Task 1

Define a function *filter_strings_containing_a* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["apple", "banana", "cherry", "date"]

When called, the function should return a new list containing only strings that contain the letter “a”.

Starter Code

```
def filter_strings_containing_a(input_strs: list[str])
    -> list[str]:
    # Your implementation here

print(filter_strings_containing_a(
    ["apple", "banana", "cherry", "date"])
)
```

Examples

```
Input:  ["apple", "banana", "cherry", "date"]
Output: ["apple", "banana", "date"]

Input:  []
Output: []

Input:  ["bbbb", "cccc"]
Output: []
```

Task 2

Define a function *sum_if_less_than_fifty* that takes two parameters:

Name	Type	Example Input
num_one	int	20
num_two	int	25

When called, the function should return either:

- The sum of the two numbers if the sum is less than 50
- *None* if the sum of the two numbers is more than or equal to 50

Starter Code

```
def sum_if_less_than_fifty(num_one: int, num_two: int)
    -> int | None:
    # Your implementation here

print(sum_if_less_than_fifty(20, 20))
```

Examples

```
Inputs:
 - num_one = 20
 - num_two = 20
Output: 40
Inputs:
 - num_one = 20
 - num_two = 30
Output: None
Inputs:
 - num_one = 20
 - num_two = 100
Output: None
```

Task 3

Define a function *sum_even* that takes one parameter:

Name	Type	Example Input
input_nums	list of int	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

When called, the function should return the sum of even integers in the list.

Starter Code

```
def sum_even(input_nums: list[int]) -> int:
    # Your implementation here

print(sum_even([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
```

Examples

```
Input: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Output: 30

Input: [10, 20, 30, 40, 50]
Output: 150

Input: [9, 7, 5, 3, 1]
Output: 0
```

Task 4

Define a function *remove_vowels* that takes one parameter:

Name	Type	Example Input
input_str	str	"Hello, World!"

When called, the function should return a new string with all the vowels removed.

Starter Code

```
def remove_vowels(input_str: str) -> str:
    # Your implementation here

print(remove_vowels("Hello, World!"))
```

Examples

```
Input: "Hello, World!"
Output: "Hll, Wrld!"

Input: "aeiouAEIOU"
Output: ""

Input: "zzxxxccvvvbbnnmmmLLKKJJHH"
Output: "zzxxxccvvvbbnnmmmLLKKJJHH"
```

Task 5

Define a function *get_longest_string* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["cat", "dog", "bird", "lizard"]

When called, the function should return the longest string in the list. If there are ties, return the string that appears first in the list.

Starter Code

```
def get_longest_string(input_strs: list[str]) -> str:
    # Your implementation here

print(get_longest_string(["cat", "dog", "bird", "lizard"]))
```

Examples

```
Input: ["cat", "dog", "bird", "lizard"]
Output: "lizard"

Input: ["cat", "dog", "bird", "wolf"]
Output: "bird"

Input: ["a", "b", "c", "d"]
Output: "a"
```

Task 6

Define a function *filter_even_length_strings* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["cat", "dog", "fish", "elephant"]

When called, the function should return a new list with all the strings that have an even number of characters.

Starter Code

```
def filter_even_length_strings(input_strs: list[str])
    -> list[str]:
    # Your implementation here

print(filter_even_length_strings(
    ["cat", "dog", "fish", "elephant"]
))
```

Examples

```
Input: ["cat", "dog", "fish", "elephant"]
Output: ["fish", "elephant"]

Input: ["q", "w", "e", "r", "t", "y"]
Output: []

Input: ["qq", "ww", "ee", "rr", "tt", "yy"]
Output: ["qq", "ww", "ee", "rr", "tt", "yy"]
```

Task 7

Define a function *reverse_elements* that takes one parameter:

Name	Type	Example Input
input_nums	list of int	[1, 2, 3, 4, 5]

When called, the function should return a new list with all of the elements in the original list reversed.

Starter Code

```
def reverse_elements(input_nums: list[int]) -> list[int]:
    # Your implementation here

print(reverse_elements([1, 2, 3, 4, 5]))
```

Examples

```
Input: [1, 2, 3, 4, 5]
Output: [5, 4, 3, 2, 1]

Input: []
Output: []

Input: [20, 15, 25, 10, 30, 5, 0]
Output: [0, 5, 30, 10, 25, 15, 20]
```

Task 8

Define a function *filter_type_str* that takes one parameter:

Name	Type	Example Input
input_list	list of str or int	["hello", 1, 2, "www"]

When called, the function should return a new list containing only the strings from the original list.

