Cats

● Graded

Group

Seong Jae Ahn
Sangwon Ji
View or edit group

Total Points

22 / 20 pts

Autograder Score

22.0 / 20.0

Autograder Results

```
==================================================================
Assignment: Project 2: Cats
OK, version v1.18.1
==================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

------------------------------------------------------------
Problem 1
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

------------------------------------------------------------
Problem 2
Passed: 2
Failed: 0
[ooooooooook] 100.0% passed

------------------------------------------------------------
Problem 3
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

------------------------------------------------------------
Problem 4
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

------------------------------------------------------------
Problem 5
Passed: 1
```

```
Failed: 0
[ooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 6
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 7
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 8
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 9
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 10
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Extra Credit
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Point breakdown
Problem 1: 1.0/1
Problem 2: 2.0/2
Problem 3: 2.0/2
Problem 4: 1.0/1
Problem 5: 2.0/2
Problem 6: 3.0/3
Problem 7: 3.0/3
Problem 8: 2.0/2
Problem 9: 2.0/2
Problem 10: 2.0/2
Extra Credit: 1.0/1
```

Score:
Total: 21.0

Cannot backup when running ok with --local.

Final Score:22.0
Early Submission. Bonus Point Included

Submitted Files

cats.py Download

```
"""Typing test implementation"""

from utils import lower, split, remove_punctuation, lines_from_file, count, deep_convert_to_tuple
from ucb import main, interact, trace
from datetime import datetime


###########
# Phase 1 #
###########


def pick(paragraphs, select, k):
    """Return the Kth paragraph from PARAGRAPHS for which SELECT called on the
    paragraph returns True. If there are fewer than K such paragraphs, return
    the empty string.

    Arguments:
        paragraphs: a list of strings
        select: a function that returns True for paragraphs that can be selected
        k: an integer

    >>> ps = ['hi', 'how are you', 'fine']
    >>> s = lambda p: len(p) <= 4
    >>> pick(ps, s, 0)
    'hi'
    >>> pick(ps, s, 1)
    'fine'
    >>> pick(ps, s, 2)
    ''
    """
    # BEGIN PROBLEM 1
    "*** YOUR CODE HERE ***"
    selected_paragraphs=[] # make new array for post-selected paragraphs
    for p in paragraphs:
        if select(p): # if it's TURE
            selected_paragraphs.append(p)

    if k>= len(selected_paragraphs):
        return ''
    # for example ,there are 'hi', 'fine' index 0,1. if(k=2) 2 >= 1 -> return ''
    return selected_paragraphs[k] # k below the len, it returns selected_paragraphs[k]


    # END PROBLEM 1


def about(subject):
    """Return a select function that returns whether
```

```
    a paragraph contains one of the words in SUBJECT.

    Arguments:
        subject: a list of words related to a subject

    >>> about_dogs = about(['dog', 'dogs', 'pup', 'puppy'])
    >>> pick(['Cute Dog!', 'That is a cat.', 'Nice pup!'], about_dogs, 0)
    'Cute Dog!'
    >>> pick(['Cute Dog!', 'That is a cat.', 'Nice pup.'], about_dogs, 1)
    'Nice pup.'
    """
    assert all([lower(x) == x for x in subject]), 'subjects should be lowercase.'
    # BEGIN PROBLEM 2
    "*** YOUR CODE HERE ***"

    def check_subject_in_paragraph(paragrah):   #same as dogs('A paragraph about cats.')
        paragrah = lower(paragrah)
        paragrah = remove_punctuation(paragrah)
        para_words=paragrah.split()

        for word_para in para_words:
            for words_subject in subject:
                if words_subject == word_para:
                    return True


        return False


    return check_subject_in_paragraph



    # # END PROBLEM 2


def accuracy(typed, source):
    """Return the accuracy (percentage of words typed correctly) of TYPED
    when compared to the prefix of SOURCE that was typed.

    Arguments:
        typed: a string that may contain typos
        source: a string without errors

    >>> accuracy('Cute Dog!', 'Cute Dog.')
    50.0
    >>> accuracy('A Cute Dog!', 'Cute Dog.')
    0.0
    >>> accuracy('cute Dog.', 'Cute Dog.')
    50.0
    >>> accuracy('Cute Dog. I say!', 'Cute Dog.')
    50.0
```

```
    >>> accuracy('Cute', 'Cute Dog.')
    100.0
    >>> accuracy('', 'Cute Dog.')
    0.0
    >>> accuracy('', '')
    100.0
    """


    typed_words = split(typed)
    source_words = split(source)
    # BEGIN PROBLEM 3
    "*** YOUR CODE HERE ***"
    #start with empty
    count_equal =0
    if len(typed_words)==0 and len(source_words)==0:
        return 100.0
    if len(typed_words)==0 and len(source_words)!=0:
        return 0.0

    for typ_word,s_word in zip(typed_words,source_words):
        if typ_word == s_word:
            count_equal+=1

    accuracy=(count_equal / len(typed_words))*100.0

    return accuracy

    # END PROBLEM 3


def wpm(typed, elapsed):
    """Return the words-per-minute (WPM) of the TYPED string.

    Arguments:
        typed: an entered string
        elapsed: an amount of time in seconds

    >>> wpm('hello friend hello buddy hello', 15)
    24.0
    >>> wpm('0123456789',60)
    2.0
    """
    assert elapsed > 0, 'Elapsed time must be positive'
    # BEGIN PROBLEM 4
    "*** YOUR CODE HERE ***"
    number_words = len(typed)/ 5
    elapsed_min = elapsed/60

    return number_words/elapsed_min


