18.2 Data Structures Practice Exercises

October 11, 2021

1 1 product.py

[ ]:
```
def product(a, b):
    """Return product of a and b.

    """
    return a*b
print(product("!", 10))
```

```
!!!!!!!!!!
```

2 2 weekday_name.py

[ ]:
```
def weekday_name(day_of_week):
    """Return name of weekday.

    """
    days = [None, "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",␣
↪"Friday", "Saturday"]
    if day_of_week > 7 or day_of_week < 1:
        return None
    else:
        return days[day_of_week]
print(weekday_name(3))
print(weekday_name(.5))
```

```
Tuesday
None
```

3 3 last_element.py

[ ]:
```
def last_element(lst):
    """Return last item in list (None if list is empty.

    """
    if not lst:
        return None
```

```
    else:
        return lst[-1]
print(last_element([1,2,3]))
print(last_element([]))
```

```
3
None
```

4 4 number_compare

[ ]:
```
def number_compare(a, b):
    """Report on whether a>b, b>a, or b==a


    """
    if a == b:
        return 'Numbers are equal'
    elif a < b:
        return 'Second is greater'
    else:
        return 'First is greater'
```

5 5 reverse string

[ ]:
```
def reverse_string(phrase):
    """Reverse string,

    """
    chars = list(phrase)
    chars.reverse()
    return "".join(chars)
print(reverse_string("hello"))
print(reverse_string("12345"))
```

```
olleh
54321
```

6 5 reverse string using start:stop:stride

6.1 items start through stop-1

`a[start:stop]` ## items start through the rest of the array `a[start:]` ## items from the beginning through stop-1 `a[:stop]` ## a copy of the whole array `a[:]` ## last item in the array `a[-1]` ## last two items in the array
`a[-2:]` ## everything except the last two items `a[:-2]`

[ ]:
```
word = "hello"
word[-1] # "o"
word[::-2] # "olh"

word[::-1] # "olleh"
def negstride(thing):
    print(thing[-1::-1], "thing[-1::-1]")
    return thing[::-1]
print(negstride(word), "word[::-1]")
```

```
olleh thing[-1::-1]
olleh word[::-1]
```

7 negative stride

[ ]:
```
word = "bonjour"
print(word[::-1]) # all items in the array, reversed
print(word[1::-1]) # first two items in reverse
print(word[:-3:-1]) # last two items (-1,-2) reversed
print(word[-3::-1]) # all items reversed, ommiting -2 and -1
```

```
ruojnob
ob
ru
ojnob
```

8 6 single letter occurrence count

[ ]:
```
def single_letter_count(word, letter):
    """How many times does letter appear in word (case-insensitively)?


    """
    lword = word.lower()
    count = lword.count(letter.lower())
    return count
print(single_letter_count("YOLO", "O"))
```

```
2
```

9 7 multiple letter count

[ ]:
```
def multiple_letter_count(phrase):
    """Return dict of {ltr: frequency} from phrase.
    """
    # make a set of each unique letter in phrase
```

```
    #  phrase.count(letter) for each unique letter in phrase
    #       store the result in dict
    unique_letters = set(phrase)
    letter_count = dict()
    for letter in unique_letters:
        letter_count[letter] = phrase.count(letter)
    return letter_count
print(multiple_letter_count("Yay"))
```

```
{'y': 1, 'a': 1, 'Y': 1}
```

10 8 list manipulation

[ ]:
```
def list_manipulation(lst, command, location, value=None):
    """Mutate lst to add/remove from beginning or end.

    - lst: list of values
    - command: command, either "remove" or "add"
    - location: location to remove/add, either "beginning" or "end"
    - value: when adding, value to add


    """
    # validate input
    if (command != 'add') and (command != 'remove'):
        #print("invalid input")
        return None
    elif (location != "beginning") and (location != "end"):
        #print("invalid location")
        return None

    # figure out which command to do
    if command == "add":
        if location == "end":
            #print('adding to the end')
            lst.append(value)
            return lst
        elif location == "beginning":
            lst.insert(0,value)
            return lst
    if command == "remove":
        if location == "beginning":
            return lst.pop(0)
        elif location == "end":
            return lst.pop()
lst = [1, 2, 3]
list_manipulation(lst, 'remove', 'end')
```

```
print(list_manipulation(lst, 'remove', 'beginning'))
```

```
1
```