Starter Code

```
def filter_type_str(input_list: list[str | int]) -> list[str]:
    # Your implementation here

print(filter_type_str(["hello", 1, 2, "www"]))
```

Examples

```
Input: ["hello", 1, 2, "www"]
Output: ["hello", "www"]

Input: []
Output: []

Input: [1, 2, 3, 4, 5]
Output: []
```

Task 9

Define a function *string_to_morse_code* that takes one parameter:

Name	Type	Example Input
input_str	str	"HELLO, WORLD!"

When called, the function should return the more code equivalent of the input string. The function should meet the following requirements:

A morse code “dot” should be represented by a full stop

A more code “dash” should be represented by a hyphen

A space should be used between each morse code letter e.g. “.-**(space)**-...”

The function should be able to support the following characters in the input string:

Alphanumeric characters, uppercase and lowercase

Special characters: , . : ? ‘ - / ( ) “ @ = + !

Should a space be encountered in the input string, it should be represented as a forward slash in the output string

Starter Code

```
def string_to_morse_code(input_str: str) -> str:
    morse_dict = {"a": ".-",    "b": "-...",  "c": "-.-.",
                  "d": "-..",   "e": ".",     "f": "..-.",
                  "g": "--.",   "h": "....",  "i": "..",
                  "j": ".---",  "k": "-.-",   "l": ".-..",
                  "m": "--",    "n": "-.",    "o": "---",
                  "p": ".--.",  "q": "--.-",  "r": ".-.",
                  "s": "...",   "t": "-",     "u": "..-",
                  "v": "...-",  "w": ".--",   "x": "-..-",
                  "y": "-.--",  "z": "--..",  "0": "-----",
                  "1": ".----", "2": "..---", "3": "...--",
                  "4": "....-", "5": ".....", "6": "-....",
                  "7": "--...", "8": "---..", "9": "----.",
                  ",": "--..--",".": ".-.-.-",":": "---...",
                  "?": "..--..","'": ".----.","-": "-....-",
                  "/": "-..-.", "(": "-.--.", ")": "-.--.-",
                  '"': ".-..-.","@": ".--.-.","=": "-...-",
                  "+": ".-.-.", "!": "-.-.--"}

    # Your implementation here


print(string_to_morse_code("HELLO, WORLD!"))
```

Examples

Input: "HELLO, WORLD!"
Output: ".... . .-.. .-.. --- --..-- / .-- --- .-. .-.. -.. -.-.--"

Input: "abcdefghijklmnopqrstuvwxyz,.:?'-/()\"@=+!"
Output: ".- -... -.-. -.. . ..-. --. ....--- -.- .-.. -- -. --- .--. --.- .-. ... - ..- ...- .-- -..- -.-- --.. --..-- .-.-.- ---... ..--.. .----. -....- -..-. -.--. -.--.- .-..-. .--.-. -...- .-.-. -.-.--"

Input: ""
Output: ""

Task 10

Define a function *get_second_largest_number* that takes one parameter:

Name	Type	Example Input
input_nums	list of int	[1, 2, 3, 4, 5]

When called, the function should return the second largest number in the list. If there is no second largest number, the function should return *None*.

Starter Code

```
def get_second_largest_number(input_nums: list[int])
    -> int | None:
    # Your implementation here

print(get_second_largest_number([1, 2, 3, 4, 5]))
```

Examples

```
Input: [1, 2, 3, 4, 5]
Output: 4

Input: [3, 45, 345, 435, 345, 43, 56, 34, 234, 34]
Output: 345

Input: [1]
Output: None
```

Task 11

Define a function *format_number_with_commas* that takes one parameter:

Name	Type	Example Input
input_num	int	1000000

When called, the function should return a string representation of the number with commas as thousand separators.

Starter Code

```
def format_number_with_commas(input_num: int) -> str:
    # Your implementation here

print(format_number_with_commas(1000000))
```

Examples

```
Input: 1000000
Output: "1,000,000"

Input: 12345
Output: "12,345"

Input: -99999999
Output: "-99,999,999"
```

Task 12

Background: ASCII is a standard data-encoding format for electronic communication between computers. ASCII assigns standard numeric values to letters, numerals, punctuation marks, and other characters used in computers.

