.

Each edit operation contributes 1 to the difference between two words.

```
>>> big_limit = 10
>>> edit_diff("roses", "arose", big_limit)     # roses -> aroses -> arose
2
>>> edit_diff("tesng", "testing", big_limit)   # tesng -> testng -> testing
2
>>> edit_diff("rlogcul", "logical", big_limit) # rlogcul -> logcul -> logicul -> logical
3
```

We have provided a template of an implementation in `typing.py`. This is a recursive function with three recursive calls. One of these recursive calls will be similar to the recursive call in `swap_diff`.

You may modify the template however you want or delete it entirely.

If the number of edits required is greater than `limit`, then `edit_diff` should return any number larger than `limit` and should minimize the amount of computation needed to do so.

These two calls to `edit_diff` should take about the same amount of time to evaluate:

```
>>> limit = 2
>>> edit_diff("rlogcul", "logical", limit) > limit
True
>>> swap_diff("rlogculabcdefghijklm", "logicalnopqrstuvwxyz", limit) > limit
True
```

Test your implementation before proceeding:

```
python3 ok -q 07
```

Try typing again. Are the corrections more accurate?

```
python3 gui.py
```

Extensions: You may optionally design your own diff function called `final_diff`. Here are some ideas for making even more accurate corrections:

- Take into account which additions and deletions are more likely than others. For example, it's much more likely that you'll accidentally leave out a letter if it appears twice in a row.

- Try to incorporate common misspellings

Phase 3: Multiplayer

Typing is more fun with friends! You'll now implement multiplayer functionality, so that when you run `gui.py` on your computer, it connects to the course server at cats.cs61a.org (https://cats.cs61a.org) and looks for someone else to race against.

To race against a friend, 5 different programs will be running:

- Your GUI, which is a program that handles all the text coloring and display in your web browser.
- Your `gui.py`, which is a web server that communicates with your GUI.
- Your opponent's `gui.py`.
- Your opponent's GUI.
- The CS 61A multiplayer server, which matches players together and passes messages around.

When you type, your GUI sends what you have typed to your `gui.py` server, which computes how much progress you have made and returns a progress update. It also sends a progress update to the multiplayer server, so that your opponent's GUI can display it.

Meanwhile, your GUI display is always trying to keep current by asking for progress updates from `gui.py`, which in turn requests that info from the multiplayer server.

Each player has an `id` number that is used by the server to track typing progress.

Problem 8 (2 pt)

Implement `report_progress`, which is called every time the user finishes typing a word. It takes a list of the words `typed`, a list of the words in the `prompt`, the user `id`, and a `send` function that is used to send a progress report to the multiplayer server.

Your progress is a ratio of the words in the `prompt` that you have typed correctly, up to the first incorrect word, divided by the number of `prompt` words. For example, this example has a progress of `0.25`:

```
report_progress(["Hello", "ths", "is"], ["Hello", "this", "is", "wrong"], ...)
```

Your `report_progress` function should return this number. Before that, it should end a message to the multiplayer server that is a two-element dictionary containing the keys `'id'` and `'progress'`. The `id` is passed into `report_progress` from the GUI. The progress is the fraction you compute. Call `send` on this dictionary to send it to the multiplayer server.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 08 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 08
```

Problem 9 (3 pt)

Implement `fastest_words`, which returns which words each player typed fastest. This function is called once both players have finished typing. It takes `word_times` and a positive `margin`.

The `word_times` argument is a list of lists of `word_time` values, one list for each player, and within each list `n+1` elements for the total elapsed time in the race after that player has finished typing each of the `n` words, as well as an entry at the beginning with zero elapsed time for the special word `'START'`.

It returns a list of lists of words, one list for each player, and within each list the words they typed the fastest.

Definition: A player typed a word the *fastest* if the difference between their elapsed time for that word and the previous word is within `margin` of the smallest difference for any player. Therefore, if two players type a word within `margin` of each other, that word will appear in both of their lists.

Be sure to use the `word` and `elapsed_time` accessor functions for the `word_time` data abstraction, rather than assuming a particular data format.

Before writing any code, unlock the tests to verify your understanding of the question.

```
python3 ok -q 09 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 09
```

Congratulations! Now you can play against other students in the course. Set `enable_multiplayer` to `True` near the bottom of `typing.py` and type swiftly!

```
python3 gui.py
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)