```

```
    # END PROBLEM 4


#################
# Phase 4 (EC) #
#################


def memo(f):
    """A general memoization decorator."""
    cache = {}

    def memoized(*args):
        immutable_args = deep_convert_to_tuple(args)  # convert *args into a tuple representation
        if immutable_args not in cache:
            result = f(*immutable_args)
            cache[immutable_args] = result
            return result
        return cache[immutable_args]
    return memoized

def memo_diff(diff_function):
    """A memoization function."""
    cache = {}

    def memoized(typed, source, limit):

        key = (typed, source)
        if (key in cache) and (limit <= cache[key][1]):
            return cache[key][0]
        result_diff = diff_function(typed, source, limit)
        cache[key] = (result_diff, limit)
        return result_diff

        # END PROBLEM EC

    return memoized



############
# Phase 2 #
############

@memo
def autocorrect(typed_word, word_list, diff_function, limit):
    """Returns the element of WORD_LIST that has the smallest difference
    from TYPED_WORD. If multiple words are tied for the smallest difference,
    return the one that appears closest to the front of WORD_LIST. If the
    difference is greater than LIMIT, instead return TYPED_WORD.

    Arguments:
```

```
206         typed_word: a string representing a word that may contain typos
207         word_list: a list of strings representing source words
208         diff_function: a function quantifying the difference between two words
209         limit: a number
210 
211     >>> ten_diff = lambda w1, w2, limit: 10 # Always returns 10
212     >>> autocorrect("hwllo", ["butter", "hello", "potato"], ten_diff, 20)
213     'butter'
214     >>> first_diff = lambda w1, w2, limit: (1 if w1[0] != w2[0] else 0) # Checks for matching first char
215     >>> autocorrect("tosting", ["testing", "asking", "fasting"], first_diff, 10)
216     'testing'
217     """
218     # BEGIN PROBLEM 5
219     if typed_word in word_list:
220         return typed_word
221     # min_diff = float('inf') # initialize as max value of python to assign curr_diff.
222     # min_diff_word = '' # empty word initialized.
223 
224     # for element in word_list:
225     #     curr_diff = diff_function(typed_word,element,limit)
226     #     if curr_diff < min_diff:
227     #         min_diff = curr_diff
228     #         min_diff_word = element # pair asssign min_diff & min_diff_word
229 
230     def limits(word):
231         return diff_function(typed_word, word, limit)
232     closest_word = min(word_list, key = limits)
233     if diff_function(typed_word, closest_word, limit) > limit:
234         return typed_word
235     else:
236         return closest_word
237 
238     # END PROBLEM 5
239 
240 
241 
242 
243 def feline_fixes(typed, source, limit):
244     """A diff function for autocorrect that determines how many letters
245     in TYPED need to be substituted to create SOURCE, then adds the difference in
246     their lengths and returns the result.
247 
248     Arguments:
249         typed: a starting word
250         source: a string representing a desired goal word
251         limit: a number representing an upper bound on the number of chars that must change
252 
253     >>> big_limit = 10
254     >>> feline_fixes("nice", "rice", big_limit)    # Substitute: n -> r
255     1
256     >>> feline_fixes("range", "rungs", big_limit)  # Substitute: a -> u, e -> s
257     2
```

```
    >>> feline_fixes("pill", "pillage", big_limit) # Don't substitute anything, length difference of 3.
    3
    >>> feline_fixes("roses", "arose", big_limit)  # Substitute: r -> a, o -> r, s -> o, e -> s, s -> e
    5
    >>> feline_fixes("rose", "hello", big_limit)   # Substitute: r->h, o->e, s->l, e->l, length difference of 1.