To convert a character to its ASCII code equivalent we can reference an ASCII table: https://www.asciitable.com

The table has five headings: Dec, Hex, Oct, Html and Chr. To convert a character to its ASCII equivalent we find it in the Chr column and then take the corresponding number from the Dec column. For example, we can see “A” has the value of 65.
The conversion can also happen the other way, if we take the ASCII code 65 and look it up in the ASCII table, we can see it matches up with the “A” character.

Task: Define a function *string_to_ascii* that takes one parameter:

Name	Type	Example Input
input_str	str	"Programming puzzles!"

When called, the function should return a list containing the ASCII numeric codes of each character of the string.

Starter Code

```
def string_to_ascii(input_str: str) -> list[int]:
    # Your implementation here

print(string_to_ascii("Programming puzzles!"))
```

Examples

```
Input: "Programming puzzles!"
Output: [80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32,
112, 117, 122, 122, 108, 101, 115, 33]

Input: ""
Output: []

Input: "aA"
Output: [97, 65]
```

Task 12.1

Define a function *ascii_to_string* that takes one parameter:

Name	Type	Example Input
input_ascii_codes	list of int	[80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32, 112, 117, 122, 122, 108, 101, 115, 33]

When called, the function should return a string consisting of the converted ASCII codes back to their char equivalents.

Starter Code

```
def ascii_to_string(input_ascii_codes: list[int]) -> str:
    # Your implementation here

print(ascii_to_string([80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32, 112, 117, 122, 122, 108, 101, 115, 33]))
```

Examples

```
Input: [80, 114, 111, 103, 114, 97, 109, 109, 105, 110, 103, 32, 112, 117, 122, 122, 108, 101, 115, 33]
Output: "Programming puzzles!"

Input: []
Output: ""

Input: [97, 65]
Output: "aA"
```

Task 13

Define a function *filter_strings_with_vowels* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["apple", "banana", "zyxvb"]

When called, the function should return a new list with all the strings that have at least one vowel.

Starter Code

```
def filter_strings_with_vowels(input_strs: list[str])
    -> list[str]:
    # Your implementation here

print(filter_strings_with_vowels(["apple", "banana", "zyxvb"]))
```

Examples

```
Input: ["apple", "banana", "zyxvb"]
Output: ["apple", "banana"]

Input: []
Output: []

Input: ["q", "w", "e", "r", "t", "y"]
Output: ["e"]
```

Task 14

Define a function *reverse_first_five_positions* that takes one parameter:

Name	Type	Example Input	Constraint
input_nums	list of int	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	len(input_nums) == 10

When called, the function should return a new list with the first five elements of the original list reversed. The solution should make use of python slicing and should not use loops.

Starter Code

```
def reverse_first_five_positions(input_nums: list[int])
    -> list[int]:
    # Your implementation here

print(reverse_first_five_positions(
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
))
```

Examples

```
Input: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Output: [5, 4, 3, 2, 1, 6, 7, 8, 9, 10]

Input: [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
Output: [60, 70, 80, 90, 100, 50, 40, 30, 20, 10]

Input: [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
Output: [-5, -4, -3, -2, -1, -6, -7, -8, -9, -10]
```

Task 15

Define a function *filter_palindromes* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["cat", "dog", "racecar", "deified", "giraffe"]

When called, the function should return a new list that contains only the strings that are palindromes.

Starter Code

```
def filter_palindromes(input_strs: list[str]) -> list[str]:
    # Your implementation here

print(filter_palindromes(
    ["cat", "dog", "racecar", "deified", "giraffe"]
))
```

Examples

```
Input: ["cat", "dog", "racecar", "deified", "giraffe"]
Output: ["racecar", "deified"]

Input: ["kayak", "deified", "rotator", "repaper", "deed", "a"]
Output: ["kayak", "deified", "rotator", "repaper", "deed", "a"]

Input: ["ab", "ba", "cd", "ef", "pt"]
Output: []
```

Task 16

Define a function *censor_python* that takes one parameter:

Name	Type	Example Input
input_strs	list of str	["python", "hello", "HELLO"]

When called, the function should return a new list of strings with the letters “P”, “Y”, “T”, “H”, “O”, “N” replaced with “X”, the solution should be case insensitive.

