Midterm Exam A

CS 101 L4 Algorithmic Problem Solving

Fall 2023

Name: ____

HU ID *(e.g., xy01042)*: ________________________________

Please read the following instructions carefully.

Exam Duration 2 hours and 30 minutes

Number of Problems This exam contains 4 problem solving and 1 debugging problems. Attempt all of them.

Grading All problems carry equal marks.

Technical Requirements You are not allowed to use code editors, compilers or online resources during this exam.

Communication During the exam, do not communicate with anyone except the course staff.

Honesty Academic honesty is expected. Do not engage in any form of cheating, plagiarism, or unauthorized collaboration. Violations will be met with disciplinary action as per the Student Code of Conduct.

Answer Format Type your answers clearly in the provided spaces.

Time Management Monitor the time carefully and pace yourself. Do not spend too much time on any one question. It is a good strategy to go over all the problems and attempt the easier ones first.

Exam Completion Before submitting, take a while to ensure that you have attempted all the problems, and that your responses are clear and accurate.

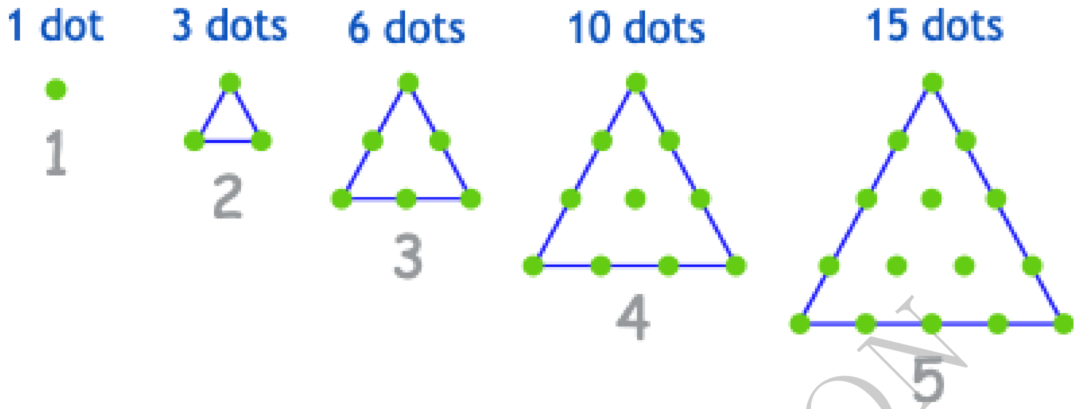
---

viel Spaß und viel Glück!

SAMPLE SOLUTION

1. Triangle Number Sequence

The *triangle number sequence* is given as: $1, 3, 6, 10, 15, 21, 28, 36, 45, \ldots$. Each term in the series can be derived as the number of dots in a triangle, as pictured below.[1]



The i-th term of the series is the total number of dots in the i-th triangle, e.g. the 1st term is 1 which is the total number of dots in triangle number 1, the 4th term is 10 which is the total number of dots in triangle number 4, etc.

Write a function, `triangle_number_sequence`, that takes an integer parameter, `n`, and prints the sequence up to and including term number `n`.

Constraints

- `isinstance(n, int)`
- `n > 0`

Interaction

The input comprises a single number denoting the value of `n`.

The output must contain the sequence up to and including the `n`-th term.

Sample

Input	Output
1	1
4	1 3 6 10

In the first case, `n == 1`. The sequence us printed up to and including the 1st term.

In the second case, `n == 4`. The sequence us printed up to and including the 4th term.

(a) $\boxed{\text{3 points}}$ **Exercise** In the space provided, indicate the outputs for the given inputs.

Input	Output
3	1 3 6
20	1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210

(b) $\boxed{\text{2 points}}$ **Problem Identification** Briefly explain the underlying problem you identified in the above question that led you to your solution.

SAMPLE SOLUTION

[1] Credit: Triangular Number Sequence, Math Is Fun.

Solution:
Input: `n`
Output: The first `n` terms of the triangle number sequence.

(c) $\boxed{\text{3 points}}$ **Pseudocode** After defining the function, make sure to call it to view its output. You may assume that the input has already been taken.

Solution:

```
def triangle_number_sequence(n):
    term = 1
    for i in range(2, n+1):
        print(term, end = ' ')
        term += i
    print(term)

# input n
triangle_number_sequence(n)
```

(d) $\boxed{\text{2 points}}$ **Dry Run** In the space below, show a dry run of your pseudocode using any input from the Exercise section. Include a column for the output.

Solution:

n	i	term	Output
3		1	
	2	3	1
	3	6	1 3
			1 3 6

SAMPLE SOLUTI

2. Real Steel

A company categorizes its produced steel according to the following criteria.