    5
    """
    # BEGIN PROBLEM 6
    # len_of_diff = abs(len(typed)-len(source))
    # subst =0
    # for typ, src in zip(typed,source):
    #     if typ!=src:
    #         subst +=1
    # total = len_of_diff + subst
    # if total <= limit :
    #     return total
    # else :
    #     return float('inf')
    # count=0
    # if not typed or not source:
    #     return abs(len(typed) - len(source))

    # if limit < 0:
    #     return float('inf')

    # count = int(typed[0] != source[0]) # first character check and move to the next index.

    # rest = feline_fixes(typed[1:], source[1:], limit - count) # index [1:]checks the rest

    # return count + rest

    if limit == 0:
        if typed == source:
            return 0
        else:
            return 1
    elif not typed or not source:
        return abs(len(typed) - len(source))
    elif typed[0] != source[0]:
        typed, source = typed[1:], source[1:]
        return 1 + feline_fixes(typed,source, limit -1)
        #return feline_fixes(typed, source, limit)
    else:
       typed, source = typed[1:], source[1:]
       return feline_fixes(typed, source, limit)
    # END PROBLEM 6

@memo_diff
def minimum_mewtations(typed, source, limit):

    """A diff function that computes the edit distance from TYPED to SOURCE.
```

```
    This function takes in a string TYPED, a string SOURCE, and a number LIMIT.
    Arguments:
        typed: a starting word
        source: a string representing a desired goal word
        limit: a number representing an upper bound on the number of edits
    >>> big_limit = 10
   >>>  minimum_mewtations("cats", "scat", big_limit)       # cats -> scats -> scat
    2
    >>> minimum_mewtations("purng", "purring", big_limit)   # purng -> purrng -> purring
    2
    >>> minimum_mewtations("ckiteus", "kittens", big_limit) # ckiteus -> kiteus -> kitteus -> kittens
    3
    """

    if typed == source or limit ==0:  # Base cases should go here, you may add more base cases as needed.
        if typed == source:
            return 0
        else:
            return limit + 1

    # Recursive cases should go below here
    if typed =="" or source == "":
        return abs(len(typed)- len(source))

    if typed[0] == source[0]:  # if the first characters are the same, skip to the next character without using a limit

        return minimum_mewtations(typed[1:], source[1:], limit)

    else:

        add = 1 + minimum_mewtations(typed,source[1:],limit -1)  # Fill in these lines
        remove = 1+ minimum_mewtations(typed[1:],source,limit -1)
        substitute = 1 + minimum_mewtations(typed[1:],source[1:],limit -1) # start from index 1 and deduct 1 from the limit

        # BEGIN

        return min(add,remove,substitute)

        # END Problem 7


# ignore the line below

minimum_mewtations = count(minimum_mewtations)


def final_diff(typed, source, limit):
    """A diff function that takes in a string TYPED, a string SOURCE, and a number LIMIT.
    If you implement this function, it will be used."""
```

```
    assert False, 'Remove this line to use your final_diff function.'


FINAL_DIFF_LIMIT = 6  # REPLACE THIS WITH YOUR LIMIT


###########
# Phase 3 #
###########


def report_progress(typed, prompt, user_id, upload):
    """Upload a report of your id and progress so far to the multiplayer server.
    Returns the progress so far.

    Arguments:
        typed: a list of the words typed so far
        prompt: a list of the words in the typing prompt
        user_id: a number representing the id of the current user
        upload: a function used to upload progress to the multiplayer server

    >>> print_progress = lambda d: print('ID:', d['id'], 'Progress:', d['progress'])
    >>> # The above function displays progress in the format ID: __, Progress: __