Starter Code

```
def censor_python(input_strs: list[str]) -> list[str]:
    # Your implementation here

print(censor_python(["python", "hello", "HELLO"]))
```

Examples

```
Input: ["python", "hello", "HELLO"]
Output: ["XXXXXX", "XellX", "XELLX"]

Input: ["abcdefg"]
Output: ["abcdefg"]

Input: []
Output: []
```

Task 17

Background: A string is happy if every three consecutive characters are distinct.

Example happy strings: “abcdefg”, “qwerty”, “abcabcabcabc”
Example unhappy strings: “aaaaaaaa”, “cbc”, “hello”

Task : Define a function *check_if_string_is_happy* that takes one parameter:

Name	Type	Example Input
input_str	str	"abcdefg"

When called, the function should return a bool indicating if the input string is happy or not.

Starter Code

```
def check_if_string_is_happy(input_str: str) -> bool:
    # Your implementation here

print(check_if_string_is_happy("abcdefg"))
```

Examples

```
Input: "abcdefg"
Output: True

Input: "abcabcabcabc"
Output: True

Input: "hello"
Output: False
```

Task 18

Define a function *get_number_of_digits* that takes one parameter:

Name	Type	Example Input	Constraint
input_num	int	1234	input_num >= 0

When called, the function should return the number of digits in the *input_num*.

Caveats:

- The function should be recursive
- The function should not convert the integer to a string

Starter Code

```
def get_number_of_digits(input_num: int) -> int:
    # Your implementation here

print(get_number_of_digits(1234))
```

Examples

```
Input: 1234
Output: 4

Input: 0
Output: 1

Input: 123456789
Output: 9
```

Task 19

Background

Tic-Tac-Toe (also known as naughts and crosses) is a two-player game played on a 3x3 grid. The objective of the game is to be the first player to get three of their symbols in a row, either horizontally, vertically, or diagonally. The game is played by alternating turns, with each player placing their symbol (usually an X or an O) on an empty space on the board until one player achieves a winning configuration or the board is filled without a winner, resulting in a tie.
A Tic-Tac-Toe board can be represented in python by a 2-dimensional list, with each of the inner lists representing a row. Let's look at an example:

```
input_board = [["X", "X", "X"],
               ["O", "X", "O"],
               ["X", "O", "O"]]

Represents the following game configuration:
X X X
O X O
X O O
```

Task

Define a function *get_tic_tac_toe_winner* that takes one parameter:

Name	Type	Example Input
input_board	list of list of str	[["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]

When called, the function should determine if X or O has won. If there is a draw the function should return *None*.

Starter Code

```
def get_tic_tac_toe_winner(input_board: list[list[str]])
    -> str | None:
    # Your implementation here

print(get_tic_tac_toe_winner(
    [["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]
))
```

Examples

```
Input: [["X", "X", "X"], ["O", "X", "O"], ["X", "O", "O"]]
Output: "X"

Input: [["X", "O", "O"], ["O", "O", ""], ["X", "O", "O"]]
Output: "O"

Input: [["X", "O", "O"], ["O", "X", ""], ["X", "O", "O"]]
Output: None
```

Task 20

Define a function *print_triangle* that takes two parameters:

Name	Type	Example Input
number_of_levels	int	4
symbol	str	“*”

When called, the function should output a centred triangle shape made up of the desired symbol. The number of symbols in each row should increase by two with each level, starting with one symbol on the first level, then three on the second level and so on until the final level is reached. For example, for *number_of_levels = 4* and *symbol = ”*”* the following triangle should be produced:

```
   *
  ***
 *****
*******
```

Starter Code

```
def print_triangle(number_of_levels: int, symbol: str) -> None:
    # Your implementation here

print_triangle(4, "*")
```

Examples

```
Inputs:
 - number_of_levels = 3
 - symbol = "*"

Output:
  *
 ***
*****

Inputs:
 - number_of_levels = 1
 - symbol = "|"

Output:
|
```

Task 21

Background : The Fibonacci sequence is a well-known mathematical series of numbers that has fascinated mathematicians and scientists for centuries. It is a sequence of numbers where each number in the sequence is the sum of the two preceding numbers. The sequence starts with 0, 1, and each subsequent number is the sum of the previous two numbers.
The sequence goes 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on.

Task : Define a function *fibonacci* that takes one parameter:

Name	Type	Example Input
sequence_number	int	4

When called, the function should return the corresponding fibonacci sequence number (starting from 0).
Caveats:

The function should be recursive.