1. *Hardness* must be greater than 50
2. *Carbon content* must be less than 0.7
3. *Tensile strength* must be greater than 5600

Based on the above, a steel is assigned a *grade* as follows.

- Grade 10 if all three conditions are met
- Grade 9 if conditions 1 and 2 are met
- Grade 8 if conditions 2 and 3 are met
- Grade 7 if conditions 1 and 3 are met
- Grade 6 if only one condition is met
- Grade 5 if none of the conditions are met

For example, a steel with `hardness == 60`, `carbon_content ==0.9`, and `tensile_strength ==5000` satisfies condition 1 only. Its `grade` is therefore 6.

Task Write a function, `steel_grade`, that takes `hardness`, `carbon_content`, and `tensile_strength` as parameters and returns the corresponding `grade` of the steel.

Constraints

- `isinstance(hardness, int)and isinstance(carbon_content, float)and isinstance (tensile_strength, int)`
- `hardness > 0`
- `0 < carbon_content <1`
- `tensile_strength >100`

Interaction

The input comprises three space-separated numbers denoting the values of `hardness`, `carbon_content` , and `tensile_strength` respectively.

The output must contain `grade`.

Sample

Input	Output
60 0.9 5000	6
50 0 5600	5

In the first case, only condition 1 is satisfied, so the grade is 6.

In the second case, no condition is satisfied, so the grade is 5.

(a) $\boxed{\text{3 points}}$ **Exercise** In the space provided, indicate the outputs for the given inputs.

Input	Output
62 0.4 5329	9
55 0.9 6000	7
40 0.5 5610	8

SAMPLE SOLUTION

(b) 2 points **Problem Identification** Briefly explain the underlying problem you identified in the above question that led you to your solution.

Solution:
Input: `hardness`, `carbon_content`, and `tensile_strength`
Output: The grade as per the given conditions.

(c) 3 points **Pseudocode** After defining the function, make sure to call it and print its result. You may assume that the input has already been taken.

Solution:

```
def steel_grade(hardness, carbon_content, tensile_strength):
  # see which conditions are true
  bool one = hardness > 50
  bool two = carbon_content < 0.7
  bool three = tensile_strength > 5600
  # assign and return grade
  if one and two and three:
    grade 10
  elif one and two:
    grade = 9
  elif two and three:
    grade = 8
  elif one and three:
    grade = 7
  elif one or two or three:
    grade = 6
  else:
    grade = 5
  return grade

# input hardness, carbon_content, tensile_strength
print(steel_grade(hardness, carbon_content, tensile_strength))
```

(d) 2 points **Dry Run** In the space below, show a dry run of your pseudocode using any input from the Exercise section.

Solution:

hardness	carbon_content	tensile_strength	one	two	three	grade
62	0.4	5329	True	True	False	9
		return 9				

SAMPLE

3. Strong Password

Meeting the following conditions makes a password *strong*.

1. Length must be between 8 and 30 letters (exclusive).
2. There must be at least one upper case character.
3. There must be at least one lower case character.
4. There must be at least one digit.
5. There must be no spaces in the password.

When a password is submitted that fails any of the conditions, an error message is displayed. The error message is the same as the description of the condition as given above. If the submitted password is strong, the displayed message is: "Strong Password!".

Write a function, `check_password_strength`, that takes a string, `password`, as a parameter and returns the corresponding message. The conditions are checked on `password` in the same order as given above, and the returned message contains only the first error, if any. If there is no error, the returned message is "Strong Password!".

Constraints

- `isinstance(password, str)`

Interaction

The input comprises a single line containing `password`.

The output must contain the corresponding message.

Sample

Input	Output
"sd2j42n 35"	"There must be at least one upper case character."
"ASO3QSA"	"Length must be between 8 and 30 letters (exclusive)"

In both cases, the message corresponding to the first unmet condition is displayed.

SAMPLE SOLUTION

Proposed Solution

```python
########## Helper functions
def has_accepted_length(s):
    return 8 < len(s) < 20

def has_uppercase(s):
    return 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' in s

def has_lowercase(s):
    return 'abcdefghijklmnopqrstuvwxyz' in s

def has_digit(s):
    for c in s:
        if c in '0123456789':
            return True
        else:
            return False

def has_space(s):
    return s in ' '

def get_error(n):
    message = ''
    if n == 1:
        message = 'Length must be between 8 and 30 letters (exclusive).'
    if n == 2:
        message = 'There must be at least one upper case character.'
    if n == 3:
        message = 'There must be at least one lower case character.'
    if n == 4:
        message = 'There must be at least one digit.'
    if n == 5:
        message = 'There must be no spaces in the password.'
    if n == 0:
        message = 'Strong Password!'
    return message

########## Main function
def check_password_strength(password):
    message = ''
    if not has_accepted_length(password):
        message = get_error(1)
    if not has_uppercase(password):
        message = get_error(2)
    if not has_lowercase(password):
        message = get_error(3)
    if not has_digit(password):
        message = get_error(4)
    if has_space(password):
        message = get_error(5)
    else:
        message = get_error(0)
    return message
```

(a) $\boxed{\text{3 points}}$ **Dry Run** Pick an input from the Sample section, and show below, the returned value from *each* of the called functions when `check_password_strength` is called with that input.