    >>> print_progress({'id': 1, 'progress': 0.6})
    ID: 1 Progress: 0.6
    >>> typed = ['how', 'are', 'you']
    >>> prompt = ['how', 'are', 'you', 'doing', 'today']
    >>> report_progress(typed, prompt, 2, print_progress)
    ID: 2 Progress: 0.6
    0.6
    >>> report_progress(['how', 'aree'], prompt, 3, print_progress)
    ID: 3 Progress: 0.2
    0.2
    """
    # BEGIN PROBLEM 8
    "*** YOUR CODE HERE ***"
    # correct_words = 0
    # for i in range(len(typed)):
    #     if typed[i] == prompt[i]:  # if the words match, increase the counter
    #         correct_words += 1
    #     else:  # stopls counting as soon as we encounter a word that does not match
    #         break

    correct = 0
    for typ,prmp in zip(typed,prompt):
        if typ == prmp:  # if the words match, increase the counter
            correct+= 1 # checks each pair of elements from the start
        else:  # even it's correct afterwards, we stop immediately.
            break # finish for loop

    progress_prption = correct / len(prompt) # proportion of the progress
    progress_rpt_id_and_prgres = {'id': user_id, 'progress': progress_prption}
```

```
    upload(progress_rpt_id_and_prgres) # upload function returns ID: # Progress: # format

    return progress_prption


    # END PROBLEM 8


def time_per_word(words, times_per_player):
    """Given timing data, return a match data abstraction, which contains a
    list of words and the amount of time each player took to type each word.

    Arguments:
        words: a list of words, in the order they are typed.
        times_per_player: A list of lists of timestamps including the time
                          the player started typing, followed by the time
                          the player finished typing each word.

    >>> p = [[75, 81, 84, 90, 92], [19, 29, 35, 36, 38]]
    >>> match = time_per_word(['collar', 'plush', 'blush', 'repute'], p)
    >>> get_all_words(match)
    ['collar', 'plush', 'blush', 'repute']
    >>> get_all_times(match)
    [[6, 3, 6, 2], [10, 6, 1, 2]]
    """
    # BEGIN PROBLEM 9
    times = [] # times[i][j] is the time it took player i to type words[j].
    for player_times in times_per_player:
        player_times_diff =[]
        for i in range(len(player_times)-1):
            player_times_diff.append( player_times[i+1] - player_times[i]) # [1,2,3,4]
        times.append(player_times_diff) #[[1,2,3,4],[2,3,4]]

    return match(words, times)




    # END PROBLEM 9


def fastest_words(match):
    """Return a list of lists of which words each player typed fastest.

    Arguments:
        match: a match data abstraction as returned by time_per_word.

    >>> p0 = [5, 1, 3]
    >>> p1 = [4, 1, 6]
    >>> fastest_words(match(['Just', 'have', 'fun'], [p0, p1]))
    [['have', 'fun'], ['Just']]
    >>> p0  # input lists should not be mutated
```

```
    [5, 1, 3]
    >>> p1
    [4, 1, 6]
    """
    player_indices = range(len(get_all_times(match)))  # contains an *index* for each player
    word_indices = range(len(get_all_words(match)))    # contains an *index* for each word

    # BEGIN PROBLEM 10

    each_times = get_all_times(match)
    each_words = get_all_words(match)
    fastest_words =[]
    for _ in range(len(player_indices)):
        fastest_words.append([])

    for word_index in word_indices:
        word = each_words[word_index]
        player_times = [each_times[player][word_index] for player in player_indices]
        fastest_player = player_times.index(min(player_times))
        fastest_words[fastest_player].append(word)

    return fastest_words

    # all_times = get_all_times(match)  # all times
    # all_words = get_all_words(match)
    # fastest_words = [[] for _ in range(len(all_times))]

    # for word_index, word in range(len(all_words)):