Starter Code

```
def fibonacci(sequence_number: int) -> int:
    # Your implementation here

print(fibonacci(4))
```

Examples

```
Input: 4
Output: 3

Input: 0
Output: 0

Input: 6
Output: 8
```

Task 22

Background: A harmonic sum is a mathematical concept that is used to calculate the sum of the reciprocals of a set of numbers. The reciprocal of a number is defined as 1 divided by the number. For example, the reciprocal of 2 is 0.5, because 1 divided by 2 is 0.5.
In the case of a harmonic sum, the set of numbers is usually represented by the natural numbers from 1 to n. The formula for the harmonic sum of n is: Hn = 1/1 + 1/2 + 1/3 + ... + 1/n.

Task: Define a function *harmonic_sum* that takes one parameter:

Name	Type	Example Input
n	int	5

When called, the function should return the harmonic sum of *n*.
Caveats:

The function should be recursive.

Starter Code

```
def harmonic_sum(n: int) -> float:
    # Your implementation here

print(harmonic_sum(5))
```

Examples

```
Input: 5
Output: 2.283

Input: 2
Output: 1.5

Input: 0
Output: 0
```

Task 23

Background

The XOR (Exclusive Or) logic gate is a digital circuit that performs a logical operation on two inputs. The output of an XOR gate is true (1) if and only if one of the inputs is true and the other is false (0). In other words, the XOR gate compares the inputs, and if they are different, the output is true. If they are the same, the output is false.
This can be represented in a truth table, which is a table that shows the output of a logic gate based on all possible combinations of inputs. Here is an example of a truth table for the XOR gate:

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	0

In this example, the left column represents input A, the middle column represents input B, and the right column represents the output. As you can see, when the inputs are different, the output is 1 and when the inputs are the same, the output is 0.

Task

Define a function *xor* that takes two parameters:

Name	Type	Example Input
input_a	str	“1101”
input_b	str	“0001”

When called, the function should return a string value that is the result of XOR’ing the two input strings together. If one string is longer than the other the excess characters should be ignored from the result.

Starter Code

```
def xor(input_a: str, input_b: str) -> str:
    # Your implementation here

print(xor("1101", "0001"))
```

Examples

```
Inputs:
 - input_a: "1111"
 - input_b: "1111"

Output: "0000"

Inputs:
 - input_a: "1111"
 - input_b: "0000"

Output: "1111"

Inputs:
 - input_a: "1101"
 - input_b: "00010"

Output: "1100"
```

Task 24

Background

In Python, the zip function takes in one or more iterable objects (such as lists, tuples, or strings) and returns an iterator of tuples. Each tuple contains elements from each iterable object in the same position. The zip function stops when it reaches the end of the shortest iterable object.
Here is an example of how the zip function works:

```
a = [1, 2, 3]
b = [4, 5, 6]

zipped = zip(a, b)
print(list(zipped)) # [(1, 4), (2, 5), (3, 6)]
```

Task

Define a function *my_zip* that takes two parameters:

Name	Type	Example Input
input_list_a	list of Any	[1, 2, 3, 4]
input_list_b	list of Any	[5, 6, 7, 8]

When called, the function should return the same result as python's built-in zip function.

Starter Code

```
from typing import Any

def my_zip(input_list_a: list[Any], input_list_b: list[Any])
    -> list[tuple[Any, Any]]:
    # Your implementation here

print(my_zip([1, 2, 3, 4], [5, 6, 7, 8]))
```

Examples

```
Inputs:
 - input_list_a: [1, 2, 3, 4]
 - input_list_b: [5, 6, 7, 8]

Output: [(1, 5), (2, 6), (3, 7), (4, 8)]

Inputs:
 - input_list_a: []
 - input_list_b: []

Output: []

Inputs:
 - input_list_a: [1, 2, 3]
 - input_list_b: [5, 6, 7, 8]

Output: [(1, 5), (2, 6), (3, 7)]
```

Task 25

Define a function *is_valid_equation* that takes one parameter:

Name	Type	Example Input	Constraint
input_equation	str	“2 + 3 = 5”	The string should consist of an integer, followed by a plus or a minus sign, followed by another integer, an equals sign, and the answer. The equation should be separated by spaces.