11 9 Is_Palindrome

[ ]:
```
def is_palindrome(phrase):
    """Is phrase a palindrome?

    Return True/False if phrase is a palindrome (same read backwards and
    forwards).


    """
    reverse = phrase[::-1].replace(" ","")
    no_spaces = phrase.replace(" ","")
    if reverse.lower() == no_spaces.lower():
        return True
    else:
        return False
is_palindrome("Noon")
is_palindrome('robert')
```

[ ]:
```
False
```

12 10 frequency

[ ]:
```
def frequency(lst, search_term):
    """Return frequency of term in lst.

    """
    return lst.count(search_term)
```

13 11 flip_case

[ ]:
```
def flip_case(phrase, to_swap):
    """Flip [to_swap] case each time it appears in phrase.



    """
    lswap = to_swap.lower()
    swapped = ""
    for letter in phrase:
        if (letter.lower() == lswap):
```

```
            swapped += letter.swapcase()
        else:
            swapped += letter
    return swapped
```

14 12 multiply even numbers

[ ]:
```
def multiply_even_numbers(nums):
    """Multiply the even numbers.


    """
    # check for even number
    product = 1
    for number in nums:
        if number % 2: # if number is odd it will have 0/False remainder
            continue
        else:
            product *= number
    return product
multiply_even_numbers([1, 3, 5])
```

[ ]:
```
1
```

15 13 capitalize

[ ]:
```
def capitalize(phrase):
    """Capitalize first letter of first word of phrase.

    """
    return phrase.capitalize()
```

16 14 compact

[ ]:
```
def compact(lst):
    """Return a copy of lst with non-true elements removed.

    """
    true_list = []
    for element in lst:
        if element:
            true_list.append(element)
    return true_list
```

17 15 intersection

[ ]:
```
def intersection(l1, l2):
    """Return intersection of two lists as a new list::

    """
    one = set(l1)
    two = set(l2)
    intersect = one & two
    return list(intersect)
```

18 16 Partition

[ ]:
```
def partition(lst, fn):
    """Partition lst by predicate.

    """
    trues = []
    falses = []
    for element in lst:
        if fn(element):
            trues.append(element)
        else:
            falses.append(element)
    divided = []
    divided.append(trues)
    divided.append(falses)
    return divided
```

19 17 Mode

[ ]:
```
def mode(nums):
    """Return most-common number in list.

    For this function, there will always be a single-most-common value;
    you do not need to worry about handling cases where more than one item
    occurs the same number of times.


    """
    the_mode = 0
    for number in nums:
        if nums.count(number) > the_mode:
            the_mode = number
    return the_mode
```

20 18 Calculate

[ ]:
```
def calculate(operation, a, b, make_int=False, message='The result is'):
    """Perform operation on a + b, ()possibly truncating) & returning w/msg.

    - operation: 'add', 'subtract', 'multiply', or 'divide'
    - a and b: values to operate on
    - make_int: (optional, defaults to False) if True, truncates to integer
    - message: (optional) message to use (if not provided, use 'The result is')

    Performs math operation (truncating if make_int), then returns as
    "[message] [result]"
    """
    if operation != "add" and operation != "subtract" and operation !=␣
↪"multiply" and operation != "divide":
        return None

    if operation == "add":
        result = a + b
    elif operation == "subtract":
        result = a - b
    elif operation == "multiply":
        result = a * b
    elif operation == "divide":
        result = a / b




    if make_int:
        message += " " + repr(int(result))
        return message
    else:
        message += " " + repr(result)
        return message
```

21 19 Friend date

[ ]:
```
def friend_date(a, b):
    """Given two friends, do they have any hobbies in common?

    - a: friend #1, a tuple of (name, age, list-of-hobbies)
    - b: same, for friend #2

    Returns True if they have any hobbies in common, False is not.

        True
```

```
    """
    l1, l2 = a[2], b[2]
    for x in l1:
        for y in l2:
            if y == x:
                return True
    return False

elmo = ('Elmo', 5, ['hugging', 'being nice'])
sauron = ('Sauron', 5000, ['killing hobbits', 'chess'])
friend_date(elmo, sauron)
```

[ ]: False

22 19.1 Friend Date Using Sets

22.1 if the intersection of setfromtuplea and setfromtupleb is non-empty

```
if set(a[2]) & set(b[2]):
```

[ ]:
```
def friend_date2(a, b):
    """Given two friends, do they have sny hobbies in common?

    - a: friend #1, a tuple of (name, age, list-of-hobbies)
    - b: same, for friend #2

    Returns True if they have any hobbies in common, False is not.

    """

    if set(a[2]) & set(b[2]):
        return True
    else:
        return False
```