    #     fastest_player = min(range(len(all_times)), key=lambda player: all_times[player][word_index])
    #     fastest_words[fastest_player].append(word)

    # return fastest_words

    # END PROBLEM 10


def match(words, times):
    """A data abstraction containing all words typed and their times.

    Arguments:
        words: A list of strings, each string representing a word typed.
        times: A list of lists for how long it took for each player to type
            each word.
            times[i][j] = time it took for player i to type words[j].

    Example input:
        words: ['Hello', 'world']
        times: [[5, 1], [4, 2]]
    """
    assert all([type(w) == str for w in words]), 'words should be a list of strings'
    assert all([type(t) == list for t in times]), 'times should be a list of lists'
    assert all([isinstance(i, (int, float)) for t in times for i in t]), 'times lists should contain numbers'
```

```
    assert all([len(t) == len(words) for t in times]), 'There should be one word per time.'
    return {"words": words, "times": times}


def get_word(match, word_index):
    """A utility function that gets the word with index word_index"""
    assert 0 <= word_index < len(get_all_words(match)), "word_index out of range of words"
    return get_all_words(match)[word_index]


def time(match, player_num, word_index):
    """A utility function for the time it took player_num to type the word at word_index"""
    assert word_index < len(get_all_words(match)), "word_index out of range of words"
    assert player_num < len(get_all_times(match)), "player_num out of range of players"
    return get_all_times(match)[player_num][word_index]


def get_all_words(match):
    """A selector function for all the words in the match"""
    return match["words"]


def get_all_times(match):
    """A selector function for all typing times for all players"""
    return match["times"]


def match_string(match):
    """A helper function that takes in a match data abstraction and returns a string representation of it"""
    return f"match({get_all_words(match)}, {get_all_times(match)})"


enable_multiplayer = False  # Change to True when you're ready to race.

##########################
# Command Line Interface #
##########################


def run_typing_test(topics):
    """Measure typing speed and accuracy on the command line."""
    paragraphs = lines_from_file('data/sample_paragraphs.txt')
    select = lambda p: True
    if topics:
        select = about(topics)
    i = 0
    while True:
        source = pick(paragraphs, select, i)
        if not source:
            print('No more paragraphs about', topics, 'are available.')
            return
```

```
        print('Type the following paragraph and then press enter/return.')
        print('If you only type part of it, you will be scored only on that part.\n')
        print(source)
        print()

        start = datetime.now()
        typed = input()
        if not typed:
            print('Goodbye.')
            return
        print()

        elapsed = (datetime.now() - start).total_seconds()
        print("Nice work!")
        print('Words per minute:', wpm(typed, elapsed))
        print('Accuracy:        ', accuracy(typed, source))

        print('\nPress enter/return for the next paragraph or type q to quit.')
        if input().strip() == 'q':
            return
        i += 1


@main
def run(*args):
    """Read in the command-line argument and calls corresponding functions."""
    import argparse
    parser = argparse.ArgumentParser(description="Typing Test")
    parser.add_argument('topic', help="Topic word", nargs='*')
    parser.add_argument('-t', help="Run typing test", action='store_true')

    args = parser.parse_args()
    if args.t:
        run_typing_test(args.topic)

```