When called, the function should return a boolean value indicating whether the equation is valid or not. The equation is classed as valid if:

- It’s in the correct format as specified above
- Both sides of the equation evaluate to the same number

Starter Code

```
def is_valid_equation(input_equation: str) -> bool:
    # Your implementation here

print(is_valid_equation("2 + 3 = 5"))
```

Examples

```
Input: "2 + 3 = 5"
Output: True
Input: "-5 + -6 = -11"
Output: True
Input: "-2 + 3 = -5"
Output: False
```

Task 26

Define a function *rotate_list_left* that takes two parameters:

Name	Type	Example Input
input_list	list of Any	[1, 2, 3, 4, 5]
rotate_amount	int	2

When called, the function should return a new list with the elements of the original list rotated left by the specified number of positions.

Caveats:

- The solution should not make use of loops
- The function should still work if the rotation amount is greater than the length of the list, e.g. a rotation amount of 6 on a list of length 5 will produce the same result as if the rotation amount was 1.

Starter Code

```
from typing import Any

def rotate_list_left(input_list: list[Any], rotate_amount: int)
    -> list[Any]:
    # Your implementation here

print(rotate_list_left([1, 2, 3, 4, 5], 2))
```

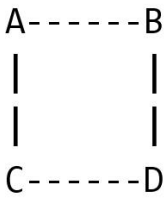
Examples

```
Inputs:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 2
Output: [3, 4, 5, 1, 2]

Inputs:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 5
Output: [1, 2, 3, 4, 5]

Inputs:
 - input_list: [1, 2, 3, 4, 5]
 - rotate_amount: 7
Output: [3, 4, 5, 1, 2]
```

Task 27

Background

An undirected graph is a mathematical representation of a set of objects in which some pairs of the objects are connected by links. The objects are represented by vertices (or nodes) and the links are represented by edges.
In an undirected graph, the edges have no direction, meaning they can be traversed in both directions. It means if there is an edge between vertex A and B, then it is also true that there is an edge between vertex B and A.
A simple example of an undirected graph would be a group of cities and the roads connecting them. Each city would be represented by a vertex, and the roads connecting the cities would be represented by edges. It would not matter whether you are travelling from city A to city B or from city B to city A, the edge between them would still be the same.
It could look something like this:



Each letter represents a vertex (city) and the lines represent edges (roads). Here, A and B are connected by an edge and A and C are also connected by an edge.
Graphs are commonly represented in python using an adjacency matrix. The concept is simple, we have the list of nodes across the top and a list of nodes down the side. If two nodes are connected we place a “1” in the connection cell. If they’re not connected we have a 0.
Example: Node A is connected to itself, Node B and Node C. Node B is only connected to node A. Node C is only connected to Node A.

	Node A	Node B	Node C
Node A	1	1	1
Node B	1	0	0
Node C	1	0	0

In python, this can be represented as a list of lists where each node is represented by an index. For example Nodes A, B, C would be represented by indexes 0, 1, 2 respectively:

```
graph = [
    [1, 1, 1],
    [1, 0, 0],
    [1, 0, 0]
]

graph[0][0] == 1 # Is Node A connected to itself?
graph[0][1] == 1 # Is Node A connected to Node B?
graph[2][1] == 1 # Is Node C connected to Node B?
```

Task

Define a function *find_adjacent_nodes* that takes two parameters:

Name	Type	Example Input
adj_matrix	list of list of int	[[1, 1, 1], [1, 0, 0], [1, 0, 0]]
start_node	int	0

When called, the function should return a list of all nodes that are adjacent to *start_node*.

Challenge: this challenge can be achieved with the function body being only one line of code, are you able to find the solution?

Starter Code

```
def find_adjacent_nodes(
    adj_matrix: list[list[int]],
    start_node: int
) -> list[int]:

    # Your implementation here

print(
    find_adjacent_nodes([[1, 1, 1], [1, 0, 0], [1, 0, 0]], 0)
)
```

Examples

```
Inputs:
 - adj_matrix: [[1, 1, 1], [1, 0, 0], [1, 0, 0]]
 - start_node: 0
Output: [0, 1, 2]

Inputs:
 - adj_matrix: [[1, 1, 1], [1, 0, 0], [1, 0, 0]]
 - start_node: 1
Output: [0]

Inputs:
 - adj_matrix: [[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 1], [0, 1, 1,
0]]
 - start_node: 1
Output: [0, 3]
```

Task 28

Background

In technical analysis, a peak refers to a high point or local maximum in the price of an asset. A valley, on the other hand, refers to a low point or local minimum in the price of an asset. When looking at a chart of the price of an asset, peaks and valleys can help traders identify potential turning points and potential changes in the overall trend of the asset.
For our sake, peaks and valleys are defined as the following:

Peak:
A peak must have one or more numbers in ascending order leading up to it.
A peak must have one or more numbers in descending order after it.
A peak can not occur at the start or end of price action.
Valley
A valley must have one or more numbers in descending order leading up to it.
A valley must have one or more numbers in ascending order after it.
A valley can not occur at the start or end of price action.