23 20 Triple and filter

[ ]:
```
def triple_and_filter(nums):
    """Return new list of tripled nums for those nums divisible by 4.

    Return every number in list that is divisible by 4 in a new list,
    except multipled by 3.


    """
    tripled = [num *3 for num in nums if num % 4 == 0]
    return tripled
```

24 21 Extract full name

[ ]:
```
def extract_full_names(people):
    """Return list of names, extracting from first+last keys in people dicts.

    - people: list of dictionaries, each with 'first' and 'last' keys for
              first and last names

    Returns list of space-separated first and last names.


    """
    space = " "
    full_names = [(person["first"] + space + person["last"]) for person in␣
 ↪people]

    return full_names
names = [
    {'first': 'Ada', 'last': 'Lovelace'},
    {'first': 'Grace', 'last': 'Hopper'},
]
extract_full_names(names)
```

[ ]:
```
['Ada Lovelace', 'Grace Hopper']
```

25 22 Sum floats

[ ]:
```
def sum_floats(nums):
    """Return sum of floating point numbers in nums.


    """

    # hint: to find out if something is a float, you should use the
    # "isinstance" function --- research how to use this to find out
```

```
    # if something is a float!
    summation = 0
    for number in nums:
        if isinstance(number, float):
            summation += number
    return summation
```

26 23 List check

[ ]:
```
def list_check(lst):
    """Are all items in lst a list?


    """
    for item in lst:
        if not isinstance(item, list):
            return False

    return True
```

27 24 Remove every other

[ ]:
```
def remove_every_other(lst):
    """return every other item in a new list"""
    every_other = lst[::2]
    return every_other
```

28 25 Sum of pairs

[ ]:
```
def sum_pairs(nums, goal):
    """Return tuple of first pair of nums that sum to goal.


    """
    # set of numbers previously used
    seen = set()

    for num2 in nums:
        # valid numbers add up goal
        match = goal - num2

        # if present valid match number in seen return the tuple
        if match in seen:
            return (match, num2)
        # else add number to set of seen numbers
```

```
        seen.add(num2)
    # if no pair sums to goal then return an empty tuple
    return ()
sum_pairs([5, 1, 4, 8, 3, 2], 7)
```

[ ]: (4, 3)

29 26 vowel count

[ ]:
```
def vowel_count(phrase):
    """Return frequency map of vowels, case-insensitive.
    """

    lphrase = phrase.lower()
    vowels = "aeiou"
    count = {}
    # returns a dict with vowels in order of occurrence
    for letter in lphrase:
        if letter in vowels:
            count[letter] = lphrase.count(letter)
            vowels.replace(letter,"")
    return count
    """
    This version does not pass the stupid tests because
    the tests order the counted vowels in the dictionary
    by occurrence.

    Version 1
    for vowel in vowels:
        if vowel in lphrase:
            count[vowel] = lphrase.count(vowel)
    """
```

30 27 titleize

[ ]:
```
def titleize(phrase):
    """Return phrase in title case (each word capitalized).

    """
    # using built-in function .title()
    titleized = phrase.title()
    return titleized
```

31 28 find factors

[ ]:
```python
def find_factors(num):
    """Find factors of num, in increasing order.

    """
    # possible integer factors are in the range such that:
    # 1 <= possible factors <= (number / 2)
    possible_factors = range(1, int(num / 2 + 1))
    factors = [number for number in possible_factors if not num % number]
    # lastly, add num to list
    factors.append(num)
    return factors
```

32 29 includes

[ ]:
```python
def includes(collection, sought, start=None):
    """Is sought in collection, starting at index start?

    Return True/False if sought is in the given collection:
    - lists/strings/sets/tuples: returns True/False if sought present
    - dictionaries: return True/False if *value* of sought in dictionary

    If string/list/tuple and `start` is provided, starts searching only at that
    index. This `start` is ignored for sets/dictionaries, since they aren't
    ordered.

    """
    # Check collection type

    if isinstance(collection, set):
        # if collection is a set return the result of this 'in' statement
        return sought in collection

    if isinstance(collection,dict):
        # look for sought in the _dict_values_ object from .values()
        return sought in collection.values()

    return sought in collection[start:]
    """
    Long Version:
    if sought in collection[start:]:
        return True
    return False"""
```

33 30 repeat

[ ]:
```
def repeat(phrase, num):
    """Return phrase, repeated num times.

    """
    # validate num value
    if not isinstance(num, int):
        return None
    if num < 0:
        return None
    echo = phrase * num
    return echo
```