Solution: The called functions and returned values when `check_password_strength ('ASO3QSA')` is called are:

has_accepted_length('ASO3QSA')	True
has_uppercase('ASO3QSA')	False
has_lowercase('ASO3QSA')	False
has_digit('ASO3QSA')	False
has_space('ASO3QSA')	False
get_error(2)	'There must be at least one upper case character.'
get_error(3)	'There must be at least one lower case character.'
get_error(4)	'There must be at least one digit.'
check_password_strength('ASO3QSA')	'There must be at least one digit.'

(b) $\boxed{\text{4 points}}$ **Error Identification** Briefly explain the errors you identified in the proposed code solution. Mention the line number and errors in each line.

Solution:

Line 3	:	the check is incorrect - 20 should be 30.
Lines 6, 9	:	the check is incorrect - it checks for the entire string of letters in `s`.
Lines 15, 16	:	can incorrectly return False at the very first letter in `s`.
Line 19	:	the check is incorrect - it checks if `s` is in space.
Line 25–33	:	not really an error but there are unnecessary checks.
Lines 42–48	:	`message` can get overwritten by a later error.
Line 52	:	returns whereas the function is required to print.

(c) $\boxed{\text{3 points}}$ **Correct Solution** Rewrite the lines of code you mentioned above with their errors corrected.

Solution:

```
def has_accepted_length(s):
    return 8 < len(s) < 30

def has_uppercase(s):
    for c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
        if c in s:
            return True
    return False

def has_lowercase(s):
    for c in 'abcdefghijklmnopqrstuvwxyz':
        if c in s:
            return True
    return False
```

```python

def has_digit(s):
    for c in s:
        if c in '0123456789':
            return True
    return False

def has_space(s):
    return ' ' in s

def get_error(n):
    message = ''
    if n == 1:
        message = 'Length must be between 8 and 30 letters (exclusive).'
    elif n == 2:
        message = 'There must be at least one upper case character.'
    elif n == 3:
        message = 'There must be at least one lower case character.'
    elif n == 4:
        message = 'There must be at least one digit.'
    elif n == 5:
        message = 'There must be no spaces in the password.'
    elif n == 0:
        message = 'Strong Password!'
    return message

def check_password_strength(password):
    message = ''
    if not has_accepted_length(password):
        message = get_error(1)
    elif not has_uppercase(password):
        message = get_error(2)
    elif not has_lowercase(password):
        message = get_error(3)
    elif not has_digit(password):
        message = get_error(4)
    elif has_space(password):
        message = get_error(5)
    else:
        message = get_error(0)
    print(message)
```

SAM

4. Goldbach Conjecture

Around 1750, the Prussian mathematician Christian Goldbach, proposed that every even number greater than 2 can be written as the sum of 2 primes. Goldbach's conjecture is one of the oldest and best-known unsolved problems in number theory and all of mathematics.

You are setting out to explore it empirically.

Write a function, `check_goldbach`, that takes an integer parameter, `N`, and prints the 2 corresponding prime numbers.

You may assume that you have access to a function, `nth_prime`, that takes an integer parameter, `n`, and returns the `n`-th prime number, with 2 being the first prime, 3 the second prime, 5 the third, and so on. The first 100 primes are given below for reference.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541

Constraints

- `isinstance(N, int)`
- `N >= 4`
- `N % 2 == 0`

Interaction

The input comprises a single number denoting the value of `N`.

The output must contain a pair of prime numbers whose sum is `N`. If multiple pairs are possible, any one pair may be output.

Sample

Input	Output
4	2 2
128	19 109

In the first case, `N == 4` which is the sum of 2 and 2.

In the second case, `N == 10` which is the sum of 19 and 109. Note that other pairs are also possible.

(a) 3 points **Exercise** In the space provided, indicate the outputs for the given inputs.

Input	Output
6	3 3
100	3 97 or 11 89 or 17 83 or 29 71 or 41 59 or 47 53

(b) 2 points **Problem Identification** Briefly explain the underlying problem you identified in the above question that led you to your solution.

Solution:
Input: `N`
Output: 2 primes that add up to N.

SAMPLE SOLUTION

(c) 3 points **Pseudocode** After defining the function, make sure to call it to view its output. You may assume that the input has already been taken.

