Question1

Write a function that takes a list and a number as arguments. Add the number to the end of the list, then remove the first element of the list. The function should then return the updated list.

**Examples**

next\_in\_line([5, 6, 7, 8, 9], 1) ➞ [6, 7, 8, 9, 1]

next\_in\_line([7, 6, 3, 23, 17], 10) ➞ [6, 3, 23, 17, 10]

next\_in\_line([1, 10, 20, 42 ], 6) ➞ [10, 20, 42, 6]

next\_in\_line([], 6) ➞ "No list has been selected"

ANS: def next\_in\_line(lst, num):

if not lst:

return "No list has been selected"

lst.pop(0) # Remove the first element of the list

lst.append(num) # Add the number to the end of the list

return lst

# Test cases

print(next\_in\_line([5, 6, 7, 8, 9], 1)) # ➞ [6, 7, 8, 9, 1]

print(next\_in\_line([7, 6, 3, 23, 17], 10)) # ➞ [6, 3, 23, 17, 10]

print(next\_in\_line([1, 10, 20, 42], 6)) # ➞ [10, 20, 42, 6]

print(next\_in\_line([], 6)) # ➞ "No list has been selected"

Question2

Create the function that takes a list of dictionaries and returns the sum of people's budgets.

### Examples

get\_budgets([

{ "name": "John", "age": 21, "budget": 23000 },

{ "name": "Steve", "age": 32, "budget": 40000 },

{ "name": "Martin", "age": 16, "budget": 2700 }

]) ➞ 65700

get\_budgets([

{ "name": "John", "age": 21, "budget": 29000 },

{ "name": "Steve", "age": 32, "budget": 32000 },

{ "name": "Martin", "age": 16, "budget": 1600 }

]) ➞ 62600

ANS: def get\_budgets(people\_list):

return sum(person["budget"] for person in people\_list)

# Test cases

print(get\_budgets([

{ "name": "John", "age": 21, "budget": 23000 },

{ "name": "Steve", "age": 32, "budget": 40000 },

{ "name": "Martin", "age": 16, "budget": 2700 }

])) # ➞ 65700

print(get\_budgets([

{ "name": "John", "age": 21, "budget": 29000 },

{ "name": "Steve", "age": 32, "budget": 32000 },

{ "name": "Martin", "age": 16, "budget": 1600 }

])) # ➞ 62600

Question3

Create a function that takes a string and returns a string with its letters in alphabetical order.

### Examples

alphabet\_soup("hello") ➞ "ehllo"

alphabet\_soup("edabit") ➞ "abdeit"

alphabet\_soup("hacker") ➞ "acehkr"

alphabet\_soup("geek") ➞ "eegk"

alphabet\_soup("javascript") ➞ "aacijprstv"

ANS: def alphabet\_soup(input\_string):

return ''.join(sorted(input\_string))

# Test cases

print(alphabet\_soup("hello")) # ➞ "ehllo"

print(alphabet\_soup("edabit")) # ➞ "abdeit"

print(alphabet\_soup("hacker")) # ➞ "acehkr"

print(alphabet\_soup("geek")) # ➞ "eegk"

print(alphabet\_soup("javascript")) # ➞ "aacijprstv"

Question4

Suppose that you invest $10,000 for 10 years at an interest rate of 6% compounded monthly. What will be the value of your investment at the end of the 10 year period?

Create a function that accepts the principal p, the term in years t, the interest rate r, and the number of compounding periods per year n. The function returns the value at the end of term rounded to the nearest cent.

For the example above:

compound\_interest(10000, 10, 0.06, 12) ➞ 18193.97

Note that the interest rate is given as a decimal and n=12 because with monthly compounding there are 12 periods per year. Compounding can also be done annually, quarterly, weekly, or daily.

### Examples

compound\_interest(100, 1, 0.05, 1) ➞ 105.0

compound\_interest(3500, 15, 0.1, 4) ➞ 15399.26

compound\_interest(100000, 20, 0.15, 365) ➞ 2007316.26

ANS: To calculate the compound interest, we can use the formula:

A = P \* (1 + r/n)^(n\*t)

Where:

A = the final amount after t years

P = the principal amount (initial investment)

r = the interest rate (in decimal form)

n = the number of compounding periods per year

t = the number of years

Let's create the `compound\_interest` function using this formula:

def compound\_interest(p, t, r, n):

A = p \* (1 + r/n) \*\* (n\*t)

return round(A, 2)

# Test cases

print(compound\_interest(10000, 10, 0.06, 12)) # ➞ 18193.97

print(compound\_interest(100, 1, 0.05, 1)) # ➞ 105.0

print(compound\_interest(3500, 15, 0.1, 4)) # ➞ 15399.26

print(compound\_interest(100000, 20, 0.15, 365)) # ➞ 2007316.26

The `compound\_interest` function takes the principal amount `p`, the term in years `t`, the interest rate `r`, and the number of compounding periods per year `n` as input. It calculates the final amount `A` using the formula and returns the result rounded to the nearest cent using `round(A, 2)`.

Question5

Write a function that takes a list of elements and returns only the integers.

### Examples

return\_only\_integer([9, 2, "space", "car", "lion", 16]) ➞ [9, 2, 16]

return\_only\_integer(["hello", 81, "basketball", 123, "fox"]) ➞ [81, 123]

return\_only\_integer([10, "121", 56, 20, "car", 3, "lion"]) ➞ [10, 56, 20, 3]

return\_only\_integer(["String", True, 3.3, 1]) ➞ [1]

ANS: You can create the function `return\_only\_integer` using list comprehension to filter out only the integer elements from the given list. Here's the implementation:

def return\_only\_integer(elements):

return [x for x in elements if isinstance(x, int)]

# Test cases

print(return\_only\_integer([9, 2, "space", "car", "lion", 16])) # ➞ [9, 2, 16]

print(return\_only\_integer(["hello", 81, "basketball", 123, "fox"])) # ➞ [81, 123]

print(return\_only\_integer([10, "121", 56, 20, "car", 3, "lion"])) # ➞ [10, 56, 20, 3]

print(return\_only\_integer(["String", True, 3.3, 1])) # ➞ [1]

The function `return\_only\_integer` takes a list of elements as input. It uses list comprehension to filter out only the elements that are integers (using `isinstance(x, int)`) and returns the new list containing only the integer elements.