34 31 truncate

[ ]:
```
def truncate(phrase, n):
    """Return truncated-at-n-chars version of  phrase.

    """

    msg = 'Truncation must be at least 3 characters.'
    if n < 3:
        return msg
    # if n is large enough for the whole phrase, do nothing
    if n >= len(phrase) + 3:
        return phrase
    truncated = phrase[:n-3] + "..."
    return truncated
```

35 32 two list dictionary

[ ]:
```
def two_list_dictionary(keys, values):
    """Given keys and values, make dictionary of those.

    """
    new_dict = {}
    for key in keys:
        index = keys.index(key)
        if index >= len(values):
            new_dict[key] = None
        else:
            new_dict[key] = values[index]


    return new_dict
```

36 33 sum range

[ ]:
```python
def sum_range(nums, start=0, end=None):
    """Return sum of numbers from start...end.

    - start: where to start (if not provided, start at list start)
    - end: where to stop (include this index) (if not provided, go through end)

    """
    # using the built in sum() function
    summation = sum(nums[start:])
    if end:
        summation = sum(nums[start:end+1])
    return summation
```

37 34 same freq

[ ]:
```python
def same_frequency(num1, num2):
    # turn each number into a string
    str1, str2 = repr(num1), repr(num2)
    count1, count2 = {}, {}
    digits = "0123456789"
    for num in digits:
        count1[num] = str1.count(num)
        count2[num] = str2.count(num)
    return count1 == count2
```

38 35 two oldest

[ ]:
```python
def two_oldest_ages(ages):
    """Return two distinct oldest ages as tuple (second-oldest, oldest)..

    """

    # NOTE: don't worry about an optimized runtime here; it's fine if
    # you have a runtime worse than O(n)

    # NOTE: you can sort lists with lst.sort(), which works in place (mutates)
    # you may find it helpful to research the `sorted(iter)` function, which
    # can take *any* type of list-like-thing, and returns a new, sorted list
    # from it.
    uniques = set(ages)
    oldest = None
    second_oldest = None
    for age in uniques:
```

```
        if oldest is None or age > oldest:
            second_oldest = oldest
            oldest = age
        elif second_oldest is None or age > second_oldest:
            second_oldest = age
    return (second_oldest, oldest)
```

39 36 find the duplicate

[ ]:
```
def find_the_duplicate(nums):
    """Find duplicate number in nums.
    Given a list of nums with, at most, one duplicate, return the duplicate.
    If there is no duplicate, return None

    """
    unique_numbers = set()
    for number in nums:
        if number not in unique_numbers:
            unique_numbers.add(number)
        elif number in unique_numbers:
            return number
```

40 37 sum up diagonals in a matrix

40.1 the key to this function's syntax is in the list notation[][] where I use

40.2 negative indices to access the 'last', 'second-to-last', 'third to last' elements

40.3 in order to access the TL-BR diagonal in a matrix. the first diagonal is easily found

40.4 because it's indices are always:

`[0][0]`, `[1][1]`, `[2][2]` ## the second diagonal is not so easily expressed without negative indices: `[0][-1] = [0][last element]`

[ ]:
```
def sum_up_diagonals(matrix):
    """Given a matrix [square list of lists], return sum of diagonals.

    Sum of TL-to-BR diagonal along with BL-to-TR diagonal:

    """
    summation = 0
    matrix_size = len(matrix)
    #for element in
    # range = [0,1,2, ... up to but excluding (n = len(matrix))]
    for element in range(matrix_size):
        # matrix[0][0] + matrix[1][1]+ matrix[highest row][highest column]
```

```
        summation += matrix[element][element]
        # matrix[0][-1] means get me the last element of row 0, i.e. the corner
        # for element = 1, this evals to matrix[1][-1-1] = matrix[1][-2]
        # i.e. the second to last element in the second row
        summation += matrix[element][-element-1]  # matrix
    return summation
```

41 38 min max key in dictionary

[ ]:
```
def min_max_keys(d):
    """Return tuple (min-keys, max-keys) in d.
    """
    keys = d.keys() # get those keys
    minim = min(keys) # call the dict method min on them keys
    maxim = max(keys) # same thing
    return (minim, maxim) # this casts a tuple
```

42 39 find greater numbers

[ ]:
```
def find_greater_numbers(nums):
    """Return # of times a number is followed by a greater number.

    For example, for [1, 2, 3], the answer is 3:
    - the 1 is followed by the 2 *and* the 3
    - the 2 is followed by the 3



    """
    count = 0
    for number in nums:
        for other_numbers in nums:
            if other_numbers > number and nums.index(other_numbers) > nums.
↪index(number):
                count += 1
    return count
```