Solution:

```
def check_goldbach(N):
    # the (N-1)-th prime is larger than N
    for i in range(1, N):
        p1 = nth_prime(i)
        for j in range(i, N):
            p2 = nth_prime(j)
            if p1 + p2 == N:
                print(p1, p2)
                return

# input N
print(check_goldbach(N))
```

(d) 2 points **Dry Run** In the space below, show a dry run of your pseudocode using any input from the Exercise section. Include a column for the output.

Solution:

N	i	p1	j	p2	Output
6	1	2	1	2	
			2	3	
			3	5	
			4	7	
			5	11	
	2	3	2	3	3 3
		return			

SAMPLE SOLUTION

5. Collatz Conjecture

In 1937, Lothar Collatz, a German mathematician, put forth the following conjecture: if you start with any positive integer, `k`, and you half the integer if it is even, or triple it and add one to it if it is odd, and you keep doing that, you will eventually arrive at one. This conjecture has, to date, not been proven to be true or false. However, it has been verified up to a very large integer.

Given `k == 5`, the generated numbers are as follows:

- 5 is odd, so we triple and add 1 to it, yielding: $5 * 3 + 1 = 16$,
- 16 is even, so we halve it, yielding, $16/2 = 8$,
- 8 is even, so we halve it, yielding, $8/2 = 4$,
- 4 is even, so we halve it, yielding, $4/2 = 2$,
- 2 is even, so we halve it, yielding, $2/2 = 1$.

The Collatz sequence corresponding to 5 is thus: 5 16 8 4 2 1. The sequence stops at 1. The Collatz conjecture is that every positive number yields a sequence that stops at 1.

We want to verify this conjecture by looking at the Collatz sequence corresponding to all the numbers from `1` to `n`. Write a function, `show_collatz`, that takes n as a parameter and prints the Collatz sequence of each number from `1` to `n`.

Constraints

- `isinstance(n, int)`
- `n > 0`

Interaction

The input contains a single number denoting the value of `n`.

The output must contain `n` lines, with the first line containing the Collatz sequence for 1, the second line for 2, and so on, with the `n`-th line containing the Collatz sequence for `n`.

Sample

Input	Output
1	1

5	1
	2 1
	3 10 5 16 8 4 2 1
	4 2 1
	5 16 8 4 2 1

In the first case, `n == 1` and the output contains the Collatz sequence for 1.

In the second case, `n == 5` and output contains the Collatz sequence for each number from 1 to 5.

(a) $\boxed{\text{3 points}}$ **Exercise** In the space provided, indicate the outputs for the given inputs.

SAMPLE SOLUTION

Input	Output

4	1
	2 1
	3 10 5 16 8 4 2 1
	4 2 1

15	1
	2 1
	3 10 5 16 8 4 2 1
	4 2 1
	5 16 8 4 2 1
	6 3 10 5 16 8 4 2 1
	7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
	8 4 2 1
	9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
	10 5 16 8 4 2 1
	11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
	12 6 3 10 5 16 8 4 2 1
	13 40 20 10 5 16 8 4 2 1
	14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
	15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1

(b) 2 points **Problem Identification** Briefly explain the underlying problem you identified in the above question that led you to your solution.

Solution:
Input: `n`
Output: `n` printed lines with line `i` containing the Collatz sequence for `i`

(c) 3 points **Pseudocode** After defining the function, make sure to call it and print its result. You may assume that the input has already been taken.

Solution:

```
def show_collatz(n):
    for i in range(1, n+1):
        j = i
        while j > 1:
            print(j, end = ' ')
            if j % 2 == 0:
                j = j // 2
            else:
                j = 3*j + 1
        print(1)

# input n
show_collatz(n)
```

PLE SOLU

(d) $\boxed{\text{2 points}}$ **Dry Run** In the space below, show a dry run of your pseudocode using any input from the Exercise section. Include a column for the output

Solution:

n	i	j	Output
4	1	1	1
	2	2	1 2
		1	1 2 1

i	j	Output

3	3	1
		2 1
		3

	10	1
		2 1
		3 10

	5	1
		2 1
		3 10 5

	16	1
		2 1
		3 10 5 16

	8	1
		2 1
		3 10 5 16 8

	4	1
		2 1
		3 10 5 16 8 4

	2	1
		2 1
		3 10 5 16 8 4 2

	1	1
		2 1
		3 10 5 16 8 4 2 1

i	j	Output

4	4	1
		2 1
		3 10 5 16 8 4 2 1
		4

	2	1
		2 1
		3 10 5 16 8 4 2 1
		4 2

	1	1
		2 1
		3 10 5 16 8 4 2 1
		4 2 1

SAMPLE