Task

Define a function *count_peaks_valleys* that takes one parameter:

Name	Type	Example Input
price_action	list of int	[1, 2, 3, 2, 1]

When called, the function should return a tuple representing how many peaks and valleys are in the given price action. The tuple should contain two integers, the first representing the number of peaks and the second representing the number of valleys.

Starter Code

```
def count_peaks_valleys(price_action: list[int])
    -> tuple[int, int]:
    # Your implementation here

print(count_peaks_valleys([1, 2, 3, 2, 1]))
```

Examples

```
Input: [1, 2, 3, 2, 1]
Output: (1, 0)

Input: [1, 2, 3, 2, 1, 2]
Output: (1, 1)

Input: [7, 6, 5, 10, 11, 12, 10, 9, 10]
Output: (1, 2)
```

Task 29

Background

Tap code is a simple method for transmitting messages through a series of taps or knocks. It was originally developed for prisoners of war to communicate with each other in secret, but it has also been used in other situations where communication is difficult, such as by hikers lost in the wilderness or by people trapped in a collapsed building. Tap code uses the following 5x5 grid to represent each letter:

	1	2	3	4	5
1	A	B	C / K	D	E
2	F	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Note: C and K are both represented by (1, 3) - when coding you can choose to either return C or K.

To transmit a message, the sender taps out the row and column of each letter in the message, with a pause between letters and a longer pause between words. The receiver then uses tap code to decode the message.

As an example, the word “water” would be represented by:

5, 2
1, 1
4, 4
1, 5
4, 2

Or when using in real life a series of taps: ••••• •• • • •••• •••• • ••••• •••• ••

Task

Define a function *tap_code_to_english* that takes one parameter:

Name	Type	Example Input

input_code	str	".. ... ."

When called, the function should return the converted sentence. Items within the string should be represented as the following:

Letters: 1-5 dots followed by a space, followed by 1-5 dots e.g. “. ..” = “b”

End of letter / start of new letter: two spaces e.g. “. .. . ..” = “bb”

End of word / start of new word: three spaces “.. ... .” = “hi hi”

Starter Code

```
tap_code_map = {
    "a": ". .", "b": ". ..", "c": ". ...", "d": ". ....",
    "e": ". .....", "f": ".. .", "g": ".. ..", "h": ".. ...",
    "i": ".. ....", "j": ".. .....", "l": "... .",
    "m": "... ..", "n": "... ...", "o": "... ....",
    "p": "... .....", "q": ".... .", "r": ".... ..",
    "s": ".... ...", "t": ".... ....", "u": ".... .....",
    "v": "..... .", "w": "..... ..", "x": "..... ...",
    "y": "..... ....", "z": "..... ....."
}

def tap_code_to_english(input_code: str) -> str:
    # Your implementation here

print(tap_code_to_english(".. ...  .. ....   .. ...  .. ...."))
```

Examples

```
Input: ".. ...  .. ....   .. ...  .. ...."
Output: "hi hi"

Input: ". ...  ... ....  ... ....  ... ."
Output: "cool"

Input: ""
Output: ""
```

Task 30

Define a function *find_zero_sum_triplets* that takes one parameter:

Name	Type	Example Input
input_nums	list of int	[1, 2, 3, 4, 5, -9]

When called, the function should return all possible combinations of the indexes of three numbers that add up to 0. The function should return a list of tuples, each tuple containing the three indexes of the numbers that add up to 0, or an empty list if no such combination exists. The function should be able to deal with duplicate numbers in the input list.

Starter Code

```
def find_zero_sum_triplets(input_nums: list[int])
    -> list[tuple[int, int, int]]:
    # Your implementation here

print(find_zero_sum_triplets([1, 2, 3, 4, 5, -9]))
```

Examples

```
Input: [1, 2, 3, 4, 5]
Output: []

Input: [1, 2, 3, 4, 5, -9]
Output: [(3, 4, 5)]

Input: [1, 2, 3, 4, 5, -9, -9]
Output: [(3, 4, 5), (3, 4, 6)]
```